System.Runtime.InteropServices.SafeHandle Class

```
[ILAsm]
.class public abstract beforefieldinit SafeHandle extends System.Object
implements System.IDisposable

[C#]
public abstract class SafeHandle: System.Object, IDisposable
```

Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 4.0.0.0
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- **System.IDisposable**

Summary

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

Inherits From: System.Object

Library: RuntimeInfrastructure

Permissions

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

Description

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

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

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

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

How and When to Override

When you inherit from `System.Runtime.InteropServices.SafeHandle`, you must override the following members: `System.Runtime.InteropServices.SafeHandle.IsInvalid` and `System.Runtime.InteropServices.SafeHandle.ReleaseHandle`.

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

SafeHandle(System.IntPtr, System.Boolean) Constructor

```
[ILAsm]
.method family hidebysig specialname rtspecialname instance void
.ctor(native int invalidHandleValue, bool ownsHandle) cil managed

[C#]
protected SafeHandle (IntPtr invalidHandleValue, bool ownsHandle)
```

Summary

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

Parameters

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

Description

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

Exceptions

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

Permissions

Permission	Description

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

1

2

SafeHandle.handle Field

```
[ILAsm]
.field family native int handle

[C#]
protected IntPtr handle
```

Summary

Specifies the handle to be wrapped.

Description

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

SafeHandle.Close() Method

```
[ILAsm]
.method public hidebysig instance void Close() cil managed

[C#]
public void Close ()
```

Summary

Marks the handle for releasing and freeing resources.

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 safe handle object, but will happen as soon as that is no longer the case. Although most classes that use the `System.Runtime.InteropServices.SafeHandle` class 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.

[*Note:* Always call `System.Runtime.InteropServices.SafeHandle.Close` or `System.Runtime.InteropServices.SafeHandle.Dispose` 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: `System.Security.Permissions.SecurityAction.LinkDemand`. Associated enumeration: `System.Security.Permissions.SecurityPermissionFlag.UnmanagedCode`

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
success	`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` methodsafely.

]

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.DangerousRelease() Method

```
[ILAsm]
.method public hidebysig instance void DangerousRelease() cil managed
internalcall

[C#]
public void DangerousRelease ()
```

Summary

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

Description

The `System.Runtime.InteropServices.SafeHandle.DangerousRelease` method is the counterpart to `System.Runtime.InteropServices.SafeHandle.DangerousAddRef`. You should always match a call to the `System.Runtime.InteropServices.SafeHandle.DangerousRelease` method with a previous successful call to `System.Runtime.InteropServices.SafeHandle.DangerousAddRef`.

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`

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: `System.Security.Permissions.SecurityAction.LinkDemand`. Associated enumeration: `System.Security.Permissions.SecurityPermissionFlag`.

	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
disposing	`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: `System.Security.Permissions.SecurityAction. LinkDemand`. Associated enumeration: `System.Security.Permissions.SecurityPermissionFlag. UnmanagedCode`

SafeHandle.Finalize() Method

```
[ILAsm]
.method family hidebysig virtual instance void Finalize() cil managed

[C#]
~SafeHandle ()
```

Summary

Frees all resources associated with the handle.

Description

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

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`

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.IsInvalid` 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

SafeHandle.SetHandle(System.IntPtr) Method

```
[ILAsm]
.method family hidebysig instance void SetHandle(native int handle) cil
managed

[C#]
protected void SetHandle (IntPtr handle)
```

Summary

Sets the handle to the specified pre-existing handle.

Parameters

Parameter	Description
handle	The pre-existing handle to use.

Description

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

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`

SafeHandle.SetHandleAsInvalid() Method

```
[ILAsm]
.method public hidebysig instance void SetHandleAsInvalid() cil managed
internalcall

[C#]
public void SetHandleAsInvalid ()
```

Summary

Marks a handle as no longer used.

Description

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

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

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`

SafeHandle.IsClosed Property

```
[ILAsm]
.property instance bool IsClosed

[C#]
public bool IsClosed { get; }
```

Summary

Gets a value indicating whether the handle is closed.

Property Value

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

Description

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

- The `System.Runtime.InteropServices.SafeHandle.SetHandleAsInvalid` method was called.
- The `System.Runtime.InteropServices.SafeHandle.Dispose` method or `System.Runtime.InteropServices.SafeHandle.Close` method was called and there are no references to the `System.Runtime.InteropServices.SafeHandle` object on other threads.

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`

SafeHandle.IsInvalid Property

```
[ILAsm]
.property instance bool IsInvalid

[C#]
public abstract bool IsInvalid { get; }
```

Summary

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

Property Value

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

Description

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

Unlike the `System.Runtime.InteropServices.SafeHandle.IsClosed` property, which reports whether the `System.Runtime.InteropServices.SafeHandle` object has finished using the underlying handle, the `System.Runtime.InteropServices.SafeHandle.IsInvalid` property calculates whether the given handle value is always considered invalid. Therefore, the `System.Runtime.InteropServices.SafeHandle.IsInvalid` property always returns the same value for any one handle value.

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`