Exploring Useful Methods in the Task Parallel Library

Filip Ekberg
Principal Consultant & CEO

@fekberg fekberg.com

Overview

- How to know if all the tasks in a collection all have been completed
- How to run a continuation when at least one task in a collection has completed
- Starting multiple tasks and process the result as it arrives
- Creating a task with a precomputed result
- Learn more about the execution context and controlling the continuation

Knowing When All or Any Task Completes

Task.WhenAll

```
var task1 Task.Run(() => { return "1"; });
var task2 Task.Run(() => { return "2"; });

var tasks = new [] { task1, task2 };

string[] result = await Task.WhenAll(tasks);
```

Precomputed Results of a Task

Task.CompletedTask

```
public override Task Run()
{
    return Task.CompletedTask;
}


await Run(); // Completes immediately
```

Adding **async** and **await**
when you don't need to
introduce **unnecessary**
complexity

Task.FromResult

```
public Task<IEnumerable<StockPrice>> Get(...)
{
    var stocks = new List<StockPrice>
    {
        new StockPrice { ... },
        new StockPrice { ... },
        ...
    };

    var task = Task.FromResult(stocks);
    return task;
}
```

Process Tasks as They Complete

Don’t use List<T>
for **parallel** operations it is
not thread-safe

ConcurrentBag<T>

```
var bag = new ConcurrentBag<StockPrice>();
```

Generic & thread-safe!

Execution Context and Controlling the Continuation

ConfigureAwait

```
var task = Task.Run(() => { ... });

await task.ConfigureAwait(false);
```

ConfigureAwait

```
var task = Task.Run(() => { ... });

await task.ConfigureAwait(false);
```

Configures how the
continuation will be executed

ConfigureAwait(false) could **slightly improve performance** as it **doesn't** have to **switch context**

Don’t rely on the captured context

```
var task = Task.Run(() => { ... });

await task.ConfigureAwait(false);

// No code below should require the original context
```

Thread static variables from the **original context won't be available**!

ConfigureAwait in ASP.NET Core

ASP.NET Core doesn't use a **synchronization context** which means it will not capture the context like traditional ASP.NET.

Thus, making **ConfigureAwait(false)** useless.

Library developer?

Always use ConfigureAwait(false)

Use ConfigureAwait in libraries

```
public async Task MyLibraryMethod()
{
    var task = ...;

    var result = await task.ConfigureAwait(false);

    // Won't go back to the original thread
    // when handling the result
}
```

Summary

- How to best use the Task Parallel Library
- Configure the continuation
- Start multiple asynchronous operations that execute in parallel
- Use Task.WhenAll and Task.WhenAny
- Construct a pre-computed result with Task.FromResult
- When a pre-computed result is necessary
- Processing Tasks as they complete
- Using the ConcurrentBag<T>
- Controlling the continuation with ConfigureAwait

You're now **ready** to **learn** about the **advance topics**!