BASTA!

.NET, WINDOWS, VISUAL STUDIO

Thomas Claudius Huber

Asynchrone Programmierung mit C#

Thomas Claudius Huber

- Principal Consultant @ Trivadis AG
  - Trainer, Coach, Developer, Architect
  - www.thomasclaudiushuber.com
- Spezialisiert auf WPF, XAML, WinApps
- Autor der umfassenden Handbücher zu WPF, Silverlight und Windows Store Apps

Microsoft® MVP Most Valuable Professional

Sessioninhalt

- Tasks und async & await
- Progress, Cancellation & more
- .NET vs. WinRT

Asynchron in .NET

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

Asynchron in .NET

- «**Event-based asynchronous pattern**» (EAP)
  - kam in .NET 2.0
- Zu einer *[...]Async*-Methode gibt es ein *[...]Completed*-Event
  - bspw. *LoadAsync* und *LoadCompleted*

Event-based asynchronous pattern

```
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 – Task Parallel Library

- Neuer Namespace System.Threading.Tasks
- Task lässt sich auf neuem Thread starten
  - **Task.Factory.StartNew**

Demo

Verschiedene Funktionen

- Warten bis der Task fertig ist
  - myTask.Wait();
  - Task.WaitAll(params Task[] tasks);
  - Task.WaitAny(params Task[] tasks);
- Task verzögern
  - Task.Delay (.NET 4.5)
- Task mit Action fortsetzen
  - myTask.ContinueWith(Action action);

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 etc.

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

Demo

Asynchron vs. Multithreading

- CPU-lastiges in neuen Thread auslagern
  - Task.Run / Task.Factory.StartNew
- Für andere APIs wird kein 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

Demo

Sessioninhalt

- Tasks und async & await
- Progress, Cancellation & more
- .NET vs. WinRT

Fortschritt / Progress

- Async-Methode nimmt IProgress<T> entgegen
- Aufrufer gibt eine Progress<T>-Instanz an
- Progress<T> kapselt Action als Callback

Demo

Abbruch eigener Async-Methoden

- Als Parameter ein **CancellationToken** entgegennehmen
- Die **IsCancellationRequested**-Property prüfen und die Methode verlassen
  - vorher kann Cleanup stattfinden
- Alternativ die **ThrowIfCancellationRequested**-Methode nutzen

Demo

Tasks abbrechen

- **CancellationTokenSource** erstellen
- Wert der **Token**-Property (CancellationToken) an Async-Methode übergeben
- Auf CancellationTokenSource die **Cancel**-Methode aufrufen

Demo

Exceptions

- Eine await-Anweisung kann in einen try/catch-Block gepackt werden.
- Im asynchronen Code geworfene Exceptions werden somit beim await im catch-Block abgefangen

Demo

Sessioninhalt

- Tasks und async & await
- Progress, Cancellation & more
- .NET vs. WinRT

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**

.NET vs. WinRT

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

Demo

Cancellation & Progress

- IAsyncOperationWithProgress mit AsTask-Extension-Methode in Task umwandeln
- Extension-Methode nimmt IProgress<T> und CancellationToken entgegen
- .NET-Knowhow lässt sich wiederverwenden.

Sessioninhalt

- Tasks und async & await
- Progress, Cancellation & more
- .NET vs. WinRT

Danke

Twitter: @thomasclaudiush

Homepage: www.thomasclaudiushuber.com

Mail: thomas.huber@trivadis.com

Slides/Demos:
www.thomasclaudiushuber.com/blog