

```

public static ThreadLocal<string> threadLocalA = new ThreadLocal<string>();
public static ThreadLocal<string> threadLocalB = new ThreadLocal<string>();
public static ThreadLocal<string> threadLocalC = new ThreadLocal<string>();

//// AsyncLocal is within a container held within ExecutionContext
public static AsyncLocal<string> asyncLocalA = new AsyncLocal<string>();
public static AsyncLocal<string> asyncLocalB = new AsyncLocal<string>();
public static AsyncLocal<string> asyncLocalC = new AsyncLocal<string>();

public static async Task AsyncAwait_A(Action<string> output
    , bool continueOnCapturedSynchronizationContext = true, int pause = 1000)
{
    // Logical Execution 1
    Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("fr-FR");
    threadLocalA.Value = "A"; asyncLocalA.Value = "A";
    LogicalExecution(1, output);

    // Logical Execution 2
    await AsyncAwait_B(output, continueOnCapturedSynchronizationContext, pause)
        .ConfigureAwait(continueOnCapturedContext: continueOnCapturedSynchronizationContext);

    // Logical Execution 9
    LogicalExecution(9, output);
}

private static async Task AsyncAwait_B(Action<string> output
    , bool continueOnCapturedSynchronizationContext, int pause = 1000)
{
    // Logical Execution 3
    Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("en-US");
    threadLocalB.Value = "B"; asyncLocalB.Value = "B";
    LogicalExecution(3, output);

    // Location Execution 4
    await AsyncAwait_C(output, continueOnCapturedSynchronizationContext, pause)
        .ConfigureAwait(continueOnCapturedContext: continueOnCapturedSynchronizationContext);

    // Logical Execution 8
    LogicalExecution(8, output);
}

private static async Task AsyncAwait_C(Action<string> output
    , bool continueOnCapturedSynchronizationContext, int pause = 1000)
{
    // Logical Execution 5
    Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("es-MX");
    threadLocalC.Value = "C"; asyncLocalC.Value = "C";
    LogicalExecution(5, output);

    // Location Execution 6
    await Task.Delay(millisecondsDelay: pause)
        .ConfigureAwait(continueOnCapturedContext: continueOnCapturedSynchronizationContext);

    // Logical Execution 7
    LogicalExecution(7, output);
}

```

## Using Async - Await in C# as designed – Keith Voels

- Understand and follow the Task-Based Asynchronous Programming (TAP) design pattern and use it in *all* layers of your application.
- Always deal with the returned Task instance from an awaitable method
  - Await it, return it, Task.Wait() or Task.Result.
- Allow TAP/Async-Await to spread in your application.
  - It's better to have async available and not need it than to need it and not have it. Use Task.FromResult<> or Task.CompletedTask to end an async flow without an 'await'.
  - Start at the entry point and connect the Async - Await flow deeper
    - ASP.NET Controller methods – async Task Method() or async Task<T> Method()
    - Unit Test Methods – async Task Method()
    - WPF / Forms Events – async void Button\_Click(obj sender, EventArgs e)
      - Only safe use of async void. Do not use async void anywhere else.
- End with a .ConfigureAwait(false) on every await method call within your libraries
  - This allows async methods to be used with .Result or .Wait() when necessary without the risk of blocking.
  - This is annoying, yes, but this is by design.
- Use the System.Threading.Tasks Namespace Toolbox instead of System.Thread.
  - Common APIs: Task.Delay, Tasks.WhenAll, Tasks.WaitAll, Tasks.CompletedTask, Task.FromResult<T>()
  - TaskCompletionSource instead of AutoResetEvent
  - AsyncLocal instead of ThreadLocal
- .Net Framework 4.6.1 or newer
  - Critical pieces like Task.CompletedTask and AsyncLocal are introduced and other important async APIs.

### WPF / Forms

```
// Use ONLY for UI events with signatures that do not allow 'async Task'
public async void AsyncAwaitExercise1_Click(object sender, RoutedEventArgs e){
    await AsyncMethod().ConfigureAwait(false);
}

public async Task ITouchUIComponents_AsyncMethod(){
    await AsyncMethod().ConfigureAwait(false);
}
```

### ASP.NET

```
// Do not use async void outside of WPF / Forms
public async void AsyncAwaitExercise1_Click(object sender, RoutedEventArgs e){
public async Task AsyncMethod(){
    await AsyncMethod().ConfigureAwait(false); // Performance improvement, optional for .NET Core
}
```

### Library

```
// Do not use async void outside of WPF / Forms
public async void AsyncAwaitExercise1_Click(object sender, RoutedEventArgs e){
public async Task AsyncMethod(){
    await AsyncMethod().ConfigureAwait(false); // Required to make use in WPF/Forms guaranteed to
not block
}
```