

1 System.Runtime.InteropServices.SafeHandle

2 Class

```
3 [ILAsm]  
4 .class public abstract beforefieldinit SafeHandle extends System.Object  
5 implements System.IDisposable  
  
6 [C#]  
7 public abstract class SafeHandle: System.Object, IDisposable
```

8 Assembly Info:

- 9 • *Name:* mscorlib
- 10 • *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- 11 • *Version:* 4.0.0.0
- 12 • *Attributes:*
 - 13 ○ CLSCompliantAttribute(true)

14 Implements:

- 15 • **System.IDisposable**

16 Summary

17 Represents a wrapper class for operating system handles. This class must be inherited.

18 Inherits From: System.Object

19 **Library:** RuntimeInfrastructure

22 Permissions

Permission	Description
System.Security.Permissions.SecurityAction.InheritanceDemand	for full trust for inheritors. This member cannot be inherited by partially trusted code.

24 Description

25 The `System.Runtime.InteropServices.SafeHandle` class provides critical finalization
26 of handle resources, preventing handles from being reclaimed prematurely by garbage
27 collection and from being recycled by Windows to reference unintended unmanaged
28 objects.

29 The `System.Runtime.InteropServices.SafeHandle` class contains a finalizer that
30

1 ensures that the handle is closed and is guaranteed to run, even during unexpected
2 `System.AppDomain` unloads when a host may not trust the consistency of the state of
3 the `System.AppDomain`.
4

5 This class is abstract because you cannot create a generic handle. To implement
6 `System.Runtime.InteropServices.SafeHandle`, you must create a derived class. To
7 create `System.Runtime.InteropServices.SafeHandle` derived classes, you must know
8 how to create and free an operating system handle. This process is different for different
9 handle types because some use `CloseHandle`, while others use more specific methods
10 such as `UnmapViewOfFile` or `FindClose`. For this reason, you must create a derived
11 class of `System.Runtime.InteropServices.SafeHandle` for each operating system
12 handle type; such as `MySafeRegistryHandle`, `MySafeFileHandle`, and
13 `MySpecialSafeFileHandle`.

14 **How and When to Override**

15 When you inherit from `System.Runtime.InteropServices.SafeHandle`, you must
16 override the following members:
17 `System.Runtime.InteropServices.SafeHandle.Invalid` and
18 `System.Runtime.InteropServices.SafeHandle.ReleaseHandle`.
19

20 You should also provide a default constructor that calls the base constructor with a value
21 that represent an invalid handle value, and a `Boolean` value indicating whether the
22 native handle will be owned by the `System.Runtime.InteropServices.SafeHandle`,
23 and consequently freed when that `System.Runtime.InteropServices.SafeHandle` has
24 been disposed.

25

1 SafeHandle(System.IntPtr, System.Boolean) 2 Constructor

```
3 [ILAsm]  
4 .method family hidebysig specialname rtspecialname instance void  
5 .ctor(native int invalidHandleValue, bool ownsHandle) cil managed  
  
6 [C#]  
7 protected SafeHandle (IntPtr invalidHandleValue, bool ownsHandle)
```

8 Summary

9 Initializes a new instance of the `System.Runtime.InteropServices.SafeHandle` class
10 with the specified invalid handle value.

11 Parameters

Parameter	Description
<i>invalidHandleValue</i>	The value of an invalid handle (usually 0 or -1). Your implementation of <code>System.Runtime.InteropServices.SafeHandle</code> . <code>IsValid</code> should return <code>true</code> for this value.
<i>ownsHandle</i>	<code>true</code> to reliably let <code>System.Runtime.InteropServices.SafeHandle</code> release the handle during the finalization phase; otherwise, <code>false</code> (not recommended).

12 13 Description

14 If the *ownsHandle* parameter is `false`,
15 `System.Runtime.InteropServices.SafeHandle.ReleaseHandle` is never called; thus,
16 it is not recommended to use this parameter value as your code may leak resources.

17 Exceptions

Exception	Condition
System.TypeLoadException	The derived class resides in an assembly without unmanaged code access permission.

