Microsoft Logo

Gray Pipe

Developer Network Developer Network Developer

Subscriber portal

Get tools

- Downloads
- Programs
- Community
- Documentation

search clear

.NET Framework Programming with the .NET Framework Including Asynchronous Calls

Including Asynchronous Calls Asynchronous Programming Overview

Asynchronous Programming Overview

Asynchronous Programming Overview

Asynchronous Programming Design Pattern

Asynchronous Delegates

TOC

This documentation is archived and is not being maintained.

Recommended Version

Asynchronous Programming Overview

.NET Framework 1.1

Other Versions

- .NET Framework (current version)
- Visual Studio 2010

- .NET Framework 4
- .NET Framework 3.5
- .NET Framework 3.0
- View all retired versions

The .NET Framework allows you to call any method asynchronously. Define a delegate with the same signature as the method you want to call; the common language runtime automatically defines **BeginInvoke** and **EndInvoke** methods for this delegate, with the appropriate signatures.

The **BeginInvoke** method is used to initiate the asynchronous call. It has the same parameters as the method you want to execute asynchronously, plus two additional parameters that will be described later. **BeginInvoke** returns immediately and does not wait for the asynchronous call to complete. **BeginInvoke** returns an IasyncResult, which can be used to monitor the progress of the call.

The **EndInvoke** method is used to retrieve the results of the asynchronous call. It can be called any time after **BeginInvoke**; if the asynchronous call has not completed, **EndInvoke** will block until it completes. The parameters of **EndInvoke** include the **out** and **ref** parameters (<Out> **ByRef** and **ByRef** in Visual Basic) of the method you want to execute asynchronously, plus the **IAsyncResult** returned by **BeginInvoke**.

> **Note** The IntelliSense feature in Visual Studio .NET displays the parameters of **BeginInvoke** and **EndInvoke**. If you are not using Visual Studio or a similar tool, or if you are using C# with Visual Studio .NET, see Asynchronous Method Signatures for a description of the parameters the runtime defines for these methods.

The code in this topic demonstrates four common ways to use **BeginInvoke** and **EndInvoke** to make asynchronous calls. After calling **BeginInvoke** you can:

- Do some work and then call **EndInvoke** to block until the call completes.
- Obtain a WaitHandle using IAsyncResult.AsyncWaitHandle, use its WaitOne method to block execution until the **WaitHandle** is signaled, and then call **EndInvoke**.
- Poll the **IAsyncResult** returned by **BeginInvoke** to determine when the asynchronous call has completed, and then call **EndInvoke**.

- Pass a delegate for a callback method to **BeginInvoke**. The method is executed on a ThreadPool thread when the asynchronous call completes, and can call **EndInvoke**.

CAUTION Always call **EndInvoke** after your asynchronous call completes.

Test Method and Asynchronous Delegate

All four samples use the same long-running test method, **TestMethod**. This method displays a console message to show that it has begun processing, sleeps for a few seconds, and then ends. **TestMethod** has an **out** parameter (<Out> **ByRef** in Visual Basic) to demonstrate the way such parameters are added to the signatures of **BeginInvoke** and **EndInvoke**. You can handle **ref** parameters (**ByRef** in Visual Basic) similarly.

The following code example shows **TestMethod** and the delegate that represents it; to use any of the samples, append the sample code to this code.

> **Note** To simplify the samples, **TestMethod** is declared in a separate class from
> Main()
> . Alternatively, **TestMethod** could be a **static** method (**Shared** in Visual Basic) in the same class that contains
> Main()
> .

VB

```
Imports System
Imports System.Threading
Imports System.Runtime.InteropServices

Public Class AsyncDemo
    ' The method to be executed asynchronously.
    '
    Public Function TestMethod(ByVal callDuration As Integer, _
            <Out> ByRef threadId As Integer) As String
        Console.WriteLine("Test method begins.")
        Thread.Sleep(callDuration)
        threadId = AppDomain.GetCurrentThreadId()
        return "MyCallTime was " + callDuration.ToString()
    End Function
End Class

' The delegate must have the same signature as the method
' you want to call asynchronously.
Public Delegate Function AsyncDelegate(ByVal callDuration As Integer, _
    <Out> ByRef threadId As Integer) As String
```

[C#]

```
using System;
using System.Threading;

public class AsyncDemo {
    // The method to be executed asynchronously.
    //
    public string TestMethod(int callDuration, out int threadId) {
        Console.WriteLine("Test method begins.");
        Thread.Sleep(callDuration);
        threadId = AppDomain.GetCurrentThreadId();
        return "MyCallTime was " + callDuration.ToString();
    }
}

// The delegate must have the same signature as the method
// you want to call asynchronously.
public delegate string AsyncDelegate(int callDuration, out int threadId);
```

Waiting for an Asynchronous Call with EndInvoke

The simplest way to execute a method asynchronously is to start it with **BeginInvoke**, do some work on the main thread, and then call **EndInvoke**. **EndInvoke** does not return until the asynchronous call completes. This is a good technique to use with file or network operations, but because it blocks on **EndInvoke**, you should not use it from threads that service the user interface.

