Blocking Application Execution by Ending an Async Operation

03/30/2017 • 2 minutes to read • +6

In this article

Applications that cannot continue to do other work while waiting for the results of an asynchronous operation must block until the operation completes. Use one of the following options to block your application's main thread while waiting for an asynchronous operation to complete:

- Call the asynchronous operations **End***OperationName* method. This approach is demonstrated in this topic.
- Use the AsyncWaitHandle property of the IAsyncResult returned by the asynchronous operation's **Begin***OperationName* method. For an example that demonstrates this approach, see Blocking Application Execution Using an AsyncWaitHandle.

Applications that use the **End***OperationName* method to block until an asynchronous operation is complete will typically call the **Begin***OperationName* method, perform any work that can be done without the results of the operation, and then call **End***OperationName*.

Example

The following code example demonstrates using asynchronous methods in the Dns class to retrieve Domain Name System information for a user-specified computer. Note that `null` (`Nothing` in Visual Basic) is passed for the BeginGetHostByName `requestCallback` and `stateObject` parameters because these arguments are not required when using this approach.

C# Copy

```
/*
The following example demonstrates using asynchronous methods to
```

```
get Domain Name System information for the specified host computer.
*/

using System;
using System.Net;
using System.Net.Sockets;

namespace Examples.AdvancedProgramming.AsynchronousOperations
{
    public class BlockUntilOperationCompletes
    {
        public static void Main(string[] args)
        {
            // Make sure the caller supplied a host name.
            if (args.Length == 0 || args[0].Length == 0)
            {
                // Print a message and exit.
                Console.WriteLine("You must specify the name of a host
computer.");
                return;
            }
            // Start the asynchronous request for DNS information.
            // This example does not use a delegate or user-supplied object
            // so the last two arguments are null.
            IAsyncResult result = Dns.BeginGetHostEntry(args[0], null, null);
            Console.WriteLine("Processing your request for information...");
            // Do any additional work that can be done here.
            try
            {
                // EndGetHostByName blocks until the process completes.
                IPHostEntry host = Dns.EndGetHostEntry(result);
                string[] aliases = host.Aliases;
                IPAddress[] addresses = host.AddressList;
                if (aliases.Length > 0)
                {
                    Console.WriteLine("Aliases");
                    for (int i = 0; i < aliases.Length; i++)
                    {
                        Console.WriteLine("{0}", aliases[i]);
                    }
                }
                if (addresses.Length > 0)
                {
                    Console.WriteLine("Addresses");
                    for (int i = 0; i < addresses.Length; i++)
                    {
                        Console.WriteLine("{0}",addresses[i].ToString());
                    }
                }
            }
            catch (SocketException e)
```

```
            {
                Console.WriteLine("An exception occurred while processing the
request: {0}", e.Message);
            }
        }
    }
}
```

See also

- Event-based Asynchronous Pattern (EAP)
- Event-based Asynchronous Pattern Overview

Is this page helpful?

Yes No