LAPPEENRANTA UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

MASTER'S THESIS

The Usage of Microsoft Push Notification Service on Mobile Devices

The topic of Master's Thesis was approved by the council of the Department of Information Technology on 04.01.2011

Supervisors: Professor Jari Porras
M.Sc (Tech) Tommi Kallonen

Lappeenranta, May 12th, 2011

Qianting Liu

Ruskonlahdenkatu 13–15 F10

53850 Lappeenranta

Mobile: +358 442110214
qianting.liu@lut.fi

ABSTRACT

Lappeenranta University of Technology

Department of Information Technology

Qianting Liu

The Usage of Microsoft Push Notification Service on Mobile Devices

Thesis for the Degree of Master of Science in Technology

2011

64 pages, 44 figures, 2 tables and 3 appendices

Examiners: Professor Jari Porras

M.Sc. (Tech) Tommi Kallonen

Keywords: Push Notification, MPNS, Push Technology

Along with the increasing in demand of mobile computing, Push Notification (PN) is widely used in mobile phones and other devices. PN allows the developer to send messages to the end users even when the client application is not running at the moment. This solves the problem produced by non-supported multi-tasking feature as well as saving battery life. Microsoft Push Notification Service (MPNS) is one solution to use PNs in Windows Phones. The thesis gives the developers an idea of how to use PNs by introducing MPNS, comparing MPNS with other Push Notification Services, usage of different PN types analysis, and PN simulation system implementation.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the essential and gracious support of many individuals. First and foremost I offer my sincerest gratitude to my supervisor, Professor Jari Porras and D.Sc. Tommi Kallonen. They supported me throughout my thesis with their patience and knowledge. There is a saying in Chinese: He who teaches me for one day is my father for life. I appreciate all the teaching and instruction from all the teachers from Lappeenranta University of Technology, especially in the department of Information Technology, who teach me all the knowledge and lead me on my way as one cup bright lamp. I also want to thank Susanna Koponen who has been helping me on my study plan.

I acknowledge the support from the colleagues in Microsoft Development Center Copenhagen, who gave me a great help on Microsoft Technology and professional skills.

My appreciation I give to my parents and my friend for their encouragement, my cousin Tommi for the supports all the way and his iPod Touch. Thanks to all the people who have been helping me these four years. With your support and help, I can do better on this thesis and all things in the future.

Qianting Liu

Table OF CONTENTS

ABBEVIATIONS

3G	3rd generation mobile telecommunications
APNS	Apple Push Notification Services
BBC	British Broadcasting Corporation
C2DM	Android Cloud to Device Messaging Framework
HTTP	Hypertext Transfer Protocol
IM	Instant Messaging
IMAP	Internet Message Access Protocol
iOS	iPhone OS
JSON	JavaScript Object Notation
MPNS	Microsoft Push Notification Service
OS	Operation Systems
PNS	Push Notification Service
POP3	Post Office Protocol 3rd
SMTP	Simple Mail Transfer Protocol
WCF	Windows Communication Foundation
WP	Windows Phone
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

1. Introduction

Google CEO Eric Schmidt said: “The future of computing is mobile.” [1] This quote indicates how powerful the mobile is and will be. Truly, the mobile is more and more multifunctional and complicated nowadays. The mobile is somehow replacing computers along with the improvement of mobile hardware, for example, internet surfing, social networking and video gaming

As demand increases, more and more mobile applications work is network-related, meaning more and more scenarios require the phone application to always be connected to a network [2]. In other words, background services need to be running when the application is not in the foreground, for example, displaying the weather or other information from the web, tracking game results, or IM applications.

All these application or scenarios have the same common feature: they are web-based application with web services, which need to interact with the device to give end users notifications that something of interest is occurring at the moment, even when the users are making a call or surfing the Internet. In the past, generally, **Polling Technology** (Figure 1) was used to meet this need so that the application was able to listen from the server on the web. One way of using polling technology to listen to the server is that the application can be scheduled to poll the web service to get information as much as it needs when the application occupies the foreground. Another way is to run a background service in order to frequently poll its corresponding web service to see whether there are any pending updates when the application is not in the foreground.

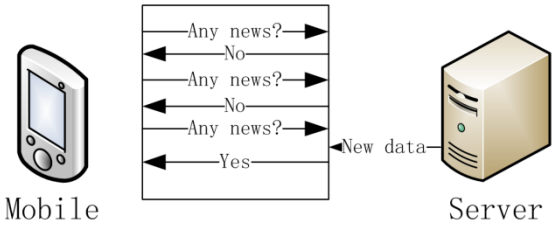

Figure 1 Polling Technology

Both of these two ways are expensive considering the battery usage and servers' workload [3]. On one hand, they both require the device to connect with the internet either through cellular network, for example 3G, or via Wi-Fi for a long time or all day to communicate with the web service. Although the first way as mentioned above, impact on battery life depends on the time and scope of how long the application is in foreground [4]. The second way to keep applications listening from the server has more problems related to battery consumption. Consider that if all the applications have a piece of code running in the background to update the information, and if they are running 24/7, the battery will be run down slowly and invisibly. The users of the device will never know how the battery is used as they cannot know if any program is running so that they can turn it off in order to prevent the battery from running out, but they will find out the phones are dead and useless when they want to make phone calls or listen to the music on the train. On the other hand, polling increases the servers' workload. Imagine that each client application is polling and accessing the server every half an hour. Mostly, there is no new data to sync to the device on the server, but the workload of the server is wasted correspondingly. The scheduled time for polling, such as every half an hour, is also not easy to estimate, as it is difficult to make the application real-time updated in half an hour increments [5][6].

But not all the mobile systems support multitasking and allow a background service from third-party applications running all the time. In this thesis, Microsoft Windows Phone is the mobile system that we will discuss about.

The problem is that the developer is not allowed to write a piece of code running in the background to listen for information from the web service for each application, because Windows Phone (WP) application model does not support third-party multitasking. This means you can make phone calls while reading the calendar (both of the programs are integrated in WP system), but you cannot read the weather from a weather application while playing chess using another application, both of which were downloaded from Marketplace, which provides tons of apps to users for Windows Phone to download. So how could WP give

the end-users a positive experience without wasting battery life of the device because of a large number of background processes? How could all the application be updated in real-time while multi-tasking is not supported by WP? The answer is **Microsoft Push Notification Service (MPNS)** since normal synchronization is not as useful at the moment [7].

The purpose of this study is to ascertain what MPNS is and how it works, to compare MPNS with other PN on other mobile systems. This topic was identified as being of importance to developers in providing them with the necessary background of MPNS and the knowledge of how to use different PNs to fit their own scenarios.

Chapter 2 introduces to the Push Technology and Push Notification Service. Chapter 3 focuses on the introduction of MPNS, MPNS notifications and payloads, and how MPNS works. Chapter 4 discusses about the comparison between MPNS and PN service on other mobile systems (iOS and Android). Chapter 5 analyses the use cases of PNs and how to choose the type of PN for a specific scenario. Chapter 6 describes the technical project details about implementing a WP7 application and WCF service for PN. Discussion comments on if WP7 will support multi-tasking in the future, whether PN is needed or useful anymore. Finally, conclusion summarizes the thesis and possible future work related to this topic.

1.1 Push Technology

In the concept of Push Notification Service, “Push” comes from Push Technology.

Push Technology, where the initial request for data originates from the server, is a kind of net-based communication protocol style. In contrast, Pull Technology is the style of communication where clients originate the initial request for data [8].

Push services, which implements Push Technology, are often used to inform the client that new information is available before the actual information reaches to the client. A client can subscribe different data channels from different web servers via a push service central server. Whenever new information is accessible via whichever channel, the push service central server informs the user to incept the new data.

Typically, Instant Messaging (IM) applications are using push services to achieve the functionality of real-time communication [9]. Whenever chat messages or even files are delivered by the IM service, they are delivered to the target user. Email system is also a typical example of push service. Simple Mail Transfer Protocol (SMTP) is based on a push protocol [10]. But then, a pull protocol, for example POP3 (Post Office Protocol 3rd) and IMAP (Internet Message Access Protocol), is used for the last step by repeatedly polling the mail server to check whether there is new mail.

HTTP server push [11], Pushlet [12], Long polling, Flash XML Socket relays [13], Push Notification Service are all related to Push Technology. We will be concentrating on Push Notification Service Technology on mobile devices.

1.2 Push Notification Service

Push Notification Service (PNS), which is used in different Operation Systems especially Mobile Operation Systems, is not a new concept. PNS allows the server to “push” messages to an application which is installed in the devices so that the web services can interact with the client application at a certain level, even when the client application is not in the foreground or not running. For the developers, a PN message is an effective way to communicate with the end user. For the end users, they never know how and when the PN messages arrive to the device until the messages are displayed visually on the phone with the same context that the provider sends, in order to notify the user that there are some events of interest to them happening, even when the application is not at the foreground state. Comparing with Polling Technology, the data flow of PNS looks much simpler (Figure 2).

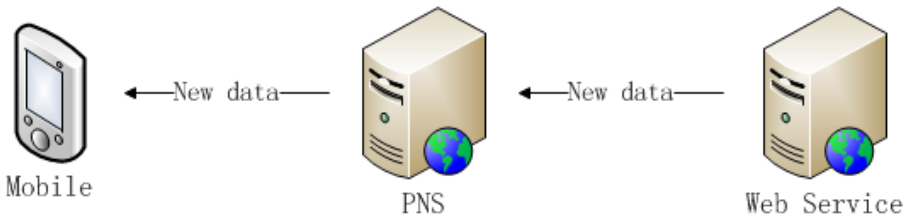

Figure 2 Data flow of PNS

PNS is widely used in different mobile systems, such as Apple Push Notification Services (APNS) [14] in iPhone and Android Cloud to Device Messaging Framework (C2DM) in Android. In this thesis, Microsoft Push Notification (MPNS) is primarily studied.

2. Microsoft Push Notification Service

Microsoft Push Notification Service (MPNS) is a service which provides functionality for the developers to "push" information from the server to users' Windows Phone via Microsoft Push Notification Server. The information pushed from the server can be received and displayed in terms of notifications, even when the application is not running, in order to indicate to the user that new information is available. Microsoft Push Notification Server is hosted in the cloud so that it is able to get the benefits that the cloud offers. "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."[15] The provider, which offers these computing resources for cloud computing, is normally called "The Cloud".

"The Microsoft PN (MPN) service includes a cool twist that gives developers the power to create the impression that their application is always connected by displaying relevant information even while not running through the Live Tile displayed on the Start Screen." [16] MPNS is an integral mechanism of WP model so that all the web services can communicate with the client application in the WP devices at all times. From the device view, only one background service is running and connecting to the internet at the same time, but this service keeps all the applications installed on the device updated in real-time. Another important purpose of MPNS is to trigger the user to launch the WP application when they get different notifications that catch their eyes, as MPNS is designed to quick launch the specific application – the notification is shown either on the top of the screen (even when the phone is locked or any other application is running at the moment) or as a tile of the application in the Quick Launch area of the main screen.

2.1 MPNS notifications and payloads

MPNS supports 3 kinds of notifications: **Toast**, **Tile** and **Raw** notifications. Each PN carries its own payload. The payload specifies how the notification is shown on the device to inform the user that data is waiting to be downloaded.

Tiles and toast notifications are means whereby a web service can deliver messages via Microsoft Push Notification Server to the device while the phone application is not running at the moment, and the end users can perform some actions according to the visible notifications shown on the WP shell. Raw notifications can be received only when the application is running or, in other words, in the foreground; otherwise they will be ignored by WP application model. Depending on the type of notification, Windows Phone can implement one of the three ways, raw data is delivered to the application, a toast notification is displayed, or the tile is updated visually.

2.1.1 Toast Notifications

A Toast notification is a special kind of notification which is displayed as an overlay on the device's current screen when it is received. It means that the notification is shown as an alert on top of any screen, including WP shell screen and any other application in the foreground, for example, when the user is calling someone or playing games, except the particular application that toast notification belongs to. The toast notification is displayed on the top of screen for 10 seconds once it is received and it disappears without any history saved. Once the user clicks the toast notification, the application is automatically launched.