VB

```
Public Class AsyncMain
    Shared Sub Main()
        ' The asynchronous method puts the thread id here.
        Dim threadId As Integer

        ' Create an instance of the test class.
        Dim ad As New AsyncDemo()

        ' Create the delegate.
        Dim dlgt As New AsyncDelegate(AddressOf ad.TestMethod)

        ' Initiate the asynchronous call.
        Dim ar As IAsyncResult = dlgt.BeginInvoke(3000, _
            threadId, Nothing, Nothing)

        Thread.Sleep(0)
        Console.WriteLine("Main thread {0} does some work.", _
             AppDomain.GetCurrentThreadId())

        ' Call EndInvoke to Wait for the asynchronous call to complete,
        ' and to retrieve the results.
        Dim ret As String = dlgt.EndInvoke(threadId, ar)

        Console.WriteLine("The call executed on thread {0}, with return value ""{1}"".",
threadId, ret)
    End Sub
End Class
```

[C#]

```
public class AsyncMain {
    static void Main(string[] args) {
        // The asynchronous method puts the thread id here.
        int threadId;

        // Create an instance of the test class.
        AsyncDemo ad = new AsyncDemo();

        // Create the delegate.
        AsyncDelegate dlgt = new AsyncDelegate(ad.TestMethod);

        // Initiate the asychronous call.
        IAsyncResult ar = dlgt.BeginInvoke(3000,
            out threadId, null, null);

        Thread.Sleep(0);
        Console.WriteLine("Main thread {0} does some work.",
            AppDomain.GetCurrentThreadId());

        // Call EndInvoke to Wait for the asynchronous call to complete,
        // and to retrieve the results.
        string ret = dlgt.EndInvoke(out threadId, ar);

        Console.WriteLine("The call executed on thread {0}, with return value \"{1}\".",
threadId, ret);
    }
}
```

Waiting for an Asynchronous Call with WaitHandle

Waiting on a WaitHandle is a common thread synchronization technique. You can obtain a **WaitHandle** using the AsyncWaitHandle property of the IAsyncResult returned by **BeginInvoke**. The **WaitHandle** is signaled when the asynchronous call completes, and you can wait for it by calling its WaitOne.

If you use a **WaitHandle**, you can perform additional processing after the asynchronous call completes, but before you retrieve the results by calling **EndInvoke**.

VB

```
Public Class AsyncMain
    Shared Sub Main()
```

```
        ' The asynchronous method puts the thread id here.
        Dim threadId As Integer

        ' Create an instance of the test class.
        Dim ad As New AsyncDemo()

        ' Create the delegate.
        Dim dlgt As New AsyncDelegate(AddressOf ad.TestMethod)

        ' Initiate the asynchronous call.
        Dim ar As IAsyncResult = dlgt.BeginInvoke(3000,
            threadId, Nothing, Nothing)

        Thread.Sleep(0)
        Console.WriteLine("Main thread {0} does some work.",
            AppDomain.GetCurrentThreadId())

        ' Wait for the WaitHandle to become signaled.
        ar.AsyncWaitHandle.WaitOne()

        ' Perform additional processing here.
        ' Call EndInvoke to retrieve the results.
        Dim ret As String = dlgt.EndInvoke(threadId, ar)

        Console.WriteLine("The call executed on thread {0}, with return value ""{1}"".",
threadId, ret)
    End Sub
End Class
[C#]
public class AsyncMain {
    static void Main(string[] args) {
        // The asynchronous method puts the thread id here.
        int threadId;

        // Create an instance of the test class.
        AsyncDemo ad = new AsyncDemo();

        // Create the delegate.
        AsyncDelegate dlgt = new AsyncDelegate(ad.TestMethod);

        // Initiate the asychronous call.
        IAsyncResult ar = dlgt.BeginInvoke(3000,
            out threadId, null, null);

        Thread.Sleep(0);
        Console.WriteLine("Main thread {0} does some work.",
            AppDomain.GetCurrentThreadId());

        // Wait for the WaitHandle to become signaled.
        ar.AsyncWaitHandle.WaitOne();

        // Perform additional processing here.
        // Call EndInvoke to retrieve the results.
        string ret = dlgt.EndInvoke(out threadId, ar);

        Console.WriteLine("The call executed on thread {0}, with return value \"{1}\".",
threadId, ret);
    }
}
```

Polling for Asynchronous Call Completion

You can use the IsCompleted property of the IAsyncResult returned by **BeginInvoke** to discover when the asynchronous call completes. You might do this when making the asynchronous call from a thread that services the user interface. Polling for completion allows the user interface thread to continue processing user input.

