"Be More Responsive with Async and Await"

Joe Hummel, PhD

Microsoft MVP Visual C++

Technical Staff: Pluralsight, LLC

Professor: U. of Illinois, Chicago

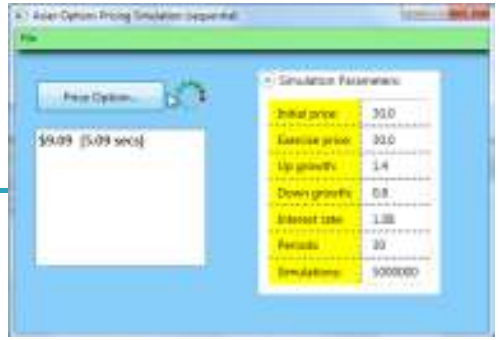
email: joe@joehummel.net

stuff: http://www.joehummel.net/downloads.html

Why use Async / Await?

- To prevent blocking on long-running operations
  - Client-side UI remains responsive vs. locking up
  - Server-side scales vs. denying service

Use-case #1



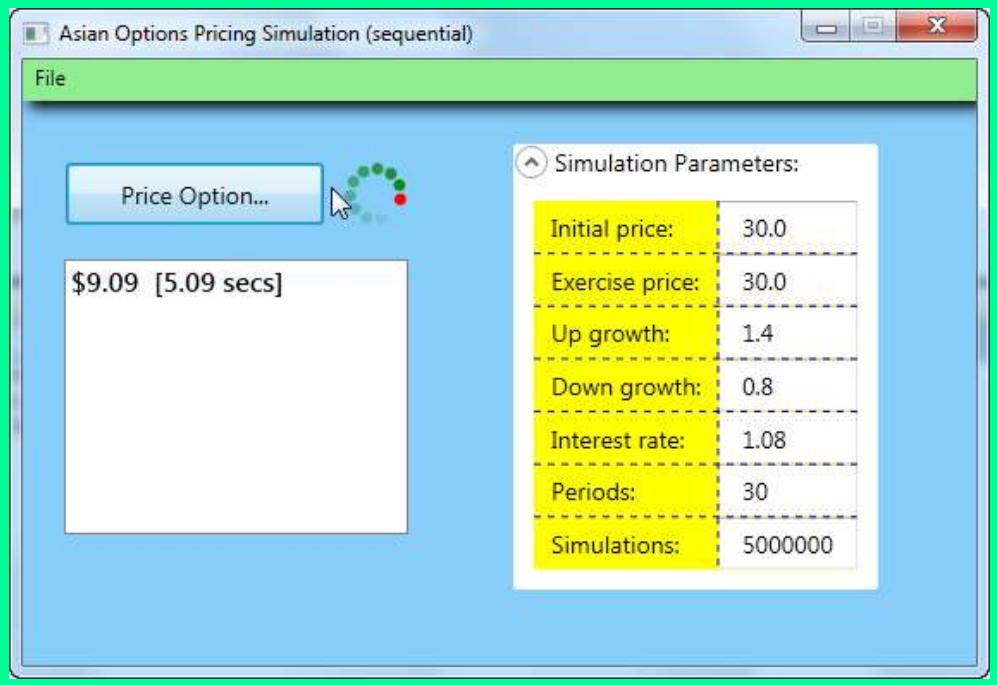
- Responsive UI…

```
void button1_Click(…)
{
   var result = DoLongLatencyOp();
   lstBox.Items.Add(result);
}
```

```
async void button1_Click(…)
{
   var result = await Task.Run(() => DoLongRunningOp());
   lstBox.Items.Add(result);
}
```

Demo — Responsiveness

- Asian options financial modeling…



Async / Await

```
async void button1_Click(…)
{
  var result = await Task.Run(() => DoLongRunningOp());
  lstBox.Items.Add(result);
}
```

*Method *may* perform async, long-latency op*

Tells compiler to setup a continuation that waits to execute rest of method on the current thread context. Meanwhile method can return while Task and continuation execute…

Observation…

- **await** must wait on a task
  - *implies underlying method must create & start a task…*

```
async void button1_Click(…)
{
  var result = await Task.Run(() => DoLongRunningOp());
  lstBox.Items.Add(result);
}
```

```
System.IO.FileStream  file = ...;

int read = await file.ReadAsync(buffer, offset, count);
```

Using Async / Await…

- Think chunky, not chatty
  - *i.e. designed for coarse-grain, long-latency operations*
  - *file I/O, network I/O, compute-bound work…*

- Async / await is *not* for
  - *Fire-and-forget (use a Task)*
  - *High-performance parallelism (use Parallel.For)*

Using Async / Await…

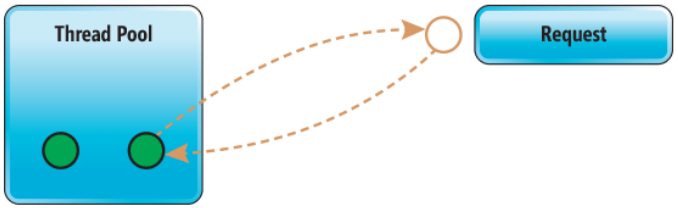
- Use with APIs exposing “TAP” pattern
  - *Async / await take advantage of Task-based Asynchronous Pattern*

