Mobile Operator Notifications and System Events

May 31, 2012

Abstract

This paper provides information about the Mobile Operator Notification system event for Windows 8 Release Preview. It provides guidelines for mobile operators to develop mobile broadband Metro style apps that effectively handle incoming SMS or USSD-based mobile operator notifications and relevant mobile broadband system events. It assumes that the reader is familiar with the Windows Mobile Broadband Platform and basic Metro style app development concepts.

This information applies to the following operating systems:

Windows 8 Release Preview

References and resources discussed here are listed at the end of this paper.

The current version of this paper is maintained on the Web at:

Mobile Operator Notifications and System Events

Disclaimer: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet website references, may change without notice. Some information relates to pre-released product which may be substantially modified before it’s commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes. You may modify this document for your internal, reference purposes.

© 2012 Microsoft. All rights reserved.
**Document History**

<table>
<thead>
<tr>
<th>Date</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>February 28, 2012</td>
<td>First publication</td>
</tr>
<tr>
<td>May 31, 2012</td>
<td>Updated information about the DataPlanThresholdReached message, Defining filtering rules in provisioning XML, getting SMS text messages, and troubleshooting. Also, updates to Sample backgroundtask.js.</td>
</tr>
</tbody>
</table>

**Contents**

Introduction.................................................................................................................. 4  
Background brokered work items in Windows 8............................................................. 4  
Terms and definitions..................................................................................................... 5  
MobileOperatorNotification scenarios............................................................................. 5  
Connect to and disconnect from mobile broadband......................................................... 5  
Network operator messages............................................................................................... 5  
Mobile network operator SMS notifications..................................................................... 6  
Network-initiated USSD.................................................................................................... 7  
Triggering data usage and roaming notifications locally............................................... 7  
Data usage notification via local data counters............................................................ 7  
Roaming notification via Windows Connection Manager................................................ 8  
Data plan expiration and usage reset............................................................................. 8  
MobileOperatorNotification Event Technical Details...................................................... 8  
Event Payload.................................................................................................................. 8  
GSM/CDMA SMS and USSD.................................................................................................. 9  
DataPlanThresholdReached.............................................................................................. 9  
DataPlanReset.................................................................................................................. 10  
DataPlanDeleted............................................................................................................. 10  
ProfileConnected and ProfileDisconnected..................................................................... 10  
RegisteredRoaming and RegisteredHome....................................................................... 10  
Registering for the MobileOperatorNotification event via metadata............................. 10  
Defining filtering rules in provisioning XML................................................................. 11  
Developing the App to Handle the MobileOperatorNotification Event............................ 13  
Best practices.................................................................................................................. 13  
Step 1: Background task contract declaration................................................................ 13  
Step 2: Background task handler.................................................................................... 15  
Show toast notification..................................................................................................... 18  
Get SMS text message.................................................................................................... 20  
Use local storage............................................................................................................. 20  
Step 3: Handle the Activation event............................................................................... 21  
Step 4: Handle background task completion handlers................................................... 22  
Troubleshooting............................................................................................................... 24  
Verifying Windows is receiving SMS and USSD............................................................... 24  
Received SMS messages are not detected as operator notifications............................. 24  
Triggering metadata parsing to register background tasks.............................................. 24
Appendix.......................................................................................................................... 24
Resources.......................................................................................................................... 27
Introduction

A customer’s primary experience of the operator’s brand on Windows 8 will be the Metro style device app for mobile broadband. This app is not expected to provide primary connection management functions, but to provide an account management experience and a service experience. In order to keep the user informed about their account status, the app will need to perform some activities even when the user is not interacting with it. These activities include responding to operator SMS or network-initiated USSD messages, notifying the user that they are approaching their data limit, notifying the user that their data plan has expired, and notifying the user of their roaming status.

Background brokered work items in Windows 8

Windows 8 introduces Metro style apps, including Metro style device apps for mobile broadband, that run on the full screen. End users are only expected to interact with the application that is in the foreground. The foreground app is assumed to be the most important to the user, so this app receives all the resources of the system. When an app is not in the foreground, it is suspended and cannot run any code. A suspended app remains suspended until the user resumes it by bringing the app back to the foreground. With this model of app behavior, the user experience is never affected by lags or delays caused by the execution of unimportant background apps. In addition, reducing unnecessary background activity optimizes battery life on a variety of form factors. The time taken to resume a suspended app is negligible and would appear to be almost unnoticeable to most users.

Windows 8 provides Windows push notifications that can be used to keep the app tile fresh and up-to-date even when the app is suspended. Push notifications are optimized for system performance and longer device battery life, so it’s best to use Windows push notifications whenever possible. If a suspended app must run its own code to do other kinds of work, Windows 8 provides apps with the ability to create background tasks.

Although Metro style apps cannot run any code if they are not running in the foreground, the System Event Broker in Windows 8 provides facilities to run code in response to events while an app is in the background. Apps can register work items with the System Event Broker to respond to specific background brokered events. Windows runs the app’s work item when background brokered events are triggered, regardless of the app’s current state (active or suspended).