VB

```
Public Class AsyncMain
    Shared Sub Main()
        ' The asynchronous method puts the thread id here.
        Dim threadId As Integer

        ' Create an instance of the test class.
        Dim ad As New AsyncDemo()

        ' Create the delegate.
        Dim dlgt As New AsyncDelegate(AddressOf ad.TestMethod)

        ' Initiate the asynchronous call.
        Dim ar As IAsyncResult = dlgt.BeginInvoke(3000,
            threadId, Nothing, Nothing)

        ' Poll while simulating work.
        While ar.IsCompleted = False
            Thread.Sleep(10)
        End While

        ' Call EndInvoke to retrieve the results.
        Dim ret As String = dlgt.EndInvoke(threadId, ar)

        Console.WriteLine("The call executed on thread {0}, with return value ""{1}"".",
threadId, ret)
    End Sub
End Class
[C#]
public class AsyncMain {
    static void Main(string[] args) {
        // The asynchronous method puts the thread id here.
        int threadId;

        // Create an instance of the test class.
        AsyncDemo ad = new AsyncDemo();

        // Create the delegate.
        AsyncDelegate dlgt = new AsyncDelegate(ad.TestMethod);
        // Initiate the asychronous call.
        IAsyncResult ar = dlgt.BeginInvoke(3000,
            out threadId, null, null);

        // Poll while simulating work.
        while(ar.IsCompleted == false) {
            Thread.Sleep(10);
        }

        // Call EndInvoke to retrieve the results.
        string ret = dlgt.EndInvoke(out threadId, ar);

        Console.WriteLine("The call executed on thread {0}, with return value \"{1}\".",
threadId, ret);
    }
}
```

Executing a Callback Method When an Asynchronous Call Completes

If the thread that initiates the asynchronous call does not need to process the results, you can execute a callback method when the call completes. The callback method is executed on a ThreadPool thread.

To use a callback method, you must pass **BeginInvoke** an AsyncCallback delegate representing the method. You can also pass an object containing information to be used by the callback method. For example, you might pass the delegate that was used to initiate the call, so the callback method can call **EndInvoke**.

VB

```
Public Class AsyncMain
    ' The asynchronous method puts the thread id here.
    Private Shared threadId As Integer

    Shared Sub Main()
        ' Create an instance of the test class.
        Dim ad As New AsyncDemo()

        ' Create the delegate.
        Dim dlgt As New AsyncDelegate(AddressOf ad.TestMethod)

        ' Initiate the asynchronous call.
        Dim ar As IAsyncResult = dlgt.BeginInvoke(3000, _
            threadId, _
            AddressOf CallbackMethod, _
            dlgt)

        Console.WriteLine("Press Enter to close application.")
        Console.ReadLine()
    End Sub

    ' Callback method must have the same signature as the
    ' AsyncCallback delegate.
    Shared Sub CallbackMethod(ByVal ar As IAsyncResult)
        ' Retrieve the delegate.
        Dim dlgt As AsyncDelegate = CType(ar.AsyncState, AsyncDelegate)

        ' Call EndInvoke to retrieve the results.
        Dim ret As String = dlgt.EndInvoke(threadId, ar)

        Console.WriteLine("The call executed on thread {0}, with return value ""{1}"".",
threadId, ret)
    End Sub
End Class
[C#]
public class AsyncMain {
    // Asynchronous method puts the thread id here.
    private static int threadId;

    static void Main(string[] args) {
        // Create an instance of the test class.
        AsyncDemo ad = new AsyncDemo();

        // Create the delegate.
        AsyncDelegate dlgt = new AsyncDelegate(ad.TestMethod);

        // Initiate the asychronous call.  Include an AsyncCallback
        // delegate representing the callback method, and the data
        // needed to call EndInvoke.
        IAsyncResult ar = dlgt.BeginInvoke(3000,
            out threadId,
            new AsyncCallback(CallbackMethod),
            dlgt );

        Console.WriteLine("Press Enter to close application.");
        Console.ReadLine();
    }

    // Callback method must have the same signature as the
    // AsyncCallback delegate.
    static void CallbackMethod(IAsyncResult ar) {
        // Retrieve the delegate.
        AsyncDelegate dlgt = (AsyncDelegate) ar.AsyncState;

        // Call EndInvoke to retrieve the results.
        string ret = dlgt.EndInvoke(out threadId, ar);

        Console.WriteLine("The call executed on thread {0}, with return value \"{1}\".",
threadId, ret);
    }
}
```

See Also

Asynchronous Programming

Show: Inherited Protected

Dev centers

- Windows
- Office
- Visual Studio
- Microsoft Azure
- More...

Learning resources

- Microsoft Virtual Academy
- Channel 9
- MSDN Magazine

Community

- Forums
- Blogs
- Codeplex

Support

- Self support

Programs

- BizSpark (for startups)
- Microsoft Imagine (for students)

United States (English)

- Newsletter
- Privacy & cookies
- Terms of use
- Trademarks

Microsoft

logo

© 2017 Microsoft