Jogging - Calorie Calculator: Demonstrating Real World usecase of WP8 features

This article needs to be updated: If you found this article useful, please fix the problems below then delete the {{ArticleNeedsUpdate}} template from the article to remove this warning.

Reasons: hamishwillee
(03 May 2013)
The article has reported errors. There is no working zip file which makes it hard to verify the code, and it is not documented sufficiently to understand the code without such a zip file. IMO this therefore does not meet wiki quality standards. If a buildable project cannot be provided it should be deleted.

This article aims to showcase the strengths of various new WP8 Features and how we can use those for adding value to the Real World.

Introduction

People are becoming more and more aware about the need for taking care of their health due to the improper and stressful modern lifestyle. And Technology can help them in achieving their goal in easy and systematic manner.And as a result of which we have seen a lot of health related mobile apps booming in mobile market.
In this tutorial we will see how easily we can implement an application using new WP8 features that can help people stay fit .

Application Description & Features

Jogging(brisk walking/running) is one of the best exercise to maintain overall fitness. And equally important is to track our progress during our daily exercise, else we will not be able to realize if we are making a right progress and taking required efforts. User will start the application when he/she starts jogging.

Application Features:
These are the features of application for which the implementation methodologies are covered in this article.

- **Track Exercise Results** - This is done by calculating distance, time, speed and calorie burned during the jogging session. This will help user understand if he/she has achieved the daily goal.
- **Voice Notification for every kilometer completed** - User will be notified about this through voice functionality so that user is aware how much more he wants to run.
- **Live Jogging Track Display** - Using map control the app will draw the route he/she has taken during the exercise and the route will be updated regularly to give a real time track display.
- **Background Processing** - User can navigate away from app, say during a call or navigate to other application, and still the application will be able to monitor the exercise.
- **Optional Music Feature** - User can play music from within the app, this feature is provided due to the fact that many people like to listen to music while jogging for fun or for motivation (fast music).

Future Enhancements:
Following are some possible future enhancements which are explained briefly in the ending section, but currently out of scope of this article.

- **Setting Targets**
- **Share results with NFC**
- **Heart Rate Monitor Integration & Safety Alerts**

Project Setup

WP8 Features that will be utilized

The newly introduced WP8 features make the implementation of this app very easy. We will be using following features for the app development:

- Location API
- Background Location Tracking Feature
- Draw Map Route on Map control
- Voice API
- Music

Capabilities

In your project, expand the Properties and double click **WMAppManifest.xml**. Choose the 'Capabilities' tab and check the following items:

```
ID_CAP_LOCATION
ID_CAP_MAP
ID_CAP_SPEECH_RECOGNITION
ID_CAP_MEDIALIB_PLAYBACK
```


(c) Copyright Nokia 2013. All rights reserved.

Enable Background Processing

One of the **key features of WP8** is **Background processing** for apps using Location API. We will see the details of its implementation aspects in later part of this article, but for now the below steps are required to make the app 'background processing enabled'.

In WMAppManifest.xml, overwrite **DefaultTasks** :

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

In **App.xaml**, overwrite the shell:PhoneApplicationService element:

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

In **App.xaml**, add the RunningInBackground event handler and create a global static variable called RunningInBackground, that will help us track if the application is in foreground or background and do the processing accordingly :

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

Project Structure

Now that we have seen the goal of the application, its functionality specs and WP8 feature usage in this application, its time to deep dive into the implementation section. Project consists of 3 pages:

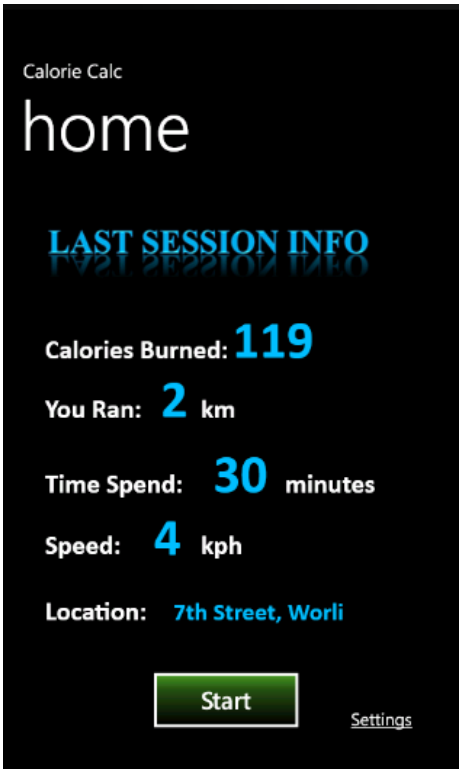
- MainPage.xaml
- SettingsPage.xaml
- StartExercisePage.xaml

The tutorial will go page wise and explain what functionalities are performed in each page respectively and how those functionalities can be implemented. The focus is on how the WP8 specific features needs to be implemented to achieve the app functionalities, so the trivial and non-wp8 specific code is not touched in this tutorial.

