

Bu sensör bir pusuladır. Hangi yöne çevirirsek onun derecesini verir. Bunu yaparken Kuzey Kutbu'nun manyetik alanının hesaplanmasından faydalanır. Compass sensörü tek bir koordinat ile çalışır.

Bu özelliği ApplicationBar içerisinde on/off düğmesine tıklayarak açıp kapatabilirsiniz. On/Off düğmesini kullanarak Compass'ı aktifleştirdiğimizde, ivmede değişiklik olma anında tetiklenecek compass\_CurrentValueChanged aksiyonunu da hazırlamış oluyoruz. Buna ek olarak kalibrasyon işlemi yapılmak istendiğinde (calibrationButton'un click anında) devreye girecek compass\_Calibrate event'i burada tanımlanıyor.

Önemli olan, yani kullanıcıya geri bildirimde bulunacak kısım ise timer\_Tick event'idir. Bu event sayfanın çalışma anında (Constructor içinde tanımlaması yapılmıştır) Timer nesnesi ile devreye alınır. Belli periyotlarda (tanımlandığı yerde 30 milisaniye olarak belirtilmiştir.)

Compass'dan aldığı bilgiyi ekrana yazdırır.

Bunu bir örnekle görelim. Yeni bir Windows Phone projesi açın ve aşağıda verilen kodları uygun yerlere yerleştirin:

C# örnek:

```
using System;
using System.Windows;
using System.Windows.Media;
using Microsoft.Phone.Controls;

using Microsoft.Devices.Sensors;
using Microsoft.Xna.Framework;
using System.Windows.Threading;

namespace sdkRawSensorDataCS
{
    public partial class CompassPage : PhoneApplicationPage
    {
        Compass compass;
        DispatcherTimer timer;
        double magneticHeading;
        double trueHeading;
        double headingAccuracy;
        Vector3 rawMagnetometerReading;
        bool isValidData;
```

```

bool calibrating = false;
Accelerometer accelerometer;
// Constructor
public CompassPage()
{
    InitializeComponent();
    Application.Current.Host.Settings.EnableFrameRateCounter = false;
    if (!Compass.IsSupported)
    {
        //compass destekleniyor mu diye kontrol ediyoruz

        statusTextBlock.Text = "Pusula desteklenmiyor";
        ApplicationBar.IsVisible = false;
    }
    else
    {
        //destekleniyorsa timer ayarlanıp başlatılıyor

        timer = new DispatcherTimer();
        timer.Interval = TimeSpan.FromMilliseconds(30);
        timer.Tick += new EventHandler(timer_Tick);
    }
}

private void ApplicationBarIconButton_Click(object sender, EventArgs e)
{
    if (compass != null && compass.IsDataValid)
    {
        compass.Stop();
        timer.Stop();
        statusTextBlock.Text = "Pusula durduruldu.";
        accelerometer.Stop();
    }
    else
    {
        if (compass == null)
        {

```

```

//Compass instance alınıyor.
compass = newCompass();

//Görüldüğü üzere kullanımı accelerometer'a çok benziyor
compass.TimeBetweenUpdates = TimeSpan.FromMilliseconds(20);
timeBetweenUpdatesTextBlock.Text =
compass.TimeBetweenUpdates.TotalMilliseconds + " ms";

//ilgili eventler oluşturuluyor, öncelikle değeri değiştikçe çalışacak event
compass.CurrentValueChanged +=
newEventHandler<SensorReadingEventArgs<CompassReading>>(compass_CurrentValueCha
nged);

//kalibrasyon talebi gelince çalıştırılacak event:
compass.Calibrate +=
newEventHandler<CalibrationEventArgs>(compass_Calibrate);
}
try
{
    statusTextBlock.Text = "Pusula başlatılıyor.";
    compass.Start();
    timer.Start();
}
catch (InvalidOperationException)
{
    statusTextBlock.Text = "Pusula başlatılamadı";
}
}
}

void compass_CurrentValueChanged(object sender,
SensorReadingEventArgs<CompassReading> e)
{
    //compass'tan gelen bilgi güncelledikçe burası çalıştırılacaktır. Data teyit edilecek
ardından sensordan gelen bilgiler ilgili değişkenlere aktarılacaktır.

    isValid = compass.IsValid;

```

```
trueHeading = e.SensorReading.TrueHeading;  
magneticHeading = e.SensorReading.MagneticHeading;  
headingAccuracy = Math.Abs(e.SensorReading.HeadingAccuracy);
```