18 19 Permissions

Permission	Description
------------	-------------

**System.Security.Permissions.
SecurityAction.InheritanceDemand**

for full trust for inheritors. This member cannot be inherited by partially trusted code.

1

2

1 SafeHandle.handle Field

```
2 [ILAsm]  
3 .field family native int handle  
4 [C#]  
5 protected IntPtr handle
```

6 Summary

7 Specifies the handle to be wrapped.

8 Description

9 Do not expose the handle publicly (that is, outside of the derived class).

10

1 SafeHandle.Close() Method

```
2 [ILAsm]  
3 .method public hidebysig instance void Close() cil managed  
4 [C#]  
5 public void Close ()
```

6 Summary

7 Marks the handle for releasing and freeing resources.

8 Description

9 Calling the `System.Runtime.InteropServices.SafeHandle.Close` or
10 `System.Runtime.InteropServices.SafeHandle.Dispose` method allows the resources
11 to be freed. This might not happen immediately if other threads are using the same safe
12 handle object, but will happen as soon as that is no longer the case. Although most
13 classes that use the `System.Runtime.InteropServices.SafeHandle` class do not need
14 to provide a finalizer, this is sometimes necessary (for example, to flush out file buffers
15 or to write some data back into memory). In this case, such classes can provide a
16 finalizer that is guaranteed to run before the
17 `System.Runtime.InteropServices.SafeHandle` critical finalizer runs.

18
19 Call the `System.Runtime.InteropServices.SafeHandle.Close` or
20 `System.Runtime.InteropServices.SafeHandle.Dispose` method when you are
21 finished using the `System.Runtime.InteropServices.SafeHandle` object.

22
23 *[Note: Always call `System.Runtime.InteropServices.SafeHandle.Close` or
24 `System.Runtime.InteropServices.SafeHandle.Dispose` before you release your last
25 reference to the `System.Runtime.InteropServices.SafeHandle` object. Otherwise, the
26 resources it is using will not be freed until the garbage collector calls the
27 `System.Runtime.InteropServices.SafeHandle` object's
28 `System.Runtime.InteropServices.SafeHandle.Finalize` method.*

29
30]

31 Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: <code>System.Security.Permissions.SecurityAction.LinkDemand</code> . Associated enumeration: <code>System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode</code>

32

33

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

SafeHandle.DangerousAddRef(System.Boolean&) Method

```
[ILAsm]  
.method public hidebysig instance void DangerousAddRef(bool success) cil  
managed internalcall  
  
[C#]  
public void DangerousAddRef (ref bool success)
```

Summary

Manually increments the reference counter on System.Runtime.InteropServices.SafeHandle instances.

Parameters

Parameter	Description
<i>success</i>	true if the reference counter was successfully incremented; otherwise, false.

Description

The System.Runtime.InteropServices.SafeHandle.DangerousAddRef method prevents the common language infrastructure from reclaiming memory used by a handle (which occurs when the runtime calls the System.Runtime.InteropServices.SafeHandle.ReleaseHandle method). You can use this method to manually increment the reference count on a System.Runtime.InteropServices.SafeHandle instance. System.Runtime.InteropServices.SafeHandle.DangerousAddRef returns a Boolean value using a ref parameter (*success*) that indicates whether the reference count was incremented successfully. This allows your program logic to back out in case of failure. You should set *success* to false before calling System.Runtime.InteropServices.SafeHandle.DangerousAddRef. If *success* is true, avoid resource leaks by matching the call to System.Runtime.InteropServices.SafeHandle.DangerousAddRef with a corresponding call to System.Runtime.InteropServices.SafeHandle.DangerousRelease.

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: System.Security.Permissions.SecurityAction.LinkDemand. Associated enumeration:

	System.Security.Permissions.SecurityPermissionFlag. UnmanagedCode
--	--

1

2

SafeHandle.DangerousGetHandle() Method

```
[ILAsm]
.method public hidebysig instance native int DangerousGetHandle() cil
managed

[C#]
public IntPtr DangerousGetHandle ()
```

Summary

Returns the value of the `System.Runtime.InteropServices.SafeHandle.handle` field.

