عنوان	**توسعه نرم افزار برای سیستم عامل ویندوز موبایل،** **بخش چهارم: کنترل های سفارشی و استفاده از سخت افزار GPS (موقعیت یاب ماهواره ای)**
عنوان اصلی	**Windows Mobile App Development Part 4: Adding Custom Controls and Making Use of GPS Hardware**
کلمات کلیدی	C#, Win Mobile, GPS, Custom Control, .NET, Dev
مؤلف	**mstruys, dougturn**
مرجع	http://www.codeproject.com
سطح	مبتدی
مترجم	مهدی عبداللهی http://m0911.wordpress.com
تاریخ انتشار	۳ فروردین ۱۳۸۹
تعداد صفحه	۱۵
مطالب مرتبط	توسعه ی نرم افزار برای سیستم عامل ویندوز موبایل، بخش اول: ایجاد نخستین برنامه توسعه ی نرم افزار برای سیستم عامل ویندوز موبایل، بخش دوم: شبیه ساز دستگاه و مدیریت شبیه ساز توسعه ی نرم افزار برای سیستم عامل ویندوز موبایل، بخش سوم: توسعه ی برنامه با WinForm توسعه ی نرم افزار برای سیستم عامل ویندوز موبایل، بخش پنجم: مقدمه ای بر SQL Server CE توسعه ی نرم افزار برای سیستم عامل ویندوز موبایل، بخش ششم: امنیت دستگاه و نصب نرم افزار توسعه ی نرم افزار برای سیستم عامل ویندوز موبایل، بخش هفتم: توسعه برای وب موبایل
فایل های ضمیمه	

مقدمه

توسعه ی نرم افزار ویندوز موبایل شباهت زیادی به توسعه ی نرم افزار در دسکتاپ دارد به ویژه زمانی که یکی از دو زبان ویژوال بیسیک یا ویژوال سی شارپ دات نت را استفاده می کنید. شما همان ابزارهای توسعه ی برنامه های ویندوز دسکتاپ را برای ویندوز موبایل هم استفاده می کنید لیکن تفاوت هایی نیز بین این دو محیط هست. دستگاه های ویندوز موبایل صفحه ی نمایش کوچک تر و منابع محدود تر دارند و همچنین قابل حمل بوده، اغلب باتری استفاده می کنند.

در این مقاله یاد می گیرید که چطور نرم افزارهای بر مبنای ویندوز فرم، برای موبایل ایجاد کنید. کنترل های سفارشی را به برنامه تان اضافه کنید و چگونگی ساختن کنترل های سفارشی با پشتیبانی کامل محیط طراحی را تمرین کنید. به علاوه کاربرد سخت افزار GPS داخل نرم افزار و همچنین به روز رسانی مقادیر کنترل های واسط کاربر را داخل یک برنامه ی چند پردازشی (Multi Threaded) را نیز فرا خواهید گرفت.

افزودن کنترل های سفارشی به برنامه ی خودتان

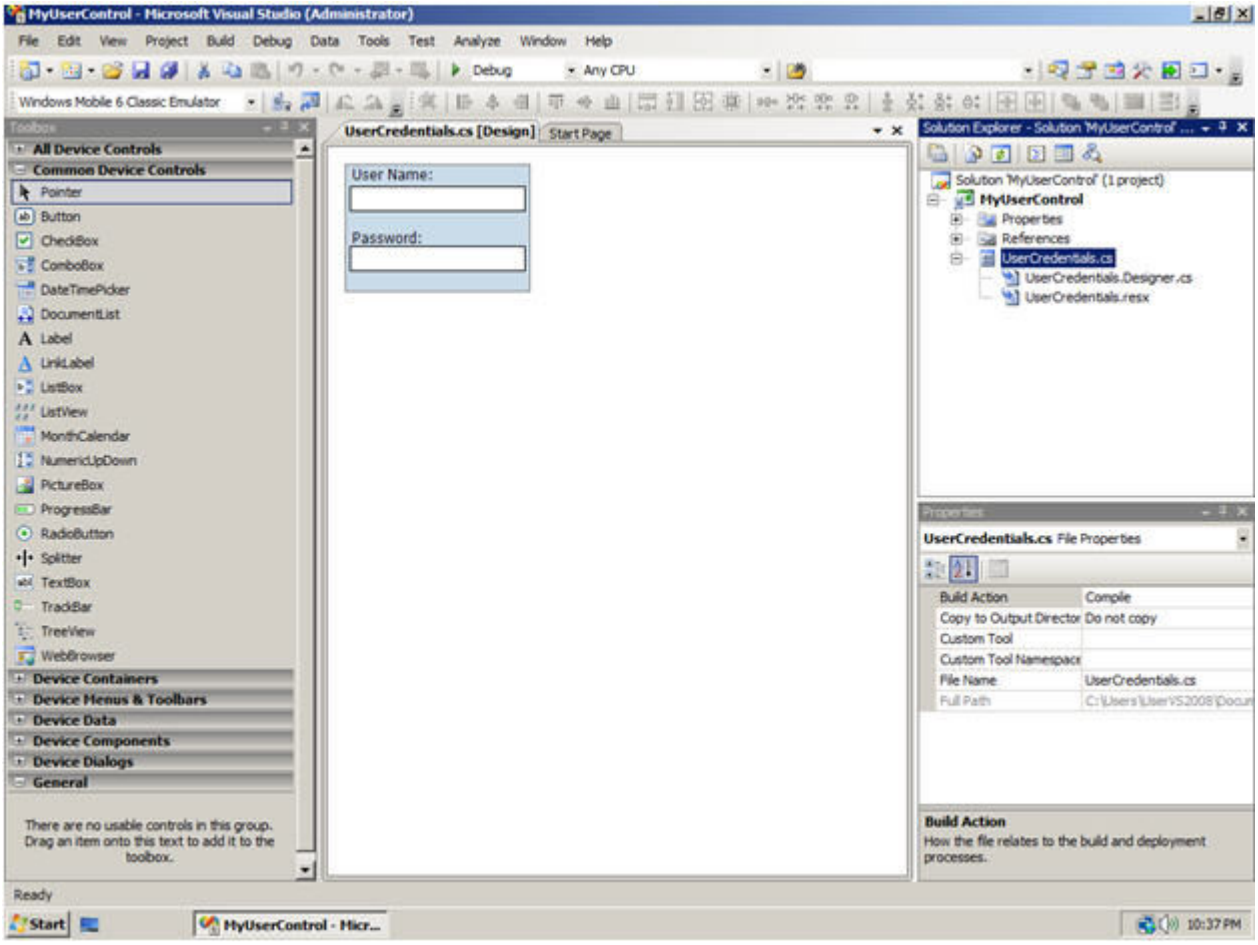
اگر به هنگام توسعه ی برنامه ی ویندوز موبایل به کنترل های خاصی نیاز پیدا کنید که در مجموعه ی کنترل های استاندارد .NET Compact Framework نیستند، راه حل های زیر را پیش رو خواهید داشت:

- یک کنترل واسط کاربر فقط برای برنامه ی فعلی تان درست کنید
- یک کنترل واسط کاربر که در برنامه ی دیگر هم قابل استفاده باشد، درست کنید
- یک کنترل واسط کاربر با امکانات مورد نظرتان خریداری کنید

در این مقاله سه روش ایجاد کنترل ها را هم از ابتدا و هم از طریق استفاده ی مجدد و ترکیب با کنترل های موجود یاد خواهید گرفت.