A toast notification is very intrusive to the end user. When the user is calling someone talking about some urgent things, and the device gets different toast notifications, which are not important things like advertisements, and each notification makes the device beep upon the update, it would be very inconvenient. Therefore, the developer that provides the notification should use it sparingly and with purpose. The developer should make sure that the notification

is as important or urgent to the user as before it was sent by web service and the user should also be provided the functionality to turn off the toast notifications from the specific application. The easiest scenario about the importance level of the alert is the weather application. When a tsunami is on the way, it is an effective way to use a toast notification to inform the people by the sea. The toast notification should primarily be used for personally relevant and time critical peer-to-peer communication.

The payload of a toast notification is made up of two strings, the notification's title and sub-title, or in other words, head and optional body text. Figure 3 shows the location and appearance of a toast notification.

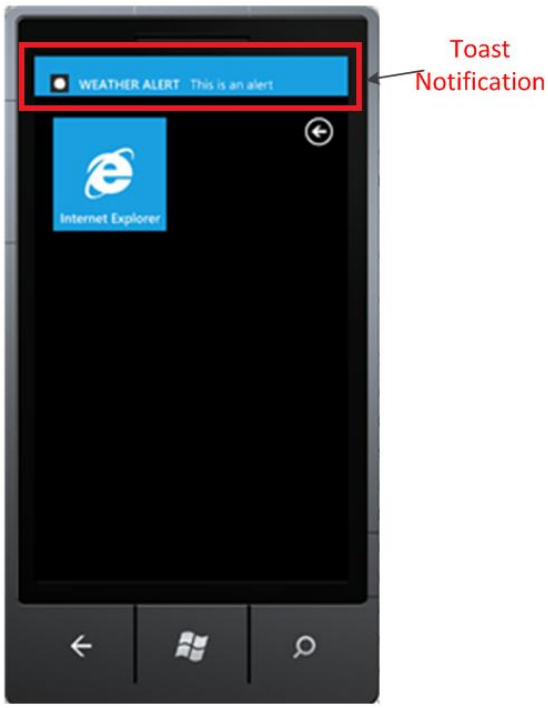

Figure 3 Toast Notification

The function that sends a PN from a web service to MPNS looks like Figure 4 [17], and the meaning of the codes can be found as comments.

```
// The URI that the Push Notification Service returns to the Push Client when creating a notifica
tion channel.
string subscriptionUri = "<Notification Channel URI>";
HttpWebRequest sendNotificationRequest = (HttpWebRequest)WebRequest.Create(subscriptionUri);

// HTTP POST is the only allowed method to send the notification.
sendNotificationRequest.Method = "POST";

// The optional custom header X-MessageID uniquely identifies a notification message. If it is pre
sent, the // same value is returned in the notification response. It must be a string that contains
a UUID.
sendNotificationRequest.Headers.Add("X-MessageID", "<UUID>");

// Sets the notification payload to send.
byte[] notificationMessage = new byte[] {<payload>};

// Sets the web request content length.
sendNotificationRequest.ContentLength = notificationMessage.Length;
using (Stream requestStream = sendNotificationRequest.GetRequestStream())
{
    requestStream.Write(notificationMessage, 0, notificationMessage.Length);
}

// Sends the notification and gets the response.
HttpWebResponse response = (HttpWebResponse)sendNotificationRequest.GetResponse();
string notificationStatus = response.Headers["X-NotificationStatus"];
string notificationChannelStatus = response.Headers["X-SubscriptionStatus"];
string deviceConnectionStatus = response.Headers["X-DeviceConnectionStatus"];
```

Figure 4 Example code for sending a PN

To indicate the notification type is toast notification, two actions need to be done.

- Add the following HTTP header (Figure 5):

```
sendNotificationRequest.ContentType = "text/xml";
sendNotificationRequest.Headers.Add("X-WindowsPhone-Target", "toast");
sendNotificationRequest.Headers.Add("X-NotificationClass", "<batching interval>");
```

Figure 5 Add HTTP header for a toast PN

- Replace <payload> with the following (Figure 6):

```
string toastMessage = "<?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>";
```

Figure 6 XML code for toast PN payload

In Figure 5, <batching interval> is used to indicate when the PN will be sent from MPNS [18]. By different batching interval setting, the message can be delivered by MPNS immediately, within 450 seconds, or within 900 seconds.

2.1.2 Tile Notification

To give the users the best use experience, dynamic tile is designed and used to represent an application or its content. A tile is a link to represent an application or a function, which can be updated in real-time. The start screen of WP7 is composed of different tiles. The end-user is allowed to pin up to 15 third-party applications tiles to the Quick Launch area of the phone's Start experience. Once the user pins one application to the Start screen, one single visual tile is associated with the specific application. The user can quick launch the application just by clicking the tile and the developer can send a tile PN message to the phone to change the content of the tile lively. Take the weather application as an example. The dynamic tile can display the local weather with a picture to show sunny or fog condition and a number to show the temperature.

Figure 7 Tile Notification

As the climate condition changes all the time, a tile notification can be used to update the weather information once the weather changes so that the user can always find the latest weather information on the start screen without any manual sync request. All the tiles are square, and have an icon, a string and a number associated with it. They are respectively called a **tile's background image**, **counter (or 'badge')**, and **tile properties**. Figure 7 shows a tile notification in the main shell. The background image can be any valid UPI to an image either a local resource, which is installed as a part of the application, or a resource from web. To use the resource from local or web is depended on the provider and the purpose of the application. Normally, if the image is always chosen from some specific pictures, it is better to pack all those pictures needed into the application so that the dynamic tile is able to have better performance and to save battery life. In another scenario, the application might show different background images every time when the tile is updated, for example, an online photos viewing application which will update the newest photo to the tile. Then the background image needs to be referencing a resource from the web. The phone should download the image before it is displayed as a tile so the size of the image is not suggested to be large.

The payload of a tile notification includes three items of data [19]:

- A background PNG or JPG image for the tile that should be 173 pixels by 173 pixels in size
- A text label (string) that overlays the background image
- A count value (string) that also overlays the background image;

The same as sending toast notification, the following HTTP headers need to be added to indicate the type of a tile notification (Figure 8) [17].

```
sendNotificationRequest.ContentType = "text/xml";
sendNotificationRequest.Headers.Add("X-WindowsPhone-Target", "token");
sendNotificationRequest.Headers.Add("X-NotificationClass", "<batching interval>");
```

Figure 8 Add HTTP header for a tile PN

The <payload> can be replaced with following example code (Figure 9):

```
string tileMessage = "<?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>";
```

Figure 9 XML code for tile PN payload

In Figure 8, toke is used to indicate the message is tile PN although the name of the type is called Tile PN. In Figure 9, <background image path>, <count> and <title> properties are in a string format. If <background image path> references a remote resource, the maximum allowed size of the Tile image is 80 KB, with a maximum download time of 15 seconds [17].

2.1.3 Raw Notification

Unlike toast and tile notifications, which are WP System defined notifications and have their own fixed format of payload, a raw notification is a developer defined notification. Each developer can create a unique and special payload format for the application and the accordingly functionality to parse and handle it. For example, the developer of a weather application can create a dictionary or a form which is used for raw notification. This dictionary contains different strings which are used for “Location”, “Temperature”, and “WeatherType”. When the raw notification is ready to send, the web service fills the dictionary with the corresponding value, writes it into an XML document in-memory and sends the notification to MPNS as a byte array. Once the raw notification reaches to the client, the functionality written in the client application for parsing the raw payload is used in order to get these three values from the dictionary and will do homologous action to handle this raw notification. It also means the WP7 system does not deal with the raw notification itself for user interface. In other word, WP7 client application is given a free hand to take action to deal with the raw message. This is suitable for the functionality to exchange data between WP7 application and web server, and for those who want to control the action on the raw data to user interface.

Another difference between raw notification and toast or tile notification is the time and scope to use. The raw notifications can only be received and handled when the application is in the foreground. If the raw notification is sent to the MPNS but the application is not running on the device, the raw notification is ignored by default and not sent to the device any more.

When sending a raw notification via MPNS, following HTTP header must be added (Figure 10):

```
sendNotificationRequest.Headers.Add("X-NotificationClass", "<batching interval>");
```

Figure 10 Add HTTP header for a raw PN

2.2 How MPNS works

To enable MPNS functionality, it needs to follow a given process so that a PN can be passed from a web service to the end user.

There are three elements playing roles in the whole PN lifecycle [20].

- The **Web application** or **cloud service**, which provides PNs and communicates with the WP7 client application.

- The **Windows Phone device** or **WP system**, which receives notifications and handles them. It is not the client application that is playing in the process. The notifications reach the WP system, and the system decides how to handle the notifications according to the payload format, not the client application. Precisely, it is MPN Client which is integrated in WP system that plays a major role. The MPN Client and MPNS implement a client-to-server protocol. The MPN Client service is running in the background on WP7 all times to listen from MPNS to make sure all the third-party applications are updated as-needed, even when the application is not running.

- **MPNS**, which is a bridge between the web service and the WP device. MPNS, specially used for WP7, is a message exchanging center running on Windows Azure that Microsoft creates and holds specially used for WP7. It passes messages from a third-party web service to WP devices through PNs.

The WP7 channel is submitted to web service from WP7 application, and then the web service sends messages to MPNS. When MPNS gets a message, it puts the message in the queue and sends to WP7 device as a notification. With this Notification Framework, the messages get to the end user easily and in time, and the developers can implement application with MPNS expediently.

Before the web service can send PNs to WP7 devices, the channel has to be registered as an identification, which then can be used for sending a message from third-party web application to MPNS. Figure 11 shows the whole process how an application on the WP7 device can register for PN in order to listen from the corresponding web service via MPNS.

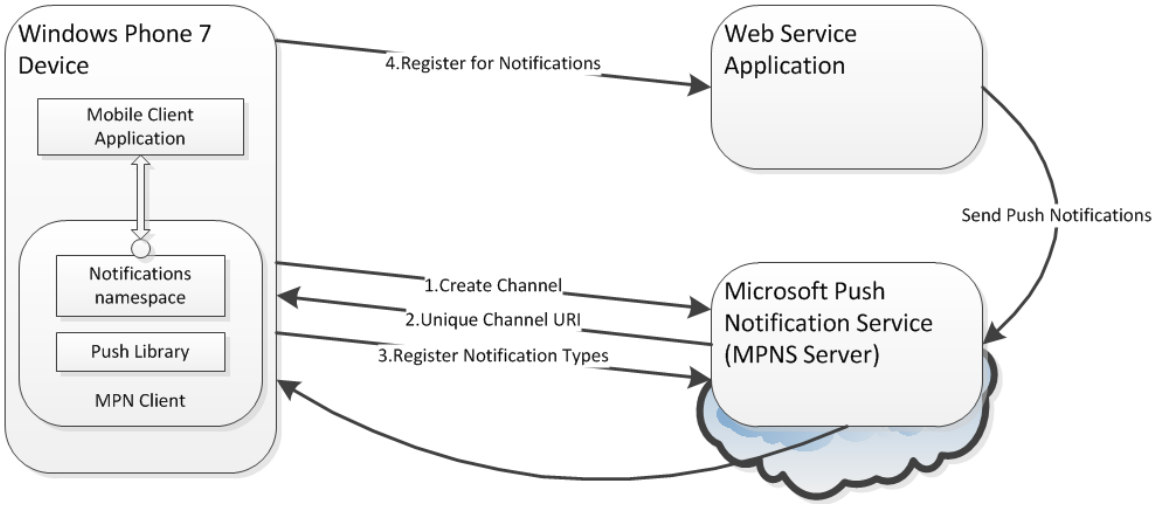

Figure 11 PNS process

The detail description of the registration process is explained as following in Figure 11 [21]:

1. **Create Channel.** The application on the WP7 device sends a request to MPNS to create a notification channel. This is a signal to tell MPNS that the application wants to receive PN messages from a specific web service.
2. **Unique Channel URI.** When MPNS gets the request from the WP application, it creates a channel for that application. A subscription end point is produced within the MPN servers once the channel is formulated. This subscription end point makes MPNS as the bridge for message transmission, as it lets the web service to POST a message to the end point when the web application pushes the message to the client, which follows by a consequent action that MPNS transmits the messages to the WP device via the channel. MPNS returns a URI, which presents the channel, back to the WP7 client application as the response of channel creating request, which is unique to that specific application on the particular WP

device. This URI contains all the information related to the subscription. In a developer's view, this URI is also the address to the device on which the client is installed, so that the provider can locate the device and send a PN to the user. This is extremely important for the scenario that the PN message is not for broadcasting but for sending to a specific person.

