Async Await Best Practices Cheat Sheet

Updated 2 years ago

Asynchronous Programming Guidelines.

Markdown | async, bestpractices, cheatsheets, csharp | Stephen Cleary | Raw

```
# Async Await Best Practices Cheat Sheet

## Summary of Asynchronous Programming Guidelines

|        Name       |                    Description                    |           Exceptions          |
|-------------------|---------------------------------------------------|-------------------------------|
| Avoid async void  | Prefer async Task methods over async void methods | Event handlers                |
| Async all the way | Don't mix blocking and async code                 | Console main method           |
| Configure context | Use `ConfigureAwait(false)` when you can          | Methods that require context  |

## The Async Way of Doing Things

|              To Do This ...              |    Instead of This ...     |       Use This       |
|------------------------------------------|----------------------------|----------------------|
| Retrieve the result of a background task | `Task.Wait or Task.Result` | `await`              |
| Wait for any task to complete            | `Task.WaitAny`             | `await Task.WhenAny` |
| Retrieve the results of multiple tasks   | `Task.WaitAll`             | `await Task.WhenAll` |
| Wait a period of time                    | `Thread.Sleep`             | `await Task.Delay`   |

## Know Your Tools

There's a lot to learn about async and await, and it's natural to get a little
disoriented. Here's a quick reference of solutions to common problems.

**Solutions to Common Async Problems**

|                    Problem                    |                                  Solution                                  |
|-----------------------------------------------|-----------------------------------------------------------------------------|
| Create a task to execute code                 | `Task.Run` or `TaskFactory.StartNew` (not the `Task` constructor or `Task.Start`) |
| Create a task wrapper for an operation or event | `TaskFactory.FromAsync` or `TaskCompletionSource<T>`                        |
| Support cancellation                          | `CancellationTokenSource` and `CancellationToken`                           |
| Report progress                               | `IProgress<T>` and `Progress<T>`                                            |
| Handle streams of data                        | TPL Dataflow or Reactive Extensions                                         |
| Synchronize access to a shared resource       | `SemaphoreSlim`                                                             |
| Asynchronously initialize a resource          | `AsyncLazy<T>`                                                              |
| Async-ready producer/consumer structures      | TPL Dataflow or `AsyncCollection<T>`                                        |

## Async and Await Guidelines

Read the [Task-based Asynchronous Pattern (TAP) document](http://www.microsoft.com/download/en/details.aspx?id=19957).
It is extremely well-written, and includes guidance on API design and the proper
use of async/await (including cancellation and progress reporting).

There are many new await-friendly techniques that should be used instead of the
old blocking techniques. If you have any of these Old examples in your new async
code, you're Doing It Wrong(TM):

|        Old        |                New                |                        Description                        |
|-------------------|-----------------------------------|-----------------------------------------------------------|
| `task.Wait`       | `await task`                      | Wait/await for a task to complete                         |
| `task.Result`     | `await task`                      | Get the result of a completed task                        |
| `Task.WaitAny`    | `await Task.WhenAny`              | Wait/await for one of a collection of tasks to complete   |
| `Task.WaitAll`    | `await Task.WhenAll`              | Wait/await for every one of a collection of tasks to complete |
| `Thread.Sleep`    | `await Task.Delay`                | Wait/await for a period of time                           |
| `Task` constructor | `Task.Run` or `TaskFactory.StartNew` | Create a code-based task                                  |

> Source http://blog.stephencleary.com/2012/02/async-and-await.html

**Source**

- [Async/Await - Best Practices in Asynchronous Programming](https://msdn.microsoft.com/en-us/magazine/jj991977.aspx)
```