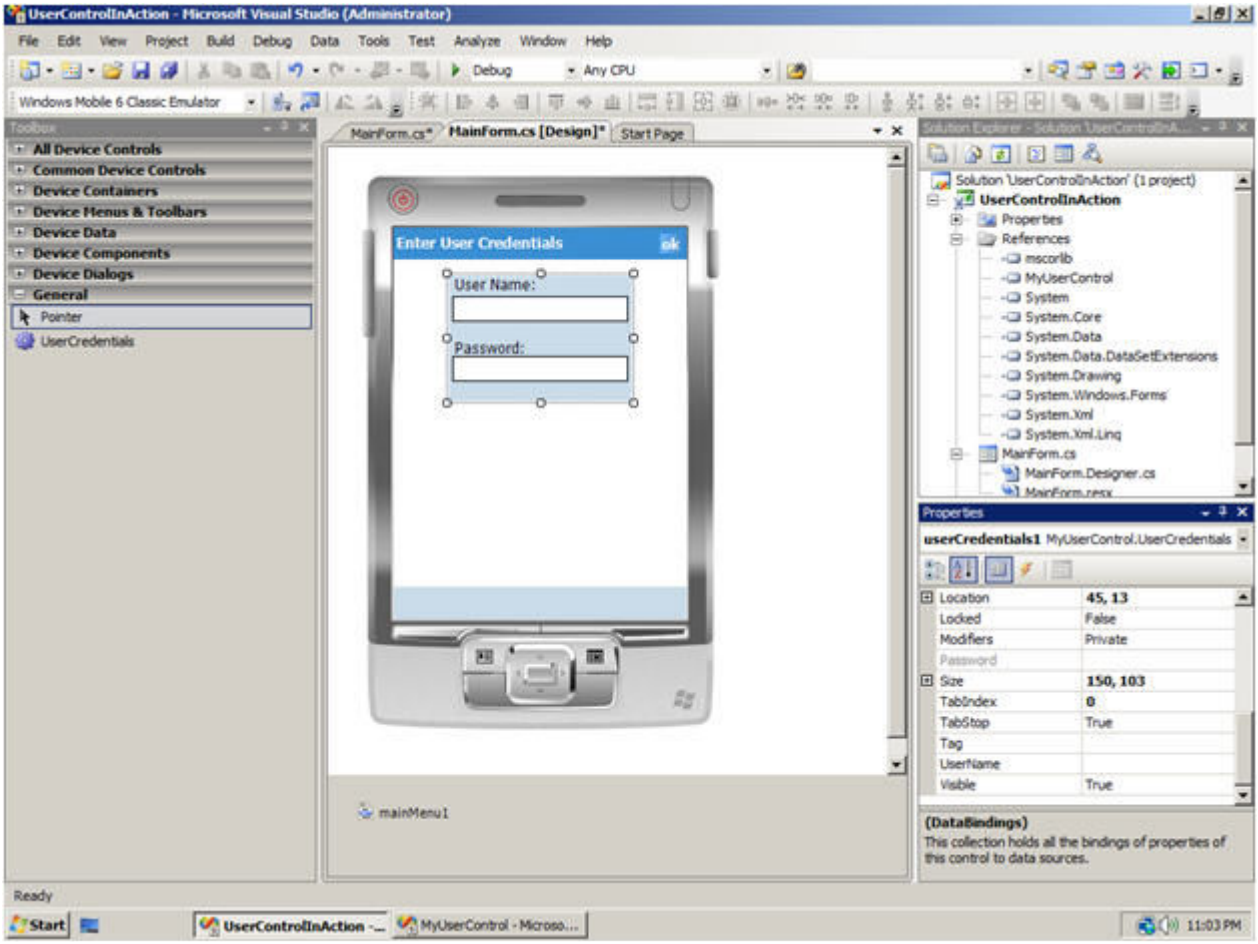
کنترل های کاربر (سفارشی)

این کنترل ها معمولا با ترکیب چند کنترل موجود و درست کردن یک کنترل جدید به کار می روند. در ویژوال استودیو ۲۰۰۸ ایجاد کنترل های کاربر به طور کامل توسط یکی از انواع پروژه ی آن (User Control Project) پشتیبانی می شود. وقتی یک کنترل جدید ایجاد می کنید یک صفحه ی طراحی مانند کانتینر پدید می آید که کنترل ها را روی آن قرار می دهید. با یک کنترل کاربر شما می توانید چند کنترل موجود را با هم ترکیب کنید به طوری که به هنگام استفاده از کنترل ترکیبی عملکرد آن مانند یک کنترل منفرد باشد. در شکل ۱ شما یک کنترل کاربر داخل ویژوال استودیو ۲۰۰۸ می بینید. کنترل مذکور شامل دو کنترل برچسب (Label) و دو کنترل جعبه متن (TextBox) است که می توانید برای اعتبار سنجی کاربر، مثلا برای ورود به یک سیستم بانک اطلاعاتی استفاده کنید. زمانی که پروژه کامپایل می شود این کنترل در اسمبلی مربوط به خودش ذخیره می شود تا در آینده توسط هر پروژه ی دیگر ویندوز موبایل استفاده گردد. در این کنترل کاربر، کدنویسی چندانی پس از قرار دادن کنترل سفارشی روی آن لازم نیست. تنها کد مورد نیاز داخل این کنترل کاربر عبارت است از تعریف دو متغیر با سطح دسترسی عمومی (Public) برای به دست آوردن نام کاربر و گذرواژه داخل یک برنامه و همچنین مقدار دهی نام کاربر است. مشخصه ی Password (گذرواژه) فقط خواندنی (Read-only) است تا کاربر را مجبور کند که حداقل یک گذرواژه ی مجاز برای خودش در نظر بگیرد. اعتبار سنجی گذرواژه در داخل برنامه انجام می شود نه داخل کنترل کاربر. دلیل این امر راحتی کار با کنترل و امکان استفاده ی مجدد از آن است.



شکل ۱: کنترل سفارشی برای اعتبار سنجی کاربر

برای استفاده از کنترل اعتبار سنجی کاربر داخل برنامه کافی است آن را از جعبه ابزار ویژوال استودیو بکشید و روی فرم بیندازید. کنترل اعتبار سنجی کاربر به طور خودکار در جعبه ابزار نشان داده نمی شود. باید نخست روی بخش General جعبه ابزار راست کلیک کنید و از منوی باز شده، گزینه ی Choose Items را بزنید و به فایل اسمبلی که حاوی کنترل اعتبار سنجی کاربر است اشاره کنید تا کنترل مذکور در جعبه ابزار ظاهر شود. در شکل ۲ شما چگونگی نمایش کنترل اعتبار سنجی (User Credentals) را داخل برنامه می بینید. اگر به جعبه ابزار هم بنگرید کنترل با نام UserCredentals را در بخش General جعبه ابزار خواهید دید که به آسانی می توانید آن را روی فرم خود قرار دهید. پس از این کار شما در پنجره ی مشخصات (Properties) یک مشخصه ی UserName - که قابل مقدار دهی در محیط طراحی و در عین حال به صورت کد نویسی داخل برنامه است- می بینید. در عین حال مشخصه ی Password را با رنگ خاکستری در پنجره ی مشخصات خواهید دید که به دلیل فقط خواندنی بودن آن است. این مشخصه قابل مقداردهی نیست ولی مقدارش پس از آن که توسط کاربر به هنگام اجرای برنامه وارد شد، قابل بازیابی (خواندن) خواهد بود.