In general, background events are intended as simple trigger points and are not intended to signal large amounts of processing. As such, quotas for each app are placed on the processing time allowed for background events. The background events offered by the Network Operator API, including the MobileOperatorNotification event and HotspotAuthentication event, are treated by Windows as critical events. Compared to general background events in Windows 8, background work items associated with MobileOperatorNotification and HotspotAuthentication events will execute for every instance of the event regardless of a processing time quota, though each instance of the background work item is subject to a processing time quota. Care should be taken to only do minimal
processing in the background event handler and to defer larger processing to the application.

For more information on the **HotspotAuthentication** background event, see [Windows 8 Integration for Wireless Hotspot Operators](#).

### Terms and definitions

This paper uses the following terms:

- **Background Task**
  - A class or JavaScript page implemented by the app to provide functionality even if the app is not in the foreground.

- **Background Trigger**
  - A system-defined event that an app can associate with a background task. When the system fires a trigger, an app background task associated with the trigger is launched.

- **Foreground app**
  - The app that the user is actively interacting with.

### MobileOperatorNotification scenarios

The **MobileOperatorNotification** event is a critical background event that targets different scenarios in which the end user must be notified of changes to their service or account. The event payload is used to differentiate between the different scenarios that trigger the **MobileOperatorNotification** event. In each case, the Metro style device app for mobile broadband will be required to run code to effectively complete the scenario. The following sections describe these scenarios in more detail.

#### Connect to and disconnect from mobile broadband

The Windows Connection Manager (WCM) monitors available networks across Wi-Fi, mobile broadband, and Ethernet. It makes automatic connect and disconnect decisions based on the available networks. When the WCM connects to and disconnects from a mobile broadband profile, a **MobileOperatorNotification** background event is triggered. This allows the operator’s app to perform any logic necessary when connecting to their network, such as verifying account status, retrieving the most recent data usage, or displaying toast notifications and tile updates to the user to inform them of the relevant account status.

#### Network operator messages

The mobile broadband platform in Windows 8 provides enhanced functionality available only to an operator’s Metro style device app for receiving and displaying incoming SMS and network-initiated USSD operator administrative messages. These messages can be intended for user notification, such as approaching data usage cap, international roaming, or low balance, or to trigger a response from the operator’s app.

The app handles the incoming message as appropriate. Likely responses include any or all of the following:

- Immediately syncing current data usage
• Updating the Metro style app’s tile
• Retrieving and applying updated operator provisioning XML
• Displaying a toast notification to the user, such as:

The user can tap or click the notification to be brought directly to the operator’s app, as illustrated by the figure below.

If an operator wants to display the message in the app, the background task triggered by the **MobileOperatorNotification** event needs to read the message contents and store the message contents in the app’s own local data storage. The mobile broadband SMS platform does not maintain a queue of past received administrative SMS notifications.

**Mobile network operator SMS notifications**

Incoming SMS messages are available to any app that has requested and been granted access to the SMS capabilities on the PC. However, some SMS messages come directly from the carrier and should be restricted to and handled by the operator’s app.

The mobile broadband SMS platform filters each new received SMS into one of two types: administrative (“silent”) SMS notifications from an MNO and general SMS messages. Administrative SMS notifications received from a mobile network operator are only accessible to the MNO Metro style app and are hidden from general SMS client apps.

Mobile network operators specify custom filtering rules for administrative SMS and USSD notifications in the account provisioning metadata. If no message filtering rules are specified, the SMS platform classifies all SMS messages as general SMS messages.
available to any app. If an incoming SMS matches the provisioned filtering rules, the
**MobileOperatorNotification** event is triggered and the background work item can
appropriately handle the incoming SMS message.

**Network-initiated USSD**

Windows 8 provides a USSD API, which is an abstraction of the underlying USSD
protocol that hides most of the details to simplify application development. Upon
receiving a network-initiated USSD that matches the provisioned filtering rules, the
**MobileOperatorNotification** event will be triggered and the corresponding background
work item can communicate over the USSD session using the USSD API.

For more information about USSD APIs, see [Windows.Networking.NetworkOperators
namespace](#).

**Triggering data usage and roaming notifications locally**

In many markets, mobile network operators are required by regulatory laws to notify
a user when she reaches her data usage limit or is roaming on a more costly network.
This consumer protection mitigates the risk of “bill shock”, or extremely excessive
usage charges. In Windows, the operator’s mobile broadband app can show toast
notifications and tile updates to make the user aware of the data usage and roaming
states. These notifications can be initiated from the operator’s network back end via
SMS or USSD, which will trigger the **MobileOperatorNotification** event as in the
examples above. Alternatively, the **MobileOperatorNotification** event will be
triggered using local information in the following cases.

**Data usage notification via local data counters**

1. Operator enables local data usage notifications via provisioning metadata.
2. Local data counters estimate that usage on the profile has changed by more
   than 5% of the user’s data limit since the last update.
3. The Data Usage and Subscription Manager (DUSM) notifies the System Event
   Broker to trigger the **MobileOperatorNotification** event.
4. The System Event Broker invokes the operator’s app to handle the
   background event.
5. The app handles the event by retrieving the most current usage information
   from the operator’s back-end infrastructure.
6. If the current usage information exceeds a threshold (such as 80%), the app
displays a toast notification to the user and updates the DUSM with the
current usage. Alternatively, if the current usage does not exceed a threshold,
the app need not display the toast notification.

