Path Prediction Service  
Programming Guide

Version 2.1 – September 13, 2012

Abstract

The Project Hawaii Path Prediction Service enables a mobile application to predict a user’s destination based on current route data. The Path Prediction Service is provided as part of the Project Hawaii SDK.

This document provides an introduction to the Path Prediction Service and walks you through a simple Windows Phone application that uses the service.

Contents

[Introduction 2](#_Toc335306903)

[Prerequisites 2](#_Toc335306904)

[Overview of the Path Prediction Service 2](#_Toc335306905)

[Path Prediction Client Library 2](#_Toc335306906)

[PathPredictionService Class 3](#_Toc335306907)

[Walkthrough: Path Prediction Sample 3](#_Toc335306908)

[Contents of the Sample 5](#_Toc335306909)

[Initialization 5](#_Toc335306910)

[Path Prediction 5](#_Toc335306911)

[Using the Path Prediction Service in an Application 9](#_Toc335306912)

[Add Required Assemblies 9](#_Toc335306913)

[Reference the Namespace 9](#_Toc335306914)

[Set Up Your Credentials and Bing Maps Key 10](#_Toc335306915)

[Resources 10](#_Toc335306916)

**Disclaimer:** This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet Web site references, may change without notice. You bear the risk of using it.

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.

© 2012 Microsoft Corporation. All rights reserved.

Microsoft, Visual Studio, Silverlight, Windows Azure, and Windows are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

# Introduction

The Project Hawaii Path Prediction Service enables a mobile application to predict a user’s destination based on current route data. The Path Prediction Service is provided as part of the Project Hawaii SDK.

This document provides an introduction to the Path Prediction Service and walks you through a simple application that uses the service.

# Prerequisites

Before you can build an application that uses the Path Prediction service, you must:

* Install the Project Hawaii SDK.
* Build the Project Hawaii SDK.
* Obtain Project Hawaii authentication credentials.

For information about installation, build procedures, and credentials, see “Hawaii Installation Guide,” which is installed with the SDK and is available on the web, as listed in “Resources” at the end of this document.

In addition, you should be familiar with the following:

* Windows Communication Foundation (WCF)
* Microsoft Silverlight®
* Windows Phone 7 SDK

# Overview of the Path Prediction Service

The Project Hawaii Path Prediction service provides an interface that enables a mobile application to predict possible destinations based on current route data. The Path Prediction service is based on work previously done at Microsoft Research by John Krumm and others. For a link to relevant research, see “Resources” later in this paper.

## Path Prediction Client Library

Applications access the service through the Path Prediction Client library. The source code for this library is installed with the Project Hawaii SDK in the following folder:

* Source\ServiceClients\PathPrediction

The library defines the **Microsoft.Hawaii.PathPrediction.Client** namespace, which includes the following classes and structures:

Classes

|  |  |
| --- | --- |
| Class | Description |
| **PathPredictionService** | Helper class to request service from the Path Prediction service. |
| **PredictLocationRequest** | Describes the parameters for a path prediction request. |
| **PredictLocationResult** | Describes the result of a path prediction request. |

Structures

|  |  |
| --- | --- |
| Structure | Description |
| **LatLong** | Represents a latitude and longitude. |
| **PossibleDestination** | Represents a destination; contains a **LatLong**, a **Probability** that the location is the destination of the current path |

## PathPredictionService Class

The **PathPredictionService** class defines the **PredictLocationAsync** method, whichreturns predicted destinations given a set of points along a route. **PredictLocationAsync** returns the latitude and longitude of a caller-specified number of possible destinations, along with a probability for each possible destination. The probability value is the service’s predicted probability that the supplied point is the actual final destination.

# Walkthrough: Path Prediction Sample

The Path Prediction sample is installed with the Project Hawaii SDK. The sample implements a simple interface that predicts the user’s destination based on current location data. This brief walkthrough introduces the components of the sample and shows you how the sample uses the Path Prediction service to determine the possible destination.

To use the Path Prediction sample, you must have a Bing Maps Account and a Bing Maps Key.