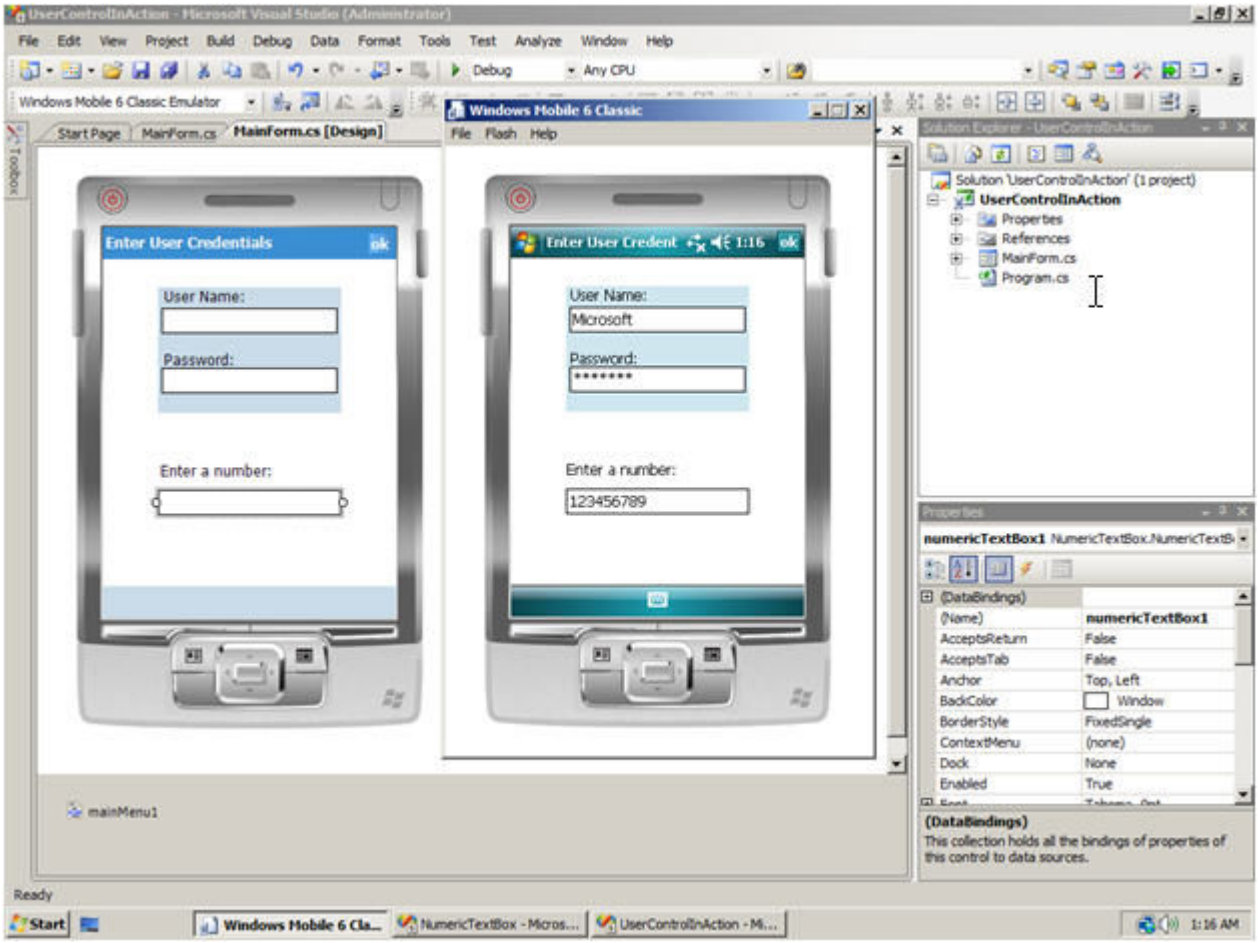
شکل ۲: کنترل سفارشی داخل یک برنامه

کنترل های ارث بری شده

گاهی اوقات مثلا یک کنترل عمومی .NET Compact Framework را استفاده می کنید لیکن لازم دارید که کارکرد آن مطابق دلخواه شما تغییر کند. مثلا یک جعبه متن (TextBox) که فقط عدد را بشود در آن وارد نمود. البته شما می توانید به کاربر اجازه دهید که هر چه خواست وارد نماید و پس از ورود اطلاعات، در داخل برنامه آن را اعتبار سنجی کنید. اما ایجاد یک کنترل سفارشی خواه داخل برنامه تان یا به صورت یک کنترل مجزای مشتق شده از جعبه متن عادی، که صرفا کاراکترهای عددی (دقیقا) به هنگام ورود داده ها بپذیرد ایده ی بهتری است. در این روش شما کدنویسی کمتری را در برنامه تان خواهید داشت چرا که اعتبار سنجی داخل کنترل ارث بری شده ی (inherited) جعبه متن صورت می گیرد. با وجود محدود شدن قابلیت های این جعبه متن جدید در مقایسه با جعبه متن معمولی، باز هم جعبه متن جدید امکانات کامل طراحی شامل همه ی مشخصه ها و رویداد های جعبه متن اصلی را در اختیار شما قرار می دهد. به علاوه اگر کنترل مزبور را به طور مجزا ایجاد و کامپایل کنید در برنامه های متعددی می توانید از آن استفاده نمایید. برای ایجاد یک کنترل با نام numericTextbox به صورت مجزا، نخست یک پروژه ی جدید از نوع Class Library را در برگه ی Smart Device Project انتخاب و ایجاد کنید. چند خط کد برای تبدیل یک جعبه متن معمولی به جعبه متن عددی مورد نظر، نیاز خواهید داشت که صرفا اجازه ی ورود داده های عددی و کلید BackSpace را به کاربر می دهد.

```
public class NumericTextBox : TextBox
{
     protected override void OnKeyPress(KeyPressEventArgs e)
     {
          if ((e.KeyChar < '0' || e.KeyChar > '9') && e.KeyChar != (char)Keys.Back)
          {
               SystemSounds.Beep.Play();
               e.Handled = true;
          }
          base.OnKeyPress(e);
     }
}
```

زمانی که شما کد باطل کننده ی متد اصلی را می نویسید باید تعیین کنید که متد کلاس پایه (اصلی) چه موقع می تواند فراخوانی شود. مطابق مستندات ارث بری در برنامه نویسی شیء گرا هنگامی که متد OnKeyPress را در یک کلاس مشتق شده باطل می کنیم باید مطمئن شویم که متد OnKeyPress متعلق به کلاس پایه فراخوانی می شود تا در اصل پیغام مربوط به رویداد دریافت گردد. از آنجا که کنترل کننده ی رویداد TextChanged فقط داده های عددی را دریافت می کند، شما پس از اعتبار سنجی کاراکتر های ورودی می توانید متد کلاس پایه را فراخوانی نمایید. استفاده از جعبه متن عددی داخل برنامه به همان روش استفاده از کنترل اعتبار سنجی کاربر است که در بخش قبلی همین مقاله دیدید.