![Image](image-url)
Roaming notification via Windows Connection Manager

1. Windows Connection Manager registers on a mobile broadband network that reports the connection as “roaming”.

2. Windows Connection Manager notifies the System Event Broker to trigger the MobileOperatorNotification event.

3. The System Event Broker invokes the operator’s app to handle the background event.

4. The app identifies whether the user will incur additional usage charges when roaming on this network and, if necessary, displays a toast notification and tile updates to the user.

Data plan expiration and usage reset

The DUSM tracks details about the user’s account or accounts, including the plan expiration date for pre-paid data plans, or the plan usage reset date for post-paid data plans. When the user’s data plan expires, the DUSM notifies the System Event Broker to trigger the MobileOperatorNotification event. The operator’s app can handle the event by displaying a toast notification and tile update to the user, informing them that their plan has expired or directing them to renew their service.

In the case of a post-paid data plan, the DUSM will reset the plan data usage to zero on a particular date, such as the first day of the month. When this occurs, the MobileOperatorNotification event will be triggered and the app can notify the user of their updated data usage.

MobileOperatorNotification Event Technical Details

Event Payload

The MobileOperatorNotification event payload includes the following fields:

- **MessageType** – Enumeration of the message that triggered the event.
- **Interface** – The GUID corresponding to the physical interface associated with the event.
- **EncodingType** – The encoding method for the message, if MessageType is SMS/ USSD.
- **MessageDataSize** – The size of the message, in bytes, if MessageType is SMS/ USSD.
- **Message** – The raw message received, if MessageType is SMS/ USSD.

The MobileOperatorNotification event enables each of the scenarios described above by differentiating them using the MessageType field in the event payload. The MessageType are enumerated as follows:

0. GSM SMS
1. CDMA SMS
2. USSD
3. DataPlanThresholdReached
4. DataPlanReset
5. DataPlanDeleted
6. ProfileConnected
7. ProfileDisconnected
8. RegisteredRoaming
9. RegisteredHome

The work item associated with the **MobileOperatorNotification** event should begin with logic to effectively differentiate the **MessageType** and execute the appropriate code for each scenario.

**GSM/CDMA SMS and USSD**

An incoming operator message, including SMS and USSD, will trigger the **MobileOperatorNotification** event with the appropriate corresponding **MessageType**. Unique to these types are **EncodingType**, **MessageDataSize**, and **Message**.

**DataPlanThresholdReached**

By default, this message type is disabled. It must be enabled by using provisioning metadata to specify the **DataUsagelnMobileOperatorNotificationEnable** field, as shown here.

```xml
<?xml version="1.0"?>
<CarrierProvisioning
xmlns="http://www.microsoft.com/networking/CarrierControl/v1">
<Global>
  <CarrierId>{2c85b76b-f859-47c4-8122-721fe8b6c25f}</CarrierId>
  <SubscriberId>012345678901234</SubscriberId>
</Global>
<MBNPProfiles>
  <DefaultProfile
xmlns="http://www.microsoft.com/networking/CarrierControl/WWAN/v1">
    <Name>Contoso</Name>
    <AssociatedPlan>SamplePlan</AssociatedPlan>
    <Context>
      <AccessString>Contoso.com</AccessString>
      <UserLogonCred>
        <UserName>User</UserName>
        <Password>secret</Password>
      </UserLogonCred>
    </Context>
  </DefaultProfile>
</MBNPProfiles>
<Plans>
  <Plan
xmlns="http://www.microsoft.com/networking/CarrierControl/Plans/v1"
  Name="SamplePlan">
    <Description PlanType="Fixed">
      <DataLimitInMegabytes>500</DataLimitInMegabytes>
      <DataUsageInMobileOperatorNotificationEnabled>
```
true
</DataUsageInMobileOperatorNotificationEnabled>
</Description>
</Plan>
</Plans>
</CarrierProvisioning>

See Providing Mobile Broadband Metadata for additional details.

The event is generated with this MessageType when the local data counters estimate that usage (bytes sent and received) on the mobile broadband interface has changed by 5% since the last occurrence, except in the following two cases:

1. When connected to a home network (non-roaming), if the data plan limit has not been specified, this event will be triggered at every 100 MB of local data usage.

2. When connected to a roaming network, the data plan limit does not apply and this event will be triggered at every 5 MB of local data usage.

The local data counters in Windows 8 are updated at a 1 minute frequency; at most, this event will be generated once per minute in all the cases described above. It is important to note that while this information is a good first-order guide, Windows cannot account for unbillled traffic, nor for usage on other devices which share the same data limits (such as family plans or SIM-swapping). Operator apps should use local data counters only to approximate usage since the last sync with the operator's own billing system. For data usage that has already been processed, the billing system should be considered authoritative.

DataPlanReset
On the plan reset date, the DUSM resets the user’s current local data usage to zero.

DataPlanDeleted
For pre-paid data plans with a fixed expiration date, the DUSM deletes the connection profile associated with the account on the expiration date and the MobileOperatorNotification event is triggered with this MessageType. When the connection profile is deleted, Windows Connection Manager will no longer attempt to automatically connect to the network described by the connection profile.

ProfileConnected and ProfileDisconnected
The MobileOperatorNotification event is generated with these MessageTypes when the Windows Connection Manager connects to the network profile provided by the Operator Experience metadata. This event will be triggered on every connect and disconnect, including the initial connection following a sleep/resume.

