BASTA!

.NET, WINDOWS, VISUAL STUDIO

Thomas Claudius Huber | trivadis makes IT easier.

Fluid User Interfaces with async / await

Thomas Claudius Huber

- Principal Consultant @ trivadis makes IT easier.
  - Developer, Trainer
- Microsoft MVP für Windows Platform Development
- Schwerpunkte: WPF, XAML, WinApps, .NET

MVP Microsoft® Most Valuable Professional

www.thomasclaudiushuber.com
@thomasclaudiush

Agenda

- Basics
- Async / Await im ViewModel
- Summary

PART 1 / 3:
Basics

Asynchron in .NET 1.0

- «**Asynchronous Programming Model**» (APM)
- *BeginInvoke* und *EndInvoke*
  - *AsyncCallback* und *IAsyncResult*
- Bekannt von Streams und Sockets

Asynchron in .NET 2.0

- «**Event-based asynchronous pattern**» (EAP)
- Zu einer *[...]Async*-Methode gibt es ein *[...]Completed*-Event
- bspw. *LoadAsync* und *LoadCompleted*

Asynchron in .NET 2.0

```
private static void EventBasedAsynchronousPattern()
{
    var wc = new WebClient();
    wc.DownloadStringCompleted += (sender, eventArgs) =>
    {
        string html = eventArgs.Result;
        // ...
    };
    wc.DownloadStringAsync(
      new Uri("http://www.thomasclaudiushuber.com"));
}
```

.NET 4.0: Einführung der Task Parallel Library (TPL)

- Neuer Namespace *System.Threading.Tasks*
- Task startet neuen Thread:
  - **Task.Factory.StartNew**
- Abstraktion über klassische Threading-Klassen

Demo

PART 2 / 3:
Async / Await im ViewModel

.NET 4.5: Task-based Asynchronous Pattern (TAP)

- Baut auf Task Parallel Library (TPL) auf
- Sprachunterstützung in C# und VB.NET
  - **async** und **await**
- Verfügbar für verschiedene Microsoft-Platformen
  - Web, Desktop, WinRT, ...

Asynchrone Methoden

- Nutzen das **async** Schlüsselwort
  - enden konventionsgemäss auf “Async”
- Geben die Kontrolle an den Aufrufer zurück
- Haben üblicherweise mindestens eine “**await**”-Anweisung

Demo

Rückgabetypen asynchroner Methoden

- **void** – der Aufrufer benötigt keine Info
  - wird nur für EventHandler verwendet
- **Task** – der Aufrufer erhält Info zum Task-Zustand
  - bspw. ob der Task «completed» ist
- **Task<T>** - neben dem Task-Zustand erhält der Aufrufer den Rückgabewert vom Typ T

Demo

Asynchron vs. Multithreading

- CPU-lastiges in neuen Thread auslagern
  - Task.Run / Task.Factory.StartNew
- Für andere APIs wird nicht zwingend ein neuer Thread erstellt, sie sind jedoch asynchron:
  - bspw. WebClient-Klasse und deren DownloadStringTaskAsync-Methode

Was passiert im Hintergrund

- Await / Async sind für den Compiler
- async markiert eine Methode, die der Compiler ändert
- In der Methode wird eine Art Statemachine generiert
  - basiert auf einer TaskAwaiter-Instanz

Async und User Interfaces

- Synchrone Eventhandler blockieren das UI
- Touch gegenüber Maus/Tastatur hochempfindlich
  - Verzeiht keine Verzögerungen
- **WinRT => alles was länger als 50ms dauern könnte, ist nur asynchron verfügbar**

Ein Async ViewModel erstellen

- Hat problemlos geklappt
- Ohne weitere Änderungen einfach async / await genutzt
- What about testing? ☺

Demo

Weitere Features der Tasks

- Progress-Support
- Abbruch mit CancellationToken

.NET vs WinRT

.NET		**WinRT**
Task	⟷	IAsyncAction
Task<T>	⟷	IAsyncOperation
		IAsyncActionWithProgress
		IAsyncOperationWithProgress

PART 3 / 3: Summary

Summary

- Async / Await ist problemlos im ViewModel einsetzbar
- DelegateCommand für Testing angepasst
- Programmierung sieht weiterhin synchron aus
  - Unter der Haube die übliche Komplexität ☺

Fragen & Antworten

Contact:

Thomas Claudius Huber
@thomasclaudiush
thomas.huber@trivadis.com

www.trivadis.com
www.thomasclaudiushuber.com