MainPage

This page is the app start page. And will provide quick information of last exercise session, options to navigate to settings and navigation to start the exercise function.

(c) Copyright Nokia 2013. All rights reserved.



SettingsPage

User needs to enter following inputs for one time:

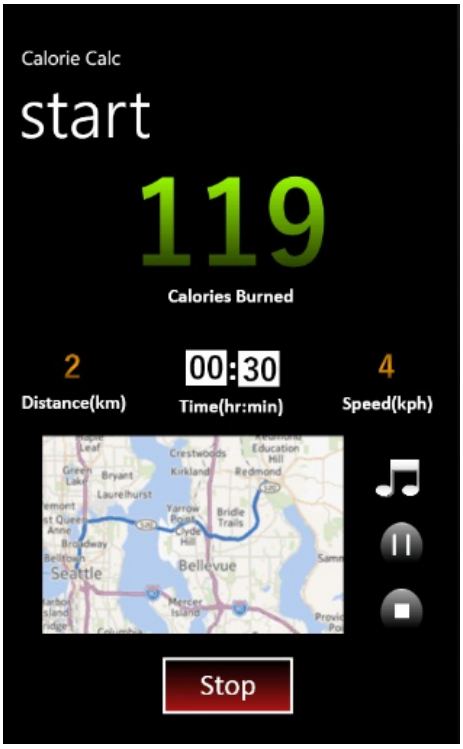
- Name (for personalized messages or status)
- Weight (needed for calorie calculation)

```
private void btnSaveSettings_Click(object sender, RoutedEventArgs e)
{
     private IsolatedStorageSettings userdata =
                              IsolatedStorageSettings.ApplicationSettings;

     // Save user configuration
     if(userdata.contains("name"))
      userdata["name"] =  tbName.Text;
    else
 userdata.Add("name", tbName.Text);

     if(userdata.contains("weight"))
      userdata["weight"] =  tbWeight.Text;
    else
 userdata.Add("weight ", tbWeight.Text);

    MessageBox.Show("Settings Saved.");
}
```

StartExercisePage

This is the place where the core functionality of application is situated. The moment user clicks on START button calculation engine will start.

(c) Copyright Nokia 2013. All rights reserved.



Tip: User Interface for such applications should be designed very aesthetically. The reason being that, some users might be lazy to use the application daily. And to make people use your application often it should be really attractive enough to make them feel like using it again and again.

Some Important Variable Declarations

There are some class level variables which we will refer throughout the StartExercisePage.

```
Geolocator geolocator;
int distanceCounter = 0;
DateTime startTime;
double timeElapsed;
int thresholdVal = 5;
Queue<GeoCoordinate> toDrawCoordinates;
RouteQuery routeQuery = null;
GeocodeQuery geocodequery = null;
IsolatedStorageSettings userdata = IsolatedStorageSettings.ApplicationSettings;
```

The Map control will be used to show the Route that was traveled by user during his exercise session.
The MediaElement control will provide an in-app option to play some music. Make sure LoadedBehavior is set to Manual to allow control by user for Play, Pause, and Stop functions. In WP8, only one mediaElement control can play music at a time.

```
<maps:Map x:Name="displayMap" ZoomLevel="11"/>
<MediaElement Source="music\exerciseMusic.wmv" Name="mediaControl"
                                        LoadedBehavior="Manual" />
```

Get user consent

An app using location API need to obtain a user consent before making any use of it. In this code section if app is running for the first time we will ask user if he/she is fine for letting the app use Location Service. If user select yes then we continue with next steps else exit. Irrespective of the decision taken by user, that decision is saved in the application cache.
If user selected allow the usage, then whenever the app runs it don't need user approval. But if user had chosen disallow, then on the next app run, user will be prompted again for permission.

```
public bool getConsentStatus()
{
    if (IsolatedStorageSettings.ApplicationSettings.Contains("LocationConsent"))
    {
        // User has already made a decision, just retrieve that decision.
        if(Convert.ToBoolean(IsolatedStorageSettings.
                                ApplicationSettings["LocationConsent"]))
         {
               return Convert.ToBoolean(IsolatedStorageSettings.
                                ApplicationSettings["LocationConsent"]);
         }
         else
        {
            MessageBoxResult result =
            MessageBox.Show("You had chosen not to allow your phone's location.
                                  Do you want to allow now?", "Location",
                                  MessageBoxButton.OKCancel);

         if (result == MessageBoxResult.OK)
         {
             IsolatedStorageSettings.ApplicationSettings["LocationConsent"] = true;
         }else
        {
             IsolatedStorageSettings.ApplicationSettings["LocationConsent"] = false;
```