RegisteredRoaming and RegisteredHome
The MobileOperatorNotification event is generated with these MessageTypes when the Windows Connection Manager registers to a network that reports as roaming. This event will be triggered on every registration, including the initial registration following a sleep/resume. The app should only notify the user once when they register on a roaming network and once when they return to their home network.
Because this event is triggered at every registration, the app is responsible for keeping track of the previous registered state in the app’s session data.

Registering for the MobileOperatorNotification event via metadata

In general, an app must be run by the user at least once before it can register work items with the System Event broker. However, because the MobileOperatorNotification events are required to complete key mobile broadband scenarios, this event is associated with the mobile broadband app via the mobile broadband service metadata. In the service metadata, add the following entry:

```
\Package\SoftwareInformation\SoftwareInfo.xml

<DeviceCompanionApplications>
  <Package>
    <Identity Name="MyOperatorNotification"
             Publisher="MyCorporation " />
    <Applications>
      <Application Id="MyOperatorNotification" />
      <DeviceNotificationHandlers>
        <DeviceNotificationHandler EventID="MobileOperatorNotificationHandler" EventAsset="backgroundtask.js" />
      </DeviceNotificationHandlers>
    </Applications>
  </Package>
</DeviceCompanionApplications>
```

In the above XML, the EventID attribute tells the system what kind of event to expect from the device. The field in the EventAsset attribute should point to the entry point that implements the background task. This will tell the system which task to run when that particular event has occurred.

For example, for the above XML, the system will create and register an event specific to that device. It will also register the Metro style app for this event. For the example above, the app must have a JavaScript file called backgroundtask.js that will be run by the system each time it receives an operator notification.

If the Metro style app is a C# app, the event asset must point to the runtime class implementing the backgroundtask interface.

```
<DeviceNotificationHandlers>
  <DeviceNotificationHandler
      EventID="MobileOperatorNotificationHandler"
      EventAsset="MNO.MessageBackground.OperatorNotification" />
</DeviceNotificationHandlers>
```

When the metadata and app are downloaded, the Device Setup Manager registers the appropriate work item with the System Event Broker, prior to the app being run once.

Defining filtering rules in provisioning XML

Windows uses metadata information from operators to customize various aspects of the Windows 8, including providing data to provision the PC. The provisioning XML specifies the custom filtering rules to differentiate operator SMS and USSD
notifications. For more information on metadata and provisioning is discussed, see Guide to Providing Mobile Broadband Metadata.

Windows accepts an XML-based provisioning file from the operator. A sample version of the provisioning XML is shown below:

```xml
<?xml version="1.0" encoding="utf-8"?>
<CarrierProvisioning
xmlns="http://www.microsoft.com/networking/CarrierControl/v1">
  <Global>
    <!-- Adjust the Carrier ID to fit match the Service Number in service
        metadata. Refer to the MSDN documentation about CarrierId. -->
    <CarrierId>11111111-1111-1111-1111-111111111111</CarrierId>
    <!-- Adjust the Subscriber ID. Refer to the MSDN documentation about
        Subscriber ID's. -->
    <SubscriberId>1234567890</SubscriberId>
  </Global>
  <MBNProfiles>
    <DefaultProfile
xmlns="http://www.microsoft.com/networking/CarrierControl/WWAN/v1">
      <!-- Adjust the profile name -->
      <Name>Contoso</Name>
      <AssociatedPlan>Limited</AssociatedPlan>
      <!-- Adjust the home provider name for the given SIM/Device -->
      <HomeProviderName>Contoso</HomeProviderName>
      <Context>
        <!-- Adjust the access string to your APN. -->
        <AccessString>Contoso.Contoso</AccessString>
        <!-- Adjust the UserLogonCred to fit your UserLogonCred.
            Refer to the MSDN documentation about UserLogonCred's. -->
        <UserLogonCred>
          <UserName>user</UserName>
          <Password>password</Password>
        </UserLogonCred>
      </Context>
    </DefaultProfile>
    <Messages
xmlns="http://www.microsoft.com/networking/CarrierControl/WWAN/v1">
      <Message RuleId="Sample1" Silent="true">
        <!-- ^ matches all messages from this sender, regardless of
            content -->
        <Pattern>^[^]*</Pattern>
        <!-- Because no fields are specified, this message will be passed
            to the operator app without parsing. -->
      </Message>
      <Message RuleId="Sample2" Silent="false">
        <!-- Parsing a simple usage message. -->
        <USSDBearer/>
        <Pattern>(\d+\.\d+)(\w+) of (\d+)(\w+) used as of (\$+)/Pattern>
        <!-- Using these field definitions, Windows will automatically
            update usage data before passing the message
            to the operator app. -->
        <Units G="GB" M="MB"/>
      </Fields>
      <!-- These fields are currently unordered, but an order will be
          required in RC. -->
      <Usage Group="1" UnitGroup="2"/>
      <UsageTimestamp Group="5" Format="%I:%M%p on %d %b"/>
      <DataLimit Group="3" UnitGroup="4"/>
    </Fields>
  </Messages>
</CarrierProvisioning>
```
The rules to identify a text message as an operator message can be defined in this XML.

- Allowed sender: The “Sender” attribute specifies the reserved sender address that the notification is allowed to arrive from. (This number must match the sender number received in the SMS message exactly, including the international format).

