Windows Phone 8 Location API

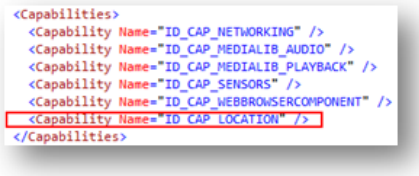
Windows Phone 8 SDK offers a set of new run-time location APIs for getting the current location of the phone.

Introduction

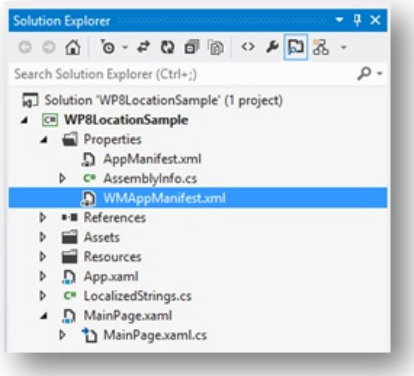
Windows Phone 8 SDK offers a set of new run-time location APIs for getting the current location of the phone. Another new feature that is available is the background location tracking, which enables apps to continue tracking location in the background even after the user exits the app.

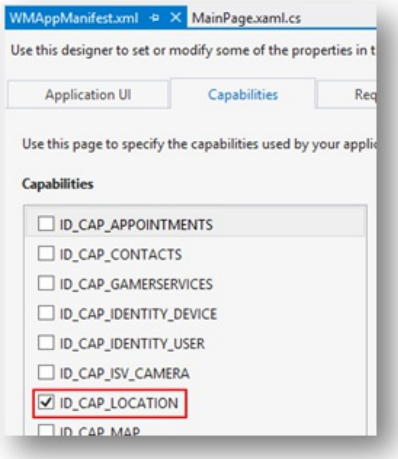
Before we begin

If you want to use the location APIs in your app, you will need to enable the location capability; otherwise your app may not work correctly or may fail certification. You can do this by editing the **WMAppManifest.xml** file and adding a Capability element for ID_CAP_LOCATION inside the Capabilities element:



Or alternatively you can use the visual designer: double click on **WMAppManifest.xml** and check the ID_CAP_LOCATION checkbox in the Capabilities tab as shown below:





Finally, before we can get to the code, we will add the following code in **MainPage.xaml**:

```
<StackPanel Orientation="Vertical">
    <Button x:Name="btnGetCurrentLocation" Content="Get current location" Click="btnGetCurrentLocation_Click" />
    <TextBlock Text="Current location:" />
    <TextBlock x:Name="txtLocation" />
    <CheckBox x:Name="cbEnableLocationTracking"
                  Content="Enable location tracking"
                  Checked="cbEnableLocationTracking_Checked"
                  Unchecked="cbEnableLocationTracking_Unchecked" />
    <TextBlock Text="Status:" />
    <TextBlock x:Name="txtStatus" />
</StackPanel>
```

New API for getting the current location

Windows Phone 8 includes new location APIs designed for apps that do not need to track the phone's location continuously but rather only rarely need to retrieve the current location. In fact, if your app does not need to continuously track location, using the API demonstrated in the following code sample is recommended as it saves battery and therefore improves user experience.

The new location APIs are exposed through the `Geolocator` class. In order to get the phone's current location we simply call the `GetGeopositionAsync` method. As demonstrated in the code below, you can also set the desired accuracy and maximum age of the returned coordinates (in meters), as well as the timeout (how much you are willing to wait). Note that this is an asynchronous method which returns `IAsyncOperation<Geoposition>` rather than just a `Geoposition` object, but calling it is easy using the new `async` and `await` keywords. For now, it is enough to know, that all code after an `await` statement will execute only after the asynchronous call has finished. In our case this is the line where we format the coordinates using the `GetCoordinateString`

method and display the result in the `txtLocation` text block.

Note: Include the following namespace: `using Windows.Devices.Geolocation;`

```
private async void btnGetCurrentLocation_Click(object sender, RoutedEventArgs e)
{
    this.btnGetCurrentLocation.IsEnabled = false;
    Geolocator geolocator = new Geolocator();
    geolocator.DesiredAccuracyInMeters = 50;
    try
    {
           Geoposition position = await geolocator.GetGeopositionAsync(
          maximumAge: TimeSpan.FromMinutes(1), timeout: TimeSpan.FromSeconds(30));

          this.txtLocation.Text = this.GetCoordinateString(position.Coordinate);
    }
    catch (UnauthorizedAccessException)
    {
        MessageBox.Show("Location is disabled in phone settings.");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
    finally
    {
        this.btnGetCurrentLocation.IsEnabled = true;
    }
}
```

```
private string GetCoordinateString(Geocoordinate coordinate)
{
    string positionString = string.Format("Lat: {0:0.0000}, Long: {1:0.0000}, Acc: {2}m",
                coordinate.Latitude, coordinate.Longitude, coordinate.Accuracy);
    return positionString;
}
```

New APIs for location tracking

Note: You should use location tracking only if it is necessary for your app to work correctly. Have in mind that this could cut down the battery life of the phone. So, if your app requires only the current location then is it recommended that you use the `GetGeopositionAsync` method instead.