شکل ۳: کنترل های UserCredentials (اعتبار سنجی کاربر) و NumericTextBox (ورود داده ی عددی) در حالت طراحی و اجرا داخل شبیه ساز

کنترل های سفارشی

اگر نمی خواهید کنترل های موجود را دوباره استفاده (بازنویسی) کنید ولی به کنترل های جدید نیاز دارید می توانید کنترل سفارشی خودتان را بسازید. معمولا این نوع کنترل ها از نقطه ی صفر نوشته می شوند حتی اگر بتوانید به عنوان مثال آن ها را از کلاس System.Windows.Forms.Control ارث بری نمایید که حداقل کارآیی های عمومی کنترل ها را داشته باشند. به این نوع کنترل ها دست ساز (Own Drawn Controls) هم می گویند. در ویژوال استودیو ۲۰۰۸ الگوی آماده ای برای کنترل های سفارشی نیست. آسان ترین راه همان ایجاد یک کنترل کاربر داخل ویژوال استودیو است. پس از ایجاد پروژه ی خالی می توانید کنترل کاربر را حذف کنید و به صورت دستی یک کنترل سفارشی را از ویژوال استودیو ۲۰۰۸ به آن اضافه نمایید.

```
public partial class CustomControl1 : Control
{
    public CustomControl1()
    {
        InitializeComponent();
    }

    protected override void OnPaint(PaintEventArgs pe)
    {
         // TODO: Add custom paint code here
         // Calling the base class OnPaint
         base.OnPaint(pe);
    }
}
```

برای ایجاد یک کنترل سفارشی از صفر باید خودتان آن را ترسیم (Paint) کنید. بدین منظور کافی است متد OnPaint را باطل کنید. البته این مورد از همان لحظه ی ایجاد کنترل سفارشی به طور خودکار توسط ویژوال استودیو ۲۰۰۸ به سورس برنامه تان اضافه می شود. از آ"جا که ترسیم کنترل تان به هنگام اجرای برنامه ی استفاده کننده از آن ممکن است بار ها و بار ها پیش بیاید، از این رو کارآیی روش پیاده سازی شما اهمیت دوچندان پیدا می کند. در واقع هم کارآیی و هم استفاده از منابع سیستم. از آن جا که معمولا از کلاس های مربوط به گرافیک در داخل متد OnPaint استفاده می کنید و در استفاده از این کلاس ها به کد اصلی مراجعه می کنید پاکسازی و آزاد سازی منابع مورد استفاده پس از هر بار ترسیم بسیار مهم است. در C# بهتر است از کلاس هایی استفاده کنید که یک متد Dispose را به صورت ترکیب با دستور using پیاده کرده باشند. دستور using تضمین می کند که Dispose حتی در بروز خطا به هنگام فراخوانی متدهای یک شئ هم اجرا می شود. در واقع شما از بابت مدیریت خطاها (Exception Handling) خیال تان راحت است. مثلا می خواهید یک متن را نمایش دهید.

می توانید قطعه کد زیر را استفاده نمایید و آن را داخل متد OnPaint فراخوانی کنید و شیء از نوع Graphics را به صورت بخشی از پارامتر PaintEventArgs به داخل آن بفرستید:

```
private void DisplayLabelInfo(Graphics g, string labelText, int xPos, int yPos)
{
    using (Font labelFont = new Font(FontFamily.GenericSerif, 10.0F, FontStyle.Regular))
    {
        using (SolidBrush labelBrush = new SolidBrush(LabelForeground))
        {
           g.DrawString(labelText,
                        labelFont,
                        labelBrush,
                        xPos, yPos);
        }
    }
}
```

آن چه دیدید یک متد ساده برای نمایش متن داخل یک کنترل بود. دو شیء از نوع Graphics دیگر هم لازم دارید، یکی Font و دیگری SolidBrush که هر دوی این ها به هنگام فراخوانی DisplayLabelInfo ایجاد می شوند. زمانی که این اشیا را با دستور using به کار ببرید پاکسازی آن ها کارآیی بالاتری خواهد داشت چون متد Dispose درست در زمانی که کنترل برنامه به خارج از محدوده ی تعریف (دسترسی) شیء می رود، فراخوانی می شود و در نتیجه دستورات کمتری به هنگام پاکسازی حافظه (Garbage Collection) اجرا می شود چون نیازی به فراخوانی Finalize نیست. این روش به ویژه زمانی که با کلاس های مربوط به گرافیک سر و کار دارید - به دلیل آن که آن ها از منابع محدود محلی استفاده می کنند که باید در اسرع وقت آزاد شوند- بسیار مفید است.

اگر شما مستقیما تمام ترسیم های گرافیکی را با استفاده از یک کپی نمونه ی Graphics انجام دهید صفحه نمایش حالت پرش خواهد داشت، به ویژه زمانی که کدهای زیادی را در متد OnPaint خود نوشته اید. در این حالت بهترین کار استفاده از تکنیک بافر است. یعنی این که شما هر چیزی را که می خواهید ترسیم کنید ابتدا در نمونه های جدا گانه ی Graphics در حافظه ی اصلی انجام دهید و پس از ترسیم همه ی آن ها محتوای شیء Graphics را از حافظه ی اصلی به شیء Graphics که ترسیم روی صفحه ی نمایش را انجام می دهد کپی کنید.

فرض کنید یک تصویر همراه با یک متن روی آن نمایش می دهید. برای استفاده از بافر مراحل زیر را باید انجام دهید:

```
protected override void OnPaint(PaintEventArgs pe)
{
    // Initialize a Graphics object with the bitmap we just created
    // and make sure that the bitmap is empty.
    Graphics memoryGraphics = Graphics.FromImage(memoryBitmap);
    memoryGraphics.Clear(this.BackColor);
    // Draw the control image in memory
    memoryGraphics.DrawImage(someImage,
                             destRect,
                             imageRect,
                             GraphicsUnit.Pixel);
    // Display some text
    DisplayLabelInfo(memoryGraphics, "Some text", 0, 0);
    // Draw the memory bitmap on the display
    pe.Graphics.DrawImage(memoryBitmap, 0, 0);
    // Calling the base class OnPaint
    base.OnPaint(pe);
}
```

