Asynchronous Programming Demystified

http://submain.com/webcasts/asynchronous-programming-demystified/

for the webcast recording, slides and demo code download

submain

1/14/2015

Webcast Housekeeping

Audio

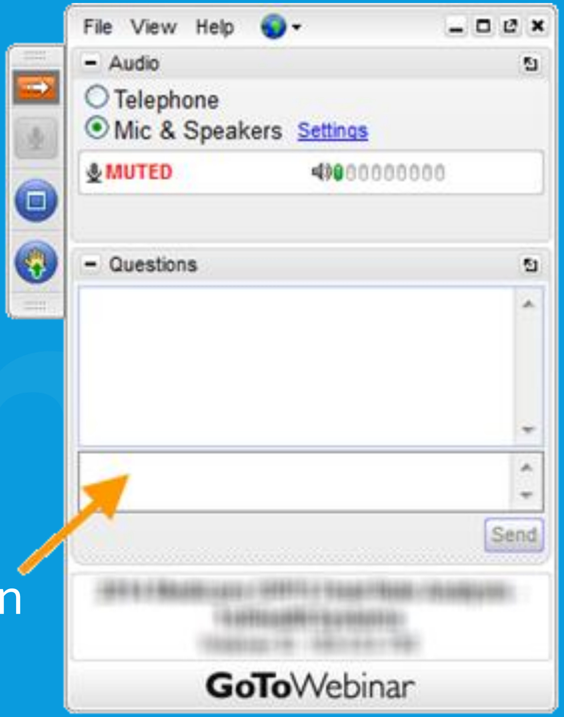
- Connect via VoIP
  - Plug in a headset or turn up your speakers
- Connect via Phone
  - Select "Use Telephone" after joining the webinar
  - Call 1 (415) 655-0053
    Access Code: 805-361-561
    Audio PIN: Shown after joining the webinar

Asking A Question

- Use the Questions window in the panel on the right of your screen
- Questions will be addressed at the end of the webcast

Recording

- A recording download link will be sent to all registrants within a few days



Agenda

- Overview of Async Patterns
- Introduction to async/await
- Asynchrony and Parallelism
- Benefits of async/await
- Demo
- Future Async webinars
- Q&A

Introduction

Presenter

Stephen Cleary

Microsoft MVP

(g)host

Serge Baranovsky

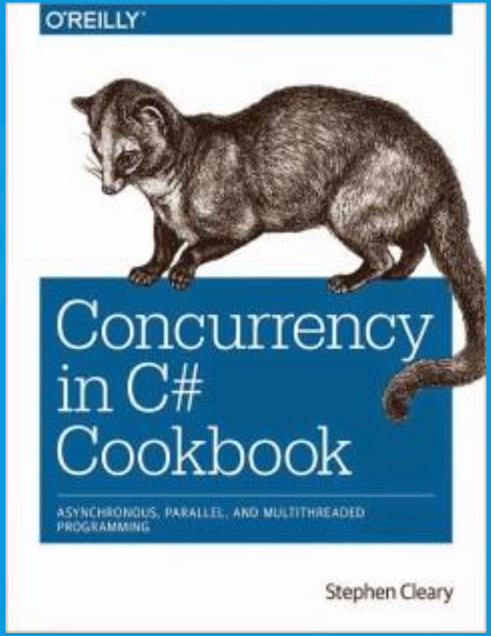
Principal, SubMain

Stephen Cleary

stephencleary.com

Microsoft® Most Valuable Professional MVP

- C# Microsoft MVP
- Concurrency in C# Cookbook (O'Reilly)


O'REILLY
Concurrency in C# Cookbook
ASYNCHRONOUS, PARALLEL, AND MULTITHREADED PROGRAMMING
Stephen Cleary

Introduction to async/await

Overview of Async Patterns

- Async/await use the Task-based Async Pattern (TAP)
  - Task SampleAsync();
- Older: Event-based Async Pattern (EAP)
  - void SampleAsync();
  - event SampleCompleted;
- Older: Async Programming Model (APM)
  - IAsyncResult BeginSample();
  - EndSample();

