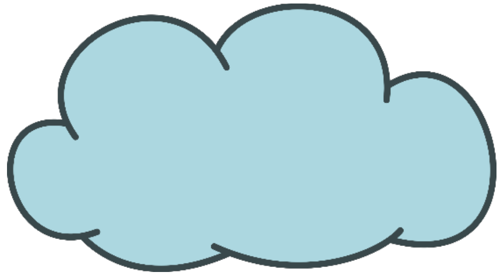
To the Cloud!

András Velvárt

@vbandi | http://vbandi.dotneteers.net

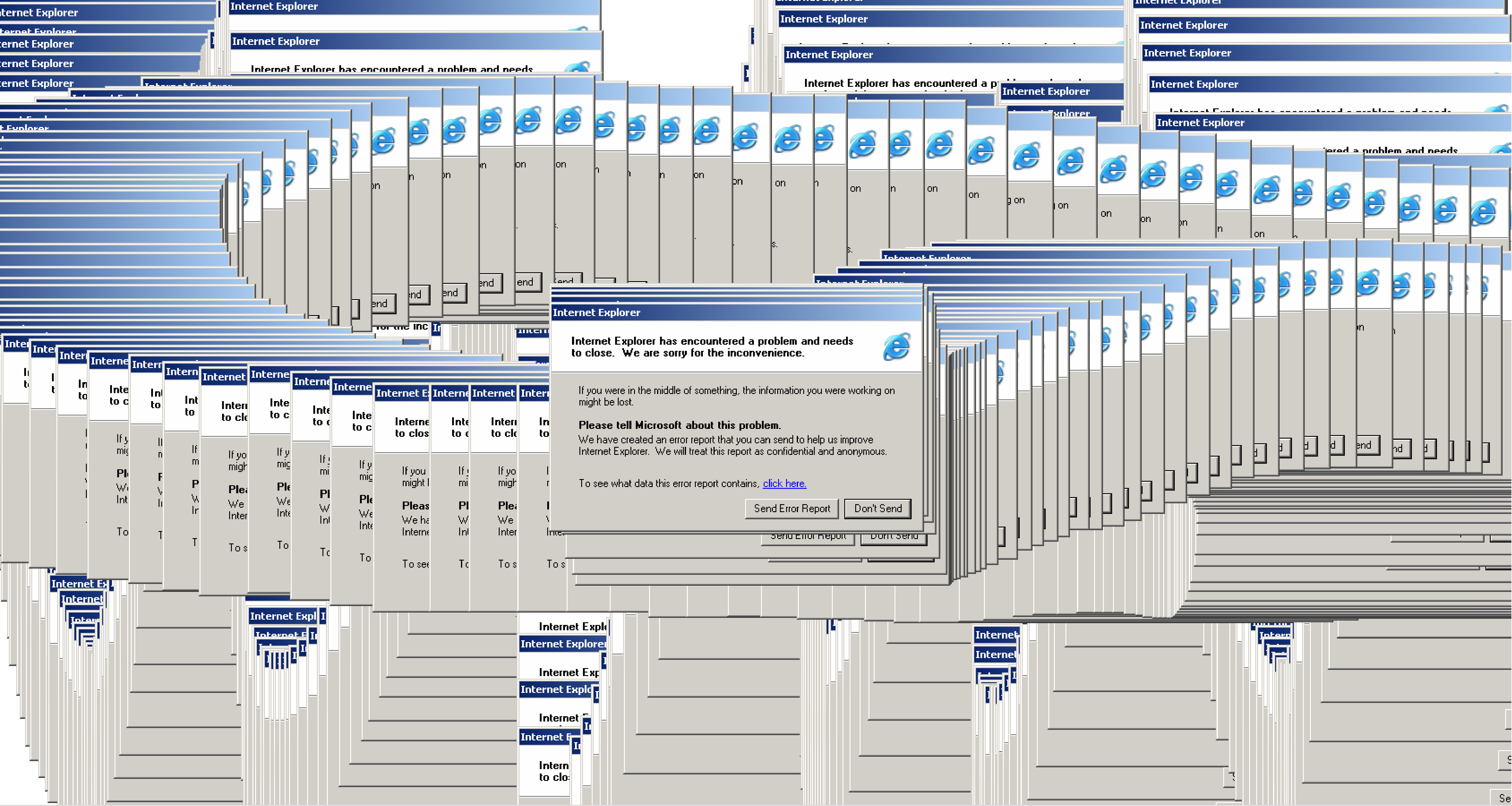
Module Overview



Downloading data from the Cloud

Download and parse JSON from a server

Asynchronous programming



Internet Explorer

Internet Explorer has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.

We have created an error report that you can send to help us improve Internet Explorer. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Created with http://mrdoob.com/lab/javascript/effects/ie6/

BACK TO THE PRESENT

Fun fact: this part was actually recorded on October 21, 2015 – on Back to the Future day!

video

Touchscreen manipulation, showing the weather app

Show mouse moving and clicking on a table

Show mouse cursor, clicking and scrolling with the scroll bar

Show pencil being pushed

Show pencil being pushed, but only moving after half a second

Keeping Your App Responsive

- Multiple threads
  - Most of it is managed by the runtime
- UI Thread
  - UI manipulation and events
  - Layout
  - Databinding
  - All of the code we’ve written so far
  - If blocked, the app freezes

The Solution

- Perform long running operations asynchronously
  - Do not block the UI Thread while waiting
  - When the result is available, return to the UI thread to display information
- Windows Runtime: anything that CAN take over 50ms can only be invoked asynchronously

