## Before Rx

### Subscribe/unsubscribe logic

if (\_accel != null)

{

statusTextBlock.Text = "accelerometer stopped.";

\_accel.ReportInterval = 0;

\_accel.ReadingChanged -= new TypedEventHandler<Accelerometer, AccelerometerReadingChangedEventArgs>(ReadingChanged);

}

else

{

\_accel.ReportInterval = 16;

\_accel.ReadingChanged += new TypedEventHandler<Accelerometer, AccelerometerReadingChangedEventArgs>(ReadingChanged);

}

### Displaying values

private void ReadingChanged(Accelerometer sender, AccelerometerReadingChangedEventArgs args)

{

var reading = args.Reading;

// filtering code is interwoven with display code

Vector current = new Vector(reading.AccelerationX, reading.AccelerationY, reading.AccelerationZ);

var delta = new Vector(current.X - \_previousValue.X, current.Y - \_previousValue.Y, current.Z - \_previousValue.Z);

\_previousValue = current;

if (delta.Length() > 1.0) {

Dispatcher.BeginInvoke(() => { // manually get back on the UI thread

statusTextBlock.Text = "receiving data from accelerometer.";

… // code elided

});

}

}

## After Rx

### Subscribe/unsubscribe logic

if (\_subscription != null) {

\_subscription.Dispose();

\_subscription = null;

statusTextBlock.Text = "Accelerometer Stopped";

}

else {

//var source = EmulateSensor.EmulateAccelerometer(); // can test against synthetic event source

var source = AccelerometerObservable.Instance;

\_subscription =

source

//.FindBigMovements() // optionally add extra filtering

.Sample(TimeSpan.FromMilliseconds(200)) // don’t be overresponsive

.ObserveOnDispatcher() // decide where to observe

.Subscribe(UpdateUI);

}

### Displaying values

private void UpdateUI(SensorData.Vector reading)

{

statusTextBlock.Text = "Receiving data from accelerometer...";

… // code elided

}

### Ensuring handlers are removed properly

if (\_accel != null) {

\_reportInterval = 16;

\_accelObs =

Observable.FromEventPattern<ReadingChangedHandler, AccelerometerReadingChangedEventArgs>

(subscribeEvent, unsubscribeEvent)

//.Do(l => Debug.WriteLine("Publishing {0}", l.EventArgs.Reading.Timestamp)) //side effect to show it is running

.Select(x => ToVector(x.EventArgs.Reading))

.Publish()

.RefCount(); // ensure that event handlers are removed when there are no more observers

}

else {

\_accelObs = Observable.Empty<Vector>();

}