To create a Bing Maps Account

* If you do not already have a Bing Maps Account, follow the instructions at “Creating a Bing Maps Account”:   
  <http://msdn.microsoft.com/en-us/library/gg650598.aspx>

To create a Bing Maps Key

* Follow the instructions at “Getting a Bing Maps Key”:  
  <http://msdn.microsoft.com/en-us/library/ff428642.aspx>

To compile and run the PathPrediction sample

1. In Visual Studio, open PathPredictionSampleApp.csproj.

2. Open the MainPage.xaml file.

3. Navigate to the <UserControl.Resources> section of the file and insert your Bing Maps Key as the value of the ApplicationID:

<UserControl.Resources>

<m:ApplicationIdCredentialsProvider x:Key="MyCredentials"   
 ApplicationId="Enter your AppID Here" />

</UserControl.Resources>

4. Save the MainPage.xaml file.

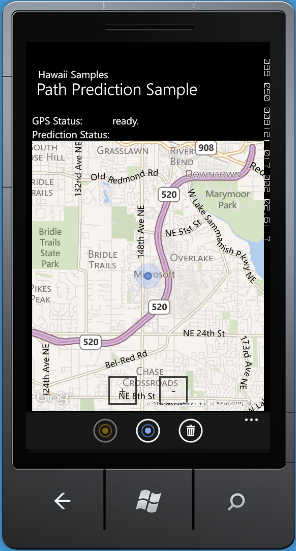
5. Open the HawaiiClient.cs file and set the **AdmClientId** and **AdmClientSecret** strings to your ADM credentials.

6. Save the HawaiiClient.cs file.

7. Build the solution.

8. Run the sample with or without the debugger, as you prefer.

The following figure shows the initial Windows Phone emulator window for the sample:



To use the Path Prediction sample

As you move, the application updates your current location on the screen. The blue circle shows your current position. By default, the starting position is set to Building 99 at Microsoft.

* Tap the gold circle to see the predicted destination. This circle is enabled only when the device is moving.
* Flick across the screen to change the view.
* Tap the **+** and **–** icons to zoom in and out on the map.
* Tap the trash can icon to delete the current route predictions.

Several sample routes are installed in the Routes subfolder of the PathPredictionSample directory. These routes contain a set of latitudes and longitudes that can be input to the Windows Phone Emulator as GPS data. You can use this data or use these XML files as models for creating your own route files.

To use a route with the sample in the Windows Phone emulator

1. Click the Other Tools icon (**>>**) on the emulator.

2. Select the **Location** tab.

3. In the **Recorded Data** text box, type the path to the route.

—OR—

Click **Load…**, navigate to the XML route data files, and click **Open**.

4. In the toolbar below the tabs, click the **Play** icon to the right of the **Fire every \_ s.** box to start sending recorded location data to the application.

## Contents of the Sample

The following table lists the C# source files for the sample:

|  |  |
| --- | --- |
| Filename | Description |
| App.xaml.cs | Implements App as the Application class object and the PhoneApplicationFrame class. |
| MainPage.xaml.cs | Implements the initial UI for the application, requests path predictions and processes the results, and manages user input. |
| MapLegendPage.xaml.cs | Implements the code for the map legend. |
| Pulsar.xaml.cs | Implements the code to display a position indicator for current and target locations. |

Because this document assumes that you are familiar with Windows Phone application development, the walkthrough focuses on the aspects of the sample that are unique to the Hawaii Path Prediction service.

## Initialization

When you start the application, the constructor for the App object performs basic initialization tasks for the application. This code appears in App.xaml and its code-behind file.

After application initialization is complete, Silverlight makes the main phone application frame active and calls the MainPage constructor method. The constructor initializes the pulsars that mark the current and target locations and activates the global positioning system (GPS) in the mobile device. The sample displays “crumbs” that show the user’s path approximately every 50 feet.

## Path Prediction

After the GPS has reported more than one location, the sample can predict the user’s path.

The sample calls the internal ExecutePrediction function in a work item and passes it the array of GPS points to use in the prediction. ExecutePrediction, in turn, calls the Path Prediction service to make the prediction. The sample locks the tripCrumbs structure and invokes ExecutePrediction in a work item, as the following code snippet from MainPage.xaml.cs shows. The lock is required to ensure data integrity because the GPS returns data continuously and asynchronously.

lock (this.tripCrumbs)

{

if (this.tripCrumbs.Count > 1 && this.predictionRequestsInProgress == 0)

{

// run a predicted location search

Interlocked.Increment(ref this.predictionRequestsInProgress);

ThreadPool.QueueUserWorkItem(new WaitCallback(this.ExecutePrediction),

this.tripCrumbs.ToArray());

}

}

The following shows the code for the ExecutePrediction function:

private void ExecutePrediction(object o)

{

this.Dispatcher.BeginInvoke(() =>

{

predictionTextBlock.Text = "predicting destination...";

});

try

{

LatLong[] crumbs = (LatLong[])o;

if (crumbs.Length == 0)

{

crumbs = new LatLong[] { currentLocation };

}

this.predictionRequest.Path = new LatLong[crumbs.Length];

for (int i = 0; i < crumbs.Length; i++)