3. **Register notification types.** As three types of PN that can be used for WP application, the developer is free to use either one type of PN or all types. The WP client application must register the types of PNs it receives with MPNS, which sets up a binding and authorizes the mobile client application to be Quick Launched when a PN reaches to the device. To set up the channel, the application on the devices must be run at least once to active the functionality of receiving PN from web service. Also, the user is able to disable or re-enable to receive toast or tile notifications, which can let the user avoid being interrupted by said PNs that the user does not want to receive.

4. **Register for notifications.** After the channel is bound between the mobile client application and MPNS, the WP application passes the URI to the corresponding web service which makes the whole channel usable. Therefore, the client application is also registered with the web service which will provide the PN messages. The web service saves all the URI acquired from all the clients in the list. This action concludes the whole registration process.

After setting up the PN channel, the mobile device is able to receive PN from the web service. To start a PN message, the web service needs to do an HTTP POST with a formatted XML which follows PN payload format to the unique URI that is provided by the mobile application in registration process. MPNS forwards the messages onto the correct device reciprocally.

3. Comparison between MPNS and other PNSs

As mentioned before, PN is not a new concept. It is already used in other mobile operation systems (OS), such as iOS – iOS3.0 and iOS 4.0, which are used widely on an iPhone, iPad, or iPod touch – and Android 2.2 (or higher), which we are taking into account here.

Although the concept of PN is same, it is used quite differently in different mobile OSs. To understand and use of PN better with MPNS, comparison between MPNS and other PNSs are extremely important, as developers can comprehend the strength and weaknesses of MPNS and implement their own applications smartly and effectively.

3.1 Apple Push Notification Service

Apple Push Notification Services (APNS) is the primary mobile service which uses a push design to deliver PNs from the server of a third-party application to the iPhone, iPod Touch and iPad. It is released with iOS 3.0, which is same as WP in that it does not support multi-tasking, which means no application can be run in the background. Beginning with iOS 4.0, the third-party application is supported to be running in the background but only for a limited time. APNS still does play an indispensable role in iOS even when multi-tasking is supported.

The way APNS works seems similar to MPNS. Web service, which is the provider in APNS's concept, prepares the notification messages in server side, and connects it with APNS through the web and sends the notification to APNS through the channel between them when needed. Once new notification arrives, APNS pushes the notification to the particular device.

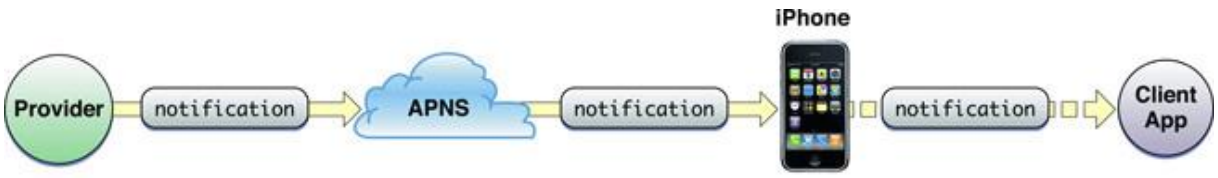

Figure 12 The notification flow from a provider to a client application [22]

Figure 12 shows the flow of a notification that is forwarded from a web service to the third-party client application on an iPhone. It is one-way flow. APNS is the background service running on an iPhone to keep all the third-party applications listening from their providers and informing the users that new data is waiting for them.

After the channel for PNs is created between a device and APNS, a device token is generated from APNS, which is used every time when the device connects with APNS. Every time a provider sends a PN to APNS, accompanied by the device token, APNS decrypts the token with token key and validate the notification with the device certificate. Then APNS pushes the PN to the target device using the device ID that comes from the device token. Figure 13 shows the big picture of how APNS works.

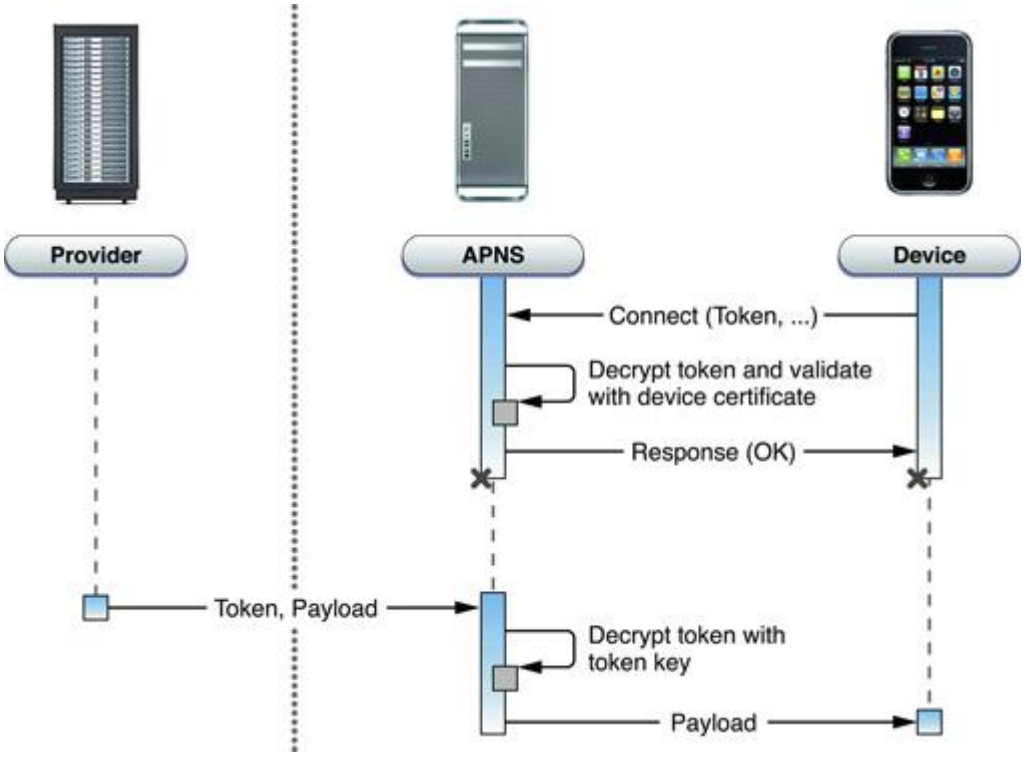

Figure 13 Big Picture of APNS [22]

With APNS, the web service can do three actions to client application, **an alert message** to display to the user, **a number** to badge the application icon with and **a sound** to play [22]. This will be discussed minutely in the later part.

On an iOS, the developers must build a JSON dictionary object for each notification. It contains three keys – alert, badge and sound. The alert is a string or a dictionary value used to show a pop-up alert on the main screen in order to display the message text and convince the user to launch the application through the alert (Figure 14). The badge is a number shown on the application icon to tell the user some number information, for example, the number of emails unread (Figure 15). A string presents the value of the sound key, which indicates the name of the sound file that comes with the application installation.

Figure 14 iPhone Push Notifications [26]

Figure 15 iPhone Push Notification Badge

3.2 Android Push Notification C2DM

Android Cloud to Device Messaging Framework (C2DM) is used for PN purposes on Android platform. C2DM is a free light-weight Android framework and a Google-run service that allows web service to send PNs to client mobile applications running on devices. It is a new framework to Android developers, and began its use only on Android 2.2 or higher version.

In the past, PN concept was already introduced into Android. To get those messages which were expected to be “pushed” from server, each application needs to be implemented in virtue of other technologies which are not push designed.

There were three ways that could be used for PNs in Android before C2DM came out [23]:

- Polling: the device keeps polling the server to see if new data comes. It is not easy to control polling frequency. Polling hurts battery life and data is not real time updated.
- Creating a service: each application leaves a background service to listen to the server persistently, which drains the phone battery and increases server’s burden.
- SMS: the server of third-party application notifies the client by SMS push, which is not free.

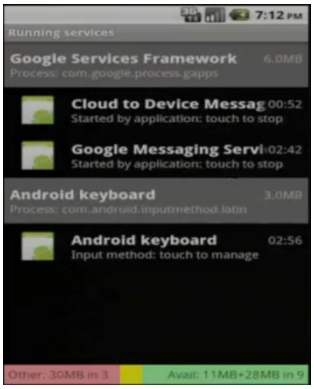

Figure 16 Running Services on Android using C2DM [24]

C2DM is the latest technology for Android system to use PN. Figure 16 shows that one background service – C2DM service – is running to receive PNs. With C2DM enabled, the third-party application server is allowed to send the message to C2DM server that is hosted in the cloud by Google. All the messages that are forwarded to C2DM server are queued, stored and sent to the device once the device is online. On the device, the message is delivered to the target application by intent system broadcasting. The application is woken up and processes the message once the message is received [25].

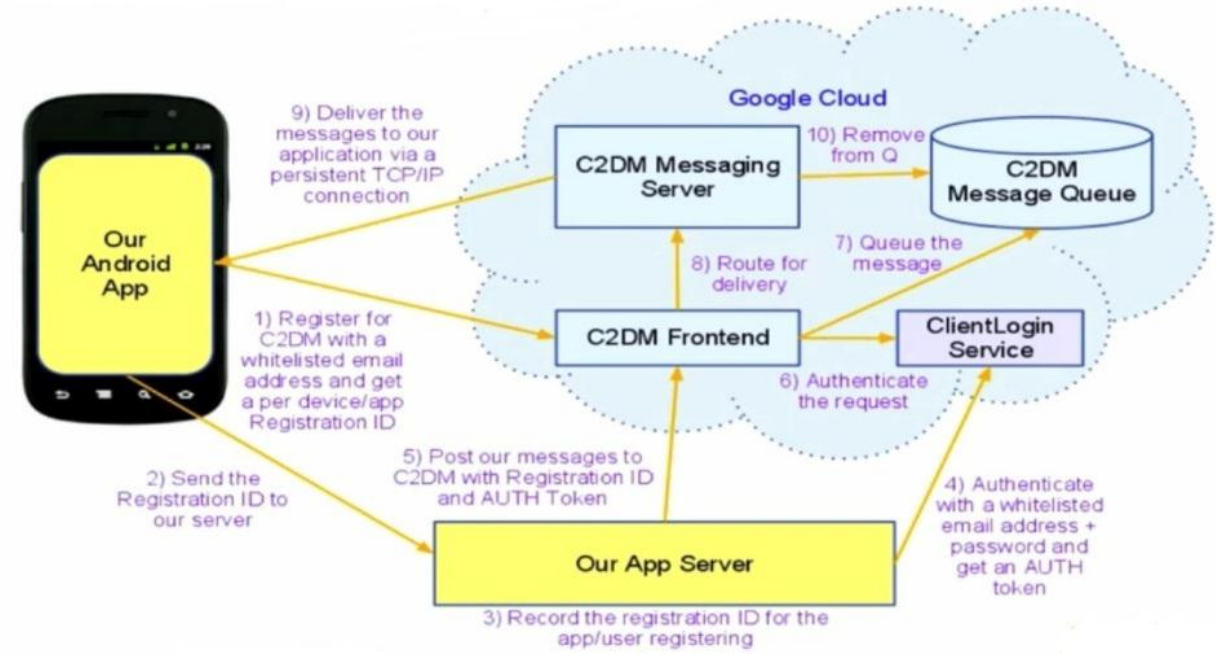

Figure 17 Big Picture of C2DM [24]

The big picture of C2DM is shown as Figure 17. As it can be seen from the figure, to use C2DM for Android to sending and receiving PNs, an application that is installed on an Android system must go through 10 steps to get it work. Each step is illustrated clearly in the figure.

