**IDisposable Interface**

**.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/system.idisposable(d=printer,v=vs.100).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/system.idisposable(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/system.idisposable(d=printer,v=vs.85).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/system.idisposable(d=printer,v=vs.80).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/system.idisposable(d=printer,v=vs.71).aspx)
* [Silverlight](http://msdn.microsoft.com/en-us/library/system.idisposable(d=printer,v=vs.95).aspx)

Defines a method to release allocated resources.

**Namespace:**  [System](http://msdn.microsoft.com/en-us/library/system(v=vs.110).aspx)  
**Assembly:**  mscorlib (in mscorlib.dll)

[Syntax](javascript:void(0))

C#

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

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

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

[ComVisibleAttribute(true)]

public interface IDisposable

The IDisposable type exposes the following members.

[Methods](javascript:void(0))

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
|  | [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) | Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. |

[Remarks](javascript:void(0))

The primary use of this interface is to release unmanaged resources. The garbage collector automatically releases the memory allocated to a managed object when that object is no longer used. However, it is not possible to predict when garbage collection will occur. Furthermore, the garbage collector has no knowledge of unmanaged resources such as window handles, or open files and streams.

Use the [Dispose](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(v=vs.110).aspx) method of this interface to explicitly release unmanaged resources in conjunction with the garbage collector. The consumer of an object can call this method when the object is no longer needed.

It is a version-breaking change to add the IDisposable interface to an existing class, because it changes the semantics of the class.

|  |
| --- |
| **Description: Important noteImportant** |
| C++ programmers should read [Destructors and Finalizers in Visual C++](http://msdn.microsoft.com/en-us/library/ms177197(v=vs.110).aspx). In the .NET Framework version, 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. |

For a detailed discussion about how this interface and the [Object.Finalize](http://msdn.microsoft.com/en-us/library/system.object.finalize(v=vs.110).aspx) method are used, see the [Garbage Collection](http://msdn.microsoft.com/en-us/library/0xy59wtx(v=vs.110).aspx) and [Implementing a Dispose Method](http://msdn.microsoft.com/en-us/library/fs2xkftw(v=vs.110).aspx) topics.

**IDisposable and the inheritance hierarchy**

A base class with subclasses that should be disposable must implement IDisposable 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 suppress finalization for performance.
* The base type should not include any finalizers.

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).

**Calling the IDisposable interface**

When calling a class that implements the IDisposable interface, use the try/finally pattern to make sure that unmanaged resources are disposed of even if an exception interrupts your application.

For more information about the try/finally pattern, see [Try...Catch...Finally Statement (Visual Basic)](http://msdn.microsoft.com/en-us/library/fk6t46tz(v=vs.110).aspx), [try-finally (C# Reference)](http://msdn.microsoft.com/en-us/library/zwc8s4fz(v=vs.110).aspx), or [try-finally Statement (C)](http://msdn.microsoft.com/en-us/library/yb3kz605(v=vs.110).aspx).

Note that you can use the using statement (Using in Visual Basic) instead of the try/finally pattern. For more information, see the [Using Statement (Visual Basic)](http://msdn.microsoft.com/en-us/library/htd05whh(v=vs.110).aspx) documentation or the [using Statement (C# Reference)](http://msdn.microsoft.com/en-us/library/yh598w02(v=vs.110).aspx) documentation.

[Examples](javascript:void(0))

The following example demonstrates how to create a resource class that implements the IDisposable interface.

C#

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

[VB](http://msdn.microsoft.com/en-us/library/system.idisposable(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_4f2eee4d-6265-446e-b6e9-8c4e932b0aa5');)

using System;

using System.ComponentModel;

// The following example demonstrates how to create

// a resource class that implements the IDisposable interface

// and the IDisposable.Dispose method.

public class DisposeExample

{

// A base class that implements IDisposable.

// By implementing IDisposable, you are announcing that

// instances of this type allocate scarce resources.

public class MyResource: IDisposable

{

// Pointer to an external unmanaged resource.

private IntPtr handle;

// Other managed resource this class uses.

private Component component = new Component();

// Track whether Dispose has been called.

private bool disposed = false;

// The class constructor.

public MyResource(IntPtr handle)

{

this.handle = handle;

}

// Implement IDisposable.