Return Value

An `IntPtr` representing the value of the `System.Runtime.InteropServices.SafeHandle.handle` field. If the handle has been marked invalid with `System.Runtime.InteropServices.SafeHandle.SetHandleAsInvalid`, this method still returns the original handle value, which can be a stale value.

Description

You can use this method to retrieve the actual handle value from an instance of the `System.Runtime.InteropServices.SafeHandle` derived class. This method is needed for backwards compatibility because some properties in the standard return `IntPtr` handle types. `IntPtr` handle types are platform-specific types used to represent a pointer or a handle.

[Note: Using the `System.Runtime.InteropServices.SafeHandle.DangerousGetHandle` method can pose security risks because, if the handle has been marked as invalid with `System.Runtime.InteropServices.SafeHandle.SetHandleAsInvalid`, `System.Runtime.InteropServices.SafeHandle.DangerousGetHandle` still returns the original, potentially stale handle value. The returned handle can also be recycled at any point. At best, this means the handle might suddenly stop working. At worst, if the handle or the resource that the handle represents is exposed to untrusted code, this can lead to a recycling security attack on the reused or returned handle. For example, an untrusted caller can query data on the handle just returned and receive information for an entirely unrelated resource. See the `System.Runtime.InteropServices.SafeHandle.DangerousAddRef` and the `System.Runtime.InteropServices.SafeHandle.DangerousRelease` methods for more information about using the `System.Runtime.InteropServices.SafeHandle.DangerousGetHandle` methods safely.]

Permissions

Permission	Description
------------	-------------

**System.Security.Permissions.
SecurityPermission**

for permission to call unmanaged code. Security action:
System.Security.Permissions.SecurityAction.
LinkDemand. Associated enumeration:
System.Security.Permissions.SecurityPermissionFlag.
UnmanagedCode

1

2

1 SafeHandle.DangerousRelease() Method

```
2 [ILAsm]  
3 .method public hidebysig instance void DangerousRelease() cil managed  
4 internalcall  
  
5 [C#]  
6 public void DangerousRelease ()
```

7 Summary

8 Manually decrements the reference counter on a
9 System.Runtime.InteropServices.SafeHandle instance.

10 Description

11 The System.Runtime.InteropServices.SafeHandle.DangerousRelease method is the
12 counterpart to System.Runtime.InteropServices.SafeHandle.DangerousAddRef. You
13 should always match a call to the
14 System.Runtime.InteropServices.SafeHandle.DangerousRelease method with a
15 previous successful call to
16 System.Runtime.InteropServices.SafeHandle.DangerousAddRef.

17 Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: System.Security.Permissions.SecurityAction.LinkDemand. Associated enumeration: System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode

18

19

SafeHandle.Dispose() Method

```
[ILAsm]
.method public hidebysig newslot virtual final instance void Dispose() cil
managed
[C#]
public void Dispose ()
```

Summary

Releases all resources used by the `System.Runtime.InteropServices.SafeHandle` class.

Description

Calling the `System.Runtime.InteropServices.SafeHandle.Close` or `System.Runtime.InteropServices.SafeHandle.Dispose` method allows the resources to be freed. This might not happen immediately if other threads are using the same instance of the safe handle, but will happen as soon as that is no longer the case. Although most classes using `System.Runtime.InteropServices.SafeHandle` do not need to provide a finalizer, this is sometimes necessary (for example, to flush out file buffers or to write some data back into memory). In this case, such classes can provide a finalizer that is guaranteed to run before the `System.Runtime.InteropServices.SafeHandle` critical finalizer runs.

Call the `System.Runtime.InteropServices.SafeHandle.Close` or `System.Runtime.InteropServices.SafeHandle.Dispose` method when you are finished using the `System.Runtime.InteropServices.SafeHandle` object. The `System.Runtime.InteropServices.SafeHandle.Close` method leaves the `System.Runtime.InteropServices.SafeHandle` object in an unusable state.