*//sensörden vektör olarak okunan verileri yine vector tipli bir değişkende saklıyoruz.*

```
rawMagnetometerReading = e.SensorReading.MagnetometerReading;  
}
```

```
void timer_Tick(object sender, EventArgs e)
```

```
{  
    if (!calibrating)  
    {  
        if (isValidData)  
        {  
            statusTextBlock.Text = "receiving data from compass.";  
        }  
    }  
}
```

*// Compasstan okunmuş bilgiler ekrandaki textblocklara basılıyor*

```
magneticTextBlock.Text = magneticHeading.ToString("0.0");  
trueTextBlock.Text = trueHeading.ToString("0.0");  
accuracyTextBlock.Text = headingAccuracy.ToString("0.0");
```

*// Gelen bilgileri ekranda görsel olarak anlatmak için alttaki şekilde line nesneleri ile çizim yapıyoruz. Matematiksel işlemler kullanacağımızdan ötürü Math sınıfının Sin ve Cos metodlarını kullanırız. MathHelper.ToRadians yardımıyla da bir açıyı dereceden double tipine dönüştürürüz.*

```
double centerX = headingGrid.ActualWidth / 2.0;  
double centerY = headingGrid.ActualHeight / 2.0;  
magneticLine.X2 = centerX - centerY *  
Math.Sin(MathHelper.ToRadians((float)magneticHeading));  
magneticLine.Y2 = centerY - centerY *  
Math.Cos(MathHelper.ToRadians((float)magneticHeading));  
trueLine.X2 = centerX - centerY *  
Math.Sin(MathHelper.ToRadians((float>trueHeading));  
trueLine.Y2 = centerY - centerY *
```

```
Math.Cos(MathHelper.ToRadians((float>trueHeading)));
```

```
// Kompasstan gelen vektor bilgilerini ekrana basıyoruz
```

```
xTextBlock.Text = rawMagnetometerReading.X.ToString("0.00");
```

```
yTextBlock.Text = rawMagnetometerReading.Y.ToString("0.00");
```

```
zTextBlock.Text = rawMagnetometerReading.Z.ToString("0.00");
```

```
// Ve yine vektor verisinden gelen değerleri ekrana çizdiriyoruz
```

```
xLine.X2 = xLine.X1 + rawMagnetometerReading.X * 4;
```

```
yLine.X2 = yLine.X1 + rawMagnetometerReading.Y * 4;
```

```
zLine.X2 = zLine.X1 + rawMagnetometerReading.Z * 4;
```

```
}
```

```
else
```

```
{
```

```
//Kalibrasyon tamamlandıysa kullanıcıyı bilgilendiriyoruz
```

```
calibrationTextBlock.Text = "Tamamlandı!";
```

```
}
```

```
}
```

```
void compass_Calibrate(object sender, CalibrationEventArgs e)
```

```
{
```

```
//kalibrasyon bitince calibrasyonstackpanel'i gösterip ilgili değişkeni true yapıyoruz.
```

*Windows Phone üzerinde uygulama geliştirirken Composition Thread yapısından söz etmiştik.*

*UI Thread üzerinde uzun sürecek işlemler yaptırmamız gerekmektedir. Bu sebepten ötürü*

*Windows Phone uygulamalarında yoğun işlem gerektiren veya UI Thread'e karışmasını*

*istemediğimiz kodları Dispatcher.BeginInvoke yapısını kullanarak gerçekleştiririz.*

```
Dispatcher.BeginInvoke(() => { calibrationStackPanel.Visibility = Visibility.Visible; });
```

```
calibrating = true;
```

```
}
```

```
private void calibrationButton_Click(object sender, RoutedEventArgs e)
```

```
{
```

*//tekrar kalibrasyon yapmak için bu düğmeyi kullanıyoruz. Visibility.Collapsed bir nesnenin görünürlüğünü tamamen ekrandan siler. Gösterilmez ama sadece kod olarak*

eklenmiştir. Hidden ile farkı da ekrandan silinmesidir. Visibility.Hidden dendiğinde kontrol ekranda yer kaplar ama görünmez.

```
        calibrationStackPanel.Visibility = Visibility.Collapsed;
        calibrating = false;
    }
}
}
```

