.NET   Languages ˅   Features ˅   Workloads ˅   APIs ˅   Resources ˅   Download .NET

Learn / .NET / Advanced programming /

Asynchronous programming patterns

Article • 02/13/2023 • 2 minutes to read • 15 contributors

Feedback

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

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

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

See also

- C# - Asynchronous programming with async and await
- Visual Basic - Asynchronous Programming with Async and Await
- F# - Asynchronous Programming

Recommended content

Managed Threading Basics

See links to other managed threading articles, covering topics such as exceptions, synchronizing data, foreground & background threads, local storage, and more.

Data Structures for Parallel Programming

Learn more about: Data Structures for Parallel Programming

Parallel Programming in .NET: A guide to the documentation

A list of articles about parallel programming in .NET.

Task-based Asynchronous Pattern (TAP): Introduction and overview

Learn about the Task-based Asynchronous Pattern (TAP), and compare it to the legacy patterns: Asynchronous Programming Model (APM) and Event-based Asynchronous Pattern (EAP).

Show more ˅

Feedback

Submit and view feedback for

[This product]   [This page]

View all page feedback