(c) Copyright Nokia 2013. All rights reserved.

```
        }

        IsolatedStorageSettings.ApplicationSettings.Save();
   return result == MessageBoxResult.OK;
     }
    }
    else
    {
        MessageBoxResult result =
            MessageBox.Show("This app accesses your phone's location. Is that ok?",
            "Location",
            MessageBoxButton.OKCancel);

        if (result == MessageBoxResult.OK)
        {
            IsolatedStorageSettings.ApplicationSettings["LocationConsent"] = true;
        }else
        {
            IsolatedStorageSettings.ApplicationSettings["LocationConsent"] = false;
        }

        IsolatedStorageSettings.ApplicationSettings.Save();
  return result == MessageBoxResult.OK;
     }
}
```

Initialize Calculation Engine

Once we get the approval from user or from cache. We now start with the core calculation engine process.

- **DesiredAccuracy** - This property is set to High because we are going require as much accuracy as possible to get the precise distance covered by the user and also we will be displaying the route taken by user on the map.**There is a consideration that must be taken into account that is power usage v/s accuracy needed.** The higher the accuracy the more power will be used, but our application needs accuracy and hence we have kept the property as High.
- MovementThreshold - Value(in meters) of this property triggers an event who's handler does some processing that is required on completion of the event. When the phone moves more than the movement threshold from the previous location this event will be triggered. We could have put a minimum possible value and that is 1 for the sake of accuracy. But instead we kept it 5 because, we want the UI to be updated at reasonable interval and in practical scenario a user wont be checking the app status every second, so its not a bad idea to delay the UI update after a few seconds instead of every second.
- PositionChanged - This is the event that is triggered once the MovementThreshold is reached every time. We will update the user exercise stats in the geolocator_PositionChanged event handler. We will look at calculation processing logic a bit later.
- Store the previousLocation to display in the HomePage.

```
private void btnStart_Click(object sender, RoutedEventArgs e)
{
    //Check user consent else exit.
    if(getConsentStatus() == true)
    {
  geolocator = new Geolocator();
        geolocator.DesiredAccuracy = PositionAccuracy.High;
        geolocator.MovementThreshold = thresholdVal;

        geolocator.PositionChanged += geolocator_PositionChanged;

         //If all the previous functionality works fine then only enable stop button.
         btnStop.IsEnabled = true;

        //Store the location origin coordinates into cache for Status History in Home
        //Page.
        userdata["previousLocation"] =  geolocator.Coordinate.Latitude.ToString("0.00")
        + ":" + geolocator.Coordinate.Longitude.ToString("0.00");
  }
  else
  {
  MessageBox.Show("You have chosen not to use Location Service”);
  }
}
```

Core Engine Processing

Every time the threshold is passed the geolocator_PositionChanged will be triggered.

First perform all the below non-UI processing:

- **Update Distance Covered** : Add the thresholdVal which we have set to 5 meters. So for instance when the user starts jogging distanceCounter =0, and when covers his first 5 meters this module will add 5 to distanceCounter to calculate the distance covered by him. Distance will be in meters.
- **Update Time Elapsed**: timeElapsed variable will calculate the time elapsed since start of session. Time will be in minutes.
- **Calorie Calculation**: Call the calorie calculation module and update the current calorie burned during jogging.
- **Voice Notification**: If user completed another kilometer then notify the user through voice notification. If the music is playing then pause it first and then notify. After notification is completed resume the music.

Now comes an interesting and critical part of the app processing - location tracking in background and UI update. This is a new and very useful WP8 feature.

As our application is Background Enabled, it might be in background if user has navigated away from our app.
If the application is in Foreground, only then perform UI updates like: Distance, Time, Calorie Burned and Map Route. PositionChanged event handler cannot directly update UI elements, and hence we use BeginInvoke(Action) for updating UI controls.
If the application is in Background then do not perform any UI processing as it is of no use. Instead show a toast notification for every kilometer complete.

