.NET 4.5 og C# 5.0

Denne artikel beskriver nogle af de nye features i .NET 4.5 og C# 5.0.

Den forudsætter et vist kendskab til .NET og C#.

Skrevet den **11. okt 2012** af **arne_v** I kategorien **Programmering / C#** | ⭐⭐⭐⭐⭐

Historie:
V1.0 - 26/08/2012 - original
V1.1 - fix typo i versions numre

Versions numre

Nu er der igen kaos i versions numrene.

C# går fra 4.0 til 5.0

.NET library går fra 4.0 til 4.5

.NET runtime forbliver på 4.0

Hvornår

.NET 4.5 (og Visual Studio 2012) blev releaset 15. augusr 2012.

Generelt

Den her opdatering er anderledes end de tidligere opdateringer.

På nogen måder er det den største opdatering nogensinde i .NET og C#.

På andre måder er det ikke så store ændringer.

Kompabilitet

Først en lille note.

Et system med kun .NET 4.0 runtime (Windows 8 eller 2012) vil som default ikke køre .NET 2.0 (og 3.5) programmer.

Enten skal der installeres 3.5 (hvilket vil blive foreslået) eller der skal tilføjes noget i app.config:

```
<configuration>
  <startup>
    <supportedRuntime version="v2.0.50727" />
    <supportedRuntime version="v4.0" />
  </startup>
```

```
</configuration>
```

eller

```
<configuration>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0" />
  </startup>
</configuration>
```

Det sidste skulle være godt for noget COM og/eller mixed mode C++ - jeg har ikke sat mig ind i detaljerne.

Store objekter

Hidtil har der været en max. størrelse for objekter på 2 GB.

Det kan nu ændres via konfiguration med .NET 4.5 for 64 bit Windows.

app.config:

```
<configuration>
    <runtime>
       <gcAllowVeryLargeObjects enabled="true" />
    </runtime>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
</configuration>
```

Bemærk at:
- 2^31-1 er stadig limit for antal elementer i array
- max. længde af string er uændret

async & await

Langt den største ændring i C# 5.0 er introduktionen af async og await.

Og selvom ideen i asynkrone kald er simpel, så er disse to keywords lidt tricky at forstå.

Og det bliver ikke bedre af at de valgte keywords er ret misvisende:

async i sig selv gør ikke en metode asynkron - den gør bare at man kan bruge await

await venter ikke - tværtimod gør den noget asynkron

Så for at udføre noget asynkront i en metode er proceduren:
1) put async på metoden for at kunne bruge await
2) put await på den linie som man vil udføre asynkront fra

C# compileren vil så generere kode hvor den awaitede linie og de efterfølgende linier udføres i en anden tråd mens metoden returnerer til kalder med det samme.

Lad os illustrere med nogle eksempeler.

Først en helt normal synkron version:

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Asy1
{
    public class Program
    {
        public static void Test()
        {
            Thread.Sleep(1000);
            Console.WriteLine("Thread {0} done at {1}",
Thread.CurrentThread.ManagedThreadId, DateTime.Now.ToString("HH:mm:ss.fff"));
        }
        public static void Main(string[] args)
        {
            Test();
            Test();
            Test();
            Console.WriteLine("Calls completed at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
            Thread.Sleep(3100);
            Console.WriteLine("Main done at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
        }
    }
}
```

Output ser som forventet ud som:

```
Thread 1 done at 11:31:36.757
Thread 1 done at 11:31:37.769
Thread 1 done at 11:31:38.769
Calls completed at 11:31:38.769
Main done at 11:31:41.869
```

Lad os så prøve med async og await:

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Asy2
{
    public class Program
    {
        public async static Task Test()
        {
            await Task.Delay(1000);
            Console.WriteLine("Thread {0} done at {1}",
Thread.CurrentThread.ManagedThreadId, DateTime.Now.ToString("HH:mm:ss.fff"));
        }
        public static void Main(string[] args)
        {
            Test();
            Test();
            Test();
            Console.WriteLine("Calls completed at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
            Thread.Sleep(3100);
            Console.WriteLine("Main done at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
        }
    }
}
```

Output ser nu ud som:

```
Calls completed at 11:37:03.154
Thread 4 done at 11:37:04.157
Thread 6 done at 11:37:04.178
Thread 5 done at 11:37:04.178
Main done at 11:37:06.265
```

Hvor vi ser at:
- alle 3 Test kald er returneret inden de faktisk udføres
- de udføres i forskellige tråde

Vi noterer også at:
* metoden skal returnere Task fremfor void
* der bruges Task.delay fremfor Thread.Sleep da sidstnævnte ikke understøtter await

Men nogen gange er det jo nødvendigt at vente på noget asynkront er færdigt. Det kan gøres via det returnerede Task objekt:

