**Implementing a Dispose Method**

**.NET Framework 4.5**

[Other Versions](javascript:;)

Description: http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/fs2xkftw(d=printer,v=vs.100).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/fs2xkftw(d=printer,v=vs.85).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/fs2xkftw(d=printer,v=vs.71).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/fs2xkftw(d=printer,v=vs.90).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/fs2xkftw(d=printer,v=vs.80).aspx)

The pattern for disposing an object, referred to as a dispose pattern, imposes order on the lifetime of an object. The dispose pattern is used only for objects that access unmanaged resources. This is because the garbage collector is very efficient at reclaiming unused managed objects.

A type's [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method should release all the resources that it owns. It should also release all resources owned by its base types by calling its parent type's [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method. The parent type's [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method should release all resources that it owns and in turn call its parent type's [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method, propagating this pattern through the hierarchy of base types.

A base class with subclasses that should be disposable must implement [IDisposable](http://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx) as follows:

* It should provide one public, non-virtual [Dispose()](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method and a protected virtual Dispose(Boolean disposing) method.
* The [Dispose()](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method must call Dispose(true) and should call the [GC.SuppressFinalize](http://msdn.microsoft.com/en-us/library/system.gc.suppressfinalize(v=vs.110).aspx) method to suppress finalization for better performance.
* The base type should not include any finalizers.

|  |
| --- |
| **Description: NoteNote** |
| See the [Example: Implementing Dispose for a base class](http://msdn.microsoft.com/en-us/library/fs2xkftw(d=printer,v=vs.110).aspx#base) section for an example that can be used as a template for implementing the dispose pattern for base classes. |

Subclasses should implement the disposable pattern as follows:

* They must override Dispose(Boolean) and call the base class Dispose(Boolean) implementation.
* They can provide a finalizer if needed. The finalizer must call Dispose(false).

To help ensure that resources are always cleaned up appropriately, a [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method should be callable multiple times without throwing an exception.

There is no performance benefit in implementing the [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method on types that use only managed resources (such as arrays) because they are automatically reclaimed by the garbage collector. Use the [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method primarily on managed objects that use native resources and on COM objects that are exposed to the .NET Framework. Managed objects that use native resources (such as the [FileStream](http://msdn.microsoft.com/en-us/library/system.io.filestream(v=vs.110).aspx) class) implement the [IDisposable](http://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx) interface.

|  |
| --- |
| **Description: Important noteImportant** |
| If you're a C++ programmer, do not implement the [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method. Instead, follow the instructions in the "Destructors and finalizers" section of [How to: Define and Consume Classes and Structs (C++/CLI)](http://msdn.microsoft.com/en-us/library/ke3a209d(v=vs.110).aspx). Starting with the .NET Framework version 2.0, the C++ compiler provides support for implementing deterministic disposal of resources and does not allow direct implementation of the [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method. |

A [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method should call the [SuppressFinalize](http://msdn.microsoft.com/en-us/library/system.gc.suppressfinalize(v=vs.110).aspx) method for the object it is disposing. If the object is currently on the finalization queue, [GC.SuppressFinalize](http://msdn.microsoft.com/en-us/library/system.gc.suppressfinalize(v=vs.110).aspx) prevents its [Finalize](http://msdn.microsoft.com/en-us/library/system.object.finalize(v=vs.110).aspx) method from being called. Remember that executing a [Finalize](http://msdn.microsoft.com/en-us/library/system.object.finalize(v=vs.110).aspx) method is costly to performance. If your [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method has already done the work to clean up the object, it is not necessary for the garbage collector to call the object's [Finalize](http://msdn.microsoft.com/en-us/library/system.object.finalize(v=vs.110).aspx) method.

The code example provided for the [GC.KeepAlive](http://msdn.microsoft.com/en-us/library/system.gc.keepalive(v=vs.110).aspx) method shows how aggressive garbage collection can cause a finalizer to run while a member of the reclaimed object is still executing. It is a good idea to call the [KeepAlive](http://msdn.microsoft.com/en-us/library/system.gc.keepalive(v=vs.110).aspx) method at the end of a lengthy [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method.

[The SafeHandle alternative](javascript:void(0))

Writing code for an object's finalizer is a complex task that can cause problems if not done correctly. Therefore, we recommend that you construct [System.Runtime.InteropServices.SafeHandle](http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.safehandle(v=vs.110).aspx) objects instead of implementing a finalizer.

The [SafeHandle](http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.safehandle(v=vs.110).aspx) class simplifies object lifetime issues by assigning and releasing handles without interruption. It contains a critical finalizer that is guaranteed to run while an application domain is unloading. For more information about the advantages of using a safe handle, see [Safe Handles and Critical Finalization](http://msdn.microsoft.com/en-us/library/fh21e17c(v=vs.110).aspx).

The [SafeHandle](http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.safehandle(v=vs.110).aspx) class in the [System.Runtime.InteropServices](http://msdn.microsoft.com/en-us/library/system.runtime.interopservices(v=vs.110).aspx) namespace is an abstract wrapper class for operating system handles. Deriving from this class is difficult. Instead, use the derived classes in the [Microsoft.Win32.SafeHandles](http://msdn.microsoft.com/en-us/library/microsoft.win32.safehandles(v=vs.110).aspx) namespace that provide safe handles for the following:

* Files and pipes
* Memory views
* Cryptography constructs
* Registry keys
* Wait handles

[Example: Implementing Dispose for a base class](javascript:void(0))

The following example shows the recommended design pattern for implementing a [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method for classes that encapsulate unmanaged resources. It defines a DisposableResource class that wraps a [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.110).aspx) object that is passed to its class constructor. The lifetime of the [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.110).aspx) object does not exceed the lifetime of the DisposableResource instance. Because the [Stream](http://msdn.microsoft.com/en-us/library/system.io.stream(v=vs.110).aspx) class implements [IDisposable](http://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx), the DisposableResource class must also provide an implementation. It does not need a finalizer because it does not directly create a native resource such as a file handle or allocate memory in the unmanaged heap.