- Pattern: The regular expression to use to identify and optionally extract the data fields from the text message.
  - To match all messages from a sender, use pattern [^]*

More information can be found in Providing Mobile Broadband Metadata.

Developing the App to Handle the MobileOperatorNotification Event

Best practices

With respect to background event handling, the following best practices should be observed:

- Do not register for background events you can’t take action on. Processing these events will consume the application quota needlessly.
- Do not do large amounts of processing on receipt of a background event.
- Consider deferring processing to the next time the app is launched.
- Consider showing a toast notification and updating tile in response to a background event. The Metro style app can be launched to process the background event payload.

For more information on the Background Event model in Windows 8, see Introduction to Background Tasks.

Step 1: Background task contract declaration

In order for Windows to recognize the background task experiences supplied by a Metro style device app for mobile operators, the app has to declare that it is providing an extension to system functionality.

To make the declaration in the package.appxmanifest file for your Visual Studio 11 project, follow these steps:

2. In the **Declarations** tab, choose **Background Tasks** from **Available Declarations**, and then click **Add**.

3. Enter app info under **Properties**:
   a. For a Metro style device app using JavaScript and HTML, enter the file name that handles the background task in the app (for example, backgroundtask.js) in the **StartPage** field of Application Settings.
   b. For **Supported Task Types**, select **System Event**.

The following figure shows this procedure in C#:

![C# Figure]

The following figure shows this procedure in JavaScript:
If done correctly, you can see the following code if you open the package.appxmanifest in Notepad:

```xml
<Application>
  <Application Id="...">
    ...
    <Extension Category="windows.backgroundTasks" StartPage="backgroundtask.js">
      <BackgroundTasks>
        <Task Type="systemEvent" />  
      </BackgroundTasks>
    </Extension>
    ...
  </Application>
</Applications>
```

Step 2: Background task handler

If your app provides a mobile operator notifications declaration, it must supply a handler for the background task activation. The handler will get the mobile operator network account ID and event data from `Windows.UI.WebUI.WebUIBackgroundTaskinstance.current.triggerDetails` ( `Windows.Networking.NetworkOperators.NetworkOperatorNotificationEventDetails`).

As the only UI supported by background task is Toast, the background task handler can show Toast or save `NetworkOperatorNotificationEventDetails` to the local storage.
The following code examples demonstrate a background task designed to run when a new administrative SMS notification is received.

**C#:**

```csharp

namespace MNOMessageBackground
{
    public sealed class MNOBggroundTask : IBackgroundTask
    {
        public void Run(Windows.ApplicationModel.Background.IBackgroundTaskInstance taskInstance)
        {
            NetworkOperatorNotificationEventDetails notifyData =
                        (NetworkOperatorNotificationEventDetails)taskInstance.TriggerDetails;

            // The network account ID is stored in notifyData.NetworkAccountId.
            switch (notifyData.NotificationType)
            {
                case NetworkOperatorEventMessageType.Gsm: // 0
                    break;
                case NetworkOperatorEventMessageType.Cdma: // 1
                    break;
                case NetworkOperatorEventMessageType.Ussd: // 2
                    break;
                case NetworkOperatorEventMessageType.DataPlanThresholdReached: // 3
                    break;
                case NetworkOperatorEventMessageType.DataPlanReset: // 4
                    break;
                case NetworkOperatorEventMessageType.DataPlanDeleted: // 5
                    break;
                case NetworkOperatorEventMessageType.ProfileConnected: // 6
                    break;
                case NetworkOperatorEventMessageType.ProfileDisconnected: // 7
                    break;
                case NetworkOperatorEventMessageType.RegisteredRoaming: // 8
                    break;
                case NetworkOperatorEventMessageType.RegisteredHome: // 9
                    break;
                default:
                    break;
            }

            // Add code to save the message to app local storage, and optionally show toast notification and tile updates.
        }
    }
}
```

**JavaScript:**

```javascript
(function () {
    "use strict";

    // The background task instance's activation parameters are available via
    // Windows.UI.WebUI.WebUIBackgroundTaskInstance.current.

    //
```

---

May 31, 2012
© 2012 Microsoft. All rights reserved.
```javascript
var backgroundTaskInstance =
Windows.UI.WebUI.WebUIMainPage.backgroundTaskInstance.current,
networkOperatorEventType =
key = null,
settings = Windows.Storage.ApplicationData.current.localSettings;

try {

    var details = backgroundTaskInstance.triggerDetails;

    // The network account ID is stored in details.networkAccountId.
    switch (details.notificationType) {
        case networkOperatorEventType.gsm:
            showToast("Mobile Broadband message", details.message);
            break;
        case networkOperatorEventType.cdma:
            showToast("Mobile Broadband message", details.message);
            break;
        case networkOperatorEventType.ussd:
            showToast("Mobile Broadband message", details.message);
            break;
        case networkOperatorEventType.dataPlanThresholdReached:
            showToast("Mobile Broadband message", "Data plan threshold reached");
            break;
        case networkOperatorEventType.dataPlanReset:
            showToast("Mobile Broadband message", "Data plan reset");
            break;
        case networkOperatorEventType.dataPlanDeleted:
            showToast("Mobile Broadband message", "Data plan deleted");
            break;
        case networkOperatorEventType.profileConnected:
            showToast("Mobile Broadband message", "Profile connected");
            break;
        case networkOperatorEventType.profileDisconnected:
            showToast("Mobile Broadband message", "Profile disconnected");
            break;
        case networkOperatorEventType.registeredRoaming:
            showToast("Mobile Broadband message", "Registered roaming");
            break;
        case networkOperatorEventType.registeredHome:
            showToast("Mobile Broadband message", "Registered home");
            break;
        default:
            showToast("Mobile Broadband message", "Unknown message");
            break;
    }

    // A JavaScript background task must call close when it is done.
    //
    close();
}
```
Show toast notification

For toast notification and tile update experience design guidelines, see Designing User Experience of Mobile Operator Apps.