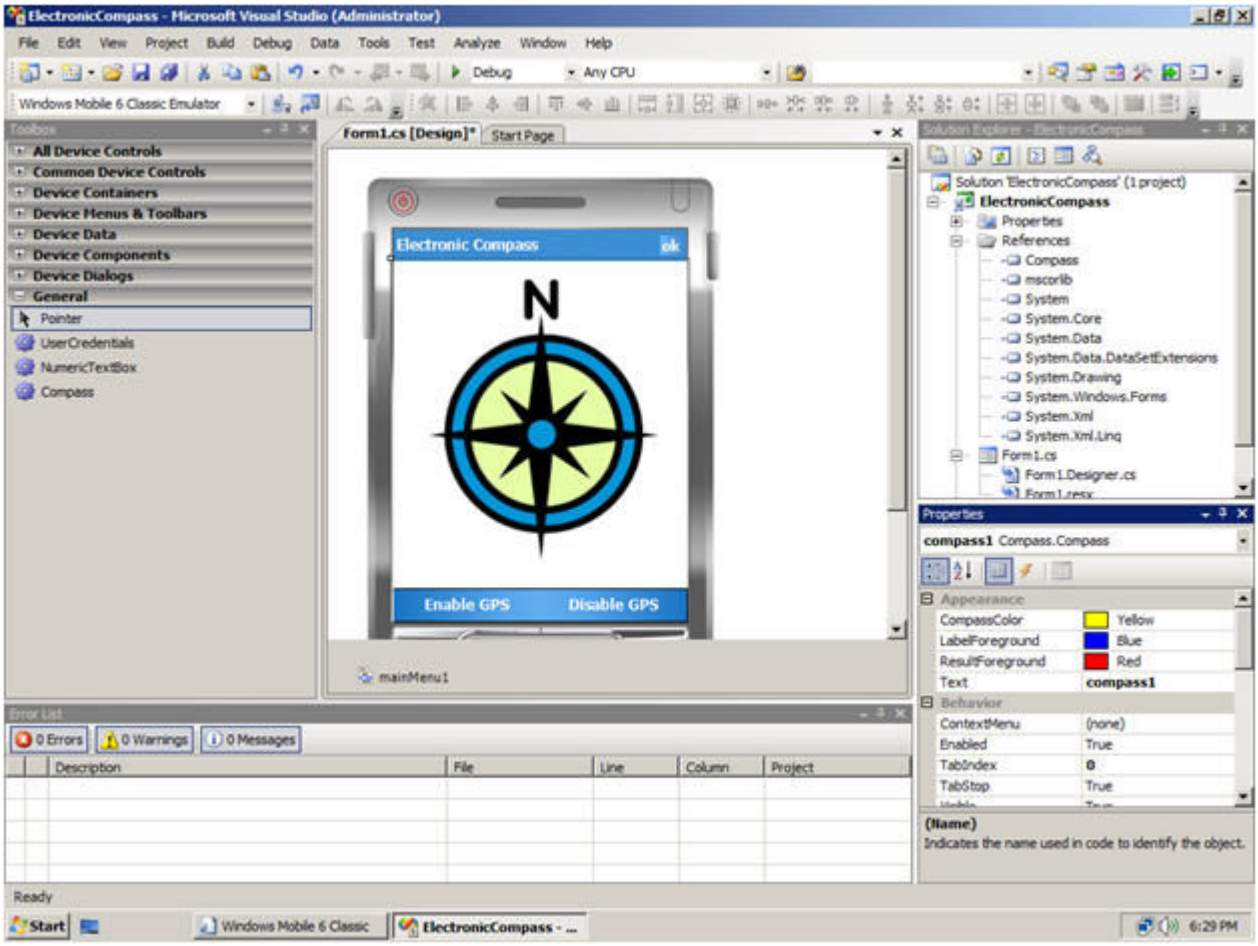
در کد قبلی فرض بر این است که شما یک نمونه از نوع Bitmap با نام memoryBitmap و همچنین یک نمونه از نوع Image با نام someImage و در نهایت یک مستطیل برای ترسیم تصویر در داخل آن ایجاد و مقدار دهی اولیه کرده اید. فراخوانی متد base.OnPaint در انتهای متد OnPaint باعث خواهد شد که آن چه در رویداد Paint از پیش از این ثبت شده است پس از نمایش محتویات توسط متد OnPaint شما، روی آن ها نمایش داده خواهد شد.

برای اضافه کردن امکانات طراحی به کنترل سفارشی تان، باید یک فایل xmta. - که نوع خاصی از فایل XML است – ایجاد نمایید. ویژوال استودیو ۲۰۰۸ با استفاده از IntelliSense (تایپ هوشمند) این امکان را به شما می دهد که یک فایل مشخصات زمان طراحی (Design-Time Attribute File) به پروژه ی تان از داخل سولوشن اکسپلورر اضافه نمایید. قطعه کد زیر یک نمونه از این نوع فایل را نشان می دهد یک مشخصه برای کنترل سفارشی شما با یک رده بندی موضوعی ایجاد می کند و در پنجره ی مشخصات ویژوال استودیو ۲۰۰۸ همراه با یک متن راهنما نشان می دهد.

```
<?xml version="1.0" encoding="utf-16"?>
<classes xmlns="http://schemas.microsoft.com/VisualStudio/2004/03/SmartDevices/XMTA.xsd">
  <class name="Compass.Compass">
    <property name="Heading">
      <category>CompassSpecific</category>
      <description>Sets the travelling speed in mph</description>
    </property>
  </class>
</classes>
```

استفاده از کنترل سفارشی داخل یک برنامه

پس از ایجاد یک کنترل سفارشی آن را داخل یک برنامه استفاده می کنید. اگر شما امکانات طراحی را برای کنترل سفارشی تان به درستی ایجاد کرده باشید و کنترل سفارشی را به جعبه ابزار ویژوال استودیو اضافه کرده باشید، قرار دادن آن در واسط کاربری برنامه تان همانند کنترل های استاندارد خواهد بود. در شکل ۴ واسط کاربر یک برنامه ی قطب نمای الکترونیکی را می بینید که از یک کنترل سفارشی Compass Control استفاده کرده است. همان طور که می بینید امکانات کامل طراحی همراه با تعدادی مشخصه برای این کنترل در دسترس قرار دارد. به علاوه می بینید که همه ی مشخصه ها مقدار پیش فرض و متن راهنما در باره ی نحوه ی استفاده از آن مشخصه دارند. برای ساختن این واسط کاربر باید یک «کنترل قطب نما» را داخل MainForm بیندازید و یک کنترل منو (Menu) هم برای فعال / غیر فعال سازی دریافت داده ها از گیرنده ی داخلی GPS به برنامه افزوده شده است.



شکل ۴: نرم افزار قطب نمای الکترونیکی که یک کنترل سفارشی را استفاده می کند

بعد از این که برنامه اطلاعات محل جغرافیایی GPS را دریافت کرد، جهت و سرعت حرکت کاربر در صفحه ی «کنترل قطب نما» نمایش داده می شود. برای به روز رسانی این داده ها «کنترل قطب نما» تعدادی مشخصه در اختیار برنامه نویس قرار داده است که از داخل برنامه قابل مقدار دهی هستند و مقادیری را که از گیرنده ی GPS خوانده می شوند، نشان می دهند.