XAML örnek:

```
<phone:PhoneApplicationPage
x:Class="sdkRawSensorDataCS.CompassPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
FontFamily="{StaticResource PhoneFontFamilyNormal}"
FontSize="{StaticResource PhoneFontSizeNormal}"
Foreground="{StaticResource PhoneForegroundBrush}"
SupportedOrientations="Portrait" Orientation="Portrait"
mc:Ignorable="d" d:DesignHeight="696" d:DesignWidth="480"
shell:SystemTray.IsVisible="True">

<!--İçeride kullanılacak bütün kontrol ve tasarım öğeleri bu gridin içerisinde barındırılacak -->
<Grid x:Name="LayoutRoot" Background="Transparent">
<!-- Grid tanımlaması yaparken düzenli bir yapı kurabilmek için satır tanımlamaları
yapılabilir. -->
<Grid.RowDefinitions>
<!-- Auto seçildiğinde bu satırın içerisinde bulunan diğer öğelere göre yüksekliğini otomatik
olarak ayarlıyor -->
<RowDefinition Height="Auto"/>
<!-- * sembolü ile Grid içerisinde bulunan diğer satırlara bakılarak bu satırın sayfanın geri
```

kalanına yayılması sağlanıyor -->

```
<RowDefinition Height="*" />
```

```
</Grid.RowDefinitions>
```

<!-- StackPanel ile içerisine eklenen öğelerin dikey veya yatay olarak dizilimi sağlanabiliyor. -->

<!-- Grid.Row=0 diyerek LayoutRoot Grid'inin ilk satırına bu StackPanel'in yerleştirilmesi sağlanıyor -->

<!-- Margin özelliği ile StackPanel'in içinde bulunduğu Grid satırındaki çerçeveden kendisinin uzaklaştırılması sağlanıyor. Bu sayede gridin başladığı yere yapışık olmak yerine ara biraz mesafe bırakılarak kullanıcı deneyimi tasarımına uygun hale getiriliyor -->

```
<StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
```

<!-- TextBlock metinsel değerler girilebilen bir kontroldür -->

<!-- Style içerisine yazılan StaticResource ataması ile farklı bir yerde tanımlanan tasarım özellikleri çağırılıp bu TextBlock'a uygulanmaktadır. StaticResource tasarım atamalarında kullanılan genel yöntemlerden birisidir. HTML ve CSS bilginize var ise, bunu CSS olarak değerlendirebilirsiniz. -->

```
<TextBlock x:Name="ApplicationTitle" Text="SENSOR APPLICATION" Style="{StaticResource PhoneTextNormalStyle}" />
```

```
<TextBlock x:Name="PageTitle" Text="compass" Margin="9,-7,0,0" Style="{StaticResource PhoneTextTitle1Style}" />
```

```
</StackPanel>
```

<!-- İç içe grid veya stackpanel kullanılabilir. LayoutRoot gridinin ilk satırına daha önce stackpanel eklemişti. Şimdi ikinci satırına içerikleri ekleyeceğimiz yeni bir grid atıyoruz. -->

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
```

<!-- StackPanel'e daha önce söylediğimiz gibi yatay ve dikey hizalama yaptırabiliyoruz.

Yapmamız gereken sadece Orientation'ı Vertical (dikey) veya Horizontal (yatay) olarak belirlemek -->

```
<StackPanel Orientation="Vertical">
```

```
<StackPanel Orientation="Horizontal">
```

```
<TextBlock>status:</TextBlock>
```

<!-- Pusula desteklenmiyor, başlatılamadı, durduruldu, başlatıldı vb. bilgilerin yazdırılacağı alan -->