// Do not make this method virtual.

// A derived class should not be able to override this method.

public void Dispose()

{

Dispose(true);

// This object will be cleaned up by the Dispose method.

// Therefore, you should call GC.SupressFinalize to

// take this object off the finalization queue

// and prevent finalization code for this object

// from executing a second time.

GC.SuppressFinalize(this);

}

// Dispose(bool disposing) executes in two distinct scenarios.

// If disposing equals true, the method has been called directly

// or indirectly by a user's code. Managed and unmanaged resources

// can be disposed.

// If disposing equals false, the method has been called by the

// runtime from inside the finalizer and you should not reference

// other objects. Only unmanaged resources can be disposed.

protected virtual void Dispose(bool disposing)

{

// Check to see if Dispose has already been called.

if(!this.disposed)

{

// If disposing equals true, dispose all managed

// and unmanaged resources.

if(disposing)

{

// Dispose managed resources.

component.Dispose();

}

// Call the appropriate methods to clean up

// unmanaged resources here.

// If disposing is false,

// only the following code is executed.

CloseHandle(handle);

handle = IntPtr.Zero;

// Note disposing has been done.

disposed = true;

}

}

// Use interop to call the method necessary

// to clean up the unmanaged resource.

[System.Runtime.InteropServices.DllImport("Kernel32")]

private extern static Boolean CloseHandle(IntPtr handle);

// Use C# destructor syntax for finalization code.

// This destructor will run only if the Dispose method

// does not get called.

// It gives your base class the opportunity to finalize.

// Do not provide destructors in types derived from this class.

~MyResource()

{

// Do not re-create Dispose clean-up code here.

// Calling Dispose(false) is optimal in terms of

// readability and maintainability.

Dispose(false);

}

}

public static void Main()

{

// Insert code here to create

// and use the MyResource object.

}

}

**IDisposable.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/system.idisposable.dispose(d=printer,v=vs.100).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(d=printer,v=vs.85).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(d=printer,v=vs.80).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(d=printer,v=vs.71).aspx)
* [Silverlight](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(d=printer,v=vs.95).aspx)

Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.

**Namespace:**  [System](http://msdn.microsoft.com/en-us/library/system(v=vs.110).aspx)  
**Assembly:**  mscorlib (in mscorlib.dll)

[Syntax](javascript:void(0))

C#

[C++](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-1)

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

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

void Dispose()

[Remarks](javascript:void(0))

Use this method to close or release unmanaged resources such as files, streams, and handles held by an instance of the class that implements this interface. By convention, this method is used for all tasks associated with freeing resources held by an object, or preparing an object for reuse.

|  |
| --- |
| **Description: Important noteImportant** |
| C++ programmers should read [Destructors and Finalizers in Visual C++](http://msdn.microsoft.com/en-us/library/ms177197(v=vs.110).aspx). In 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 method. |

When implementing this method, ensure that all held resources are freed by propagating the call through the containment hierarchy. For example, if an object A allocates an object B, and object B allocates an object C, then A's Dispose implementation must call Dispose on B, which must in turn call Dispose on C.

An object must also call the Dispose method of its base class if the base class implements [IDisposable](http://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx). For more information about implementing [IDisposable](http://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx) on a base class and its subclasses, see the "IDisposable and the inheritance hierarchy" section in the [IDisposable](http://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx) topic.

If an object's Dispose method is called more than once, the object must ignore all calls after the first one. The object must not throw an exception if its Dispose method is called multiple times. Instance methods other than Dispose can throw an [ObjectDisposedException](http://msdn.microsoft.com/en-us/library/system.objectdisposedexception(v=vs.110).aspx) when resources are already disposed.

Users might expect a resource type to use a particular convention to denote an allocated state versus a freed state. An example of this is stream classes, which are traditionally thought of as open or closed. The implementer of a class that has such a convention might choose to implement a public method with a customized name, such as Close, that calls the Dispose method.