Syntax

- Pair of keywords (async + await)

```
int Test()
{
  Thread.Sleep(100);
  return 13;
}

async Task<int> TestAsync()
{
  await Task.Delay(100);
  return 13;
}
```

What "async" means

- Enables the await keyword in that method.
- Creates a state machine for the method.

```
async Task<int> TestAsync()
{
  await Task.Delay(100);
  return 13;
}
```

What “await” means

- An await expression takes one argument – an “awaitable” (usually a Task or Task<T>).
  - This “awaitable” represents an *asynchronous operation*.

```
async Task<int> TestAsync()
{
  await Task.Delay(100);
  return 13;
}
```

```
async Task<int> TestAsync()
{
  var delayTask = Task.Delay(100);
  await delayTask;
  return 13;
}
```

What “await” means

- Await will *pause* its async method until the operation completes.
  - If operation is already completed -> don’t pause.
  - If operation faults -> raise exception.
- Resume executing in a *captured context* by default.

```
async Task<int> TestAsync()
{
  await Task.Delay(100);
  return 13;
}
```

What “await” means

- When await pauses:
  - Returns an incomplete task to its caller.
- When async method completes:
  - Completes the task that was returned earlier.
- The task *represents the method*.

```
async Task<int> TestAsync()
{
  await Task.Delay(100);
  return 13;
}
```

Captured Context

- When await resumes:
  - SynchronizationContext.Current or TaskScheduler.Current
- What that means in practice:
  - UI context, ASP.NET request context, or thread pool.
- Avoiding context: use ConfigureAwait(false).

```
async Task<int> TestAsync()
{
  await Task.Delay(100);
  return 13;
}
```

