



MOTODEV
The Motorola developer network

MOTO Q 9 WINDOWS MOBILE GPS APPLICATION
V 1.0

TECHNICAL ARTICLE



MOTODEV

Copyright © 2008, Motorola, Inc. All rights reserved. This documentation may be printed and copied solely for use in developing products for Motorola products. In addition, two (2) copies of this documentation may be made for archival and backup purposes. Except for the foregoing, no part of this documentation may be reproduced or transmitted in any form or by any means or used to make any derivative work (such as translation, transformation, or adaptation) without express written consent from Motorola, Inc.

Motorola reserves the right to make changes without notice to any products or services described herein. "Typical" parameters, which may be provided in Motorola Data sheets and/or specifications, can and do vary in different applications and actual performance may vary. Customer's technical experts will validate all "Typicals" for each customer application.

Motorola makes no warranty in regard to the products or services contained herein. Implied warranties, including without limitation, the implied warranties of merchantability and fitness for a particular purpose, are given only if specifically required by applicable law. Otherwise, they are specifically excluded.

No warranty is made as to coverage, availability, or grade of service provided by the products or services, whether through a service provider or otherwise. No warranty is made that the software will meet your requirements or will work in combination with any hardware or application software products provided by third parties, that the operation of the software products will be uninterrupted or error free, or that all defects in the software products will be corrected.

In no event shall Motorola be liable, whether in contract or tort (including negligence), for any damages resulting from use of a product or service described herein, or for any indirect, incidental, special or consequential damages of any kind, or loss of revenue or profits, loss of business, loss of information or data, or other financial loss arising out of or in connection with the ability or inability to use the Products, to the full extent these damages may be disclaimed by law.

Some states and other jurisdictions do not allow the exclusion or limitation of incidental or consequential damages, or limitation on the length of an implied warranty, therefore the above limitations or exclusions may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights, which vary from jurisdiction to jurisdiction.

Motorola products or services are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product or service could create a situation where personal injury or death may occur.

Should the buyer purchase or use Motorola products or services for any such unintended or unauthorized application, the buyer shall release, indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the designing or manufacturing of the product or service.

Motorola recommends that if you are not the author or creator of the graphics, video, or sound, you obtain sufficient license rights, including the rights under all patents, trademarks, trade names, copyrights, and other third party proprietary rights.

If this documentation is provided on compact disc, the other software and documentation on the compact disc are subject to the license agreement accompanying the compact disc.

MOTO Q 9 Windows Mobile GPS Application

Version 1.0

May 2008

For the latest version of this document, visit <http://developer.motorola.com>

Motorola, Inc.

<http://www.motorola.com>



Contents

Introduction	3
Purpose	3
Background	3
History	3
GPS Intermediate Driver	4
Introduction to the GPS location wrapper	4
Preparing the GPS location wrapper	5
Example C# Code	5
Summary	10
Points to remember	10
GPS chip detection	10
Power management	11
Conclusion and references	11

Introduction

Imagine having an application for your MOTO Q 9 that calculates not just your location (latitude, longitude, and elevation) but also the speed at which you travel. This power is available today on a select set of MOTO Q 9 handsets. These handsets provide an integrated GPS (Global Position System) radio and a standard Windows Mobile Location Based Services API that allows application developers to easily develop GPS solutions for their MOTO Q 9 customers.

Combining the power of knowing where you are with information of relevant interest in a mobile Smartphone (e.g., directions to a coffee shop) further enhances the experience of the mobile user!

Purpose

This article demonstrates how to create a simple location detection application with minimum programming complexity on a MOTO Q 9 handset, with integrated GPS, using Windows Mobile Location Based Service APIs.

Background

GPS is all about geosynchronous satellites sending out time signals. Based on the time delay from satellites with known geostationary locations, the receiver's location can be determined in a predefined grid. A GPS receiver needs to receive signals from four or more of these satellites to calculate a three-dimensional location. GPS receivers were expensive a couple of years ago. Devices with GPS receivers have become more popular and their prices have dropped significantly. More and more devices appearing on the market are equipped with a GPS receiver. The MOTO Q 9c and MOTO Q 9h Windows Mobile Smartphones from Motorola are equipped with a GPS receiver.

NOTE: The availability of integrated GPS support is carrier and region-dependent.

History

Before there was a dedicated API, you needed to obtain information from a GPS receiver following a standard established by the National Marine Electronics Association (NMEA)¹. You had to parse the data stream, which included a large amount of extra information, to find the key data elements to identify the location of the receiver and then translate this raw data into useful information (hours, minutes, and seconds defining longitude and latitude).