```
<TextBlock Name="statusTextBlock"></TextBlock>
```

```
</StackPanel>
```

```
<StackPanel Orientation="Horizontal">
```

```

<TextBlock>time between updates:</TextBlock>
<TextBlock Name="timeBetweenUpdatesTextBlock"></TextBlock>
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock>magnetic heading:</TextBlock>
<TextBlock Name="magneticTextBlock"></TextBlock>
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock>>true heading:</TextBlock>
<TextBlock Name="trueTextBlock"></TextBlock>
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock>heading accuracy:</TextBlock>
<TextBlock Name="accuracyTextBlock"></TextBlock>
</StackPanel>
<StackPanel Orientation="Horizontal">
<TextBlock>compass orientation mode:</TextBlock>
<TextBlock Name="orientationTextBlock"></TextBlock>
</StackPanel>
<Grid Height="200" Name="headingGrid">
<TextBlock Foreground="Yellow" FontSize="16">magnetic heading</TextBlock>
<TextBlock Foreground="Orange" FontSize="16" Margin="0,18">>true heading</TextBlock>
<!-- çizgi çizmek için kullandığımız kontrol. Çizginin kalınlığını ve rengini Stroke ve
StrokeThickness özellikleriyle tanımlayabiliyoruz. -->
<Line x:Name="magneticLine" X1="240" Y1="100" X2="240" Y2="0" Stroke="Yellow"
StrokeThickness="4"></Line>
<Line x:Name="trueLine" X1="240" Y1="100" X2="240" Y2="0" Stroke="Orange"
StrokeThickness="4"></Line>
</Grid>
<TextBlock Text="raw magnetometer data:"></TextBlock>
<Grid>
<TextBlock Height="30" HorizontalAlignment="Left" Name="xTextBlock" Text="X: 1.0"
VerticalAlignment="Top" Foreground="Red" FontWeight="Bold"/>
<TextBlock Height="30" HorizontalAlignment="Center" Name="yTextBlock" Text="Y: 1.0"
VerticalAlignment="Top" Foreground="Green" FontWeight="Bold"/>
<TextBlock Height="30" HorizontalAlignment="Right" Name="zTextBlock" Text="Z: 1.0"
VerticalAlignment="Top" Foreground="Blue" FontWeight="Bold"/>

```



```
</Grid>
<Grid Height="140">
<Line x:Name="xLine" X1="240" Y1="40" X2="240" Y2="40" Stroke="Red"
StrokeThickness="14"></Line>
<Line x:Name="yLine" X1="240" Y1="70" X2="240" Y2="70" Stroke="Green"
StrokeThickness="14"></Line>
<Line x:Name="zLine" X1="240" Y1="100" X2="240" Y2="100" Stroke="Blue"
StrokeThickness="14"></Line>
</Grid>
</StackPanel>
```

```
<!--Calibration UI-->
<StackPanel Name="calibrationStackPanel" Background="Black" Opacity="1"
Visibility="Collapsed">
<Image Source="/Images/calibrate_compass.png" Opacity=".95"
HorizontalAlignment="Center"/>
<TextBlock TextWrapping="Wrap" TextAlignment="Center">The compass on your device
needs to be calibrated.

        Hold the device in front of you and sweep it through a figure 8 pattern as shown
        until the calibration is complete.</TextBlock>
<StackPanel Orientation="Horizontal" Margin="0,10" HorizontalAlignment="Center">
<TextBlock>heading accuracy:</TextBlock>
<!-- Pusula kalibrasyonu durumunu kullanıcıya bildirmek için kullanılacak yazı alanı -->
<TextBlock Name="calibrationTextBlock">0.0°</TextBlock>
</StackPanel>
<!-- Pusula kalibrasyon sürecini başlatmak için kullanılacak Button. -->
<!-- Content alanına yazılanlar ön yüzde kullanıcıya gösterilmektedir. Yani button'un üzerinde
yazacak metindir. -->
<!-- Kontrollerin bir çok işlevsel kısımları vardır. Bunlara, kontrol ile etkileşime girildiğinde arka
planda kod blokları oluşturup fonksiyonellik katmakta dahildir. Örneğin button'un fare ile
üzerine gelip tıklanma anında Click aksiyonu tetiklenmektedir. Ve arka tarafta içinde C# kodları
barındıran kod bloğu çalışmaktadır. Bunu sağlayan ise Click event handler'ıdır. -->
<Button Name="calibrationButton" Content="Done"
Click="calibrationButton_Click"></Button>
</StackPanel>
<!--End Calibration UI-->
</Grid>
```

</Grid>

<!-- ApplicationBar ile içeride bulunan öğeler ile ilgili tekil veya çoğul işlemler yapabileceğimiz ayar tuşları barındırabilir veya uygulamanın kendisiyle ilgili genel ayarlara yönlendirmeler yapabiliriz. -->

<phone:PhoneApplicationPage.ApplicationBar>

<shell:ApplicationBar.IsVisible="True" IsMenuEnabled="True">

<shell:ApplicationBarIconButton.IconUri="/ApplicationIcon.png" Text="on/off"

Click="ApplicationBarIconButton\_Click"/>

</shell:ApplicationBar>

</phone:PhoneApplicationPage.ApplicationBar>

</phone:PhoneApplicationPage>