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Asy3
{
    public class Program
    {
        public async static Task Test()
        {
            await Task.Delay(1000);
            Console.WriteLine("Thread {0} done at {1}",
Thread.CurrentThread.ManagedThreadId, DateTime.Now.ToString("HH:mm:ss.fff"));
        }
        public static void Main(string[] args)
        {
            Task t1 = Test();
            t1.Wait();
            Task t2 = Test();
            t2.Wait();
            Task t3 = Test();
            t3.Wait();
            Console.WriteLine("Calls completed at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
            Thread.Sleep(3100);
            Console.WriteLine("Main done at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
        }
    }
}
```

Nu ser output ud som:

```
Thread 4 done at 11:42:08.110
Thread 4 done at 11:42:09.123
Thread 4 done at 11:42:10.137
Calls completed at 11:42:10.137
Main done at 11:42:13.237
```

Men bemærk at man kan sagtens udføre en masse kode mellem await og Wait kaldet.

Wait er vigtigt når man skal have returneret en værdi.

Første forsøg:

```
using System;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace Asy4
{
    public class Program
    {
        public async static Task<int> Test()
        {
            await Task.Delay(1000);
            Console.WriteLine("Thread {0} done at {1}",
Thread.CurrentThread.ManagedThreadId, DateTime.Now.ToString("HH:mm:ss.fff"));
            return 123;
        }
        public static void Main(string[] args)
        {
            int v1 = Test().Result;
            int v2 = Test().Result;
            int v3 = Test().Result;
            Console.WriteLine("Calls completed at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
            Console.WriteLine("{0} {1} {2}", v1, v2, v3);
            Thread.Sleep(3100);
            Console.WriteLine("Main done at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
        }
    }
}
```

Giver nemlig:

```
Thread 4 done at 11:45:19.336
Thread 4 done at 11:45:20.351
Thread 4 done at 11:45:21.363
Calls completed at 11:45:21.363
123 123 123
Main done at 11:45:24.463
```

Result laver simpelthen en Wait.

Så derfor:

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Asy5
{
    public class Program
    {
        public async static Task<int> Test()
        {
```

```
            await Task.Delay(1000);
            Console.WriteLine("Thread {0} done at {1}",
Thread.CurrentThread.ManagedThreadId, DateTime.Now.ToString("HH:mm:ss.fff"));
            return 123;
        }
        public static void Main(string[] args)
        {
            Task<int> t1 = Test();
            Task<int> t2 = Test();
            Task<int> t3 = Test();
            Console.WriteLine("Calls completed at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
            int v1 = t1.Result;
            int v2 = t2.Result;
            int v3 = t3.Result;
            Console.WriteLine("{0} {1} {2}", v1, v2, v3);
            Thread.Sleep(3100);
            Console.WriteLine("Main done at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
        }
    }
}
```

Som giver det ønskede:

```
Calls completed at 11:47:13.962
Thread 4 done at 11:47:14.963
Thread 5 done at 11:47:14.964
Thread 6 done at 11:47:14.964
123 123 123
Main done at 11:47:18.064
```

For at vise lidt om hvad der sker behind the scene prøver vi nu at lave det samme uden async og await:

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Asy6
{
    public class Program
    {
        public static Task Test()
        {
            return Task.Run( () => {
                Thread.Sleep(1000);
                Console.WriteLine("Thread {0} done at {1}",
Thread.CurrentThread.ManagedThreadId, DateTime.Now.ToString("HH:mm:ss.fff"));
```

```
            });
        }
        public static void Main(string[] args)
        {
            Test();
            Test();
            Test();
            Console.WriteLine("Calls completed at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
            Thread.Sleep(3100);
            Console.WriteLine("Main done at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
        }
    }
}
```

og:

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace Asy7
{
    public class Program
    {
        public static Task Test()
        {
            TaskCompletionSource<string> tcs = new
TaskCompletionSource<string>();
            Timer t = new Timer(cb => {
                Console.WriteLine("Thread {0} done at {1}",
Thread.CurrentThread.ManagedThreadId, DateTime.Now.ToString("HH:mm:ss.fff"));
                tcs.SetResult("not used");
            });
            t.Change(1000, -1);
            return tcs.Task;
        }
        public static void Main(string[] args)
        {
            Test();
            Test();
            Test();
            Console.WriteLine("Calls completed at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
            Thread.Sleep(3100);
            Console.WriteLine("Main done at {0}",
DateTime.Now.ToString("HH:mm:ss.fff"));
        }
    }
}
```

Udover compiler supporten har Microsoft:
- dokumenteret en standard for brug kaldet TAP (Task-based Asynchronous Pattern)
- implementeret TAP rigtigt mange steder i Frameworket

Der er bl.a. en navne konvention om at asybc metoder har et suffix Async.