The `Geolocator` class also exposes new APIs for continuous location tracking. In order to do continuous location tracking in your app, you just need to subscribe to the `PositionChanged` event of an instance of the `Geolocator` class. Similar to retrieving one-off location data, you can control settings like the desired accuracy and movement threshold as demonstrated in the following code snippet.

```
private void cbEnableLocationTracking_Checked(object sender, RoutedEventArgs e)
{
    if (App.Geolocator == null)
    {
        App.Geolocator = new Geolocator();
        App.Geolocator.DesiredAccuracy = PositionAccuracy.High;
        App.Geolocator.MovementThreshold = 100; // 100 meters
        App.Geolocator.StatusChanged += geolocator_StatusChanged;
        App.Geolocator.PositionChanged += geolocator_PositionChanged;
    }
}
```

The `geolocator_PositionChanged` event handler will be called whenever the change in the phone's location is greater than the value set to the `MovementThreshold` property. In the handler of the `PositionChanged` event, we have access the phone's current location through the event arguments as demonstrated in the next code snippet. It is important to check if the app is tracking location in the background, and do not do any UI updates if it is. Also, if the app is running in the foreground, then we must use the `Dispatcher` to update the UI since the `PositionChanged` event handler is not executed on the main thread.

```
void geolocator_PositionChanged(Geolocator sender, PositionChangedEventArgs args)
{
    string positionString = this.GetCoordinateString(args.Position.Coordinate);
    if (!App.RunningInBackground)
    {
        // IMPORTANT: the event handler is called on a worker thread,
        // so we must use the Dispatcher to update the UI from the main thread
        Dispatcher.BeginInvoke(() =>
        {
            this.txtLocation.Text = positionString;
        });
    }
    else
    {
        Microsoft.Phone.Shell.ShellToast toast = new Microsoft.Phone.Shell.ShellToast();
        toast.Content = positionString;
        toast.Title = "Location: ";
        toast.NavigationUri = new Uri("/MainPage.xaml", UriKind.Relative);
        toast.Show();
    }
}
```

When implementing location tracking in your app, you can also subscribe to the `StatusChanged` event of the **Geolocator** to be notified of changes in the status of location tracking. You can then access the current status from the arguments of the `StatusChanged` event handler as demonstrated in the following code snippet. This event handler is again not executed on the main thread so it is important to use the `Dispatcher` if you want to update the UI.

```
geolocator_StatusChanged(Geolocator sender, StatusChangedEventArgs args)
{
    if (App.RunningInBackground)
```

```
    {
            return;
    }
    string status = "";

    switch (args.Status)
    {
        case PositionStatus.Disabled:
        // the application does not have the right capability or the location master switch is off
        status = "location is disabled in phone settings";
        break;

        case PositionStatus.Initializing:
        // the geolocator started the tracking operation
        status = "initializing";
        break;

        case PositionStatus.NoData:
        // the location service was not able to acquire the location
        status = "no data";
        break;

        case PositionStatus.Ready:
        // the location service is generating geopositions as specified by the tracking parameters
        status = "ready";
        break;

        case PositionStatus.NotAvailable:
        status = "not available";
        // not used in WindowsPhone, Windows desktop uses this value to signal that there is no hardware capable to acquire location
        break;

        case PositionStatus.NotInitialized:
        // the initial state of the geolocator, once the tracking operation is stopped by the user the geolocator moves back to this
        status = "not initialized";
        break;
    }
    // IMPORTANT: the event handler is called on a worker thread,
    // so we must use the Dispatcher to update the UI from the main thread
    Dispatcher.BeginInvoke(() =>
    {
        this.txtStatus.Text = status;
    });
}
```

Implementing background location tracking

To implement background location tracking in your app, you must first edit the **WMAppManifest.xml** file and change the `DefaultTask` element to look like this:

```
<DefaultTask Name="_default"
NavigatioPage="MainPage.xaml">
    <BackgroundExecution>
        <ExecutionType Name="LocationTracking" />
    </BackgroundExecution>
</DefaultTask>
```