3.3 Comparisons on PN payloads

PN payloads are quite different on each mobile OS because payloads decide the usage of PNs.

As we mentioned before, each notification can do all three actions on iOS, appearing a pop-up alert message, changing the badge of the application icon, and playing a sound. Comparing with APNS, MPNS has multiple classifications on payloads. As mentioned before, MPNS has tile, toast and raw notification. Each time, web service can only send one kind notification among the three. Only one action becomes the consequent action, showing a toast alert, updating the tile or sending raw data to the application. These classifications make all notifications, which developers provide, are sent with the selected type according to the content and usage of the notification. For example, if the notification content is an urgent alert to the user, the web service can send a toast notification to the device so that the message is informed more effectively and evidently. On opposite side, when the new information does not need to catch the user's eye, a tile notification can update the tile of the application quietly and visually.

No matter the specific application is running or not in the foreground, PNs are always forwarded by APNS from server side to client side. The difference regarding the condition is that the alert, sound or badge value is played or shown if the application is not running, and the message is delivered to the application when it is running at the moment. In this regard, the classification of MPNS payloads decreases the workload of both Microsoft Push Notification Server and user's device. Because PNs are sent selectively depending on the application's running condition. If the application is not running, raw notifications are ignored to be sent to the device which decreases the burden of Microsoft Push Notification Server.

C2DM only supports one kind of notification on Android. All the notifications are demonstrated on the notification board. When the notification reaches the device via C2DM, a notification icon, which represents the application, is added to the notification status bar at the

top of main screen, and ticker text is shown on the bar for one second. The user can pull down the notification status bar to reveal the detailed notification information, including an icon, notification title, notification message and the time when it is received. The user can either wake up the application to see what the notification tries to inform, or ignore the notification by clicking “Clear” button. (Figure 18)

Figure 18 Android notifications [27]

Compared with Android, toast notification works for the same usage of PN to the end user. Tile notification brings better visual and quiet user experience, which Android does not have. For those low urgent level but visualized information, which no action is to be expected by users, such as weather information, web service can send the messages to clients as tile notification. Whenever a user catches a glimpse of main screen, the tile notifies the user with the latest news of the application. If the notice catches the user’s eye, quick launching the application to read more is just a click away. While Android PN always needs to be handled by the third-party application, MPNS client provides the developer a visual way to show the information so that the developer does not need to create a program to handle all PNs.

As a whole comparison of PN payloads among APNS, C2DM and MPNS, it is obvious to see that – APNS has standards but the standards are not open and too restrictive to implement; C2DM is open but relies on third-party applications’ handling too much; MPNS has standard notifications, such as toast and tile notification, but it is also open to developers since raw notification is used to handle particular messages. With different types of PNs, MPNS provides developers diversified implementation methods.

3.4 Logging of PNs

In real life, a user has dozens of applications installed on the mobile device and each application is capable to receive a PN informing the user that new data is available. The user might attend a meeting or sleep at night, but will never be playing the mobile phone 24/7. On Android system, notification board (Figure 19) is used to store all the PNs either from web services or from local applications. The user is able to read and clear all the pending notifications when he or she is available. Notification board makes it possible that the user never misses any notifications.

Figure 19 Android Notification Board [28]

It is different on WP system and iOS that no logs and history is kept for PNs. On WP system, toast notification is only shown on the top of the screen for 10 seconds without any history; tile notification is overwritten when new tile notification comes for the same application; raw notification is even ignored when the application is not running. On iOS, the alert is shown when it is received, and only when the user taps the button on the alert – either to ignore or launch the application – the next alert that is received after this one can be shown on the screen.

Considering this point mentioned above, whether to choose PN as the way to inform the end users or the purpose of PN is suggested to be thought over before implementing the application. For example, a mobile application is used to remind old people to take medicines on time in case they should forget. The doctor sets the timetable and prescription on the software for each patient, and the devices of patients get a toast PN when the time comes. In this case, because the toast notification is only shown for 10 seconds without any log, the message is quite easily dismissed. Taking another example, the developer of a weather application tries to warn those people – who are on the beach – that tsunami is on the way. It is an emergency scenario. The notification has to been sent over and over again in order people get notified as the phone keeps ringing or shaking.

3.5 Push notification and local notification

iOS 4.0 and Android system both have local notification that WP system does not have. Local and push notifications serve different design needs [22]. Different to PNs, which are “pushed” from outside a device remotely, local notification are actually another kind of PN, which are “pushed” from an application locally on a device. With local notifications, the user is able to be notified locally with the setting of the application even when the device is not online. Figure 20 shows APNS local notification.

Figure 20 APNS local notification [22]

For example, an application on an iPhone manages a list that includes all the food and its overdue-time. The application is set to alert the user before some food's due date expires. To implement this functionality, the application posts a local notification for that event to show an alert message on the main screen.

MPNS only supports remote PN on WP7. It is impossible to notify the user according to a local setting because WP7 does not support multitasking. The only way to get a scheduled alert, as we mentioned above, is to upload all the records to the third-party application server in order to get PNs at the specific time set. To design an application to get notification real time depending on local condition of the device is impossible. For example, an application, which is used to indicate battery life visually and inform the user when the battery life is less than a set level, will not work on WP7.

3.6 Configuration setting for PN

As shown in Figure 21, it is the screen shot of iPhone notification configuration setting. The user is able to turn on or off all notifications, and also sounds, alerts or badges for each application. With this configuration setting, it is convenient to set whether to receive a particular kind of notification from a particular application at any time.

Figure 21 iPhone's Notification Setting System

On WP system, PN is set to default and the user cannot find any place to set it on the device. Therefore, the developers have to implement the functionality for the users to turn on or turn off PN and inform the user that PN is enabled when the application is launched the first time.

3.7 Summary of comparison between MPNS and APNS or C2DM

To summarize the comparison between MPNS and APNS or C2DM discussed above, Table 1 presents the comparison result.

Table 1 Comparison between MPNS and APNS, C2DM

PN model	MPNS	APNS	C2DM
Payload classifications	Multiple	Single	Single
PN formats	Multiple	Multiple	Single
PN is forwarded in condition	Yes	No	No
Whether PN needs to be handled by the application	Yes/no, depending on PN type	No	Yes
Standards, open to developer	Yes, open	Yes, not open	No, open
Logging of PNs	No	No	Yes
Local notification	No	Yes	Yes
Configuration setting for PN	No	Yes	No

As the table shows, MPNS has multiple payload classifications – toast, tile and raw, and supports multiple PN formats, for example string and image. Depends on the type of PN, the application does not need to implement codes to handle all the PNs and MPNS client handles toast and tile PNs.

4. Usage of different PN types analysis

As mentioned above, MPNS supports three types of PN, which are toast notification, tile notification and raw notification. Different types of PN have different properties and functionalities. Depending on the application types and the scenarios it covered, the developer might use one or all of the PN types.

When to choose a particular type of PN? What kind of scenarios is each PN type suitable? It is a really controversial topic. This thesis will discuss it from following four aspects.

4.1 Emergent and urgent levels

An application tries to help people to stop worrying about the expiration date of foods. Once the users buy different foods from the market, they launch the client application on the WP to input the expiration date of the foods. The foods information are sent by internet and stored on the server. Along with the expiration date getting closer, the application server will send PNs to notify the users that some kinds of food are closing to the expiration date and reminds the users to finish them before they are overdue. The developer of this application designed two approaches to inform the users via PNS. One approach that is mentioned before to inform the users about the expiration date is to send a PN message to the client machine just one day before the expiration date, and we called this approach as Approach A. Another one is to send the users PNs when some foods are going bad in one week, and we called this approach as Approach B.

The largest difference between these two approaches, Approach A and Approach B, is the emergent and urgent levels. This level is also the largest issue to impact on choosing PN type. Compared with Approach B, Approach A is much more urgent than the other. Approach A needs a more notable and remarkable PN to come into notice. Therefore, the developer decides to choose toast notification for Approach A and tile notification for Approach B. The toast notification works as an alert to warn the users that some foods need to be finished

before tomorrow. At the meantime, the tile of this application shows the picture of the food which will expire within a week and the number of days left on the shell. Once the user turns to the main screen, the tile gives the user visual and dynamic information of the food type and days.

According to the emergent and urgent levels, it is better to choose toast notification for higher level and tile notification for lower level, as toast notification can provide a more remarkable message and catch the users' eyes. For the unimportant information, the tile notification will update the information stilly and silently.

Take the weather forecast application as an example again. The normal climate change and weather forecast is not so important that the user have to read it as immediately as the notification reaches. Tile notification is useful enough for normal weather forecast scenario, unless a specific scenario happens, when a cyclone or tsunami is on the way to where the end user is. For the latter scenario which is mentioned above, toast notification must be chosen to draw the user's attention. Figure 22 and 23 shows the weather forecast application sends a tile notification and a toast notification for two purposes.

Figure 22 Tile shows New York's weather

Figure 23 Toast notification informs the user of tsunami

4.2 Action expected from the user

Depending on the scenario that the application covers and the content of the PN that the message brings to the user, some PNs expect the user to do some actions after the PN reaches but some do not. For example a meeting booking application sends a PN message to the user to inform the user that a meeting request is waiting to be confirmed and expects the user to launch the application to check the meeting information and calendar to decide whether the meeting is accepted or refused. The action from the user is important as the meeting organizer is waiting for the answer from the user. Take another example, when a user is reading BBC news from a running BBC (British Broadcasting Corporation) news application, a raw notification is delivered from the web service to the client application on the WP. The notification is just used to update more news from the server for the end user. It does not require any action from the user but just update more information for the application.

By this token, toast notification and tile notification are suggested to be used for those PNs which expect the end user to quick launch the application itself to read more information or do other actions; raw notification is suggested to be used for those PNs that are not waiting for the users to do any actions and to be handled by the application itself.

An online game application, which requires a group of users to finish one game, chooses toast notification and raw notification for informing the user about the status of the game. The game is play by one followed by another in sequence. If the user turns back to the main screen or suffers a cut off from the internet after he/she finishes his/her step, a toast notification will alert the user that it is his/her turn to finish the next step in the game, so that the user does not need to wait for all the others to play their part of the game. If the user keeps the game application running even if it is time for another player to play, a raw notification can update the game status once another player finishes his/her part.

4.3 Content of update

With MPNS, the format of a notification is not unitary anymore. Multiform notifications provide the best user experience on notifications to the WP consumers. As introduced before, a toast notification sends strings to the user which can be a word of advertisement or a notification of a friend's birthday. A tile notification sends a background image, a count and the title of the notification. There-in-to, the background image can be the image of the weather at that time; the count can be the number of unread emails available; and the title can be a short description of the tile notification. A raw notification sends any information that is needed to the application directly which can be handled. It could be a trigger to ask the application to download more information, or any formats of information that will be shown to the user automatically.

The diversity of PN format that MPNS supports provides more choices and convenience to the developer. Depends on the content of the PN, the developer can choose the right type of PN among these types.

In addition, because of the particularity of raw notification, that it can only be delivered when the specific application is running, the content of the notification is also an issue to choose between toast or tile notification and raw notification. If the content of the notification is expected to be read at any time even when the application is not in the foreground, the toast or tile notification might be chosen; otherwise, raw notification is a choice for the developer.

4.4 Updating frequency level

A client can subscribe different channels for PNs via MPNS server. Once new information reaches to MPNS server on any one of the channels, the information is pushed to the client, which means the device keeps getting PNs from different web services for different applications. If the user has installed dozens of applications, and each application sends hundreds of toast notifications to inform the user that new data of interest is waiting to be

downloaded, then the device would keep receiving PNs and it would be inconvenient for the user to use the phone. When the developer designs application, updating frequency of PNs must be taken into account according to the functionality of PN.