Resource classes are typically derived from complex native classes or APIs and must be customized accordingly. Use this code pattern as a starting point for creating a resource class and provide the necessary customization based on the resources you are encapsulating.

C#

[VB](http://msdn.microsoft.com/en-us/library/fs2xkftw(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_a37d6d28-8f7d-474c-a422-dc106f8bbad3');" \o "Copy to clipboard.)

using System;

using System.IO;

class Program

{

static void Main()

{

try

{

// Initialize a Stream resource to pass to the DisposableResource class.

Console.Write("Enter filename and its path: ");

string fileSpec = Console.ReadLine();

if (! File.Exists(fileSpec)) {

Console.WriteLine("Enter a valid path and filename.");

return;

}

FileStream fs = File.OpenRead(fileSpec);

DisposableResource TestObj = new DisposableResource(fs);

// Use the resource.

TestObj.DoSomethingWithResource();

// Dispose the resource.

TestObj.Dispose();

}

catch (FileNotFoundException e) {

Console.WriteLine(e.Message);

}

}

}

// A base class that implements the Dispose pattern.

public class DisposableResource : IDisposable

{

private Stream \_resource;

private bool \_disposed;

public DisposableResource(Stream stream)

{

// Ensure stream is non-null and readable.

if (stream == null)

throw new ArgumentNullException("Stream is null.");

if (! stream.CanRead)

throw new ArgumentException("Stream must be readable.");

\_resource = stream;

\_disposed = false;

}

// Method that uses the resource.

public void DoSomethingWithResource() {

// The resource must not already be disposed.

if (\_disposed)

throw new ObjectDisposedException("Resource was disposed.");

// Show the number of bytes.

Console.WriteLine("Number of bytes: {0}", \_resource.Length);

}

public void Dispose()

{

Dispose(true);

// Call SupressFinalize in case a subclass implements a finalizer.

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

// If you need thread safety, use a lock around these

// operations, as well as in your methods that use the resource.

if (!\_disposed) {

if (disposing) {

if (\_resource != null) {

\_resource.Dispose();

Console.WriteLine("Object disposed.");

}

}

\_resource = null;

// Indicate that the instance has been disposed.

\_disposed = true;

}

}

}

[Example: Implementing Dispose for a derived class](javascript:void(0))

The following example shows the recommended design pattern for implementing the Dispose method for a subclass of a base class that implements [IDisposable](http://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx).

C#

[VB](http://msdn.microsoft.com/en-us/library/fs2xkftw(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

[Copy](javascript:if%20(window.epx.codeSnippet)window.epx.codeSnippet.copyCode('CodeSnippetContainerCode_98b683d2-9c41-4aeb-89a2-a629f3601f1a');)

using System;

using System.IO;

public class Example

{

public static void Main()

{

try {

Console.Write("Enter filename and its path: ");

string fileSpec = Console.ReadLine();

DisposableResource2 TestObj = new DisposableResource2(fileSpec);

// Use the resource.

TestObj.DoSomethingWithResource();

// Dispose the resource.

TestObj.Dispose();

}

catch (FileNotFoundException e) {

Console.WriteLine(e.Message);

}

}

}

public class DisposableResource2 : DisposableResource

{

private bool \_disposed;

public DisposableResource2(string filename)

: base(new FileStream(filename, FileMode.Open))

{ }

private DisposableResource2(Stream s) : base(s)

{ }

protected override void Dispose(bool disposing)

{

Console.WriteLine("Derived class Dispose(disposing) method");

if (! \_disposed) {

if (disposing) {

// Dispose of any managed resources of the derived class here.

// Call the base class implementation.

base.Dispose(disposing);

\_disposed = true;

}

// Dispose of any unmanaged resources of the derived class here.

}

}

}

// A base class that implements the Dispose pattern.

public class DisposableResource : IDisposable

{

private Stream \_resource;

private bool \_disposed;

public DisposableResource(Stream stream)

{

// Ensure stream is non-null and readable.

if (stream == null)

throw new ArgumentNullException("Stream is null.");

if (! stream.CanRead)

throw new ArgumentException("Stream must be readable.");

\_resource = stream;

\_disposed = false;

}

// Method that uses the resource.

public void DoSomethingWithResource()

{

// The resource must not already be disposed.

if (\_disposed)

throw new ObjectDisposedException("Resource was disposed.");

// Show the number of bytes.

Console.WriteLine("Number of bytes: {0}", \_resource.Length);

}

public void Dispose()

{

Dispose(true);

// Call SupressFinalize in case a subclass implements a finalizer.

GC.SuppressFinalize(this);

}

protected virtual void Dispose(bool disposing)

{

if (! \_disposed) {

// If you need thread safety, use a lock around these

// operations, as well as in your methods that use the resource.

if (disposing) {

// Free the necessary managed disposable objects.

if (\_resource != null) {

\_resource.Dispose();

Console.WriteLine("Object disposed.");

}

}

// Free unmanaged resources.

\_resource = null;

// Indicate that the instance has been disposed.

\_disposed = true;

}

}

}