# Asynchronous Support

Mohamad Halabi
Microsoft Integration MVP
@mohamadhalabi

pluralsight
hardcore dev and IT training

# What is This Module About?

- **Not an in-depth discussion about:**
  - Parallel programming
  - Asynchronous model
  - Thread pool and threading
- **This module covers just enough detail to understand streaming asynchronous support**

# Operation Types

- **CPU intensive (computer-bound)**
  - Executed on a thread pool thread
  - Ex: mathematic operations, image processing, compression
- **I/O bound operations**
  - Ex: database calls and web service calls, file access, network access
  - Executed by hardware devices controlled by Windows device driver

# Synchronous Execution

## CPU Operations

Synchronous CPU operations are executed on the main program thread

```
static void Main(string[] args)
{
    //program execution
    //..
    //..

    ProcessImage();

    //continue process execution
    //..
    //..
}
```

Synchronous model is fine if you must wait for results of ProcessImage()

If ProcessImage() can run in parallel, then synchronous model reduces performance

## I/O Operations

Synchronous I/O operations are executed on hardware devices

Application thread that issues I/O call is blocked

```
static void Main(string[] args)
{
    //program execution
    //..
    //..

    AccessDB();

    //continue process execution
    //..
    //..
}
```

Negatively affects application scalability

# Asynchronous Execution

## CPU Operations

On multi-core machines asynchronous operations are executed on a separate thread pool thread

Application monitors the separate thread to know when operation is completed

## I/O Operations

Utilize I/O threads (the other type of threads held by the thread pool)

Application thread does not block waiting for the result

Application is notified when operation is completed

# How Does This Relate to Streaming?

## CPU Operations

MemoryStream is an exception to the backing store streams as data is in memory

Stream adapter that is talking to a MemoryStream

Decorator streams

## I/O Operations

Backing store streams

Ex: FileStream, NetworkStream, PipeStream

Web streams (WebRequest, WebClient, HttpClient)

Stream adapters

Backing store streams, decorator streams, and stream adapters (except binary adapters) expose asynchronous operations

# So When Should We Use the Asynchronous Model?

- **Decision factors for decorator streams and MemoryStream CPU operations:**
  - Does the program logic benefit from asynchronous execution?
    - Ex: Does your program need the result of a GZipStream operation before it carries on? If yes, then asynchronous model does not make sense
  - Even if logic benefits from asynchronous model, asynchronous execution incurs threading overhead
    - Ex: Time saved by executing compression asynchronously must outweigh threading overhead

# So When Should We Use the Asynchronous Model?

- **Decision factors for backing store streams and stream adapters I/O operations:**
  - Does the program logic benefit from asynchronous execution?
  - How long does the I/O operation block?
    - If long time then scalability gain outweighs threading overhead
    - If I/O operation is fast then asynchronous model might not be appropriate

# Asynchronous Operations in .NET 4.5

- **Pre .NET 4.5 you would use methods prefixed with "Begin" and "End"**
  - Ex: FileStream.BeginRead and FileStream.EndRead
  - Available for backward compatibility
- **.NET 4.5 simplifies the programming model**
  - Methods are appended with "Async"
    - Ex: FileStream exposes ReadAsync, WriteAsync, and FlushAsync
  - "async" modifier marks a method that contains asynchronous operation
  - "await" operator is applied to the result of an asynchronous method

# More of the Same

- **Using the asynchronous model across stream operations is an identical process:**
  - Look for “Async” methods
  - Use “async” modifier and “await” operator

```
private async void Process100Bytes()
{
    byte[] data = new byte[100];

    using (FileStream fs = new FileStream(@"c:\files\data.txt", FileMode.Open))
    {
        await fs.ReadAsync(data, 0, data.Length);
    }

    //process data
}
```

# Summary

- **CPU and I/O bound operation types**
- **CPU operations are executed on the thread pool**
- **I/O operations are executed on hardware devices**
- **Decorator stream actions are CPU bound**
- **Backing store streams and stream adapters actions are I/O bound**
- **CPU and I/O operations can be performed asynchronously**
  - CPU operations are executed on a separate thread pool thread
  - I/O operations do not cause main program thread to block