{

this.predictionRequest.Path[i] =

new LatLong((float)crumbs[i].Latitude,

(float)crumbs[i].Longitude);

}

this.predictionRequest.MaxDestinations = MaxDestinations;

PathPredictionService.PredictLocationAsync(

HawaiiClient.AdmClientId,

HawaiiClient.AdmClientSecret,

this.predictionRequest,

new ServiceAgent<PredictLocationResult>.OnCompleteDelegate(

this.PredictionRequest\_Completed), null);

}

catch

{

Interlocked.Decrement(ref this.predictionRequestsInProgress);

this.Dispatcher.BeginInvoke(() =>

{

predictionTextBlock.Text = "unable to make a prediction";

});

}

}

This function begins by displaying a notice to the user. Notice the use of **Dispatcher.BeginInvoke**, which ensures that the text update occurs on the main UI thread, as Silverlight requires.

Because the **PredictLocationAsync** method can throw exceptions, the code that calls it appears in a **try**-**catch** statement. To prepare to request a prediction, the sample sets up a **LatLong** structure to hold the current crumbs, which are passed in the o **object** parameter to ExecutePrediction. The sample then populates predictionRequest—a **PredictLocationRequest** structure—with the path information to pass to the Path Prediction service. Both **LatLong** and **PredictLocationRequest** are defined in the **Microsoft.Hawaii.PathPrediction.Client** namespace.

To request a prediction, an application passes a **PredictLocationRequest** structure that contains the following information:

* In the **Path** member, a list of **LatLong** structures that contain the floating-point latitude and longitude of the crumbs.
* In the **MaxDestinations** member, the number of possible destinations requested. The sample program requests 1000 destinations by default. More destinations results in a more accurate prediction but uses more bandwidth and slows the prediction process.

The sample then calls **PathPredictionService.PredictLocationAsync** to request a prediction. This method takes the following arguments:

* The ADM client ID, which is set in the HawaiiClient.cs file.
* The ADM client secret, which is set in the HawaiiClient.cs file.
* The completed **PredictLocationRequest** structure.
* A pointer to a delegate for callback when the method has completed.
* A user-defined object, which the application does not use in this call.

When the method is complete, the Path Prediction service calls the PredictionRequest\_Completed callback, which appears in the MainPage.xaml.cs file. The callback function checks the returned status from **PredictLocationAsync** and notifies the user whether the prediction succeeded or failed. This part of the code is straightforward, so this document does not show it.

If prediction succeeds, **PredictLocationAsync** returns a list of possible locations in the **PredictLocationResult.PossibleDestinations** member. **PossibleDestinations** contains a list of **PossibleDestination** structures; each such structure, in turn,contains a **LatLong**, a **Probability** that the location is the destination of the current path.

Using the returned probability for each possible destination, the sample calculates a weighted mean to represent the center of the target location, and calculates the standard deviation to determine the radius of the pulsar circle that surrounds the target on the display. The following shows the code from MainPage.xaml.cs:

if (this.possibleDestinations.Length > 0)

{

// Find the centroid of the possible destinations using weighted means

// for more info on weighted means see:

// http://en.wikipedia.org/wiki/Weighted\_mean

LatLong centroid = new LatLong(0, 0);

double probabilitySum = 0;

// Compute the weighted mean with the normalized probabilities of

// the top 25 percentile.

int topPercentile =

Math.Max(this.possibleDestinations.Length / 4, 1);

this.possibleDestinations =

this.possibleDestinations.OrderByDescending(d =>

d.Probability).Take(topPercentile).ToArray();

foreach (PossibleDestination pd in this.possibleDestinations)

{

centroid.Latitude += (float)(pd.Location.Latitude \* pd.Probability);

centroid.Longitude += (float)(pd.Location.Longitude \* pd.Probability);

probabilitySum += pd.Probability;

}

centroid.Latitude = (float)(centroid.Latitude / probabilitySum);

centroid.Longitude = (float)(centroid.Longitude / probabilitySum);

// save the target location

this.TargetLocation = centroid;

// enable the "center on target" menu button

this.Dispatcher.BeginInvoke(() =>

{

ApplicationBarIconButton btn =

(ApplicationBarIconButton)ApplicationBar.Buttons[0];

btn.IsEnabled = true;

});

// now get the standard deviation

// (i.e. radius of the target location pulsar)

double weightedMean = 0;

foreach (PossibleDestination pd in this.possibleDestinations)

{

weightedMean = pd.Probability \* Math.Pow(centroid.DistanceMeters(

new LatLong(pd.Location.Latitude, pd.Location.Longitude)), 2);