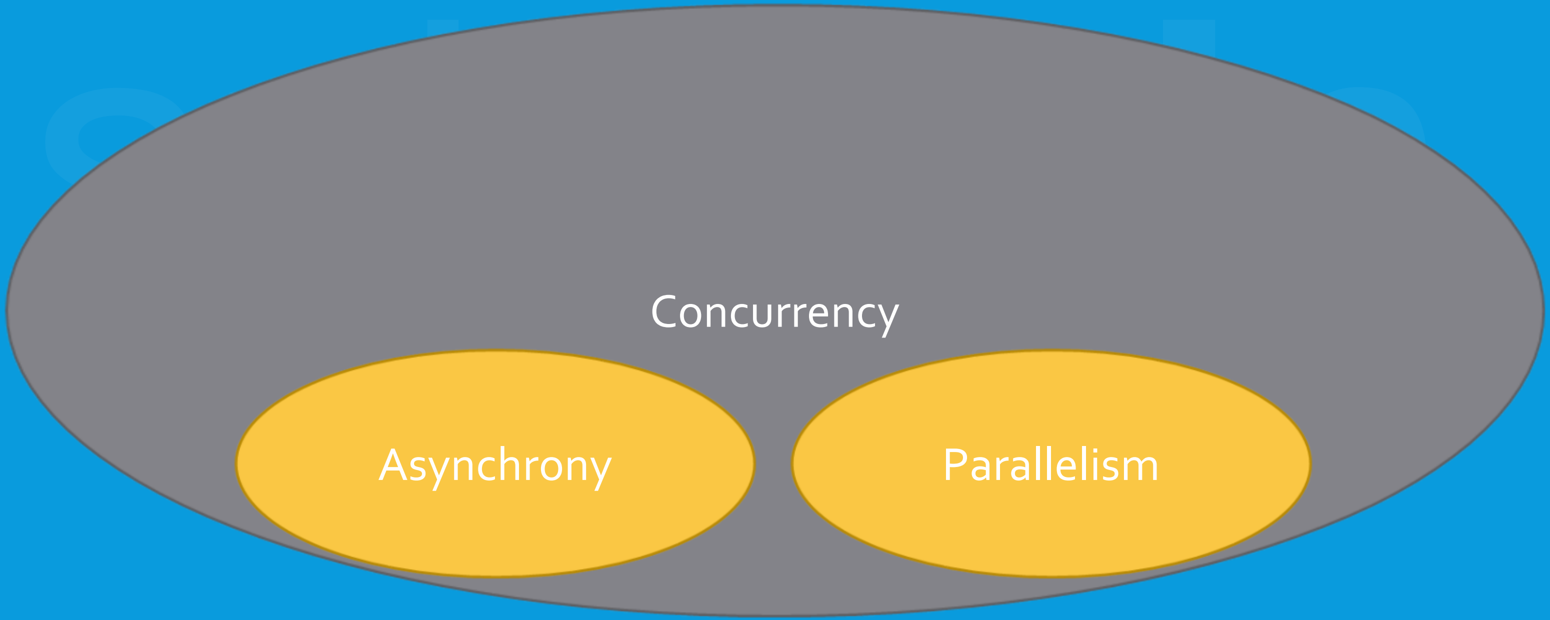
What “await” doesn’t mean

- Nothing to do with threads.
  - Not at all like “asynchronous delegates”.
  - Does not run the code on a background thread.
  - Does not parallelize your code.

```
async Task<int> TestAsync()
{
  await Task.Delay(100);
  return 13;
}
```

Asynchrony != Parallelism

- Each have their uses.
  - Asynchrony – I/O, events.
  - Parallelism – CPU-bound processing.



Benefits of async/await

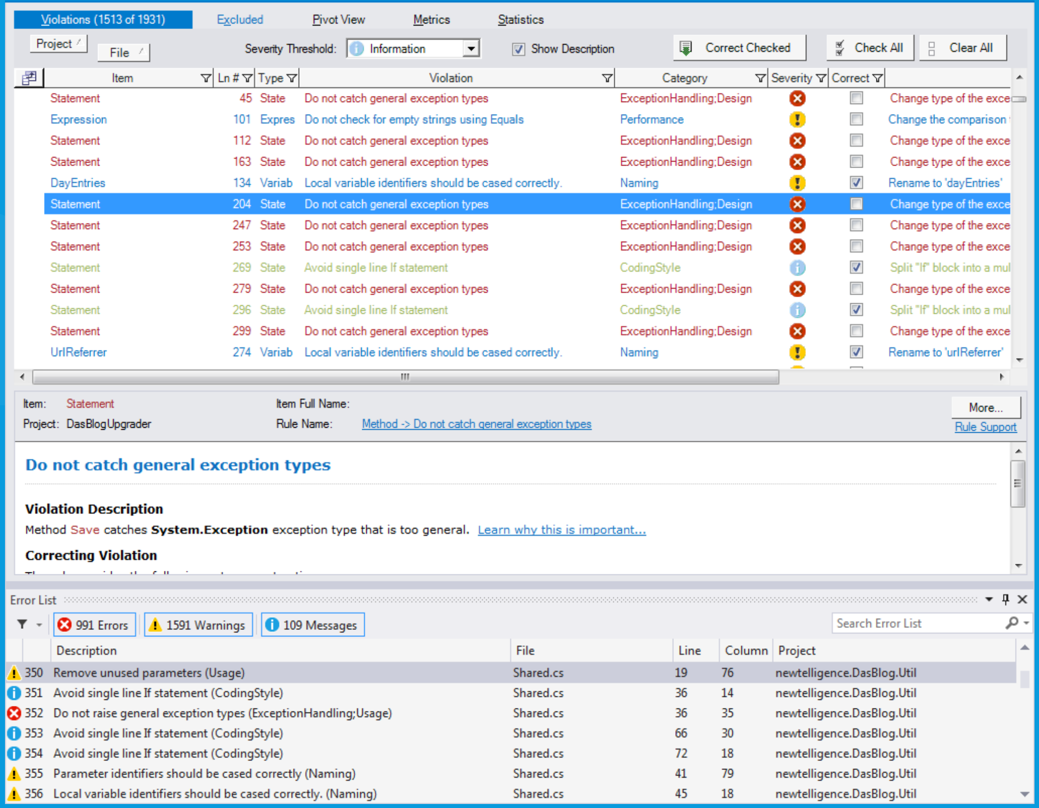
- ☑ Responsiveness on the client side
- ☑ Scalability on the server side
- ☑ Naturally-asynchronous code has async APIs
- ☑ Naturally-synchronous code stays synchronous
- ☑ Asynchronous code is clean – no callbacks!

