THE ASYNC
INVASION
WHO IS THIS GUY?
How do I do this thing?

I'm not aware of how to use Google, how do I do this basic thing in Language X?

5 Answers

Lazy but functional answer with no extensibility

That's perfect! I'm never checking back here again! – 1337z0r Feb 1 '10 at 16:42

A lot of people think you should do it in a lazy way, however in the long run it will help you if you read this well eloquent wall of text that acutely describes problems you will inevitably face but not take the time to read about here; enjoy these code samples and illustrations I pulled from thin air anyway!

The official way to do this is link.

Uh. No thanks, this is too hard. Can you give me an example of how my code should look when complete? – 1337z0r Feb 1 '10 at 16:41
THE ASYNC REVOLUTION!
THE ASYNC REVOLUTION!

INVASION?
FUTURE TIMELINE?

- C++ (n4680 Coroutines) – C++20?
- Kotlin (experimental coroutines in 1.1)
- Rust (nightly since 2017-08)
MUA HA HA HA HA!!
What does "asynchronous" really mean?
• Concurrent
  ■ Multithreaded
    ○ Parallel
  ■ Asynchronous
YOU'RE TELLING ME I CAN BE CONCURRENT

WITHOUT THREADS?
There Is No Thread

For I/O, which is more natural?

- Synchronous APIs
- Asynchronous APIs
WHY?
BENEFITS OF ASYNCHRONY

- UI: responsiveness.
- Server: scalability.
WHY IS IT IMPORTANT TODAY?

- Mobile
- Cloud
BUT WHY ASYNC/AWAIT?
ASYNCHRONY: AN ARCHAEOLOGICAL TOUR

- Events
- Callbacks / CPS
- Futures
- Async/Await
Our application will...

1. Download a string from the internet
2. Save it to a database
SYNCHRONOUS SOLUTION

PROVIDED API

```csharp
string Download();
void Save(string);
```

IMPLEMENTATION

```csharp
void DownloadAndSave() {
    string data = Download();
    Save(data);
}
```
EVENTS

PROVIDED API

```c
void Download();
event<string> DownloadCompleted;

void Save(string);
event<void> SaveCompleted;
```

SECRET SAUCE API

```c
void DownloadAndSave();
event<void> DownloadAndSaveCompleted;
```
event<void> DownloadAndSaveCompleted;

void DownloadAndSave() {
    DownloadCompleted += downloadResult => {
        if (downloadResult.error) {
            trigger DownloadAndSaveCompleted with downloadResult.error;
            return;
        }
    }
    SaveCompleted += saveResult => {
        if (saveResult.error) {
            trigger DownloadAndSaveCompleted with saveResult.error;
            return;
        }
        trigger DownloadAndSaveCompleted;
    };
    Save(downloadResult.data);
}
EVENTS: PROBLEMS

- Have to read code backwards.
- Manual error handling.
- Deep nesting.
- Non-trivial logic (loops, joins) require manual state machines.
CALLBACKS / CPS

PROVIDED API

```c
void Download(callback<string>);
void Save(string, callback<void>);
```

SECRET SAUCE API

```c
void DownloadAndSave(callback<void>);
```
void DownloadAndSave(callback<void> cb) {
    Download(downloadResult => {
        if (downloadResult.error) {
            cb(error = downloadResult.error);
            return;
        }
    })
    Save(downloadResult.data, saveResult => {
        if (saveResult.error) {
            cb(error = saveResult.error);
            return;
        }
        cb();
    });
    cb();
}
CALLBACKS / CPS: PROBLEMS

- Have to read code backwards.
- Manual error handling.
- Deep nesting.
- Non-trivial logic (loops, joins) require manual state machines.
FUTURES

A “Future” represents a future value.

Futures complete exactly once, either with a value or with an error.

Futures support continuations.

Futures are object representations of asynchronous operations.

Futures are monads.
A "Future" can be anything...

- File download
- Database write
- Timeout
- "Join" of other futures
- Mutual exclusion
FUTURES
PROVIDED API

Future<string> Download();
Future<void> Save(string);

SECRET SAUCE API

Future<void> DownloadAndSave();
FUTURES: IMPLEMENTATION

```typescript
Future<void> DownloadAndSave() {
    return Download()
        .then(data => { return Save(data); });
}
```
FUTURES: PROBLEMS

- Have to read code backwards.
- Manual error handling.
- Deep Shallow nesting.
- Non-trivial logic (loops, joins) require manual state machines multiple methods.
ASYNC / AWAIT

PROVIDED API

```
Future<string> Download();
Future<void> Save(string);
```

SECRET SAUCE API

```
Future<void> DownloadAndSave();
```
ASYNC/AWAIT: IMPLEMENTATION

```csharp
Future<
void
>
DownloadAndSave() {
    string data = await Download();
    await Save(data);
}

void DownloadAndSave() {
    string data = Download();
    Save(data);
}
```
FUTURES: PROBLEMS

- Have to read code backwards.
- Manual error handling.
- Deep nesting.
- Non-trivial logic (loops, joins) require state machines or multiple methods.
SUMMARY

Asynchrony is important today... Because of cloud and mobile...
But asynchronous code is hard...
So languages are adopting async/await...
To make asynchronous code easier.