upGrad ENTERPRISE

Advanced Async Workflows in C#

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

1. What is an Async Workflow?

Definition:

An **async workflow** is a sequence of asynchronous operations executed in an order, where each operation can depend on the previous result and be non-blocking.

Purpose:

- Enables **scalable, responsive** applications.
- Prevents thread blocking during I/O-bound or long-running operations.

2. Chaining Async Workflows

Using `await` to chain tasks:

```
public async Task<string> ProcessDataAsync()
{
    string rawData = await ReadDataAsync();
    string transformed = await TransformDataAsync(rawData);
    await SaveDataAsync(transformed);
    return "Done!";
}
```

Using `.ContinueWith()`:

Less readable; use cautiously.

```
Task.Run(() => Step1())
    .ContinueWith(t => Step2(t.Result))
    .ContinueWith(t => Step3(t.Result));
```

Using `Task.WhenAll()`

Parallel Async Execution

Executes multiple async tasks concurrently and waits for **all** to complete.

```
public async Task DownloadAllFilesAsync()
{
    var urls = new[] { "file1", "file2", "file3" };
    var tasks = urls.Select(DownloadFileAsync);
    await Task.WhenAll(tasks);
}
```

- Great for **parallel I/O**.
- Returns an array of results when using `Task<T>`.

4. Async File Access in a Loop

Inefficient (sequential):

```
foreach (var file in files)
{
    string content = await File.ReadAllTextAsync(file);
}
```

Efficient (parallel):

```
var readTasks = files.Select(f => File.ReadAllTextAsync(f));
var contents = await Task.WhenAll(readTasks);
```

5. Integrating Async in Larger Programs

Async Logging

```
public async Task LogAsync(string message)
{
    await File.AppendAllTextAsync("log.txt", message + "\n");
}
```

Async Loaders (e.g., config loading)

```
public async Task LoadAllConfigsAsync()
{
    var configs = await Task.WhenAll(
        LoadDbConfigAsync(),
        LoadApiConfigAsync(),
        LoadUserSettingsAsync()
    );
}
```

6. `ConfigureAwait(false)`

What it does:

- Prevents capturing the **original context** (e.g., UI or ASP.NET context).
- Improves performance and avoids deadlocks in **library** or **non-UI** code.

Example:

```
await SomeIOOperationAsync().ConfigureAwait(false);
```

When to use:

✅ In **library code**
✅ In **ASP.NET** (non-UI)
❌ Not in UI thread (e.g., WinForms/WPF)

7. UI Simulation for Async in Console

Even though console apps don't have UI threads, you can simulate UI delays:

```
public static async Task SimulateUIAsync()
{
    Console.WriteLine("Loading...");
    await Task.Delay(2000); // simulate loading screen
    Console.WriteLine("Done!");
}
```

8. Understanding `Task.Run()`

Purpose:

Executes **CPU-bound** work on a **background thread**.

```
await Task.Run(() =>
{
    DoCPUIntensiveWork();
});
```

Don't use `Task.Run()` for:

- I/O-bound tasks (they already run asynchronously)
- Web APIs or ASP.NET (unnecessary thread usage)

9. Valid Scenarios for `Task.Run()`

Use Case	Avoid
Heavy CPU-bound work in UI apps	Wrapping `HttpClient`, `File.ReadAsync`, etc.
Offloading work to thread pool	Async by nature methods
Blocking legacy code	ASP.NET Core request pipeline

10. Alternatives to `Task.Run()`

a) Make code async end-to-end

```
// Instead of blocking:
var result = GetDataAsync().Result; ❌

// Use:
var result = await GetDataAsync(); ✅
```

b) Use `ConfigureAwait(false)`

Use it in library code to avoid resuming on captured context.

11. Real-Time Application Scenarios

Scenario	Async Use Case
Web scraping	Download multiple pages in parallel
File processing service	Async I/O for thousands of files
Game engines	Async asset loading, networking
Logging/Telemetry system	Non-blocking write to storage
Mobile apps	Background sync, notifications

12. Quiz Time

Quiz Questions

1. **What does `ConfigureAwait(false)` do?**
    - a) Blocks the thread
    - b) Avoids resuming on the captured context
    - c) Disables exception handling

2. **When should you use `Task.Run()`?**
    - a) For async I/O
    - b) For CPU-bound work
    - c) Inside ASP.NET API controllers

3. **What does `Task.WhenAll()` return?**
    - a) A list of exceptions
    - b) A `Task` that completes when all supplied tasks complete
    - c) Only the first completed result

4. **Which is more efficient for reading 100 files concurrently?**
    - a) `foreach` with `await`
    - b) `Task.WhenAll()` with `Select`

5. **What's a potential downside of misusing `Task.Run()` in ASP.NET Core?**

   - a) More responsiveness
   - b) Thread starvation
   - c) Faster I/O

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com