}

weightedMean = Math.Sqrt(weightedMean / probabilitySum);

// set the target location of the pulsar

this.Dispatcher.BeginInvoke(() =>

{

// make the pulsar visible

this.TargetLocationPulsar.Visibility =

System.Windows.Visibility.Visible;

this.TargetLocationPulsarRadiusInMeters = weightedMean \* 3.0;

if (this.TargetLocationPulsarRadiusInMeters /

GetMapResolution(centroid.Latitude, MyMap.ZoomLevel) > 25)

{

this.TargetLocationPulsar.Radius =

this.TargetLocationPulsarRadiusInMeters /

GetMapResolution(centroid.Latitude, MyMap.ZoomLevel);

}

else

{

this.TargetLocationPulsar.Radius = 25;

}

GeoCoordinate gc = new GeoCoordinate(centroid.Latitude,

centroid.Longitude);

if (StaticLocations.Children.Contains(this.TargetLocationPulsar))

{

StaticLocations.Children.Remove(this.TargetLocationPulsar);

}

StaticLocations.AddChild(this.TargetLocationPulsar, gc,

PositionOrigin.TopLeft);

});

The calls to GetMapResolution invoke a local method on the MainPage class that calculates the zoom level on the current map. This calculation ensures that the radius of the target pulsar display is appropriate for the resolution of the map. For more information about zoom levels on Bing maps, see “Map Control: Zoom Levels -&gt; Resolution,” which is listed in “Resources.”

# Using the Path Prediction Service in an Application

* Add required assemblies to the Visual Studio project.
* Reference the namespace in your source code.
* Set up your authentication credentials and, if necessary, the Bing Maps key.

## Add Required Assemblies

Applications that use the Path Prediction service depend on the following libraries, which are built as part of the Project Hawaii SDK:

* Microsoft.Hawaii.ClientBase.dll
* Microsoft.Hawaii.PathPrediction.Client.dll

To add the libraries to your application

* Build the Hawaii SDK, as described in “Getting Started with the Project Hawaii SDK.”
* Add references to the following DLLs to your Visual Studio project:
* Microsoft.Hawaii.ClientBase.dll
* Microsoft.Hawaii.PathPrediction.Client.dll

## Reference the Namespace

The Path Prediction service is defined in the **Microsoft.Hawaii.PathPrediction.Client** namespace. For ease of reference, include the following in your code:

using Microsoft.Hawaii;

using Microsoft.Hawaii.PathPrediction.Client;

## Set Up Your Credentials and Bing Maps Key

Your application authenticates itself with the service by using an Azure Data Market (ADM) client ID and secret. If you do not already have ADM credentials, obtain them as described in “Getting Started with the Project Hawaii SDK.”

The easiest way to use the ADM credentials in your code is to copy the HawaiiClient.cs file from one of the sample applications, add it to your project, and change the values of the following strings to your ADM client ID and secret, respectively:

* **AdmClientId**
* **AdmClientSecret**

You can then use **HawaiiClient.AdmClientId** and **HawaiiClient.AdmClientSecret** wherever the service requires ADM credentials.

If your application uses the Bing Maps service, you must have a Bing Maps Account and a Bing Maps Key, as described earlier in “Walkthrough: Path Prediction Sample.”

# Resources

This section provides links to additional information about the Project Hawaii Path Prediction Service and related topics.

Microsoft Research Project Hawaii

<http://research.microsoft.com/en-us/projects/hawaii/default.aspx>

Getting Started with the Project Hawaii SDK

<http://research.microsoft.com/en-US/projects/hawaii/docs.aspx>

Microsoft Research Project Hawaii on Facebook

<http://www.facebook.com/pages/Microsoft-Research-Project-Hawaii/164295863611699>

Research Papers

John Krumm and Eric Horvitz, "Predestination: Inferring Destinations from Partial Trajectories", Eighth International Conference on Ubiquitous Computing (UbiComp 2006), September 2006.  
<http://research.microsoft.com/en-us/um/people/jckrumm/Publications%202006/predestination%20preprint%20final.pdf>

MSDN

Bing Maps  
<http://msdn.microsoft.com/en-us/library/dd877180>

Map Control: Zoom Levels -&gt; Resolution  
<http://blogs.msdn.com/b/virtualearth/archive/2006/02/25/539239.aspx>

Programming Windows Phone 7  
<http://blogs.msdn.com/b/microsoft_press/archive/2010/10/28/free-ebook-programming-windows-phone-7-by-charles-petzold.aspx>

How to: Create Your First Silverlight Application for Windows Phone  
<http://msdn.microsoft.com/library/ff402526(v=VS.92).aspx>