Microsoft added an intermediate GPS driver to Windows Mobile 5.0² that enables easier access to a GPS receiver and thus frees you from the burden of inventing your own GPS library, allowing you to focus your attention on how to apply the information from the GPS receiver to your application.

GPS Intermediate Driver

Accessing GPS capabilities using the GPS Intermediate Driver can become quite cumbersome. To do that you would need to be an experienced Windows programmer. But we have made it simpler for you by using the wrapper for location DLL, provided by Microsoft, to access the low-level GPS APIs. Using this wrapper, we can easily access the capabilities of the GPS Intermediate Driver to get the GPS coordinates of the handset.

Introduction to the GPS location wrapper

Before developing an application, it is good to know what functionality the wrapper includes and how to access it. The wrapper includes ten classes that can be used by the user interface application. This code utilizes two classes: `Gps` and `GpsPosition`. Public properties, methods, and events of these two classes are shown in the Figure 1.

`Gps` is the interface to the managed GPS API. This class can be used to open, close, and query the device state and to query the position data from your GPS handset.

The `GpsPosition` class contains the GPS position data received from the GPS handset.³

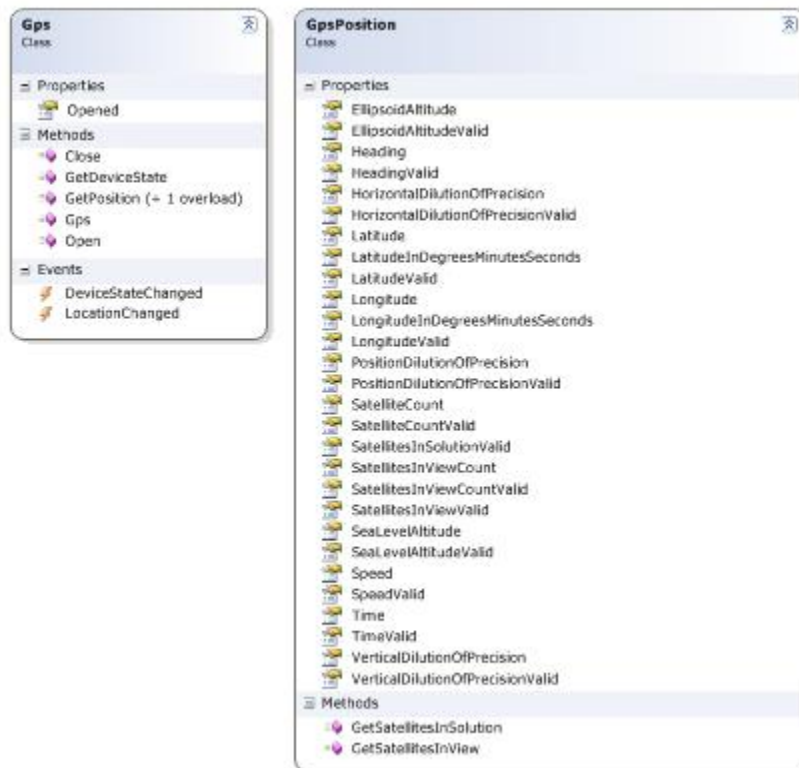


Figure 1: Public properties, methods, and events for `Gps` and `GpsPosition` classes

Preparing the GPS location wrapper

Windows Mobile 6 Standard SDK refresh version sample code includes a well packaged DLL module that can be used for this lower level functionality implementation. Older SDKs have various issues with less mature versions of this module. Microsoft recommends installing the Windows Mobile 6 SDK and using the GPS sample from the installation, even if your application is targeting Windows Mobile version 5.0 based devices.⁴

The following points describe the steps to prepare the DLL module:

- 1 Open the sample project file C:\Program Files\Windows Mobile 6 SDK\Samples\Smartphone\CS\GPS\GPS.sln with Visual Studio.
- 2 The Solution Manager shows two projects: GpsSample and Microsoft.WindowsMobile.Samples.Location. GpsSample is the user interface part and Microsoft.WindowsMobile.Samples.Location is the module that handles the lower level functionality.
- 3 Set the project type to Release.
- 4 Compile the project.
- 5 Browse to C:\Program Files\Windows Mobile 6 SDK\Samples\Smartphone\CS\GPS\bin\Release directory and find a recently compiled Microsoft.WindowsMobile.Samples.Location.dll file. This file is a low level GPS functionality wrapper that hides actual location detection functionality and lets the user interface layer be driven by events. It will be utilized by our application as a Reference module.