Traditional Async Pattern

```
private void CountBytes()
{
    WebClient wc = new WebClient();
    wc.Headers.Add("Foo", "Bar");
    wc.DownloadDataCompleted += Wc_DownloadDataCompleted;
    wc.DownloadDataAsync(new Uri("http://www.facebook.com"));
}

private void Wc_DownloadDataCompleted(object sender, DownloadDataCompletedEventArgs e)
{
    if (MessageBoxResult.OK == MessageBox.Show($"Downloaded {e.Result.Length} bytes.",
                                "Try again?",
                                MessageBoxButton.OKCancel))
        CountBytes();
}
```

Traditional Async Pattern

```
private void CountBytes()
{
    WebClient wc = new WebClient();
    wc.Headers.Add("Foo", "Bar");
    wc.DownloadDataCompleted += Wc_DownloadDataCompleted;
    wc.DownloadDataAsync(new Uri("http://www.facebook.com"));
}

private void Wc_DownloadDataCompleted(object sender, DownloadDataCompletedEventArgs e)
{
    if (MessageBoxResult.OK == MessageBox.Show($"Downloaded {e.Result.Length} bytes.",
                                "Try again?",
                                MessageBoxButton.OKCancel))
        CountBytes();
}
```

Async – Await Solution

```
private async Task CountBytesAsync()
{
    HttpClient hc = new HttpClient();
    hc.DefaultRequestHeaders.Add("Foo", "Bar");
    string result;
    do
    {
        var data = await hc.GetByteArrayAsync("http://www.facebook.com");
        result = await ShowOKCancelMessageAsync(
            $"Downloaded {data.Length} bytes.",
            "Try again?");
    }
    while (result == "OK");
}
```

Async – Await Solution

```
private async Task<string> ShowOKCancelMessageAsync(string text, string caption)
{
    var dlg = new MessageDialog(text, caption);
    dlg.Commands.Add(new UICommand("OK"));
    dlg.Commands.Add(new UICommand("Cancel"));
    return (await dlg.ShowAsync()).Label;
}
```

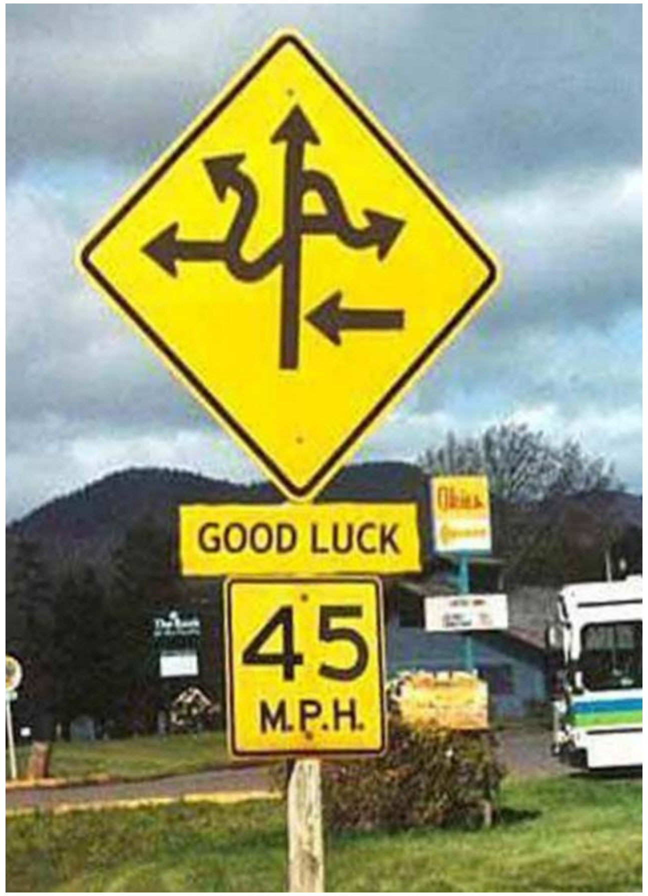
Async – Await Solution

```
private async Task CountBytesAsync()
{
    HttpClient hc = new HttpClient();
    hc.DefaultRequestHeaders.Add("Foo", "Bar");
    string result;
    do
    {
        var data = await hc.GetByteArrayAsync("http://www.facebook.com");
        result = await ShowOKCancelMessageAsync(
            $"Downloaded {data.Length} bytes.",
            "Try again?");
    }
    while (result == "OK");
}
```

Changing a Method to async

- Add the **async** keyword
- Wrap return value into a **System.Threading.Tasks.Task**
  - Example: public **string** Foo() -> public **Task<string>** FooAsync()
- Avoid **async void** methods:
  - Instead use: public **async Task** Foo()
  - The caller won’t know when it’s finished
  - Exceptions may disappear

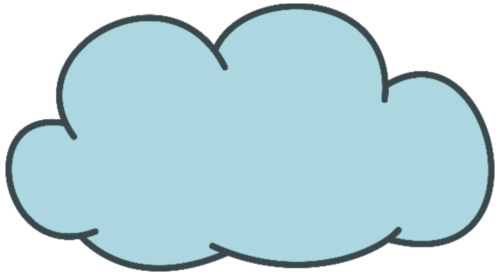
Async Can Be Confusing at First



Downloading the Actual Data

Design Time

Summary



Async frees up the UI thread while waiting for long running operations

In the Windows Runtime, all system calls that may run longer than 50ms are async

The **async** - **await** keywords help a lot