What is CodeIt.Right

CodeIt.Right

What is CodeIt.Right

- Code Quality Analysis and Metrics
- Automated Code Review process
- Automated way to discover and fix code smells
- Automatic and safe refactoring of issues into conforming code
- Ensure your code adheres to (your) predefined design requirements and best coding practices



Severity Threshold: Information
[✓] Show Description

Item	Ln #	Type	Violation	Category	Severity	Correct	
Statement	45	State	Do not catch general exception types	ExceptionHandling;Design	✖	☐	Change type of the exce
Expression	101	Expres	Do not check for empty strings using Equals	Performance	!	☐	Change the comparison
Statement	112	State	Do not catch general exception types	ExceptionHandling;Design	✖	☐	Change type of the exce
Statement	163	State	Do not catch general exception types	ExceptionHandling;Design	✖	☐	Change type of the exce
DayEntries	134	Variab	Local variable identifiers should be cased correctly.	Naming	!	☑	Rename to 'dayEntries'
Statement	204	State	Do not catch general exception types	ExceptionHandling;Design	✖	☐	Change type of the exce
Statement	247	State	Do not catch general exception types	ExceptionHandling;Design	✖	☐	Change type of the exce
Statement	253	State	Do not catch general exception types	ExceptionHandling;Design	✖	☐	Change type of the exce
Statement	269	State	Avoid single line If statement	CodingStyle	i	☑	Split "If" block into a mul
Statement	279	State	Do not catch general exception types	ExceptionHandling;Design	✖	☐	Change type of the exce
Statement	296	State	Avoid single line If statement	CodingStyle	i	☑	Split "If" block into a mul
Statement	299	State	Do not catch general exception types	ExceptionHandling;Design	✖	☐	Change type of the exce
UrlReferrer	274	Variab	Local variable identifiers should be cased correctly.	Naming	!	☑	Rename to 'urlReferrer'

Item: Statement
Project: DasBlogUpgrader
Item Full Name:
Rule Name: Method -> Do not catch general exception types

Do not catch general exception types

Violation Description

Method Save catches **System.Exception** exception type that is too general. Learn why this is important...

Correcting Violation

Error List

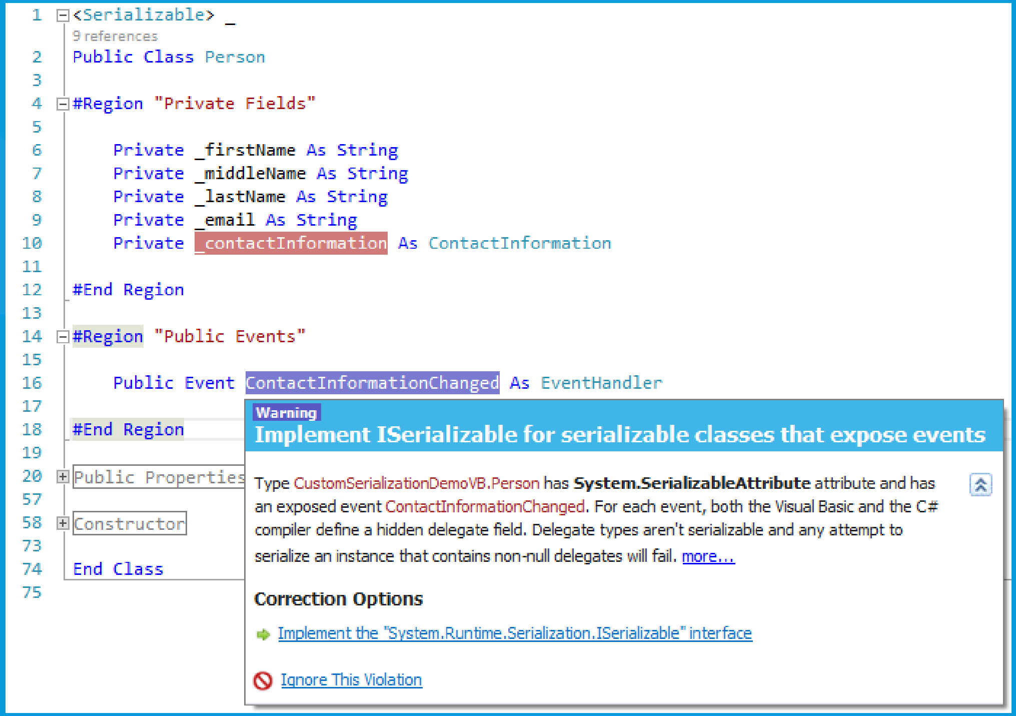
991 Errors | 1591 Warnings | 109 Messages