Eksempel med HTTP:

```
using System;
using System.Net;
using System.Threading.Tasks;

namespace AsyIO1
{
    public class Program
    {
        public static void Main(string[] args)
        {
            WebClient wc = new WebClient();
            for(int i = 0; i < 3; i++)
            {
                DateTime dt1 = DateTime.Now;
                string google = wc.DownloadString("http://www.google.com/");
                DateTime dt2 = DateTime.Now;
                Console.WriteLine((int)(dt2 - dt1).TotalMilliseconds);
                DateTime dt3 = DateTime.Now;
                Task<string> th =
wc.DownloadStringTaskAsync("http://www.google.com/");
                DateTime dt4 = DateTime.Now;
                string asygoogle = th.Result;
                DateTime dt5 = DateTime.Now;
                Console.WriteLine((int)(dt4 - dt3).TotalMilliseconds + " + " +
(int)(dt5 - dt4).TotalMilliseconds);
            }
        }
    }
}
```

Output:

```
3230
10 + 231
158
0 + 224
154
0 + 159
```

Vi ser at det er Result der tager tiden ikke DownloadStringTaskAsync som returnerer med det samme.

Eksempel med StreamReader:

```
using System;
using System.IO;
using System.Threading.Tasks;

namespace AsyIO2
{
    public class Program
    {
        public static void Main(string[] args)
        {
            using(StreamWriter sw = new StreamWriter(@"C:\work\big.txt") )
            {
                for(int i = 0; i < 10000000; i++)
                {
                    sw.WriteLine("12345678");
                }
            }
            for(int i = 0; i < 3; i++)
            {
                StreamReader sr = new StreamReader(@"C:\work\big.txt");
                DateTime dt1 = DateTime.Now;
                string src = sr.ReadToEnd();
                DateTime dt2 = DateTime.Now;
                sr.Close();
                Console.WriteLine((int)(dt2 - dt1).TotalMilliseconds);
                StreamReader asysr = new StreamReader(@"C:\work\big.txt");
                DateTime dt3 = DateTime.Now;
                Task<string> th = asysr.ReadToEndAsync();
                DateTime dt4 = DateTime.Now;
                string asysrc = th.Result;
                DateTime dt5 = DateTime.Now;
                asysr.Close();
                Console.WriteLine((int)(dt4 - dt3).TotalMilliseconds + " + " +
(int)(dt5 - dt4).TotalMilliseconds);
            }
        }
    }
}
```

Output:

```
476
79 + 6266
511
0 + 6200
533
```

```
0 + 6191
```

Igen er det Result og ikke ReadToEndAsync der tager tiden.

Ny HttpClient klasse

Den nye HttpClient klasse bruger TAP.

Eksempel:

```
using System;
using System.Net.Http;

namespace E
{
    public class Program
    {
        public static void Main(string[] args)
        {
            HttpClient hc = new HttpClient();
            String google =
hc.GetStringAsync("http://www.google.com//").Result;
            Console.WriteLine(google);
        }
    }
}
```

Her bruges der så ikke await, men det kan man bruge.

WinRT og Modern UI (Metro UI)

En anden stor ting i .NET 4.5 er at der understøttes WinRT (og det nye UI framework som er kendt som Metro men kommer til at hedde Modern).

WinRT (Windows Runtime) er et nyt framework i Windows 8 og 2012. Det er ikke det samme som Windows RT der er en tablet version af Windows 8.

WinRT er native (unmanaged code).

WinRT er COM baseret. WinRT komponenter implementerer IUnknown som alle COM komponenter gør og IInspectable som er specifik for WinRT komponenter. De implementerer ikke IDispatch (som bruges af VBS og JScript).

Meta data gemmes ikke i en .tlb fil som i gamle dage, men i en .winmd fil, som bruger samme format som .NET komponenter.

.NET kode kan tilgå WinRT komponenter ligesom .NET komponenter og frameworket sørger for den nødvendige glue code.

WinRT bruges af de såkalder Modern UI apps.

Modern UI apps og traditionelle desktop apps er to vidt forskellige programmerings modeller.

Illustrativ figur:
http://i.stack.imgur.com/y7uoI.png

Men skiftet fra C# WPF&SL til C# WinRT skulle være meget nemt. Der er nogle tips her:
http://msdn.microsoft.com/en-us/library/windows/apps/xaml/br229571.aspx

Andet

Der er mange andre nye features, men dem må du selv slå op i docs.

God kode lyst.

Foregående artikler

http://www.eksperten.dk/guide/694 om .NET 2.0 og C# 2.0
http://www.eksperten.dk/guide/1153 on .NET 3.5 og C# 3.0
http://www.eksperten.dk/guide/1330 on .NET 4.0 og C# 4.0