And there can be critical impacts of this kind of considerations. For example, our real time draw route gets impacted by it. In regular scenario if the app was always in foreground then we would have kept updating UI control - map for every coordinate changes tracked through PositionChanged event handler. But now that our application can be in background and when the app is in background we need to avoid the UI changes, the solution to this problem is that we kept storing the coordinated in a Queue data-structure - toDrawCoordinates. We keep adding map coordinates to toDrawCoordinates,

(c) Copyright Nokia 2013. All rights reserved.

so that when user resumes back to our application, route can be drawn on the Map for those coordinates that were not drawn on map controlled due to the fact that application was in background.

```
void geolocator_PositionChanged(Geolocator sender, PositionChangedEventArgs args)
{
    double currentCal=0;
    bool notifyFlag = false;
    List<GeoCoordinate> drawCoordinates;

    distanceCounter = distanceCounter + thresholdVal;
    timeElapsed = DateTime.Now.Subtract(startTime).TotalMinutes;
    currentCal = updateCalorieCalc(distanceCounter, timeElapsed, userdata["weight"]);

    if(distanceCounter % 1000 ==  0)
    {
         notifyFlag = true;
    }

    GeoCoordinate currentCoordinate = geolocator.Coordinate;
    toDrawCoordinates.Enqueue(currentCoordinate);

     //Give a voice notification to user that one more km is covered.
     if(notifyFlag)
     {
         // Pauses media if it is currently running.
         bool flag =  mediaControl.Playing;
         if(flag)
               mediaControl.Pause();

         //Voice Notify
         SpeakText("Jogging Notification, You have covered " + distanceCounter / 1000 +
                                                    " Kilometer distance");

         //Restart Music if it was paused
         if(flag)
               mediaControl.Play();
     }

     if (!App.RunningInBackground)
     {
          //If App is in foreground only then update the UI.
          Dispatcher.BeginInvoke(() =>
          {
                txtDistanceCovered.text = distanceCounter;
                txtCalorie = currentCal;

                //Call the drawRoute Function only if Queue has more than 2 coordinates,
                //This will avoid calling drawRoute on the first run of the event handler
                if(toDrawCoordinates.Count > 2)
                {
                      drawRoute(drawCoordinates);

                      //Clear all coordinates and add the currentCoordinate which will
                      //act as start coordinate in next route mapping.
                      toDrawCoordinates.Clear();
                      toDrawCoordinates.Enqueue(currentCoordinate);
                  }
            });
     }
     else
     {
            // If the app is running in background and then notify the user with a Toast
            //for every Km completed.
            if(notifyFlag)
            {
               Microsoft.Phone.Shell.ShellToast t =
                                                 new Microsoft.Phone.Shell.ShellToast();
               t.Title = "Jogging Notification: ";
               t.Content = "Distance Covered: "+ distanceCounter / 1000 + "Km";
               t.NavigationUri = new Uri("/StartExercisePage.xaml", UriKind.Relative);
               t.Show();
             }
      }

      // Reset the flag
      if(notifyFlag)
              notifyFlag = false;
}
```

This is a generic function to convert Text to Voice Notification.

```
public async void SpeakText(string Text)
{
    SpeechSynthesizer ss = new SpeechSynthesizer();
    ss.SetVoice(InstalledVoices.Default);
    await ss.SpeakTextAsync(Text);
}
```

Calorie Calculation Technique

There are various techniques to calculate calories burned during a workout. Some of which can predict the same using just distance and weight of user. But this will not be much precise as the calculation will ignore the activity level and speed of workout which will affect how much calories have burned.

In the below technique we will consider following data points for calorie calculation:

- Distance (meters)
- Time (minutes)
- Speed, derived from above two factors (meters per minute)
- Activity level (as we are aware that activity is walk/running)
- Weight (Kg)

The most optimal calculation will require additional input i.e. Average Heart Rate of the user during the exercise period, but in spite of the absence of this data point, we can be able to predict a decently precise Calorie calculation using above technique.

(c) Copyright Nokia 2013. All rights reserved.