	Synchronous	Asynchronous
System.IO.FileStream	`Read`	**`ReadAsync`**
	`Write`	**`WriteAsync`**
	`CopyTo`	**`CopyToAsync`**
System.Net.HttpWebRequest	`GetResponse`	**`GetResponseAsync`**
	`GetRequestStream`	**`GetRequestStreamAsync`**
System.Net.Http.HttpClient	*N/A*	**`GetAsync`**
	N/A	**`PostAsync,PutAsync,SendAsync`**

```
System.IO.FileStream  file = ...;

int read = await file.ReadAsync(buffer, offset, count);
```

Use-case #2



- Scalable server-side…

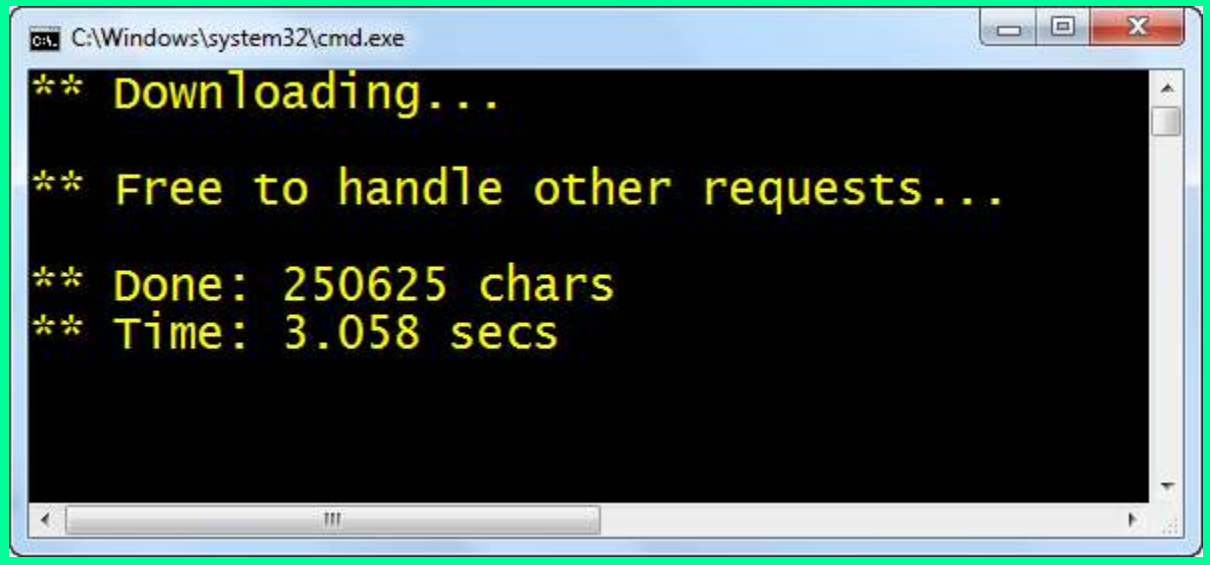
```
string GetData(string url)
{
  var client = new WebClient();
  var page   = client.DownloadString(url);
  return page;
}
```

"asynchrony" bubbles up to caller…

```
async Task<string> GetDataAsync(string url)
{
  var client = new WebClient();
  var page   = await client.DownloadStringTaskAsync(url);
  return page;
}
```

Demo

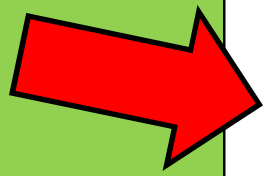
- Asynchronous web requests…



Making your own TAP methods

- Return a task for caller to await upon…

```
public int SomeOperation(...)
{
   int result;
   result = ...;
   return result;
}
```



```
public Task<int> SomeOperationAsync(...)
{
   return Task.Run<int>( () =>
     {
        int result;
        result = ...;
        return result;
     }
   );
}
```

That's it!

Summary: Async/Await

- Use cases:
  - ***Responsiveness***: *prevent blocking of the UI*
  - ***Scalability***: *prevent blocking of request-handling threads*

- Works with:
  - *.NET on Windows server / desktop / tablet / phone*
  - *Silverlight 4 and 5*
  - *ASP.NET server-side*

Thank you for attending!

- Presenter: Joe Hummel
  - *Email*: joe@joehummel.net
  - *Materials*: http://www.joehummel.net/downloads.html

- For more info, see these issues of MSDN magazine:
  - October 2011: series of intro articles
  - March 2013: best practices
  - October 2014: async / await on ASP.NET

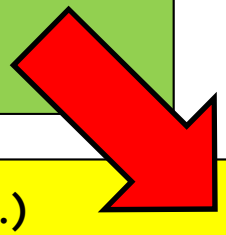
Task-based

- Programming model based on concept of a **Task**

> **Task** == *a unit of work; an object denoting an ongoing operation or computation.*