افزدون اطلاعات مربوط به محل جغرافیایی به برنامه

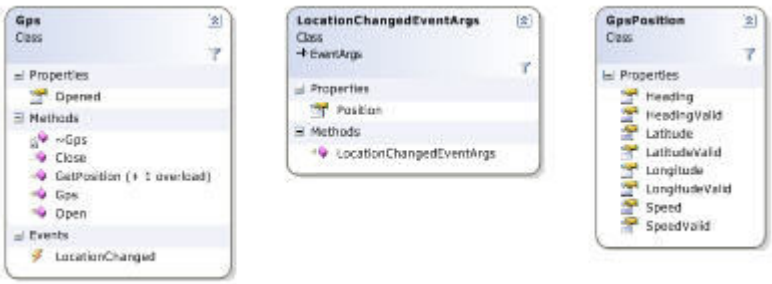
دستگاه های ویندوز موبایل ۵ و ۶ دارای یک راه انداز GPS (GPS Intermediate Driver=GPSID) هستند که برای شما به عنوان یک توسعه دهنده، دریافت اطلاعات مکان جغرافیایی از طریق GPS و همچنین به اشتراک گذاری سخت افزار آن بین برنامه های مختلف بسیار آسان خواهد بود. با گسترش دستگاه های مجهز به GPS لازم است که اطلاعاتی راجع به تولید نرم افزار بر مبنای آن داشته باشید. برای به دست آوردن اطلاعاتی در زمینه راه انداز GPS میکروسافت به آدرس زیر مراجعه نمایید:

http://msdn.microsoft.com/en-us/library/ms850332.aspx

ابزار های توسعه ی ویندوز موبایل ۵ و ۶ تعداد زیادی کد برنامه ی نمونه برای GPS دارند. یکی از این کد ها صرفا استفاده از GPSID نیست بلکه یک کلاس مدیریتی از کارکرد های GPSID را در اختیار تان قرار خواهد داد. این مثال ها را می توانید در پوشه های زیر پیدا کنید:

<Installation Folder>\<Windows Mobile SDK>\Samples\PocketPC\Cs\Gps
<Installation Folder>\<Windows Mobile SDK>\Samples\Smartphone\Cs\Gps

پیش از استفاده از این شما باید ابتدا یک سولوشن GPS درست کنید که در یکی از پوشه های فوق، ذخیره شده است. نخست ویژوال استودیو ۲۰۰۸ از شما راجع به تبدیل آن به نگارش جدید خواهد پرسید چون این کدهای نمونه در ویژوال استودیو ۲۰۰۵ نوشته شده اند و بدون تبدیل به نگارش جدید کار نخواهند کرد. پس از تبدیل می توانید سولوشن را درست کنید و از امکانات فایل اسمبلی Microsoft.WindowsMobile.Samples.Location استفاده نمایید. بدین منظور باید این اسمبلی را در سولوشن خودتان وارد (import) کنید. یک مرجع به کلاس GPSID را به سولوشن تان بیفزایید تا استفاده از آن در برنامه تان امکان پذیر گردد. دیاگرام داده ها و متد های مرتبط با کلاس استفاده شده در این مقاله در شکل ۵ نمایش داده شده است.



شکل ۵: دیاگرام کلاس GPSID

کلاس Gps نقطه ی آغاز کار شما با سخت افزار آن است. در این کلاس متدهایی برای دریافت اطلاعات مکان جغرافیایی GPS به صورت بلادرنگ و هماهنگ وجود دارد و در ضمن یک رویداد که به هنگام تغییر اطلاعات مکان، فراخوانی می شود. متدهایی نیز برای فعال/غیر فعال سازی سخت افزار GPS هست.

پیش از دریافت اطلاعات مکانی از گیرنده ی GPS نخست لازم است که یک شیء جدید از نوع Gps ایجاد و متد Open را برای آن فراخوانی کنید. این کار باعث می شود که اگر برنامه ی شما اولین استفاده کننده ی سخت افزار GPS روی این دستگاه است، سخت افزار GPS فعال گردد. پس از پایان استفاده از آن، اگر برنامه ی شما آخرین استفاده کننده ی GPS روی دستگاه است باید متد Close را برای شیء GPS فراخوانی کنیم تا سخت افزار GPS را غیرفعال کند. این کار باعث می شود که به هنگام نیاز نداشتن به GPS باتری دستگاه بیهوده مصرف نگردد.

قطعه کد زیر چگونگی فعال/غیر فعال سازی و اتصال به سخت افزار GPS را نشان می دهد و در مدیر رویداد های جداگانه اجرا می شود:

```
private void menuEnableGPS_Click(object sender, EventArgs e)
{
    gps.Open();
    gps.LocationChanged +=
      new LocationChangedEventHandler(gps_LocationChanged);
    menuDisableGPS.Enabled = true;
    menuEnableGPS.Enabled = false;
}
private void menuDisableGPS_Click(object sender, EventArgs e)
{
    gps.LocationChanged -= gps_LocationChanged;
    gps.Close();
    menuEnableGPS.Enabled = true;
    menuDisableGPS.Enabled = false;
}
```

اگر کاربر بخواهد برنامه را ببندد باز هم باید اتصال GPS را قطع کنید به همین دلیل باید در مدیر رویداد Closing (بستن برنامه) بررسی کنید که اگر هنوز GPS فعال است آن را غیر فعال نمایید و از فهرست رویداد LocationChanged خارج شوید (unsubscribe) و متد Close از شیء Gps را فراخوانی کنید. کار اصلی برنامه ی قطب نمای الکترونیکی در مدیر رویداد gps_LocationChanged انجام می شود. هر برنامه ای می تواند با استفاده از مدیر رویداد در آن ثبت نام (subscribe) کند. هر زمان که اطلاعات مکانی GPS تغییر کند، مدیر رویداد برنامه برای کار روی این تغییرات فراخوانی می شود. ثبت نام در رویداد LocationChanged تضمین می کند که برنامه ی شما در هر لحظه به آخرین اطلاعات مکانی دسترسی دارد، البته با این فرض که سخت افزار GPS اطلاعات ماهواره ای را به درستی دریافت می کند.

قطعه کد زیر مدیر رویداد gps_LocationChanged و نسخه ی اصلی آن را نشان می دهد:

```
void gps_LocationChanged(object sender, LocationChangedEventArgs args)
{
     GpsPosition pos = args.Position;
     compass1.HeadingValid = pos.HeadingValid;
     if (pos.HeadingValid)
     {
         compass1.Heading = pos.Heading;
     }
     compass1.SpeedValid = pos.SpeedValid;
     if (pos.SpeedValid)
     {
         compass1.Speed = pos.Speed * 1.152;    // convert knots to MPH
     }
     if (pos.HeadingValid | pos.SpeedValid)
     {
         compass1.Invalidate();
     }
}
```

با این که مشخصه های زیادی برای کلاس Gps هست، شما در این مدیر رویداد فقط مشخصه های Heading (جهت) و Speed (سرعت) را می بینید. چون نرم افزار صرفا یک قطب نمای الکترونیکی است، به همین خاطر مشخصه هایی مانند Lattitude (عرض جغرافیایی) و Longitude (طول جغرافیایی) مهم نیستند. از آن جا که اطلاعات دریافتی از ماهواره ممکن است به درستی خوانده نشوند، لازم است که درست

بودن مقادیر مشخصه های مورد نظر تان را بررسی کنید. در نهایت باید متد Invalidate فراخوانی کنید تا اگر حداقل یکی از داده های خوانده شده تغییر کرده باشد، «کنترل قطب نما» را به روز نماید.