It is suggested to choose toast notification for higher updating frequency usage and tile notification for lower updating frequency. Because raw notification is only delivered when the particular application is running and not a standard notification, the updating frequency might have slight impact on choosing raw notification.

Take a BBC news application as an example. The application updates the BBC news to the client once a new piece of news is published by BBC. BBC is the largest broadcaster in the world [29], and publishes a vast amount of news pieces every day. It is a higher updating frequency usage scenario compared with other scenarios. Therefore it is suggested to choose tile notification for keeping with the updates.

4.5 Summary of usage of different PN types analysis

To summarize the analysis discussed above of use cases, Table 2 presents the suggestion on how to choose the type of PN to fit a specific scenario.

Table 2 Use case analysis on how to choose PN type

Notification Type	Toast Notification	Tile Notification	Raw Notification
Emergent and urgent levels	High	Low	--
Action expected from the user	Yes	Yes	No
Format of updated content	String	String, image or count	No limitation on format
Updating frequency level	Low	High	--
Application needs to be running	No	No	Yes

5. NonOverdue System Implementation

This chapter gives details for developing the application for simulate PN sending and receiving. This system is called **NonOverdue** Application, which collects users' foods expiration date information and sends PNs to remind people of these dates so that they could finish the specific food in time. Once the user logs into the WP7 client application of the system, he/she can add, edit or delete an item – a type of food and its expiration date – and the data is store in the database on the server side. Along with the expiration date getting closer, the application server will send toast PNs to notify the users that some kinds of food will be expired in one day and tile PNs to show those food which are closing to the expiration date (for example one week) to remind the users to finish them before they are overdue. A developer can also use the WPF client of the system to send all three types of PNs to the clients directly. The Figure 24 shows the finalized solution developed for simulating PN working process with used technologies, .Net Framework v4.0, and Visual Studio 2010 platform.

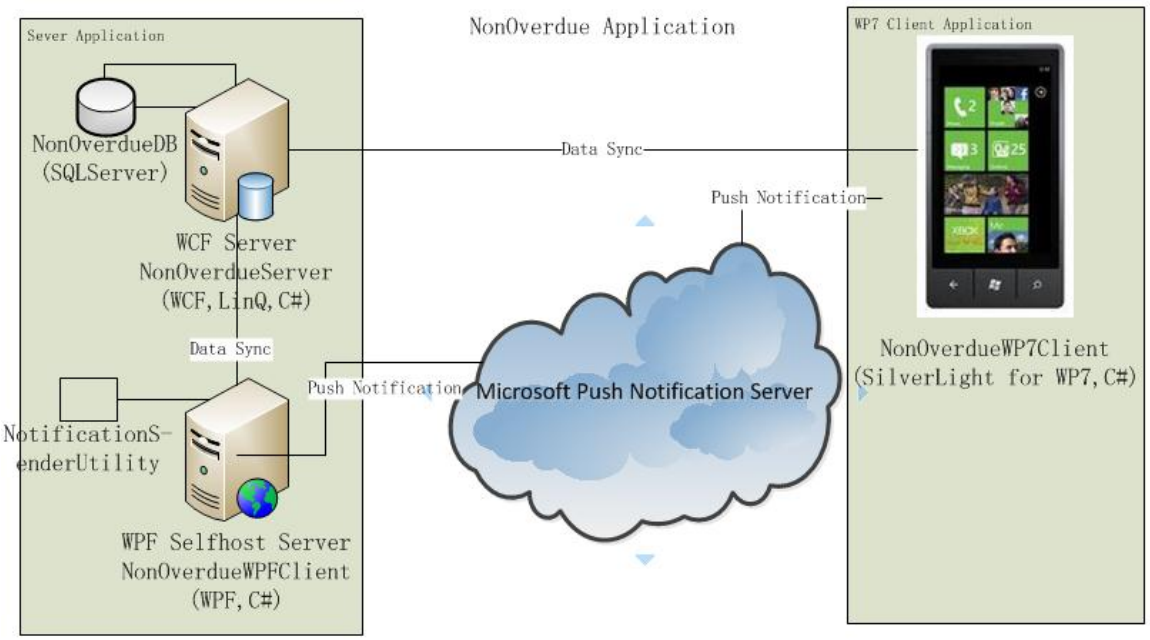

Figure 24 System architecture of NonOverdue Application

The solution contains 4 projects, which are **NonOverdueServer**, **NonOverdueWP7Client**, **NonOverdueWPFClient** and **NotificationSenderUtility**. NonOverdueServer is a WCF (Windows Communication Foundation) Service Application which connects database – NonOverdueDB – directly mainly used for data transferring to NonOverdueWPFClient and NonOverdueWP7Client. NonOverdueWPFClient is a WPF self-host server which is used to send PNs to WP7 client applications via MPNS. NonOverdueWP7Client is a client application on WP7 device. NotificationSenderUtility is a utility class which holds all the communication logic for PNS. The source code for this solution is given in the appendix.

5.1 NonOverdueServer

NonOverdueServer is a WCF service application which connects NonOverdueDB directly to provide data transferring service to NonOverdueWP7Client and NonOverdueWPFClient.

NonOverdueDB is a SQL Server Database which is hold in SQLServer 2008 R2. The project is communicating with the database with LINQ to SQL classes, which is shown in Figure 25. Inside this figure, User table is used to store users' information; Food table stores different food types and corresponding images; Item stores records which are initialized by users to hold a specific food and its expiration date.

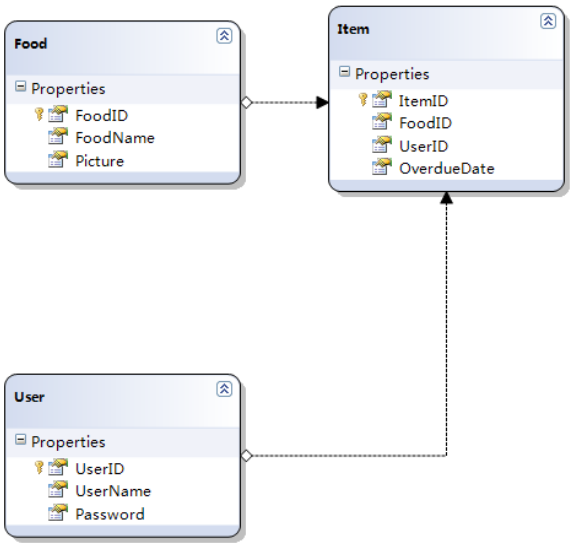

Figure 25 DataClasses of NonOverdueDB

Iservice1.cs and Service1.svc contain OperationContracts and DataContracts which are used for data communication with database for other projects. The service provides functions for searching, deleting, inserting etc. records on database. For example, the clients can get a list of food types via this service.

To use WCF service is good when the solution contains different clients and the service is easy to move to the cloud.

5.2 NonOverdueWPFClient

NonOverdueWPFClient is a self-host WPF client. It communicates with database via service reference form NonOverdueServer and communicates with MPNS with the service from the project – RegistrationService. It simulates a third-party web service sending PNs to WP7 client application [30]. Figure 26 shows the file structure of this project.

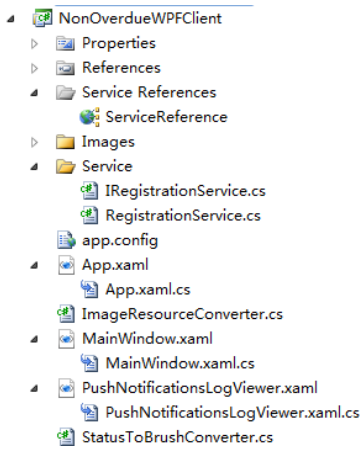

Figure 26 File structure of NonOverdueWPFClient

In Figure 26, ServiceReference is from NonOverdueServer; IRegistrationService.cs and RegistrationService.cs contain functions to register/unregister clients, hold all the registered clients in a list, check clients' requests on registration to avoid double registration; app.config contains the information of the service; App.xaml contains style information as resources and App.xaml.cs sets the host of the service which is running when the application starts up.

MainWindows is the main window of the WPF client which can send tile, toast and raw PNs to WP clients. It contains a PushNotificationLogViewer to show the logs of PNs which is sent by this client, including PN information and responds from MPNS.

The client can get food types from the database via NonOverdueServer service. Once the user chooses the type of the food and types the number of days left on this client, it shows the tile and HTTP message that are created according to the item data (Figure 27). User can also type the toast message to send to the clients.

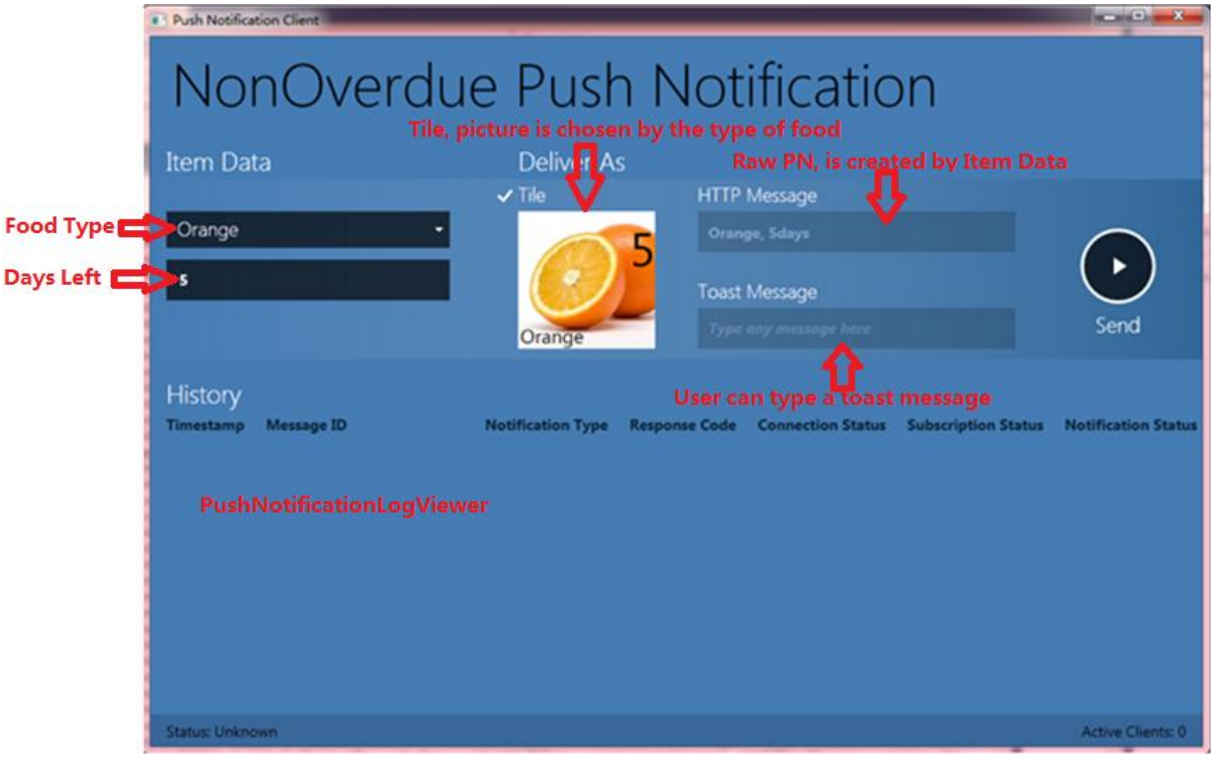

Figure 27 NonOverdue Push Notification WPF Client

5.3 NonOverdueWP7Client

NonOverdueWP7Client is a Silverlight WP7 application. It communicates with database via service reference form NonOverdueServer and receives PNs from NonOverdueWPFClient. It simulates a third-party WP7 client application receiving PNs from web service. Figure 28 shows the file structure of this project.