[Note: Always call the `System.Runtime.InteropServices.SafeHandle.Close` or `System.Runtime.InteropServices.SafeHandle.Dispose` method before you release your last reference to the `System.Runtime.InteropServices.SafeHandle` object. Otherwise, the resources it is using will not be freed until the garbage collector calls the `System.Runtime.InteropServices.SafeHandle` object's `System.Runtime.InteropServices.SafeHandle.Finalize` method.]

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: <code>System.Security.Permissions.SecurityAction.LinkDemand</code> . Associated enumeration: <code>System.Security.Permissions.SecurityPermissionFlag</code> .

	UnmanagedCode
--	---------------

1

2

SafeHandle.Dispose(System.Boolean) Method

```
[ILAsm]  
.method family hidebysig newslot virtual instance void Dispose(bool  
disposing) cil managed  
  
[C#]  
protected virtual void Dispose (bool disposing)
```

Summary

Releases the unmanaged resources used by the `System.Runtime.InteropServices.SafeHandle` class specifying whether to perform a normal dispose operation.

Parameters

Parameter	Description
<i>disposing</i>	true for a normal dispose operation; false to finalize the handle.

Description

You should never explicitly call the `System.Runtime.InteropServices.SafeHandle.Dispose` method with the *disposing* parameter set to false.

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: <code>System.Security.Permissions.SecurityAction.LinkDemand</code> . Associated enumeration: <code>System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode</code>

1 SafeHandle.Finalize() Method

```
2 [ILAsm]  
3 .method family hidebysig virtual instance void Finalize() cil managed  
4 [C#]  
5 ~SafeHandle ( )
```

6 Summary

7 Frees all resources associated with the handle.

8 Description

9 The `System.Runtime.InteropServices.SafeHandle.Finalize` method is the
10 destructor for the `System.Runtime.InteropServices.SafeHandle` class. Application
11 code should not call this method directly.

12 Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: <code>System.Security.Permissions.SecurityAction.LinkDemand</code> . Associated enumeration: <code>System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode</code>

13

14

SafeHandle.ReleaseHandle() Method

```
[ILAsm]  
.method family hidebysig newslot abstract virtual instance bool  
ReleaseHandle() cil managed  
  
[C#]  
protected abstract bool ReleaseHandle ()
```

Summary

When overridden in a derived class, executes the code required to free the handle.

Return Value

true if the handle is released successfully; otherwise, in the event of a catastrophic failure, false. In this case, it generates a ReleaseHandleFailed Managed Debugging Assistant.

Description

The System.Runtime.InteropServices.SafeHandle.ReleaseHandle method is guaranteed to be called only once and only if the handle is valid as defined by the System.Runtime.InteropServices.SafeHandle.IsValid property. Implement this method in your System.Runtime.InteropServices.SafeHandle derived classes to execute any code that is required to free the handle. Because one of the functions of System.Runtime.InteropServices.SafeHandle is to guarantee prevention of resource leaks, the code in your implementation of System.Runtime.InteropServices.SafeHandle.ReleaseHandle must never fail. The garbage collector calls System.Runtime.InteropServices.SafeHandle.ReleaseHandle after normal finalizers have been run for objects that were garbage collected at the same time. The garbage collector guarantees the resources to invoke this method and that the method will not be interrupted while it is in progress.

Additionally, for simple cleanup (for example, calling the Win32 API CloseHandle on a file handle) you can check the return value for the single platform invoke call. For complex cleanup, you may have a lot of program logic and many method calls, some of which might fail. You must ensure that your program logic has fallback code for each of those cases.

Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: System.Security.Permissions.SecurityAction.LinkDemand. Associated enumeration: System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode

1

2

1 SafeHandle.SetHandle(System.IntPtr)

2 Method

```
3 [ILAsm]  
4 .method family hidebysig instance void SetHandle(native int handle) cil  
5 managed  
  
6 [C#]  
7 protected void SetHandle (IntPtr handle)
```

8 Summary

9 Sets the handle to the specified pre-existing handle.

10 Parameters

Parameter	Description
<i>handle</i>	The pre-existing handle to use.