Because the Dispose method must be called explicitly, objects that implement [IDisposable](http://msdn.microsoft.com/en-us/library/system.idisposable(v=vs.110).aspx) must also implement a finalizer to handle freeing resources when Dispose is not called. By default, the garbage collector automatically calls an object's finalizer prior to reclaiming its memory. However, once the Dispose method has been called, it is typically unnecessary for the garbage collector to call the disposed object's finalizer. To prevent automatic finalization, Dispose implementations can call the [GC.SuppressFinalize](http://msdn.microsoft.com/en-us/library/system.gc.suppressfinalize(v=vs.110).aspx) method.

For more information on implementing finalizers and the Dispose method, see the [GC](http://msdn.microsoft.com/en-us/library/system.gc(v=vs.110).aspx) class, the [Object.Finalize](http://msdn.microsoft.com/en-us/library/system.object.finalize(v=vs.110).aspx) method, and [Dispose Pattern](http://msdn.microsoft.com/en-us/library/b1yfkh5e(v=vs.110).aspx).

When you use an object that accesses unmanaged resources, such as a [StreamWriter](http://msdn.microsoft.com/en-us/library/system.io.streamwriter(v=vs.110).aspx), a good practice is to create the instance with a using statement. The using statement automatically closes the stream and calls Dispose on the object when the code that is using it has completed. For an example, see the [StreamWriter](http://msdn.microsoft.com/en-us/library/system.io.streamwriter(v=vs.110).aspx) class.

[Examples](javascript:void(0))

The following example shows how you can implement the Dispose method.

C#

[C++](http://msdn.microsoft.com/en-us/library/system.idisposable.dispose(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-2)

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

using System;

using System.ComponentModel;

// The following example demonstrates how to create

// a resource class that implements the IDisposable interface

// and the IDisposable.Dispose method.

public class DisposeExample

{

// A base class that implements IDisposable.

// By implementing IDisposable, you are announcing that

// instances of this type allocate scarce resources.

public class MyResource: IDisposable

{

// Pointer to an external unmanaged resource.

private IntPtr handle;

// Other managed resource this class uses.

private Component component = new Component();

// Track whether Dispose has been called.

private bool disposed = false;

// The class constructor.

public MyResource(IntPtr handle)

{

this.handle = handle;

}

// Implement IDisposable.

// Do not make this method virtual.

// A derived class should not be able to override this method.

public void Dispose()

{

Dispose(true);

// This object will be cleaned up by the Dispose method.

// Therefore, you should call GC.SupressFinalize to

// take this object off the finalization queue

// and prevent finalization code for this object

// from executing a second time.

GC.SuppressFinalize(this);

}

// Dispose(bool disposing) executes in two distinct scenarios.

// If disposing equals true, the method has been called directly

// or indirectly by a user's code. Managed and unmanaged resources

// can be disposed.

// If disposing equals false, the method has been called by the

// runtime from inside the finalizer and you should not reference

// other objects. Only unmanaged resources can be disposed.

protected virtual void Dispose(bool disposing)

{

// Check to see if Dispose has already been called.

if(!this.disposed)

{

// If disposing equals true, dispose all managed

// and unmanaged resources.

if(disposing)

{

// Dispose managed resources.

component.Dispose();

}

// Call the appropriate methods to clean up

// unmanaged resources here.

// If disposing is false,

// only the following code is executed.

CloseHandle(handle);

handle = IntPtr.Zero;

// Note disposing has been done.

disposed = true;

}

}

// Use interop to call the method necessary

// to clean up the unmanaged resource.

[System.Runtime.InteropServices.DllImport("Kernel32")]

private extern static Boolean CloseHandle(IntPtr handle);

// Use C# destructor syntax for finalization code.

// This destructor will run only if the Dispose method

// does not get called.

// It gives your base class the opportunity to finalize.

// Do not provide destructors in types derived from this class.

~MyResource()

{

// Do not re-create Dispose clean-up code here.

// Calling Dispose(false) is optimal in terms of

// readability and maintainability.

Dispose(false);

}

}

public static void Main()

{

// Insert code here to create

// and use the MyResource object.

}

}