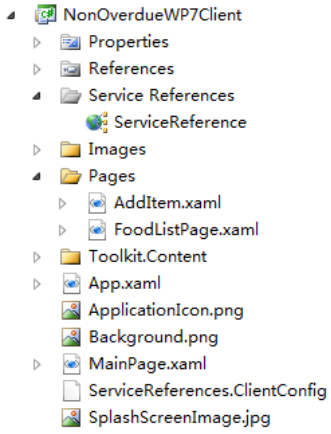

Figure 28 File structure of NonOverdueWP7Client

ServiceReference is a service reference from NonOverdueServer to communicate with the database. MainPage is used for a user to log in to the application (Figure 29). FoodListPage shows all the items of the specific user in a list box and contain the status of PN channel information (Figure 30).

Figure 29 MainPage

Figure 30 FoodListPage

User can use this application to add, edit and delete items (Figure 31). When a user adds an item, the food type can be selected from a list picker or added as a new type if it does not exist in the database, and the expiration data can be chosen from a date picker (Figure 32).

Figure 31 Add, edit and delete menu on FoodListPage

Figure 32 AddItem page

5.4 NotificationSenderUtility

It is a c# class library which holds all the functionality need to communicate with MPNS [30]. It is used by NonOverdueWPFClient to send PNs to the registered clients.

The class prepares toast and tile notification to corresponding payloads, checks the payloads of PNs, and uses HttpWebRequest class to prepare and write the request stream to channel URI which completes sending the message to MPNS. After sending the information, it waits for response from MPNS and informs the caller. As the code can be found in the appendix, we will not go into more detailed explanation here.

5.5 Simulation of PNS

When the system is running, a WPF client and a WP7 client are open as Figure 33. The user must login to be activated. The status is shown as unknown on WPF client and no active client is connected. To have more clear status of channel creating, the function for creating a new channel is put on the FoodListPage of WP7 client after the user logs in to the application.

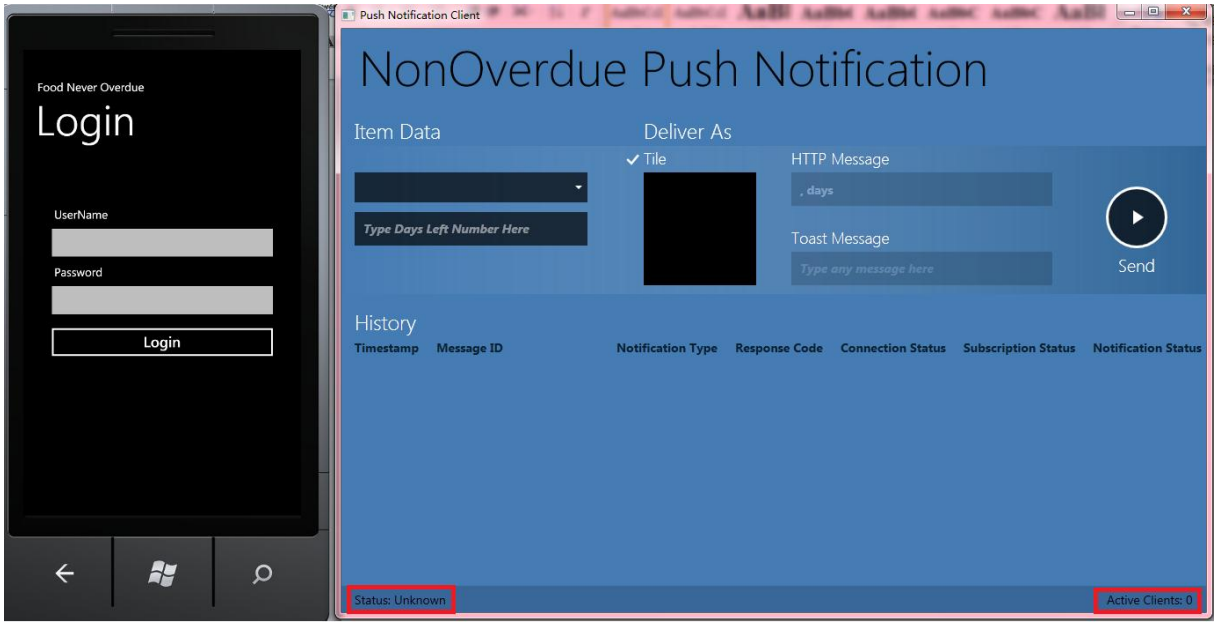

Figure 33 WPF and WP7 clients initialized

Once the user logs in, the WP7 clients try to check the channel data in isolated storage for

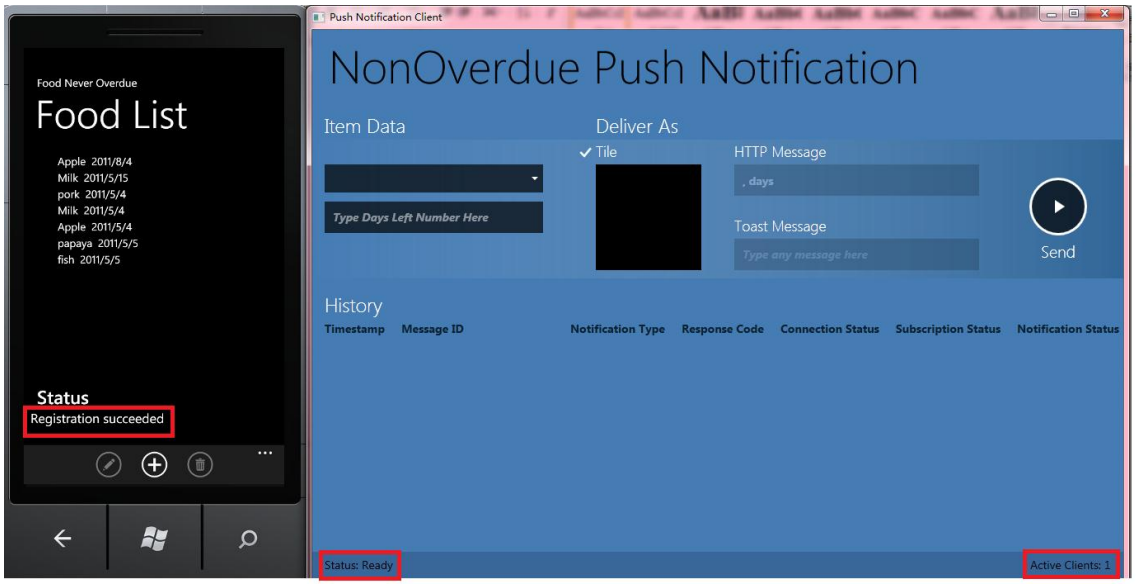

Figure 34 Registration succeeded for WP7 application

the current application. If the channel data is found, the WP7 clients load the channel that is found and uses it as PN channel (Figure 35). Otherwise, if the channel data is not found, the WP7 tries to create a new one. After creating a new PN channel, the channel data is saved in isolated storage for this application so that it can be used directly next time (Figure 36). The status on the WP7 client shows different status of registration process. The status shows as "Ready" when the channel is created and registration is completed (Figure 34).

```
Getting IsolatedStorage for current Application
Checking channel data
Channel data exists! Loading...
Finding channel
```

Figure 35 Channel data exists in isolated storage

```
Getting IsolatedStorage for current Application
Checking channel data
Channel data not found
Trying to create a new channel...
New Push Notification channel created successfully
Trying to open the channel
Channel opened. Got Uri:
http://db3.notify.live.net/throttledthirdparty/01.00/AAGtySdAdfwXQrsDJ5DdQAhCAgAAAAADAQAAAAQUZm52OkJCMjg1QTg1QkZDMkUxREQ
Subscribing to channel events
Getting IsolatedStorage for current Application
Creating data file
Saving channel data...
Saving done
Registering to Toast Notifications
Registering to Tile Notifications
```

Figure 36 Creating a new channel for the application

When the developer wants to send a raw notification from the WPF client, the developer

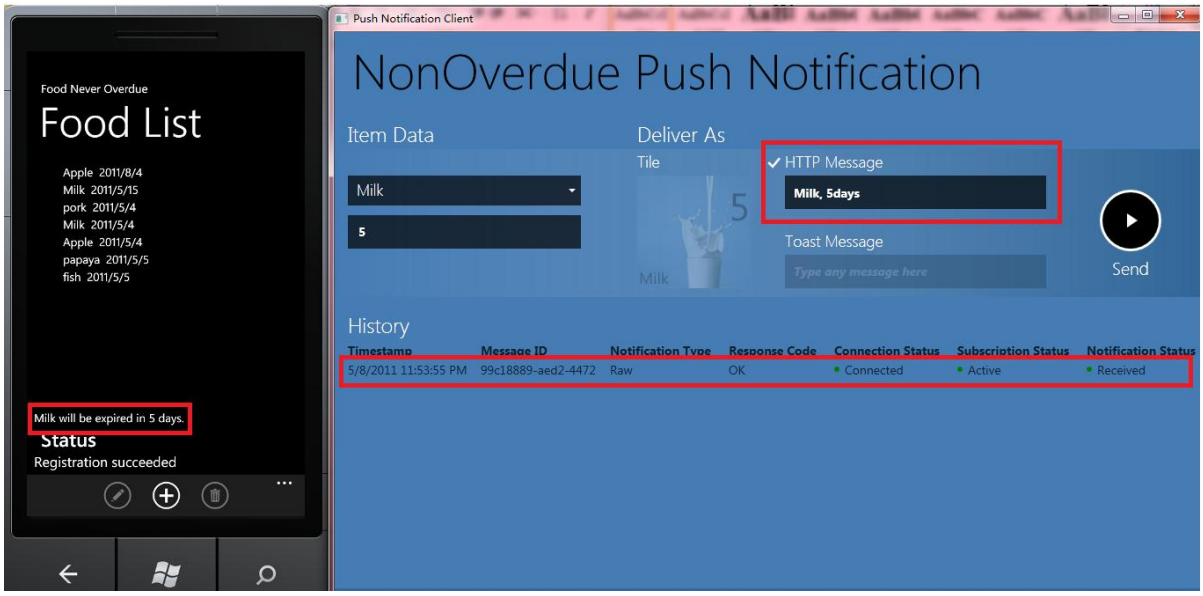

Figure 37 Sending a raw notification

chooses the food type and types the number of days left on the client. An HTTP message is created with these two data and written into XML to send to the WP7 client via MPNS. If the application is running at the moment, the raw notification is delivered to the application and the WPF client gets a received call back for this PN. A string which contains the HTTP message is shown on the WP7 client (Figure 37). Raw notification information is also traced as Figure 38.

```
==========================================
RAW notification arrived:
Got foodType: Milk
Got daysLeft: 5
Milk will be expired in daysLeft 5
==========================================
```

Figure 38 Trace of a raw notification

To send a tile notification through the WPF client, a simulated tile is shown on the WPF client with the item date. The picture is chosen by the type of food, the title of the tile is the type of food and the count is the number of days left. When the tile of the WP7 client is pinned to the main shell, the tile is able to deliver to the device, which is shown as Figure 39.