Now we are ready to build our user interface for the GPS application.

Example C# Code

Step 1: Create a new C# Smart Device project. Choose the Compact Framework version that you prefer.

Step 2: In order to use the GPS wrapper in our application, the wrapper needs to be added to the project as a reference. Copy the `Microsoft.WindowsMobile.Samples.Location.dll` from the C:\Program Files\Windows Mobile 6 SDK\Samples\Smartphone\CS\GPS\bin\Release directory to the folder <Project Dir>\<Project Dir>, that is, the project name folder that contains the `bin` and `obj` folders. Select Menu Project ->Add Reference. Select a Browse Tab and browse to the directory where you copied the `dll` file. Select the `Microsoft.WindowsMobile.Samples.Location.dll` file and click OK.

Step 3: Import the location wrapper to the project using `Microsoft.WindowsMobile.Samples.Location`.

Find this at the top of the `Form1.cs` code, right under the already defined using statements. Compile the project to see that there are no problems. Now the application can utilize all GPS wrapper functionality.

Step 4: Define class `Form1` attributes for `Gps`, `GpsPosition`, and private `EventHandler` variable types. Use the attribute names shown in the code sample below.

```
Gps gps;
GpsPosition position;
private EventHandler updateDataHandler;
```

Step 5: Create the `Gps` object at the `Form1` class constructor after the `InitializeComponent()` function. This way the object is created once and it lives as long as the application is running.

```
public Form1()
{
    InitializeComponent();
    gps = new Gps();
}
```

Step 6: Open the GPS device with a GPS class method `Open()`. Please note that the GPS chip detection is not possible with this function because the `Open` function does not return any value. For more information about the built-in GPS chip detection, see the “Points to remember” section on page 10.

```
public Form1()
{
    InitializeComponent();
    gps = new Gps();
    gps.Open();
}
```

The location can be determined with three additional commands. Next is a quick demonstration.

```
GpsPosition position = gps.GetPosition();
double lat = position.Latitude;
double lon = position.Longitude;
```

Our application uses sophisticated event driven position detection.

Step 7: Drag and drop three Labels and three Textboxes from a Toolbox window to the `Form1`. Set the layout as shown in Figure 2.

- Modify Labels' Text attribute to: “Latitude”, “Longitude”, and “Speed”.
- Name the TextBoxes ‘textLatitude’, ‘textLongitude’, and ‘textSpeed’.



Figure 2: Labels and text boxes for GPS coordinates

Step 8: Connect the Wrapper's `LocationChanged` event to the application's location change handling function `gps_Locationchanged` at the `Form1` constructor.

```
public Form1()
{
    InitializeComponent();
    gps = new Gps();
    gps.Open();
    gps.LocationChanged += new
        LocationChangedEventHandler(gps_Locationchanged);
}
```

Step 9: Create the `gps_Locationchanged` method.

```
void gps_Locationchanged(object sender, LocationChangedEventArgs args)
{
}
```

Step 10: The GPS position information is in the `LocationChangedEventArgs` parameter's property `Position`. Assign that value to the already defined position variable.

```
void gps_Locationchanged(object sender, LocationChangedEventArgs args)
{
    position = args.Position;
}
```

Step 11: Add an `Invoke` function with the `updateDataHandler` parameter to end of the `gps_Locationchanged` method. This method is needed to pass the location data to the UI thread.

```
void gps_Locationchanged(object sender, LocationChangedEventArgs args)
{
    position = args.Position;
    Invoke(updateDataHandler);
}
```

Step 12: Create the `updateDataHandler` event handler at the end of the `Form1` class constructor. Connect that handler to the method `UpdateData`.

```
public Form1()
{
    InitializeComponent();
    gps = new Gps();
    gps.Open();
    gps.LocationChanged += new
        LocationChangedEventHandler(gps_Locationchanged);
    updateDataHandler = new EventHandler(UpdateData);
}
```

Step 13: Add the `UpdateData` method. This function can modify the property values of the user interface controls. It is a good idea to check that the `gps` and `position` variables are not null before accessing the location data property values.



```
void UpdateData(object sender, System.EventArgs args)
{
    if(gps.Opened)
    {
        if (position != null)
        {
            //get numeric values
            double lat = position.Latitude;
            double lng = position.Longitude;
            double spd = position.Speed;

            //update UI texts
            textLatitude.Text = lat.ToString();
            textLongitude.Text = lng.ToString();
            textSpeed.Text = spd.ToString();
        }
    }
}
```

Step 14: Create an Exit item on the left soft menu button; create an event handler for it. This code should check whether the GPS device is open. If the GPS device is open, disconnect from the location change events and close the GPS handset. After doing this, the application must exit.

```
private void Event_Exit(object sender, EventArgs e)
{
    if (gps.Opened)
    {
        gps.LocationChanged -= gps_Locationchanged;
        gps.Close();
    }
    Application.Exit();
}
```

Step 15: Compile the project and deploy it to the phone. Once the application is running and it has detected enough GPS satellite signals, coordinates of the location are shown on the display.

