


VS08
This One Goes to ||
Going Parallel with PFX, PLINQ, TPL
and Async Keywords
Brian Noyes
Chief Architect, IDesign Inc (www.idesign.net)
brian.noyes@idesign.net, @briannoyes



About Brian

Chief Architect
IDesign Inc. (www.idesign.net)

Microsoft Regional Director
(www.theregion.com)

Microsoft MVP
Silverlight

E-mail: brian.noyes@idesign.net
Twitter: @briannoyes
Blog: <http://briannoyes.net>

Publishing

Developers Guide to Microsoft Prism 4, O'Reilly & Assoc., March 2011

Developing Applications with Windows Workflow Foundation, LiveLessons training DVD, June 2007

Smart Client Deployment with ClickOnce, Addison Wesley, January 2007

Data Binding in Windows Forms 2.0, Addison Wesley, January 2006

MSDN Magazine, MSDN Online, CoDe Magazine, The Server Side .NET, asp.netPRO, Visual Studio Magazine

Speaking

Microsoft TechEd US, Europe, Malaysia, DevConnections, DevTeach, others



Agenda

- Why Go Async?
- PFX and PLINQ
- TPL Overview
- Working with Tasks
- Async Keywords



Why Go Async?

- Because you have to...
 - Silverlight
 - WinRT/Metro
- Because you can



Why Not Go Async?

- **Sequential logic is not sequential anymore**
 - Results are produced non-deterministically
 - Code structure has to be modified to accommodate that fact
- **Synchronization**
 - Cannot access variables (memory) from multiple threads concurrently
 - Some objects have thread affinity



Async Patterns

- **Async Programming Model (APM)**
 - Begin/End pattern
- **Event-based Async Pattern (EAP)**
 - Async/Completed
- **Task-based Async Pattern (TAP)**
 - Task represents a unit of work to be executed async
 - Compiler/language features allow us to program sequentially



Parallel Framework (PFX)

- Shipped in .NET 4
- Allows simple data parallelization
 - Iteration over collections
- **Parallel.For** and **Parallel.ForEach** methods

```
Parallel.ForEach(inputCollection, coll =>  
{  
    // Work to do on each item  
});
```

- Smart scheduling of the “appropriate” number of threads
 - Based on cores, target method complexity, collection size, etc
- Items in collection need to be independent



PLINQ

- **PLINQ .AsParallel** extension
 - Executes subsequent LINQ operations concurrently
 - Some LINQ operations force sequential processing
 - Ordering
- **Control over**
 - Degree of parallelism
 - Cancellation
 - Sequential iterations
 - Partitioning, etc.



Task Parallel Library

- Shipped in .NET 4
- Only available in full .NET Framework at this time
 - Silverlight 5 contains subset
- Use Task class to point to asynchronous work
 - Thread dispatching
 - Enhances thread management in pool
 - Continuations
 - Can schedule on the user interface thread
 - Exception callbacks
 - Cancellation



Task-based Async Pattern (Async CTP)

- New keywords being added to C# and VB
- Will be released with the next version of framework and tools
- Allows you to write code that looks synchronous, but executes asynchronous
 - Compile time code generation into async patterns
- Public CTP available
- Will ship as part of .NET 4.5
- Also the async programming model of WinRT



Task-based Async Pattern (Async CTP)

- Uses Task class from Task Parallel Library
- New keywords
 - await
 - async
- Naming and return type conventions
 - Task<T>, XXXAsync



Awaiting a Task

```
static async Task<byte[]> TryFetchAsync(string url)
{
    var client = new WebClient();
    try
    {
        return await client.DownloadDataTaskAsync(url);
    }
    catch (WebException) { }
    return null;
}
```



```
static Task<byte[]> TryFetchAsync(string url)
{
    var __builder = new AsyncTaskMethodBuilder<byte[]>();
    int __state = 0;
    Action __moveNext = null;
    TaskAwaiter<byte[]> __awaiter1;

    WebClient client = null;

    __moveNext = delegate
    {
        try
        {
            if (__state == 1) goto Resume1;
            client = new WebClient();
            try
            {
                __awaiter1 = client.DownloadDataTaskAsync(url).GetAwaiter();
                if (!__awaiter1.IsCompleted) {
                    __state = 1;
                    __awaiter1.OnCompleted(__moveNext);
                    return;
                }
            }
            Resume1:
            __builder.SetResult(__awaiter1.GetResult());
        }
        catch (WebException) {}
        __builder.SetResult(null);
    }
    catch (Exception exception)
    {
        __builder.SetException(exception);
    }
};

__moveNext();
return __builder.Task;
}
```

Resources

- Pause and Play with Await, Mads Torgersen, MSDN Magazine, <http://msdn.microsoft.com/en-us/magazine/hh456403.aspx>
- Patterns of Parallel Programming, Stephen Toub, <http://www.microsoft.com/download/en/details.aspx?id=19222>
- Threading in C#, Joseph Albahari, <http://www.albahari.com/threading/>
- Async CTP: <http://blogs.msdn.com/b/visualstudio/archive/2011/04/13/async-ctp-refresh.aspx>

E-mail: brian.noyes@idesign.net

Twitter: @briannoyes

Blog: <http://briannoyes.net>



Your Feedback is Important

Please fill out a session evaluation form
drop it off at the conference registration
desk.

Thank you!



SynchronizationContexts

- Dispatcher uses DispatcherSynchronizationContext under the covers
- Derived from SynchronizationContext
 - Standard thread dispatch pattern in .NET for objects with thread affinity
 - Windows Forms, ASP.NET, WPF, Workflow also have implementations
- Multithreaded mechanisms can use this to auto-dispatch to the UI thread
 - Async/Completed pattern
 - Task class from TPL
 - Reactive Extensions
 - Task-based Async Pattern (Async CTP)
 - WCF message dispatching



Reactive Extensions (Rx)

- AKA LINQ to Events
- Available now as add-ons
 - .NET Framework, Silverlight, SL for Windows Phone, JavaScript
- Allows you to treat events and async operations as collections
 - Allows use of LINQ operations to logically filter, join, order, etc



Reactive Extensions

- `IObservable<T>` / `IObserver<T>` interfaces
 - Duals of `IEnumerable<T>` / `IEnumerator<T>`
- Allows LINQ syntax
- You provide an `IObserver` implementation that handles
 - `OnNext` – the next event occurred
 - `OnCompleted` – Events or async operation is done
 - `OnError` – exception handling
- Clean API for events and async ops
- Things that are inherently a chain of asynchronously produced data that you want to consume