Also in the later section I have covered additional Safety usage of Heart Rate Monitor integration with this application, but for now below are the steps for calorie calculation:

1. **Calculate Speed**: Use the distance and time elapsed data points.
2. **Temporary Result**: If speed <= 99.16 mpm (meters per minute) then use the following equation: (0.1 x speed) + (1.8 x speed x activity level) + 3.5 else (0.2 x speed) + (0.9 x speed x activity level) + 3.5
3. **Calculate CPM [Calories Per Minute]**: (Temporary Result * weight) / 200
4. **Calculate Total Calories**: CPM * time elapsed

```
public double updateCalorieCalc(int distance,int timeM, double weight)
{
     double speed = distance/timeM;
     double calExpPerMin, totalCalExp;
     double temp;

     if(speed <= 99.16)
     {
          temp = (0.1 * speed) + (1.8 * speed * 0.01) + 3.5;
     }
     else
     {
          temp = (0.2 * speed) + (0.9 * speed * 0.01) + 3.5;
     }
     calExpPerMin = (temp*weight) / 200;
     totalCalExp = calExpPerMin * timeM;

     return totalCalExp;
}
```

Draw a continuous route traveled by user during jogging

This function will keep drawing the route on map control till user don't stop exercise. To draw the route it will use the list of coordinates stored in drawCoordinates, which we have collected in the geolocator_PositionChanged event handler.

```
public void drawRoute(List<GeoCoordinate> drawCoordinates)
{
      routeQuery = new RouteQuery();
      routeQuery.Waypoints = toDrawCoordinates;
      routeQuery.QueryCompleted += routeQuery _QueryCompleted;
      routeQuery.QueryAsync();
      geocodequery.Dispose();
}

void routeQuery _QueryCompleted(object sender, QueryCompletedEventArgs<Route> e)
{
   if (e.Error == null)
   {
      Route drawRoute = e.Result;
      MapRoute mapRoute = new MapRoute(drawRoute);
      displayMap.AddRoute(mapRoute);
      routeQuery.Dispose();
   }
}
```

Stop Current Exercise Session

The above calculation engine will continue to calculate till user decides to stop the tracking by pressing the STOP button. On click of this button we will remove the geolocator_PositionChanged event handler and thus location tracking and calculation engine will stop. Then we will stop the music as well.

```
private void btnStop_Click(object sender, RoutedEventArgs e)
{
 geolocator.StatusChanged -= geolocator_PositionChanged;
        geolocator = null;
        mediaControl.Stop();

        //To-do: Save the current data to Isolated Storage so that it can be shown
        //on next run on MainPage
}
```

Future Enhancements

The core functionality and some additional feature are seen above. But there can be some interesting enhancements that can make this app even more useful and creative. Let us have a look at those possible enhancements.

Setting Targets

Option to set the a Target before starting the app, like today you want to reduce calories by 500 and the app will notify you through voice once you reach your goal.

Share results with NFC

People might be jogging with their friends, so not a bad idea to share each others results just by tapping. Healthy competition can motivate them to put more efforts for achieving their fitness goals.

Heart Rate Monitor Integration & Safety Alerts

(c) Copyright Nokia 2013. All rights reserved.

Every person has a Resting Heart Rate(RHR) i.e. the heart rate under normal condition. During exercise our heart rate increases, but this increase should not exceed a certain limit which is called as maximum heart rate(MHR). If one exceeds the max limit that can be dangerous and can cause **cardiovascular problems**.
One of various ways to calculate the approx. MHR is = 220 - your age.
Heart Rate Monitor Integration can help us detect if the heart rate of user is below safe levels (offcourse approximately) and the app can immediately give a Safety Alert through voice, display and vibration. This will be a very valuable add on to the application.

Apart from Safety Alert, this feature can also be used to calculate the Average Heart Rate during each session and those with serious goals like sportsperson and athletes can make use of this readings to judge their progress.

WP8 offers Bluetooth device connectivity and the current stack is 3.1, but WP8 is listed as 4.0 compatible. So when WP8 gets a stack upgrade to 4.0 and BLE(Bluetooth Low Energy) support we would be able to connect to various health devices like Heart Rate Monitor, ECG Monitor and so on.
The reason I choose the Heart Rate monitor integration is because its very small and easy for carrying along for a jogging session and can help us determine user safety using above information which is very easy to calculate and **pro-actively** give **Safety Alerts**.

Summary

The article intended to make you familiar with some features of WP8 like Location, Background processing, Maps and Music. The code in the tutorial will help you getting started but as you can easily notice there are things like Exception handling, best practices for background apps,etc. that you will need to take care of during development.
We certainly know the capabilities of WP8 and how effectively we can use it to solve the real world problems and to improvise our living. So put your thinking caps on and get started !!!

References:
http://msdn.microsoft.com/en-us/library/windowsphone/develop/system.windows.controls.mediaelement
http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj247548
http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662935%28v=vs.105%29.aspx
http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj244363%28v=vs.105%29.aspx
http://www.ncbi.nlm.nih.gov/pubmed/17468581

(c) Copyright Nokia 2013. All rights reserved.

(c) Copyright Nokia 2013. All rights reserved.