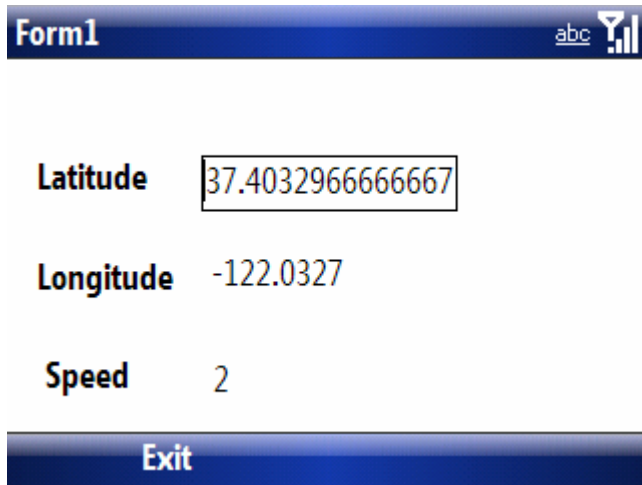


Figure 3: Screenshot of a working application.

Summary

This was a simplified implementation of a GPS application to get GPS coordinates with a C# Smartphone application. This application has been kept simple for readability and understandability. You can add more functionality for more advanced applications.

Points to remember

GPS chip detection

The method of detecting the presence of the GPS Receiver is not consistently applied across handset models. Though not detailed in the example above, we have found the following method to work on the MOTO Q 9h and MOTO Q 9c handsets.

All Windows Mobile 5 and 6 phones include the intermediate drive, whether or not the GPS chip exists. Because of that, the function `GPSOpenDevice` can not be used to detect the built-in chip. Also, registry values are not a reliable way to detect the built-in chip unless the application knows the device-specific GPS related registry keys.

There are no known methods to detect the built-in GPS chip. The `Gps` class has an event `DeviceStateChanged` that can be used to check the `GpsDeviceState` class state after the device state has changed. The class has two property values: `DeviceState` and `ServiceState`. The `DeviceState` property value stays at value 2 (`StartingUp`) on phones that do not have a GPS chip while the value is 1 (`On`) for phones with the chip. Check the Standard 6 SDK C# GPS sample project to see how to use the `DeviceStateChanged` event and `GpsDeviceState` class.

Power management

In most devices, GPS hardware is one of the largest consumers of system resources on the entire handset. GPS hardware may use as much power as the device display.⁵ From the power management viewpoint, keep the GPS hardware open only when necessary.

Conclusion and references

The combination of GPS and Windows Mobile application development has created an immense opportunity in terms of improving GPS operations on the go. From improving the end-user experience to gaining better control over activities, GPS promises to change the way a mobile application is perceived.

We hope this article has provided some insights and a basic understanding of how to create an application to use the Intermediate GPS Driver to obtain the required GPS data. We give you a basic understanding of how to start development with GPS on the MOTO Q 9h and MOTO Q 9c handsets. However, GPS application development on these handsets can be done at more advanced levels using the same building blocks provided here. Our goal is to you give you a jump start exploring this opportunity to tap into the GPS resources and using it optimally to create new applications.

¹ Writing your own GPS Application: Part I
<http://www.codeproject.com/KB/vb/WritingGPSApplications1.aspx>

² A first glance at the GPS API in Windows Mobile 5.0,
[http://www.pocketpcdn.com/articles/articles.php?&atb.set\(c_id\)=45&atb.set\(a_id\)=6509&atb.perform\(details\)=](http://www.pocketpcdn.com/articles/articles.php?&atb.set(c_id)=45&atb.set(a_id)=6509&atb.perform(details)=)

³ Windows Mobile Version 5.0 SDK > GPS: <http://msdn2.microsoft.com/en-us/library/ms881362.aspx>

⁴ Using the GPS Intermediate Driver from Managed Code,
<http://msdn2.microsoft.com/en-us/library/bb158708.aspx>

⁵ GPS Intermediate Driver Power Management, <http://msdn2.microsoft.com/en-us/library/bb202066.aspx>