C# Language Support for Task

Ian Griffiths

http://www.interact-sw.co.uk/iangblog/

ian@interact-sw.co.uk

pluralsight
hardcore developer training

async and await

- **async**
  - Enables use of asynchronous features
  - Applied to methods
- **await**

```
void Work()
{
    Task<string> ts = Get();
    ts.ContinueWith(t =>
    {
        string result = t.Result;
        Console.WriteLine(result);
    });
}
```

```
async void Work()
{
    Task<string> ts = Get();
    string result = await ts;
    Console.WriteLine(result);
}
```

Returning Tasks

- **Allowable async method return types:**
  - void
  - Task
  - Task<T>

```
static async void ShowQuote()
{
  Quote q = await GetQuote("MSFT");
  Console.WriteLine("{0} ('{1}'): {2}",
    q.Name, q.Symbol, q.LastTrade);
}
```

Exception Handling

- **Task<T>.Result throws if faulted**
  - Throws AggregateException

- **await throws original exception**

Handling All InnerExceptions

- **Compiler-generated Tasks have at most 1 exception**
- **Task.WhenAll can have many**
  - await rethrows the 1st
  - Task's Wait, Result and Exception return everything

```
try
{
    await compositeTask.ContinueWith(() => { },
            TaskContinuationOptions.ExecuteSynchronously);
    string[] results = compositeTask.Result;
}
catch (AggregateException ax)
{
    ...
}
```

Argument Validation

- Immediate vs deferred
- Always deferred in async methods

```
try
{
  Task<string> t1 = DoAsync(p1, p2);
}
catch (Exception x)
{
                Immediate exceptions
}                  caught here

try
{
  string result = await t1;
}
catch (Exception x)
{
                Deferred exceptions
}                  caught here
```

Missing an Exception

- Double await

```
try
{
   Task<int> t1 = StartSomething();
   Task<int> t2 = StartSomethingElse();

   int r1 = await t1;
   int r2 = await t2;
   return r1 + r2;

   int[] r = await Task.WhenAll(t1, t2);
   return r[0] + r[1];

   var t = Task.WhenAll(t1, t2);
   await t.ContinueWith(x => { });
   return t.Result[0] + t.Result[1];
}
catch (Exception x)
{
   ...
}
```

- Void return type

SynchronizationContext

- await is context-aware
- Async methods ‘just work’ in the UI

Summary

- async
- await
- Return void, Task or Task<T>
- Unwrapped exceptions