اگر با کد بالا –که برای مدیر رویداد gps_LocationChanged نوشته اید- برنامه تان را کامپایل و اجرا نمایید با خطای زمان اجرا مواجه خواهید شد. مفهوم پیغام خطا این است که شما نمی توانید داده های «کنترل قطب نما» را به روز کنید. متن زیر را در متن پیغام خواهید دید:

Control.Invoke must be used to interact with controls created on a separate thread

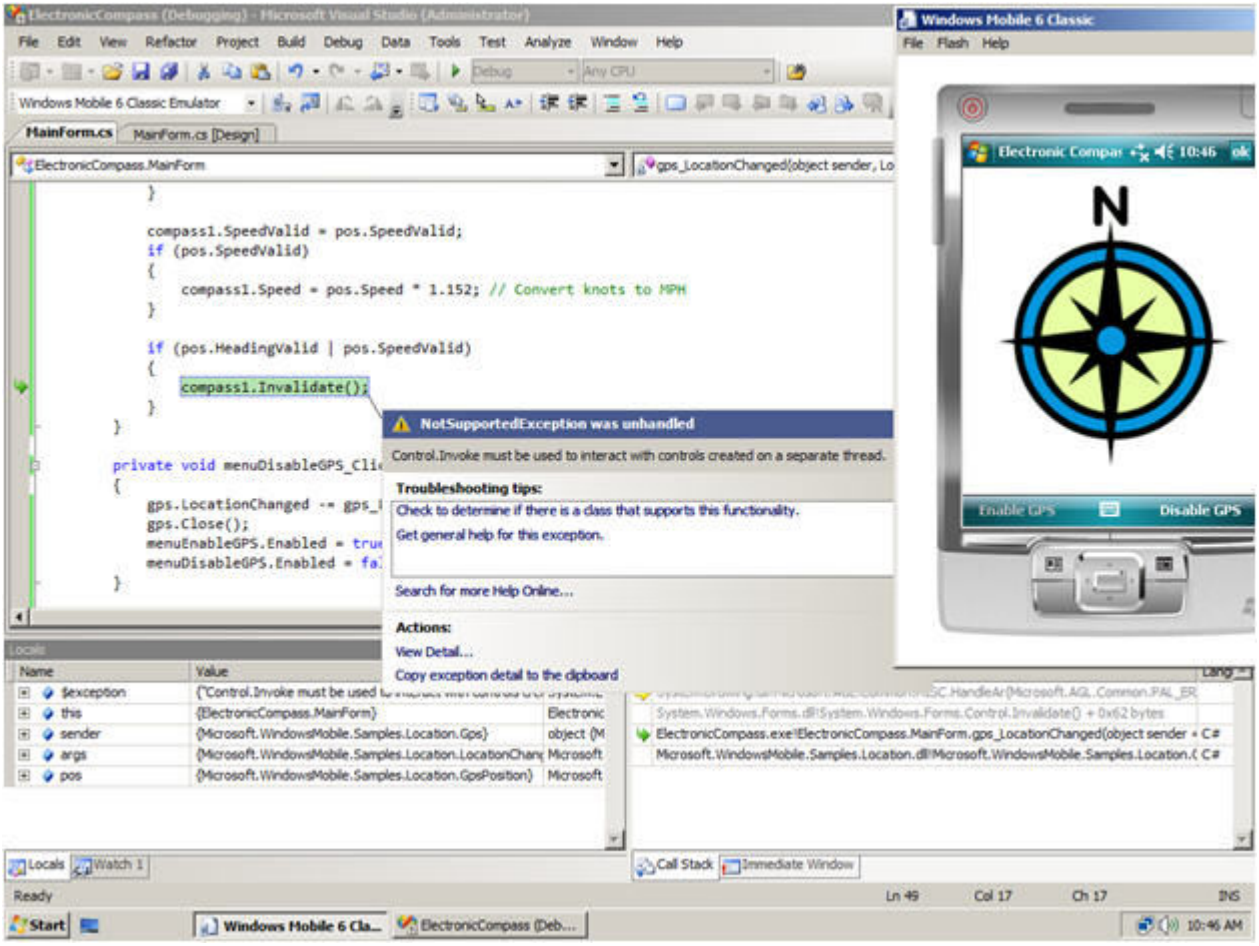
Control.Invoke باید با اشیایی در ارتباط باشد که در پردازش های مجزا ایجاد شده اند.

بازبینی پشته، اطلاعات زیر را نشان خواهد داد:

StackTrace:

```
   at Microsoft.AGL.Common.MISC.HandleAr(PAL_ERROR ar)
   at System.Windows.Forms.Control.Invalidate()
   at ElectronicCompass.MainForm.gps_LocationChanged(Object sender,
      LocationChangedEventArgs args)
   at Microsoft.WindowsMobile.Samples.Location.Gps.WaitForGpsEvents()
```

متن پیغام خطا اطلاعات مهمی برای شناسایی مشکل به ما می دهد. به نظر می رسد که چند پردازش هم زمان در برنامه ی شما اجرا می شوند، در حالی که شما فقط یک پردازش ایجاد کرده اید. اگر به Stack Trace دقت کنید متوجه می شوید که متد Invalidate –که خطا در آن رخ داده است – از داخل مدیر رویداد فراخوانی شده است که آن هم توسط یک متد داخل کلاس مدیریت GPSID فراخوانی شده است. ظاهرا GPSID یا کلاسی که آن را مدیریت می کند از چند پردازش استفاده می کنند و این همان چیزی است که باید مواظبش باشید.



شکل ۶: تغییر مشخصه های کنترل قطب نما (Compass) به هنگام بروز خطا

به روز رسانی کنترل های واسط کاربر در یک برنامه ی چند پردازشی

اشتباهی که بیشتر توسعه دهنده گان نرم افزار به آن دچار می شوند این است که سعی می کنند به طور مستقیم از داخل پردازش فعال، کنترل های واسط کاربر را تغییر دهند یا به آن ها دسترسی پیدا کنند. این کار منجر به واکنش غیر منتظره می شود. در نگارش ۱.۰ NET Compact Framework. برنامه پشت سر هم قفل می شود. در نگارش ۲.۰ یا بالاتر نتیجه اندکی بهتر است. هنگامی که بخواهید یک کنترل واسط کاربر را از داخل پردازش دیگری (غیر از پردازشی که کنترل مورد نظر در داخل آن ایجاد شده است) تغییر دهید خطای NotSupportedException ایجاد می گردد. در شکل ۶ شما این مورد را در برنامه ی قطب نمای الکترونیکی می بینید.

برای رفع این مشکل، باید خودتان را ملزم به رعایت این قانون بکنید که: *«فقط پردازشی (یا متدی) که کنترل واسط کاربر را ایجاد می کند می تواند آن را تغییر دهد»* زمانی که بخواهید یک کنترل را داخل پردازش فعال تغییر دهید همیشه باید متد Control.Invoke را فراخوانی کنید. این متد یک نماینده ی مخصوص را در پردازشی که به نگهدارنده ی پنجره ی (window handle) حاوی آن کنترل دسترسی دارد (و در واقع آن را ایجاد کرده است)، فراخوانی می کند. حال می توانید مدیر رویداد gps_LocationChanged را تغییر دهید. بدین منظور نخست باید یک delegate (نماینده) تعریف کنید که بتواند یک متد را به صورت آرگومان به متد Control.Invoke ارسال کند. یک delegate یک نوع داده ی ساده است که یک نشانه (اشاره گر) از متد را تعریف می کند که می تواند توسط هر متد با اشاره گر سازگار با آن مقدار دهی شود.

```
private delegate void UpdateDelegate();
void gps_LocationChanged(object sender, LocationChangedEventArgs args)
{
    GpsPosition pos = args.Position;
    compass1.HeadingValid = pos.HeadingValid;
    if (pos.HeadingValid)
    {
        compass1.Heading = pos.Heading;
    }
    compass1.SpeedValid = pos.SpeedValid;
    if (pos.SpeedValid)
    {
        compass1.Speed = pos.Speed * 1.152;
        // convert knots to MPH before displaying
    }
    if (pos.HeadingValid | pos.SpeedValid)
    {
        compass1.Invoke((UpdateDelegate)delegate()
        {
            compass1.Invalidate();
        });
    }
}
```