Then in **App.xaml**, add an event handler for the `RunningInBackground` event as in the following code snippet:

```
<shell:PhoneApplicationService
            Launching="Application_Launching" Closing="Application_Closing"
            Activated="Application_Activated" Deactivated="Application_Deactivated"
            RunningInBackground="PhoneApplicationService_RunningInBackground" />
```

Next in **App.xaml.cs** add the following properties and the event handler for the `RunningInBackground` event:

```
public static Geolocator Geolocator { get; set; }
public static bool RunningInBackground { get; set; }

private void PhoneApplicationService_RunningInBackground(object sender, RunningInBackgroundEventArgs e)
{
    RunningInBackground = true;
}
```

Finally, in the Activated event handler in **App.xaml.cs** set the `RunningInBackground` property to false:

```
private void Application_Activated(object sender, ActivatedEventArgs e)
{
    RunningInBackground = false;
}
```

You should check the value of the `RunningInBackground` property in your Geolocator event handlers and make sure that you are not updating the UI when your app is tracking the phone's location in the background. It is recommended that you unsubscribe from Geolocator events when your app is removed from the navigation journal as demonstrated in the following code snippet:

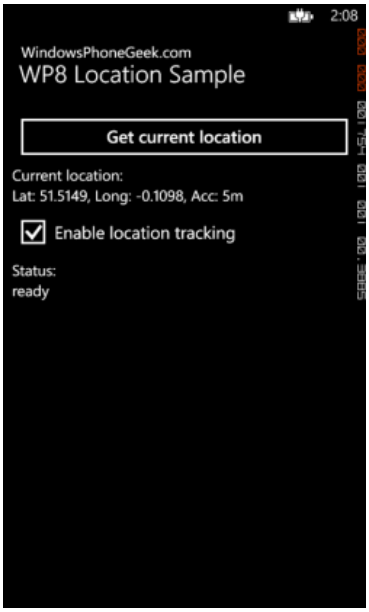
```
protected override void OnRemovedFromJournal(JournalEntryRemovedEventArgs e)
{
    this.StopLocationTracking();
}

private void StopLocationTracking()
{
    App.Geolocator.PositionChanged -= geolocator_PositionChanged;
    App.Geolocator.StatusChanged -= geolocator_StatusChanged;
    App.Geolocator = null;
}
```

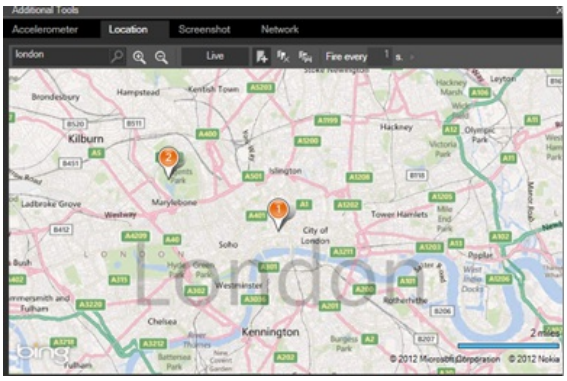
Finally, when your app is re-launched it is recommended that, in order to provide consistent experience for the user, you clear the navigation history. If

you have created your app using the new Windows Phone 8 project templates, your project already includes code that does this and you do not need to add such code manually. However, if you are upgrading an existing app, you might need to implement this in the `RootFrame.Navigated` event handler in **App.xaml.cs**, as shown in the following code snippet.

```
void RootFrame_Navigated(object sender, NavigationEventArgs e)
{
    if (e.NavigationMode == NavigationMode.Reset)
    {
        RootFrame.Navigated += ClearBackStackAfterReset;
    }
}

void ClearBackStackAfterReset(object sender, NavigationEventArgs e)
{
    RootFrame.Navigated -= ClearBackStackAfterReset;

    if (e.NavigationMode != NavigationMode.New && e.NavigationMode != NavigationMode.Refresh)
    {
        return;
    }

    while (RootFrame.RemoveBackEntry() != null) ;
}
```

Here is how our sample app looks. Note that on the second screenshot, the location is shown using toast notification when the app is tracking location in the background.





Conclusion

Windows Phone 8 exposes new location APIs that allow developers to use the hardware more easily and more efficiently. This translates in increased battery life and improved user experience. Windows Phone 8 also allows developers to implement background location tracking which enables scenarios like navigation apps and more.