In order to show a toast notification, the app needs to specify Toast Capable in the Notifications section of .appxmanifest.

The following code shows that Toast Capable has been set in the package.appxmanifest file if viewed in Notepad.

```xml
<Application>
    <Application Id="...">
        <VisualElements ToastCapable="true"/>
    </Application>
</Applications>
```
The following code demonstrates how to show a toast notification in a background task handler with JavaScript.

JavaScript:
```javascript
function showToast(title, body) {
    var notifications = Windows.UI.Notifications;
    var toastNotificationManager = notifications.ToastNotificationManager;
    var toastXml = toastNotificationManager.getTemplateContent(notifications.ToastTemplateType.toastText02);

    var temp = "the parameter will pass to app when app activated from tap Toast ";
    toastXml.selectSingleNode("/toast").setAttribute("launch", temp);

    var textNodes = toastXml.getElementsByTagName("text");
    textNodes[0].appendChild(toastXml.createTextNode(title));
    textNodes[1].appendChild(toastXml.createTextNode(body));

    var toast = new notifications.ToastNotification(toastXml);
    toastNotificationManager.createToastNotifier().show(toast);
}
```

Get SMS text message

If the background task was triggered due to an incoming SMS message, the background task details will carry the SMS object in its payload.

JavaScript:
```javascript
(function () {
    "use strict";

    // The background task instance's activation parameters are available via Windows.UI.WebUIWebUIBackgroundTaskInstance.current.
    var backgroundTaskInstance = Windows.UI.WebUIWebUIBackgroundTaskInstance.current,
        try {
            var details = backgroundTaskInstance.triggerDetails;
            if (details.notificationType === networkOperatorEventValueType.gsm
                || details.notificationType === networkOperatorEventValueType.cdma)
                var textMessage = new Windows.Devices.Sms.SmsTextMessage.fromBinaryMessage(details.smsMessage);

                // textMessage can be used to get other SmsMessage properties like sender number, timestamp, message part count etc.
                showToast("From: " + textMessage.from + "; TimeStamp: " + textMessage.timestamp, details.message);
        }
```
Use local storage
The background task can use local storage to save the message you get from the background event, so that the app can use that information later.

The following code demonstrates how to use local storage to store the message.

```
///
/// Save the message
///
var settings = Windows.Storage.ApplicationData.current.localSettings;
var keyMessage = "BA5857FA-DE2C-4A4A-BEF2-49D8B4130A39";

///
/// The background task instance's activation parameters are available via
/// Windows.UI.WebUI.WebUIBackgroundTaskInstance.current
///
var backgroundTaskInstance = Windows.UI.WebUI.WebUIBackgroundTaskInstance.current;
var details = backgroundTaskInstance.triggerDetails;
settings.values[keyMessage] = details.message;
```

The following code demonstrates how to retrieve the message stored by the background task handler in the app.

```
var settings = Windows.Storage.ApplicationData.current.localSettings;
var keyMessage = "BA5857FA-DE2C-4A4A-BEF2-49D8B4130A39";
var operatorMessage = settings.values[keyMessage];
```

Step 3: Handle the Activation event
If the background task handler shows toast, the app can be launched by tap toast. If the toast sets a parameter, it will be passed to app through `detail.arguments`.

In a JavaScript or C# implementation, you handle the `WinJS.Application.activated` event, and then examine the event arguments that are passed to the event handler. Activation from toast passes the event argument of type `Windows.UI.WebUI.WebUILaunchActivatedEventArgs`. If the event argument’s `detail.kind` property is `Windows.ApplicationModel.Activation.ActivationKind.launch`, the app will provide the Start experience or the Notification experience, respectively, depending on whether the event argument’s `detail.argument` property is set to `null`.

The following code demonstrates how to attach the event handler and determine the app context using JavaScript.

```javascript
WinJS.Application.addEventListener("activated", activated; false);

function activated(eventArgs) {
  if (eventArgs.detail.kind ==
    if (eventArgs.detail.arguments) {
      // Initialize logic for the Start experience here.
    } else {
      // Initialize logic for the Notification experience here.
    }
  }
```
Step 4: Handle background task completion handlers

The foreground app can also register a completion handler to be notified when the background task completes. The completion status or any exception that occurs in the Run method of the background task is passed to the completion handler in the foreground app. If the app was suspended when the task completed, it will receive the completion notification next time the app is resumed. If the app was in the Terminated state, it does not receive the completion notification. If the background task needs to preserve the information that it ran successfully, it must persist the information using State Manager or another means, such as a file that the app can read when it comes back to the Running state.

Though the mobile operator background event is registered automatically by the system to the app, the app still needs to run at least once if it wants to register to the background completion or progress handlers.