11 Description

13 Use the `System.Runtime.InteropServices.SafeHandle.SetHandle` method only if you
14 need to support a pre-existing handle.

15 Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: <code>System.Security.Permissions.SecurityAction.LinkDemand</code> . Associated enumeration: <code>System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode</code>

16
17

1 SafeHandle.SetHandleAsInvalid() Method

```
2 [ILAsm]  
3 .method public hidebysig instance void SetHandleAsInvalid() cil managed  
4 internalcall  
  
5 [C#]  
6 public void SetHandleAsInvalid ( )
```

7 Summary

8 Marks a handle as no longer used.

9 Description

10 Call the `System.Runtime.InteropServices.SafeHandle.SetHandleAsInvalid` method
11 only when you know that your handle no longer references a resource. Doing so does
12 not change the value of the `System.Runtime.InteropServices.SafeHandle.handle`
13 field; it only marks the handle as closed. The handle might then contain a potentially
14 stale value. The effect of this call is that no attempt is made to free the resources.

15
16 As with the `System.Runtime.InteropServices.SafeHandle.SetHandle` method, use
17 `System.Runtime.InteropServices.SafeHandle.SetHandleAsInvalid` only if you need
18 to support a pre-existing handle.

19 Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: <code>System.Security.Permissions.SecurityAction.LinkDemand</code> . Associated enumeration: <code>System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode</code>

20
21

1 SafeHandle.IsClosed Property

```
2 [ILAsm]  
3 .property instance bool IsClosed  
4 [C#]  
5 public bool IsClosed { get; }
```

6 Summary

7 Gets a value indicating whether the handle is closed.

8 Property Value

9 true if the handle is closed; otherwise, false.

10 Description

11 The `System.Runtime.InteropServices.SafeHandle.IsClosed` method returns a value
12 indicating whether the `System.Runtime.InteropServices.SafeHandle` object's handle
13 is no longer associated with a native resource. This differs from the definition of the
14 `System.Runtime.InteropServices.SafeHandle.IsInvalid` property, which computes
15 whether a given handle is always considered invalid. The
16 `System.Runtime.InteropServices.SafeHandle.IsClosed` method returns a true value
17 in the following cases:

- 18 • The `System.Runtime.InteropServices.SafeHandle.SetHandleAsInvalid` method
19 was called.
- 20 • The `System.Runtime.InteropServices.SafeHandle.Dispose` method or
21 `System.Runtime.InteropServices.SafeHandle.Close` method was called and there
22 are no references to the `System.Runtime.InteropServices.SafeHandle` object on
23 other threads.

24 Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: <code>System.Security.Permissions.SecurityAction.LinkDemand</code> . Associated enumeration: <code>System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode</code>

25

26

1 SafeHandle.IsInvalid Property

```
2 [ILAsm]  
3 .property instance bool IsInvalid  
4 [C#]  
5 public abstract bool IsInvalid { get; }
```

6 Summary

7 When overridden in a derived class, gets a value indicating whether the handle value is
8 invalid.

9 Property Value

10 true if the handle value is invalid; otherwise, false.

11 Description

12 Derived classes must implement the
13 System.Runtime.InteropServices.SafeHandle.IsInvalid property so that the
14 common language infrastructure can determine whether critical finalization is required.
15 Derived classes must provide an implementation that suits the general type of handle
16 they support (0 or -1 is invalid). These classes can then be further derived for specific
17 safe handle types.

18 Unlike the System.Runtime.InteropServices.SafeHandle.IsClosed property, which
19 reports whether the System.Runtime.InteropServices.SafeHandle object has finished
20 using the underlying handle, the
21 System.Runtime.InteropServices.SafeHandle.IsInvalid property calculates
22 whether the given handle value is always considered invalid. Therefore, the
23 System.Runtime.InteropServices.SafeHandle.IsInvalid property always returns
24 the same value for any one handle value.
25

26 Permissions

Permission	Description
System.Security.Permissions.SecurityPermission	for permission to call unmanaged code. Security action: System.Security.Permissions.SecurityAction.LinkDemand. Associated enumeration: System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode

27
28