Asynchronous programming patterns

10/16/2018 • 2 minutes to read • +4

In this article

Comparison of patterns
See also

.NET provides three patterns for performing asynchronous operations:

- **Task-based Asynchronous Pattern (TAP)**, which uses a single method to represent the initiation and completion of an asynchronous operation. TAP was introduced in the .NET Framework 4. **It's the recommended approach to asynchronous programming in .NET.** The async and await keywords in C# and the Async and Await operators in Visual Basic add language support for TAP. For more information, see Task-based Asynchronous Pattern (TAP).

- **Event-based Asynchronous Pattern (EAP)**, which is the event-based legacy model for providing asynchronous behavior. It requires a method that has the `Async` suffix and one or more events, event handler delegate types, and `EventArg`-derived types. EAP was introduced in the .NET Framework 2.0. It's no longer recommended for new development. For more information, see Event-based Asynchronous Pattern (EAP).

- **Asynchronous Programming Model (APM)** pattern (also called the IAsyncResult pattern), which is the legacy model that uses the IAsyncResult interface to provide asynchronous behavior. In this pattern, synchronous operations require `Begin` and `End` methods (for example, `BeginWrite` and `EndWrite` to implement an asynchronous write operation). This pattern is no longer recommended for new development. For more information, see Asynchronous Programming Model (APM).

Comparison of patterns

For a quick comparison of how the three patterns model asynchronous operations, consider a `Read` method that reads a specified amount of data into a provided buffer starting at a specified offset:

C# Copy

```csharp
public class MyClass
{
    public int Read(byte [] buffer, int offset, int count);
}
```

The TAP counterpart of this method would expose the following single `ReadAsync` method:

C# Copy

```csharp
public class MyClass
{
    public Task<int> ReadAsync(byte [] buffer, int offset, int count);
}
```

The EAP counterpart would expose the following set of types and members:

C# Copy

```csharp
public class MyClass
{
    public void ReadAsync(byte [] buffer, int offset, int count);
    public event ReadCompletedEventHandler ReadCompleted;
}
```

The APM counterpart would expose the `BeginRead` and `EndRead` methods:

C# Copy

```csharp
public class MyClass
{
    public IAsyncResult BeginRead(
        byte [] buffer, int offset, int count,
        AsyncCallback callback, object state);
    public int EndRead(IAsyncResult asyncResult);
}
```

See also

- Async in depth
- Asynchronous programming in C#
- Async Programming in F#

- Asynchronous Programming with Async and Await (Visual Basic)

Is this page helpful?

Yes No