The following code demonstrates how to attach the event handler to the background events. For more information on background task progress reporting and completion handlers, see Introduction to Background Tasks.

**Note:** In Windows 8 Consumer Preview, the event name may be blank. As a workaround, compare the task name to “”.

**C#**:

```csharp
foreach (var cur in BackgroundTaskRegistration.AllTasks)
{
    if (cur.Value.Name == "MobileOperatorNotificationHandler")
    {
        cur.Value.Progress += new BackgroundTaskProgressEventHandler(OnProgress);
        cur.Value.Completed += new BackgroundTaskCompletedEventHandler(OnCompleted);
    }
}

// Handle background task completion.
private void OnCompleted(IBackgroundTaskRegistration sender, BackgroundTaskCompletedEventArgs e)
{
    var taskCompletion = task as IBackgroundTaskRegistration;
    var completionArgs = args.Context as BackgroundTaskCompletedEventArgs;

    // If the background task threw an exception, display the exception in
    // the error text box.
    if (completionArgs.Status != null)
    {
        throw completionArgs.Status;
    }
}
```
// Handle background task progress.
//
private void OnProgress(IBackgroundTaskRegistration sender,
BackgroundTaskProgressEventArgs e)
{
    var taskRegistration = task as IBackgroundTaskRegistration;
    var progressArgs = args.Context as BackgroundTaskProgressEventArgs;
    // progressArgs.Progress has the progress percentage
}

JavaScript:

var iter =
Windows.ApplicationModel.Background.BackgroundTaskRegistration.allTasks.first();
var hascur = iter.hasCurrent;
while (hascur) {
    var cur = iter.current.value;
    if (cur.name === “MobileOperatorNotificationHandler”) {
        cur.addEventListener(“progress”, new ProgressHandler(cur).onProgress);
        cur.addEventListener(“completed”, new
CompleteHandler(cur).onCompleted);
    }
    hascur = iter.moveNext();
}

// Handle background task progress.
//
function ProgressHandler(task) {
    this.onProgress = function (args) {
        try {
            var progress = “Progress: “ + args.progress + “%”;
        } catch (ex) {
            displayError(ex);
        }
    }
}

// Handle background task completion.
//
function CompleteHandler(task) {
    this.onCompleted = function (args) {
        try {
            var key = task.taskId;
        } catch (ex) {
            displayError(ex);
        }
    }
}
Troubleshooting

Triggering metadata parsing to register background tasks
For end users, when the mobile broadband device is connected, Windows 8 automatically installs the Metro style device app for mobile broadband and associated service metadata, and registers background tasks defined in the Service metadata.

Developers can manually trigger Windows 8 to parse service metadata and register background tasks by pressing F5 key (or right-click and select Refresh) in the Devices and Printers window on the desktop. Background task registration through service metadata parsing succeeds only when app is deployed.

Verifying that background tasks are registered correctly
Developers can verify that the Device Setup Manager (DSM) has properly parsed the Service metadata by viewing the event logs under Application and Services Logs\Microsoft\Windows\DeviceSetupManager:

1. Open Event Viewer.
   a. On the Start menu, type “Event Viewer” and select settings in the Search pane.
   b. Click View Event Logs.
2. On the menu tabs, select View, and then Show Analytic and Debug Logs.
3. Browse to Applications and Services Logs\Microsoft\Windows\DeviceSetupManager.

Events of interest include Event ID 220 that indicates that DSM has successfully registered the background task for the MobileOperatorNotification event and Event ID 7900, which indicates any errors found with the metadata package. For more information about these events and other relevant events logged during device and service metadata installation, see “Device Metadata Package Installation Debug” in Device Metadata Package Pipeline.

Verifying that background tasks are being executed by the System Event Broker
Developers can verify that Windows is generating the MobileOperatorNotification event and that the app’s background task is being executed by the event broker by checking the Event Viewer. Logging for these events is off by default and can be enabled by:

1. Open Event Viewer.
   a. On the Start menu, type “Event Viewer” and select settings in the Search pane.
   b. Click View Event Logs.
2. On the menu tabs, select View, and then Show Analytic and Debug Logs.
3. Browse to Applications and Services Logs\Microsoft\Windows\BackgroundTaskInfrastructure.
4. Right-click the Diagnostic log and select Enable Log.
After enabling the logs, a successful execution of the background task will result in an event with an Event ID = 1, having the following description: “An instance of a background task with entry point <background_task_namespace_name>.<background_task_class_name> and name MobileOperatorNotificationHandler has been created in session 1 and given an ID of {11111111-1111-1111-1111-111111111111}.”

If the background task is not being executed, first verify that the names of your background tasks specified in the Service metadata match the names in the AppXManifest.xml file of your package. Then verify that parsing the service metadata has been triggered after deploying the app and inserting the mobile broadband device.

Verifying Windows is receiving SMS and USSD

Developers can verify Windows is receiving SMS and USSD notifications by checking for SmsRouter events in Event Viewer.

In Event Viewer under Application and Services Logs\Microsoft\Windows \Mobile-Broadband-Experience-SmsRouter\Microsoft-Windows-SMSRouter are entries such as “The SmsRouter received a SMS Operator Notification message” and “The SmsRouter received a Text message”. Under Application and Services Logs\Microsoft\Windows \Mobile-Broadband-Experience-SmsApi\SmsAppl are entries such as “App: Microsoft.SDKSamples.SmsSendReceive sent SMS text message on mobile broadband device: {11111111-1111-1111-1111-111111111111}”.