داخل مدیر رویداد gps_LocationChanged می بینید که compass1.Invalidate به طور مستقیم فراخوانده نمی شود بلکه از داخل متد دیگری به نام compass1.Invoke فراخوانی می شود. اگر با بحث نماینده های بی نام (anonymous delegates) آشنا نباشید این روش برای تان نا مفهوم خواهد بود. در مدیر رویداد gps_LocationChanged این قابلیت C# 2.0 برای فراخوانی compass1.Invalidate از داخل compass1.Invoke استفاده می شود. در عین حال این امکان هم هست که متد جداگانه ای برای فراخوانی compass1.Invalidate و فراخوانی آن از داخل UpdateDelegate تعریف شود. چون کنترل Compass فقط از یک جا فراخوانی می شود، استفاده از نماینده ی بی نام باعث خواهد شد که کدنویسی کوتاه تر شود.

با این تغییرات می توانید برنامه ی قطب نمای الکترونیکی را بدون خطا اجرا و اطلاعات ماهواره ای GPS را از طریق GPSID دریافت نمایید.

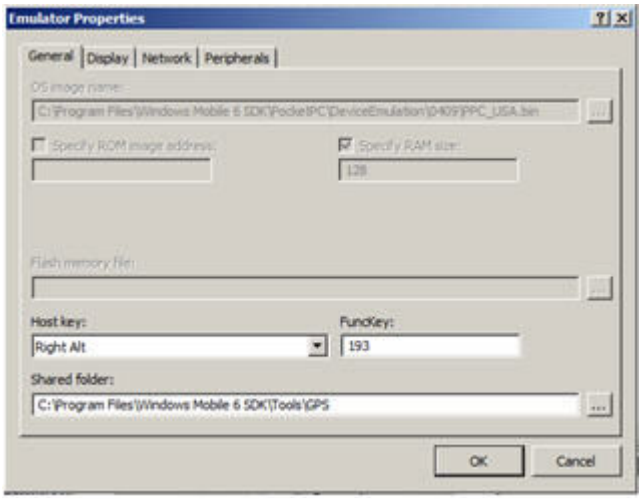
شکل ۷: قطب نمای الکترونیکی در حال اجرا و نمایش داده های GPS

حال این سؤال در ذهن تان هست که این نرم افزار الان داخل شبیه ساز دستگاه اجرا می شود ولی اطلاعات ماهواره ای GPS را دریافت می کند. پس چگونه کار می کند؟

آزمایش برنامه های حساس به محل جغرافیایی

اگر برنامه ای بنویسید که از GPS استفاده می کند آزمایش آن برنامه هم داستان دیگری است. اغلب گیرنده های GPS در فضای بسته یعنی همان جایی که شما الآن نشسته اید و برنامه می نویسید عمل نمی کنند. به علاوه اگر هم کار کنند اطلاعات دریافتی از ماهواره مربوط به یک محل است و شما هنگام توسعه و آزمایش برنامه نمی توانید حرکت کنید تا اطلاعات جدید را دریافت نمایید. برای حل این مشکل، ابزار توسعه ی ویندوز موبایل ۶ یک ابزار به نام FakeGPS دارد که فایل های داده ی متنی حاوی اطلاعات GPS را برای شبیه سازی کارکرد دستگاه GPS استفاده می نماید. برنامه هایی که از GPSID استفاده می کنند بدون نیاز به دستکاری می توانند از برنامه ی FakeGPS برای شبیه سازی حالت واقعی سخت افزار GPS استفاده نمایند. از آن جا که FakeGPS روی شبیه ساز دستگاه کار می کند شما می توانید نرم افزار های دیگر را نیز با استفاده از آن اجرا و آزمایش کنید.

برای استفاده از FakeGPS کافی است که آن را روی دستگاه یا شبیه ساز آن نصب نمایید. این برنامه به صورت یک فایل CAB می باشد. برای نصب آن روی شبیه ساز دستگاه کافی است که پوشه ی حاوی فایل FakeGPS را در پنجره ی Emulator Properties به اشتراک بگذارید تا همانند کارت حافظه ی جانبی عمل کند.



شکل ۸: اشتراک پوشه به هنگام مکان یابی شبیه سازی شده با FakeGPS

داخل شبیه ساز دستگاه می توانید با File Explorer به داخل کارت حافظه (Storage Card) بروید که به پوشه ی اشتراکی اشاره مـی کنـد. فایل CAB مربوط به FakeGPS را انتخاب و اجرا کنید تا برنامه اش در شبیه ساز نصب گردد. برنامـه ی FakeGPS همـراه بـا تعـدادی فایـل متنی حاوی اطلاعات GPS نصب خواهد شد. شما می توانید فایل های متنی خودتان را –که حاوی اطلاعـات GPS هسـتند- بـرای حالـت هـای مختلف تست برنامه تان به برنامه اضافه نمایید. یک فایل متنی FakeGPS می تواند چیزی شبیه به این باشد:

```
$GPGLL,4738.0173,N,12211.1874,W,191934.767,A*21
$GPGSA,A,3,08,27,10,28,13,19,,,,,,,2.6,1.4,2.3*3E
$GPGSV,3,1,9,8,71,307,43,27,78,59,41,3,21,47,0,10,26,283,40*77
$GPGSV,3,2,9,29,13,317,0,28,37,226,37,13,32,155,36,19,37,79,42*42
$GPGSV,3,3,9,134,0,0,0*46
$GPRMC,191934.767,A,4738.0173,N,12211.1874,W,0.109623,12.14,291004,,*21
$GPGGA,191935.767,4738.0172,N,12211.1874,W,1,06,1.4,32.9,M,-17.2,M,0.0,0000*75
$GPGLL,4738.0172,N,12211.1874,W,191935.767,A*21
$GPGSA,A,3,08,27,10,28,13,19,,,,,,,2.6,1.4,2.3*3E
$GPRMC,191935.767,A,4738.0172,N,12211.1874,W,0.081611,15.81,291004,,*2A
```

برای شروع دریافت داده ها از FakeGPS برنامه را در شبیه ساز اجرا کنید، سپس آن را فعال کنید و فایل داده ی GPS مورد نظر تان را انتخاب نمایید. در نهایت روی دکمه ی Done کلیک کنید تا ارسال داده ها به داخل GPSID آغاز گردد.

شکل ۹: انتخاب فایل داده ی GPS داخل برنامه ی FakeGPS

این همه ی آن چیزی بود که برای آزمایش برنامه های مبتنی بر GPS به آن نیاز داشتید. حال روی شبیه ساز دستگاه و هم روی دستگاه فیزیکـی (واقعی) ویندوز موبایل می توانید برنامه تان را آزمایش کنید. پس از نصب و تنظیم FakeGPS می توانیـد برنامـه ی قطـب نمـای الکترونیکـی را اجرا کنید.