Search Error List

		Description	File	Line	Column	Project
!	350	Remove unused parameters (Usage)	Shared.cs	19	76	newtelligence.DasBlog.Util
i	351	Avoid single line If statement (CodingStyle)	Shared.cs	36	14	newtelligence.DasBlog.Util
✖	352	Do not raise general exception types (ExceptionHandling;Usage)	Shared.cs	36	35	newtelligence.DasBlog.Util
i	353	Avoid single line If statement (CodingStyle)	Shared.cs	66	30	newtelligence.DasBlog.Util
i	354	Avoid single line If statement (CodingStyle)	Shared.cs	72	18	newtelligence.DasBlog.Util
!	355	Parameter identifiers should be cased correctly (Naming)	Shared.cs	41	79	newtelligence.DasBlog.Util
!	356	Local variable identifiers should be cased correctly. (Naming)	Shared.cs	45	18	newtelligence.DasBlog.Util

What is CodeIt.Right - continued

- Instant Code Review – real-time code checking
- OnDemand Analysis
- Source Control Check-In Policy
- Build Process Integration
- Hundreds of rules
  - Security, Performance, Usage, Design, Maintainability, Exception Handling, Globalization, Async, and more



```
<Serializable> _
9 references
Public Class Person

#Region "Private Fields"

    Private _firstName As String
    Private _middleName As String
    Private _lastName As String
    Private _email As String
    Private _contactInformation As ContactInformation

#End Region

#Region "Public Events"

    Public Event ContactInformationChanged As EventHandler

#End Region

Public Properties
Constructor

End Class
```

Warning

Implement ISerializable for serializable classes that expose events

Type CustomSerializationDemoVB.Person has **System.SerializableAttribute** attribute and has an exposed event ContactInformationChanged. For each event, both the Visual Basic and the C# compiler define a hidden delegate field. Delegate types aren't serializable and any attempt to serialize an instance that contains non-null delegates will fail. more...

Correction Options

Implement the "System.Runtime.Serialization.ISerializable" interface

Ignore This Violation

Demo

http://submain.com/webcasts/asynchronous-programming-demystified/
for the webcast recording, slides and **demo code download**

- Demo #1 – **Correct** async code
- Demo #2 – Async naming convention
- Demo #3 – Async void
- Demo #4 – Blocking on async code
- Demo #5 – Async code blocking
- Demo #6 – Fake-async code
- Demo #7 – Using ContinueWith

Asynchronous Programming

- Async confusing? **CodeIt.Right will guide**
- CodeIt.Right Async rule set:
  - Async method should have "Async" suffix
  - Async method should have await statement
  - Async method should return Task or Task<T>
  - Async method - avoid "out" and "ref" parameters
  - Async method - await for completion
  - Await statement - method should be async
  - Async method - call Start on the Task
  - Async method - do not use Task.Yield
  - Async method - do not use Task.Wait
  - Async method should not be Sub
  - Async method parameters should be the same to synchronous counterpart
  - Async method - transform to non-async if simple

Poll

- Future Async webinars

Q&A

http://submain.com/webcasts/asynchronous-programming-demystified/

for the webcast recording, slides and demo code download

Questions?

Email - **customer-service@submain.com**

Video - **submain.com/codeit.right/video**

1 (800) 936-2134

Download the free CodeIt.Right trial at **submain.com/codeit.right**

Contact Stephen Cleary: **stephencleary.com**