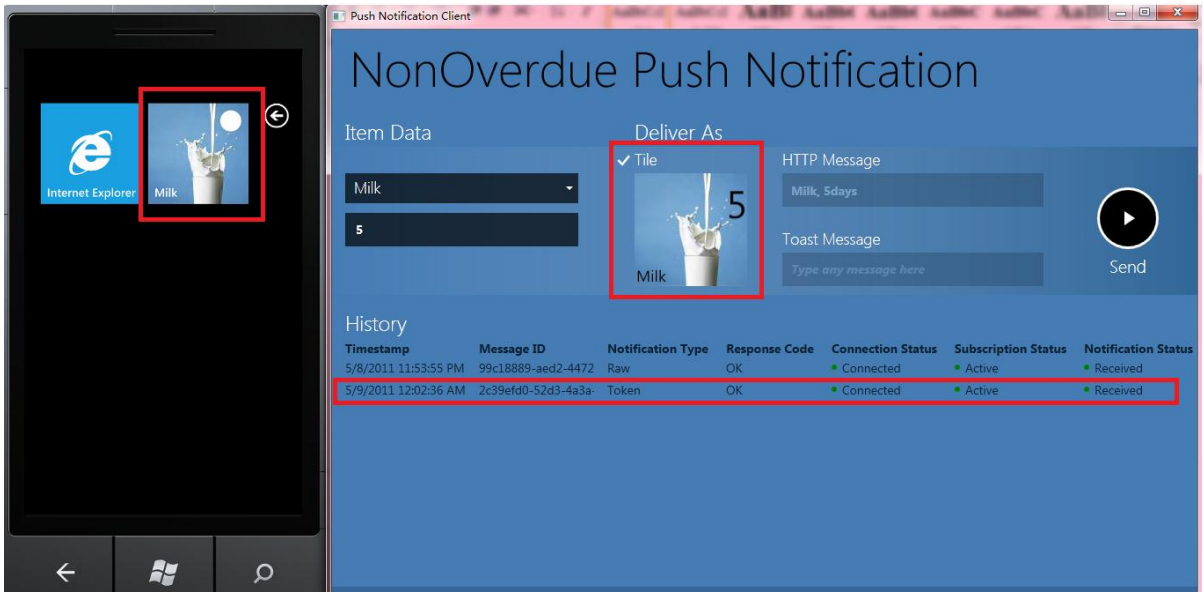

Figure 39 Sending a tile notification

To send a toast notification, a message must be typed in the textbox before sending (because the textbox is cleared after sending, we cannot see the message in the textbox any more). A toast notification is shown once it is received (Figure 40).

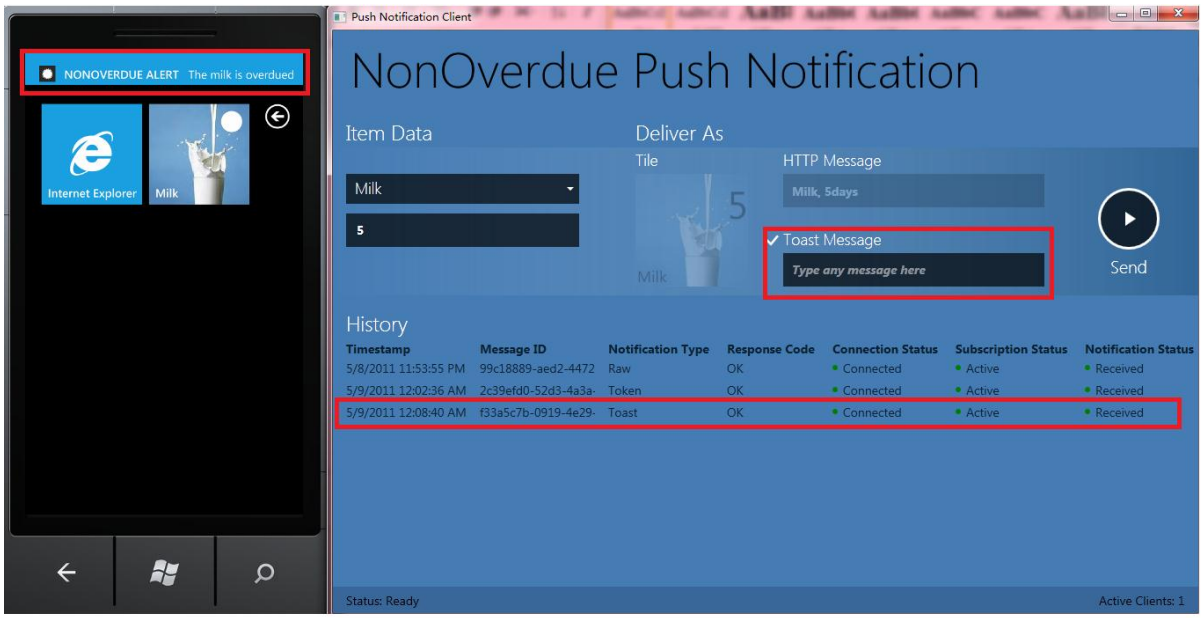

Figure 40 Sending a toast notification

6. Conclusion

PNS is a solution to low battery consumption to inform mobile users that new information of interest is available. MPNS is one of PNS used in WP7. This thesis presents the concept of MPNS, the three types of PN (toast, tile and raw notifications) and how MPNS works. By comparison with other PNSs, such as iOS and Android, and use case analysis, it indicates the different usage among different PN services and types, and gives the developer an idea of using MPNS. Use cases are discussed to investigate the motivation of PN and requirements of applications and scenarios to use PNS. This article is useful for the developers who wish to use MPNS when implementing applications for windows phone.

NonOverdue system project is implemented for simulate PN sending and receiving between a web service and a WP7 client application. It is programmed with WCF, LINQ, C#, VS2010, WPF, Silverlight v4, SQL Server 2008 R2, etc. technologies. The system provides clear process of registration to MPNS and sending/receiving PNs via MPNS, and code examples of key functions to use MPNS.

The whole research on MPNS is based on that WP does not support multi-tasking right now. MPNS is a great solution to inform the end users about new information of interest, but it does not replace multi-tasking. On the other hand, when the phone supports multi-tasking, it does not mean MPNS is unnecessary. It is possible to study on the topic whether we still need PN at that moment for further research when WP supports multi-tasking someday in the future.

Reference

1. “Eric Schmidt: Mobile Is The Future, And There’s No Such Thing As Communication Overload”. http://techcrunch.com/2010/04/12/eric-schmidt-mobile-is-the-future-and-theres-no-such-thing-as-communication-overload/. Retrieved 9 May 2011

2. Qiu, Liankui, Panlong He, and Lei Luo. “The Strategy of Advancing Mobile Web Application’s Layout and Drawing”. In Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, 2148–2152. CIT ’10. Washington, DC, USA: IEEE Computer Society, 2010.

3. Buennemeyer, Timothy, Theresa Nelson, Randy Marchany, and Joseph Tront. “Polling the smart battery for efficiency: Lifetime optimization in battery-sensing intrusion protection systems”. In Proceedings 2007 IEEE SoutheastCon, 740-745. Richmond, VA, USA, 2007.

4. Hoikkanen, A.; “A Techno-Economic Analysis of 3G Long-Term Evolution for Broadband Access” Telecommunication Techno-Economics, 2007. CTTE 2007. 6th Conference on 14-15 June 2007 Page(s):1 – 7 Digital Object Identifier 10.1109/CTTE.2007.4389887

5. Junghoon Lee ; Changik Kang ; Mikyung Kang ; “An error control scheme based on adaptation of polling schedule for real-time communication on IEEE 802.11 wireless LANs” Wireless Personal Multimedia Communications, 2002. The 5th International Symposium on. Volume: 3. Digital Object Identifier: 10.1109/WPMC.2002.1088340. Publication Year: 2002 , Page(s): 1058 - 1062 vol.3

6. Hou, Fen, James She, Pin-Han Ho, and Xuemin Shen. “A flexible resource allocation and scheduling framework for non-real-time polling service in IEEE 802.16 networks”. IEEE Transactions on Wireless Communications 8, no. 2 (2009): 766-775.

7. Miller, Matthew. Windows Phone 7 Companion. John Wiley and Sons, 2010.

8. Wang, Jingyang, Xiaohong Wang, Liwei Guo, Min Huang, and Huiyong Wang. “A New Web Services Model Based on CORBA and Push Technology in Network Management System”. In Proceedings of the 2008 Second International Conference on Genetic and Evolutionary Computing, 193–196. Washington, DC, USA: IEEE Computer Society, 2008.

9. Tosi, Davide. “An advanced architecture for push services”. In Proceedings of the Fourth international conference on Web information systems engineering workshops, 193–200. WISEW’03. Washington, DC, USA: IEEE Computer Society, 2003.

10. Khare, Rohit. “The Transfer Protocols”. IEEE Internet Computing 2 (March 1998): 80–82.

11. Martin-Flatin, J. P. “Push vs. pull in web-based network management”. In Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on, 3–18, 1999.

12. “Remote Instrument Control with CIMA Web Services and Web 2.0 Technology”. Text.Serial.Journal, February 5, 2008.

13. Myman, D. M, and C. C Watkins. Destroyable Instant Message (IM). Google Patents, 2008.

14. Bruzenak, Dylan, Ben Smith, Joachim Bondo, Owen Goss, Peter Honeder, Ray Kiddy, Steve Finkelstein, et al. iPhone Advanced Projects. Demystifying Apple’s Push Notification Service. 1st ed. Apress, 2009.

15. "NIST.gov – Computer Security Division – Computer Security Resource Center". Csrc.nist.gov.

16. Yochay Kiriaty. Understanding Microsoft Push Notifications for Windows Phones. http://windowsteamblog.com/windows_phone/b/wpdev/archive/2010/05/03/understanding-microsoft-push-notifications-for-windows-phones.aspx. Retrieved 9 May 2011

17. “How to: Send a Push Notification for Windows Phone”. http://msdn.microsoft.com/en-us/library/ff402545(v=vs.92).aspx. Retrieved 9 May 2011

18. “Push Notification Custom HTTP Headers for Windows Phone”. http://msdn.microsoft.com/en-us/library/ff941105(v=VS.92).aspx. Retrieved 9 May 2011

19. Dominic Betts, Federico Boerr, Scott Densmore, Jose Gallardo Salazar, Alex Homer. Windows® Phone 7 Developer Guide: Building connected mobile applications with Microsoft Silverlight®. Microsoft Press, February 8. 160. ISBN: 978-0-7356-5609-3

20. Yochay Kiriaty. Understanding How Microsoft Push Notification Works – Part 2. http://windowsteamblog.com/windows_phone/b/wpdev/archive/2010/05/04/understanding-how-microsoft-push-notification-works-part-2.aspx. Retrieved 9 May 2011

21. Dominic Betts, Federico Boerr, Scott Densmore, Jose Gallardo Salazar, Alex Homer. Windows® Phone 7 Developer Guide: Building connected mobile applications with Microsoft Silverlight®. Microsoft Press, February 8. 147. ISBN: 978-0-7356-5609-3

22. Apple digital library. "Local and Push Notification Programming Guide: Apple Push Notification Service". http://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/ApplePushService/ApplePushService.html#//apple_ref/doc/uid/TP40008194-CH100-SW1. Retrieved 9 May 2011

23. "Android Push Notification | Start IT up".

    http://www.techjini.com/blog/2009/12/01/android-push-notification/. Retrieved 9 May 2011

24. "Video: Learn about C2DM - the Android Cloud to Device Messaging Framework -

    Marakana". http://marakana.com/forums/android/general/299.html. Retrieved 9 May 2011

25. "Android Cloud to Device Messaging Framework - Google Projects for Android".

    http://code.google.com/android/c2dm/. Retrieved 9 May 2011

26. "iPhone 'Push Notification' Official, Testing Starts With The AP App | PhoneDig".

    http://www.phonedig.com/2009/05/iphone-push-notification-official-testing-starts-with-the-ap-app.html. Retrieved 9 May 2011

27. "Android Notifications". http://mobilepearls.com/labs/android-notifications/. Retrieved 9

    May 2011

28. "Notification Notes Free - Android".

    http://www.androidzoom.com/android_applications/productivity/notification-notes-free_cfpe.html. Retrieved 9 May 2011

29. "BBC: World's largest broadcaster & Most trusted media brand".

    http://www.medianewsline.com/news/151/ARTICLE/4930/2009-08-13.html. Retrieved 9 May 2011

30. “Using Push Notifications | Silverlight for Windows Phone | Windows Phone 7 Developer Training Kit | Courses | Learn | Channel 9”. http://channel9.msdn.com/Learn/Courses/WP7TrainingKit/WP7Silverlight/UsingPushNotificationsLab.

APPENDIX 1

The code below shows the code in NonOverdueWP7Client project in programming language C#.

// FoodListPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;
using NonOverdueWP7Client.ServiceReference;
using Microsoft.Phone.Shell;
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;

namespace NonOverdueWP7Client.Pages
{
    public partial class FoodListPage : PhoneApplicationPage
    {
        Service1Client srv = new Service1Client();


        private HttpNotificationChannel httpChannel;
        const string channelName = "ItemUpdatesChannel";
        const string fileName = "PushNotificationsSettings.dat";
        const int pushConnectTimeout = 30;

        public FoodListPage()
        {
            InitializeComponent();
            if (!TryFindChannel())
                DoConnect();

            this.Loaded +=new RoutedEventHandler(FoodListPage_Loaded);
            srv.GetItemListCompleted                                    +=new
EventHandler<GetItemListCompletedEventArgs>(srv_GetItemListCompleted);
            srv.DeleteItemCompleted                                +=                  new
EventHandler<DeleteItemCompletedEventArgs>(srv_DeleteItemCompleted);
        }

        #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

#region Misc logic
private void DoConnect()
{
    try
    {
        //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();

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

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

private void ParseRAWPayload(Stream e, out string foodType, out string daysLeft)
{
    XDocument document;
    using (var reader = new StreamReader(e))
    {
        string payload = reader.ReadToEnd().Replace('\0',
          ' ');
        document = XDocument.Parse(payload);
    }
    foodType = (from c in document.Descendants("ItemUpdate")
```

```
                    select c.Element("FoodType").Value).FirstOrDefault();
            Trace("Got foodType: " + foodType);
            daysLeft = (from c in document.Descendants("ItemUpdate")
                    select c.Element("DaysLeft").Value).FirstOrDefault();
            Trace("Got daysLeft: " + daysLeft);
        }
        #endregion

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

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

            //general error handling for push channel
            httpChannel.ErrorOccurred += new
EventHandler<NotificationChannelErrorEventArgs>(httpChannel_ExceptionOccurred);

            //subscrive to toast notification when running app
            httpChannel.ShellToastNotificationReceived += new
EventHandler<NotificationEventArgs>(httpChannel_ShellToastNotificationReceived);
        }

        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));
        }

        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)
```

```
            {
                // 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)
            {
                //handle error here
            }
        }
        #endregion

        #region Channel event handlers
        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"));
        }

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

        void httpChannel_HttpNotificationReceived(object sender, HttpNotificationEventArgs e)
        {

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

            string foodType, daysLeft;
            ParseRAWPayload(e.Notification.Body, out foodType, out daysLeft);
            Dispatcher.BeginInvoke(() => this.txtFoodTypeReceived.Text = foodType + " will be expired in
");
            Dispatcher.BeginInvoke(() => this.txtDaysLeftReceived.Text = daysLeft + " days.");
            Trace(string.Format("{0} will be expired in daysLeft {1}", foodType, daysLeft));

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

        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("==========================================");
        }
        #endregion

        #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

            void AddItem_OnClick(object sender, EventArgs e)
            {
                this.NavigationService.Navigate(new Uri("/Pages/AddItem.xaml", UriKind.Relative));
            }

            private void DeleteItem_OnClick(object sender, EventArgs e)
            {
                if (lstItem.SelectedItem != null)
                    srv.DeleteItemAsync((lstItem.SelectedItem as CompositeItemType).ItemID);
            }

            void srv_DeleteItemCompleted(object sender, DeleteItemCompletedEventArgs e)
            {
                if (e.Result == true && (App.Current as App).UserID != 0)
                    srv.GetItemListAsync((App.Current as App).UserID);
            }

            void srv_GetItemListCompleted(object sender, GetItemListCompletedEventArgs e)
            {
                if (e.Result != null)
                {
                    lstItem.ItemsSource = e.Result;
                }
            }

            public void FoodListPage_Loaded(object sender, RoutedEventArgs e)
            {
                srv.GetItemListAsync((App.Current as App).UserID);
            }

            private void lstItem_SelectionChanged(object sender, SelectionChangedEventArgs e)
            {
                if (lstItem.SelectedItem != null)
                {
                    (this.ApplicationBar.Buttons[0] as ApplicationBarIconButton).IsEnabled = true;
                    (this.ApplicationBar.Buttons[2] as ApplicationBarIconButton).IsEnabled = true;
                }
                if (lstItem.Items.Count == 0)
                {
                    (this.ApplicationBar.Buttons[0] as ApplicationBarIconButton).IsEnabled = false;
                    (this.ApplicationBar.Buttons[2] as ApplicationBarIconButton).IsEnabled = false;
                }
            }
        }
}
```

APPENDIX 2

The code below shows the code in NonOverdueWPFClient project in programming language C#.

// RegistrationService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace NonOverdueWPFClient.Service
{
    public class RegistrationService : IRegistrationService
    {
        #region Register Service
        public static event EventHandler<SubscriptionEventArgs> Subscribed;
        private static List<Uri> subscribers = new List<Uri>();
        private static object obj = new object();

        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);
        }

        #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

        #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

        #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

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

        #endregion
    }
}
```

// App.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Windows;
using System.ServiceModel;
using NonOverdueWPFClient.Service;
using System.ServiceModel.Description;

namespace NonOverdueWPFClient
{
    /// <summary>
    /// Interaction logic for App.xaml
    /// </summary>
    public partial class App : Application
    {
        ServiceHost host;

        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);

            try
            {
                host = new ServiceHost(typeof(RegistrationService));
                host.Open();
            }
            catch (TimeoutException timeoutException)
            {
                MessageBox.Show(String.Format("The service operation timed out. {0}",
timeoutException.Message));
            }
            catch (CommunicationException communicationException)
            {
                MessageBox.Show(String.Format("Could not start service host. {0}",
communicationException.Message));
            }
        }
        protected override void OnExit(ExitEventArgs e)
        {
            if (host != null)
            {
                try
                {
                    host.Close();
                }
                catch (TimeoutException)
                {
                    host.Abort();
                }
                catch (CommunicationException)
                {
                    host.Abort();
                }
            }
            base.OnExit(e);
        }
    }
}
```

// MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using NonOverdueWPFClient.ServiceReference;
using System.Collections.ObjectModel;
using System.Threading;
using System.IO;
using System.Xml;
using WindowsPhone.PushNotificationManager;
using NonOverdueWPFClient.Service;


namespace NonOverdueWPFClient
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        #region Private variables
        private ObservableCollection<CallbackArgs> trace = new ObservableCollection<CallbackArgs>();
        private NotificationSenderUtility notifier = new NotificationSenderUtility();
        private string[] lastSent = null;
        Service1Client srv = new Service1Client();
        #endregion

        public MainWindow()
        {
            InitializeComponent();

            InitializeFoodType();
            Log.ItemsSource = trace;
            RegistrationService.Subscribed += new
EventHandler<RegistrationService.SubscriptionEventArgs>(RegistrationService_Subscribed);
        }

        private void InitializeFoodType()
        {
            if (srv.GetFoodTypeList() != null)
            {
                cmbFoodName.ItemsSource = srv.GetFoodNameList();
                cmbFoodName.DisplayMemberPath = "Value";
                cmbFoodName.SelectedValue = "Key";
                cmbFoodName.SelectedIndex = 0;
            }
        }

        #region Event Handlers
```

```
        void RegistrationService_Subscribed(object sender, RegistrationService.SubscriptionEventArgs e)
        {
            //Check previous notifications, and resent last one to connected client
            if (null != lastSent)
            {
                string foodType = lastSent[0];
                string daysLeft = lastSent[1];
                List<Uri> subscribers = new List<Uri>();
                subscribers.Add(e.ChannelUri);
                byte[] payload = prepareRAWPayload(foodType, daysLeft);
                ThreadPool.QueueUserWorkItem((unused)    =>    notifier.SendRawNotification(subscribers,
payload, OnMessageSent));
            }

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

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

        private void btnSend_Click(object sender, RoutedEventArgs e)
        {
            if ((bool)rbnTile.IsChecked) sendTile();
            else if ((bool)rbnHttp.IsChecked) sendHttp();
            else if ((bool)rbnToast.IsChecked) sendToast();
        }

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

        private void sendTile()
        {
            string foodType = cmbFoodName.Text;
            int daysLeft = Int32.Parse(txtDaysLeft.Text);
            List<Uri> subscribers = RegistrationService.GetSubscribers();
            ThreadPool.QueueUserWorkItem((unused)        =>        notifier.SendTileNotification(subscribers,
"PushNotificationsToken",
                "/Images/" + foodType + ".jpg", daysLeft, foodType, OnMessageSent));
        }

        private void sendHttp()
        {
            //Get the list of subscribed WP7 clients
            List<Uri> subscribers = RegistrationService.GetSubscribers();
            //Prepare payload
            byte[] payload = prepareRAWPayload(
                cmbFoodName.Text,
                txtDaysLeft.Text);
            //Invoke sending logic asynchronously
            ThreadPool.QueueUserWorkItem((unused) => notifier.SendRawNotification(subscribers,
                                        payload,
```

```
                                                    OnMessageSent)
                    );

                //Save last RAW notification for future usage
                lastSent = new string[2];
                lastSent[0] = cmbFoodName.SelectedValue as string;
                lastSent[1] = txtDaysLeft.Text;
            }
            #endregion

            #region Private functionality
            private void UpdateStatus()
            {
                int activeSubscribers = RegistrationService.GetSubscribers().Count;
                bool isReady = (activeSubscribers > 0);
                txtActiveConnections.Text = activeSubscribers.ToString();
                txtStatus.Text = isReady ? "Ready" : "Waiting for connection...";
            }

            private static byte[] prepareRAWPayload(string foodType, string daysLeft)
            {
                MemoryStream stream = new MemoryStream();
                XmlWriterSettings settings = new XmlWriterSettings() { Indent = true, Encoding =
Encoding.UTF8 };
                XmlWriter writer = XmlTextWriter.Create(stream, settings);
                writer.WriteStartDocument();
                writer.WriteStartElement("ItemUpdate");
                writer.WriteStartElement("FoodType");
                writer.WriteValue(foodType);
                writer.WriteEndElement();
                writer.WriteStartElement("DaysLeft");
                writer.WriteValue(daysLeft);
                writer.WriteEndElement();
                writer.WriteStartElement("LastUpdated");
                writer.WriteValue(DateTime.Now.ToString());
                writer.WriteEndElement();
                writer.WriteEndElement();
                writer.WriteEndDocument();
                writer.Close();
                byte[] payload = stream.ToArray();
                return payload;
            }
            #endregion
    }
}
```

// PushNotificationsLogViewer.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Collections;

namespace NonOverdueWPFClient
{
    /// <summary>
    /// Interaction logic for PushNotificationsLogViewer.xaml
    /// </summary>
    public partial class PushNotificationsLogViewer : UserControl
    {
        public PushNotificationsLogViewer()
        {
            InitializeComponent();
            tracelog.LostFocus += (sender, e) => { tracelog.SelectedIndex = -1; };
        }

        public IEnumerable ItemsSource
        {
            get
            {
                return this.tracelog.ItemsSource;
            }

            set
            {
                this.tracelog.ItemsSource = value;
            }
        }
    }
}
```

APPENDIX 3

The code below shows the code in NotificationSenderUtility project in programming language C#.

```
// NotificationSenderUtility.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.IO;
using System.Xml;

namespace WindowsPhone.PushNotificationManager
{
    #region NotificationType Enum
    public enum NotificationType
    {
        Token = 1,
        Toast = 2,
        Raw = 3
    }
    #endregion

    #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_HEADER];
            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

    public class NotificationSenderUtility
    {
        #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

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

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

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

        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);
        }
        #endregion

        #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

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

        #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
                //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";

                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);
            }
            catch (WebException ex)
            {
                if (ex.Status == WebExceptionStatus.ProtocolError)
                {
                    //Notify client on exception
                    OnNotified(notificationType, (HttpWebResponse)ex.Response, callback);
                }

                throw;
            }
        }
        #endregion

    }
}
```