

INotifyPropertyChanged-Logik automatisiert testen

Zauberwort

DataBinding ist eine tolle Sache: Objekt an Formular binden und wie von Zauberhand stellen die Controls die Eigenschaftswerte des Objekts dar. DataBinding ist aber auch knifflig. Stefan, kannst du dazu eine Aufgabe stellen?

Databinding ist beliebt. Lästig daran ist: Man muss die *INotifyPropertyChanged*-Schnittstelle implementieren. Sie fordert, dass bei Änderungen an den Eigenschaften eines Objekts das Ereignis *PropertyChanged* ausgelöst wird. Dabei muss dem Ereignis der Name der geänderten Eigenschaft als Parameter in Form einer Zeichenkette übergeben werden. Die Frage, die uns diesmal beim dotnetpro.dojo interessiert, ist: Wie kann man die Implementierung der *INotifyPropertyChanged*-Schnittstelle automatisiert testen?

Die Funktionsweise des Events für eine einzelne Eigenschaft zu prüfen ist nicht schwer. Man bindet einen Delegate an den *PropertyChanged*-Event und prüft, ob er bei Änderung der Eigenschaft aufgerufen wird. Außerdem ist zu prüfen, ob der übergebene Name der Eigenschaft korrekt ist, siehe Listing 3.

Um zu prüfen, ob der Delegate aufgerufen wurde, erhöhen Sie im Delegate beispielsweise eine Variable, die außerhalb definiert ist. Durch diesen Seiteneffekt können Sie überprüfen, ob der Event beim Ändern der Eigenschaft ausgelöst und dadurch der Delegate aufgerufen wurde. Den Namen der Eigenschaft prüfen Sie innerhalb des Delegates mit einem *Assert*.

Solche Tests für jede Eigenschaft und jede Klasse, die *INotifyPropertyChanged* implementiert, zu schreiben, wäre keine Lösung, weil Sie dabei Code wiederholen würden. Da die Eigenschaften einer Klasse per Reflection ermittelt werden können, ist es nicht schwer, den Testcode so zu verallgemeinern, dass damit alle Eigenschaften einer Klasse getestet werden können. Also lautet in diesem Monat die Aufgabe: Implementieren Sie eine Klasse zum automatisierten Testen der *INotifyPropertyChanged*-Logik. Die zu implementierende Funktionalität ist ein Werkzeug zum Testen von ViewModels. Dieses Werkzeug soll wie folgt bedient werden:

```
NotificationTester.Verify<MyViewModel>();
```

Die Klasse, die auf *INotifyPropertyChanged*-Semantik geprüft werden soll, wird als generischer Typparameter an die Methode über-

geben. Die Prüfung soll so erfolgen, dass per Reflection alle Eigenschaften der Klasse gesucht werden, die über einen Setter und Getter verfügen. Für diese Eigenschaften soll geprüft werden, ob sie bei einer Zuweisung an die Eigenschaft den *PropertyChanged*-Event auslösen und dabei den Namen der Eigenschaft korrekt übergeben. Wird der Event nicht korrekt ausgelöst, muss eine Ausnahme ausgelöst werden. Diese führt bei der Ausführung des Tests durch das Unit-Test-Framework zum Scheitern des Tests.

Damit man weiß, für welche Eigenschaft die Logik nicht korrekt implementiert ist, sollte die Ausnahme mit den notwendigen Informationen ausgestattet werden, also dem Namen der Klasse und der Eigenschaft, für die der Test fehlschlug.

In einer weiteren Ausbaustufe könnte das Werkzeug dann auch auf Klassen angewandt werden, die ebenfalls per Reflection ermittelt wurden. Fasst man beispielsweise sämtliche ViewModels in einem bestimmten Namespace zusammen, kann eine Assembly nach ViewModels durchsucht werden. Damit die so gefundenen Klassen überprüft werden können, muss es möglich sein, das Testwerkzeug auch mit einem Typ als Parameter aufzurufen:

```
NotificationTester.Verify
    (typeof(MyViewModel));
```

Im nächsten Heft finden Sie eine Lösung des Problems. Aber versuchen Sie sich zunächst selbst an der Aufgabe. **[ml]**

Listing 3

Property changed?

```
[Test]
public void Name_Property_loest_PropertyChanged_Event_korrekt_aus() {
    var kunde = new Kunde();
    var count = 0;
    kunde.PropertyChanged += (o, e) => {
        count++;
        Assert.That(e.PropertyName, Is.EqualTo("Name"));
    };
    kunde.Name = "Stefan"; Assert.That(count, Is.EqualTo(1));
}
```

Wer übt, gewinnt

In jeder dotnetpro finden Sie eine Übungsaufgabe von Stefan Lieser, die in maximal drei Stunden zu lösen sein sollte. Wer die Zeit investiert, gewinnt in jedem Fall – wenn auch keine materiellen Dinge, so doch Erfahrung und Wissen.

Es gilt:

- Falsche Lösungen gibt es nicht. Es gibt möglicherweise elegantere, kürzere oder schnellere Lösungen, aber keine falschen.
- Wichtig ist, dass Sie reflektieren, was Sie gemacht haben. Das können Sie, indem Sie Ihre Lösung mit der vergleichen, die Sie eine Ausgabe später in dotnetpro finden.

Übung macht den Meister. Also – los geht's. Aber Sie wollten doch nicht etwa sofort Visual Studio starten ...