Asynchronous programming with Tasks

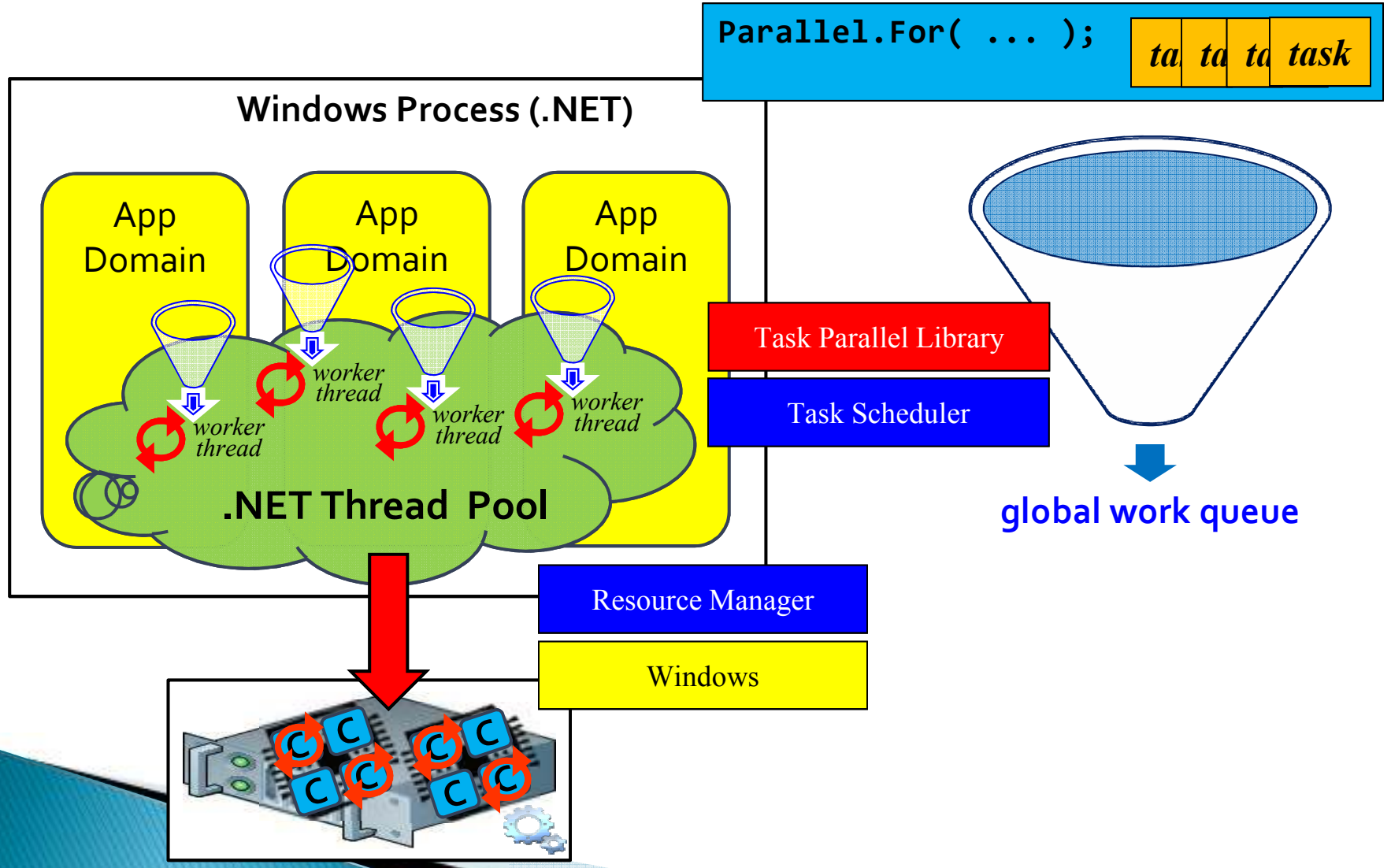
```
void button1_Click(…)
{
   var result = DoLongLatencyOp();
   lstBox.Items.Add(result);
}
```



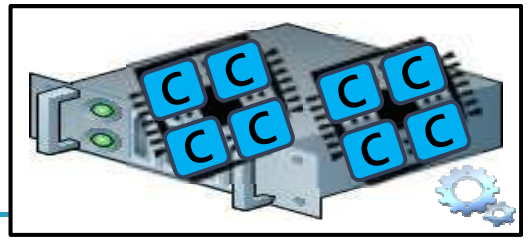
```
void button1_Click(…)
{
   var uictx =  // grab UI thread context to run UI task:
       TaskScheduler.FromCurrentSynchronizationContext();

   Task.Factory.StartNew(()=>
     {
       return DoLongLatencyOp();
     }
   ).ContinueWith((antecedent) =>
     {
       lstBox.Items.Add(antecedent.Result);
     },
     uictx  // execute this task on UI thread:
   );
}
```

Task-based execution model



Async vs. Parallel?



- Async programming:
  - *Better responsiveness…*
  - GUIs (desktop, web, mobile)
  - Cloud
  - Windows 8
- Parallel programming:
  - *Better performance…*
  - Engineering
  - Oil and Gas
  - Pharma
  - Science
  - Social media

Disk and network I/O

number crunching and big data processing