Received SMS messages are not detected as operator notifications

If received SMS are not being detected as operator notifications, verify the custom filtering rules for administrative SMS notifications in the account provisioning metadata. For more information, see Providing Mobile Broadband Metadata.

Appendix

Sample backgroundtask.js file:

```javascript
//
// A JavaScript background task runs a specified JavaScript file.
//
(function () {
  "use strict";

  //
  // The background task instance’s activation parameters are available
  // via Windows.UI.WebUI.WebUIBackgroundTaskInstance.current.
  //
  var backgroundTaskInstance = Windows.UI.WebUI.WebUIBackgroundTaskInstance.current;
  key = null,
  settings = Windows.Storage.ApplicationData.current.localSettings;

  try {
    var details = backgroundTaskInstance.triggerDetails;
```
switch (details.notificationType) {
    case networkOperatorEventType.gsm:
        var textMessage = new Windows.Devices.Sms.SmsTextMessage.FromBinaryMessage(details.smsMessage);
        showToast("Gsm Msg From: " + textMessage.from + ";
        TimeStamp: " + textMessage.timestamp, details.message);
        break;
    case networkOperatorEventType.cdma:
        showToast("Mobile Broadband message", details.message);
        break;
    case networkOperatorEventType.ussd:
        showToast("Mobile Broadband message", details.message);
        break;
    case networkOperatorEventType.dataPlanThresholdReached:
        showToast("Mobile Broadband message", "Data plan threshold reached");
        break;
    case networkOperatorEventType.dataPlanReset:
        showToast("Mobile Broadband message", "Data plan reset");
        break;
    case networkOperatorEventType.dataPlanDeleted:
        showToast("Mobile Broadband message", "Data plan deleted");
        break;
    case networkOperatorEventType.profileConnected:
        showToast("Mobile Broadband message", "Profile connected");
        break;
    case networkOperatorEventType.profileDisconnected:
        showToast("Mobile Broadband message", "Profile disconnected");
        break;
    case networkOperatorEventType.registeredRoaming:
        showToast("Mobile Broadband message", "Registered roaming");
        break;
    case networkOperatorEventType.registeredHome:
        showToast("Mobile Broadband message", "Registered home");
        break;
    default:
        showToast("Mobile Broadband message", "Unknown message");
        break;
}
} taskSucceeded();
}
catch (exception) {
    taskFailed();
}

function showToast(title, body) {
    var notifications = Windows.UI.Notifications;
    var toastNotificationManager = Windows.UI.Notifications.ToastNotificationManager;
    var toastXml = toastNotificationManager.getTemplateContent(notifications.ToastTemplateType.toastText02);
// Pass to app through eventArguments.arguments.
//
var temp = """"Title"""" + ":" + """" + title + """" + ";" + """"Message"""" + ":" + """" + body + """";
if (temp.length > 251) {
    temp = temp.substring(0, 251);
}
toastXml.selectSingleNode("/toast").setAttribute("launch", """" +
temp + ");"

var textNodes = toastXml.getElementsByTagName("text");
textNodes[0].appendChild(toastXml.createTextNode(title));
textNodes[1].appendChild(toastXml.createTextNode(body));

var toast = new notifications.ToastNotification(toastXml);
toastNotificationCenter.createToastNotifier().show(toast);

//
// This function is called when the background task is completed
// successfully.
//
function taskSucceeded() {
    //
    // Use the succeeded property to indicate that this background
    // task completed successfully.
    //
    backgroundTaskInstance.succeeded = true;
    backgroundTask.taskInstance.progress = 100;
    console.log(Background " + backgroundTask.taskInstance.task.name + " Completed");

    //
    // Write to localSettings to indicate that this background task
    completed.
    //
    key = backgroundTaskInstance.task.taskId.toString();
    settings.values[key] = Completed;

    //
    // A JavaScript background task must call close when it is done.
    //
    close();

    //
    // If the task was canceled or failed, stop the background task.
    //
    function taskFailed() {
        console.log(Background " + backgroundTask.taskInstance.task.name + " Failed";
        backgroundTaskInstance.succeeded = false;
        key = backgroundTaskInstance.task.taskId.toString();
        settings.values[key] = Failed;
        close();}
Resources

**Overview of Mobile Broadband in Windows 8**
http://go.microsoft.com/fwlink/?linkid=242052

**Preparing to Develop Mobile Operator Apps**
http://go.microsoft.com/fwlink/?linkid=242058

**Development Guide to Creating Mobile Operator Apps**
http://go.microsoft.com/fwlink/?linkid=242058

**Introduction to Background Tasks**
http://go.microsoft.com/fwlink/?LinkId=227329&clcid=0x409

**Designing User Experience of Mobile Operator Apps**
http://go.microsoft.com/fwlink/?linkid=242066

**Providing Mobile Broadband Metadata**
http://go.microsoft.com/fwlink/?linkid=242064

**Overview of Mobile Broadband Windows Runtime API**
http://go.microsoft.com/fwlink/?linkid=242060

**USSD APIs**

**Windows 8 Integration for Wireless Hotspot Operators**
http://go.microsoft.com/fwlink/?linkid=242068

**Device Metadata Package Pipeline**