System.Delegate Class

[ILAsm]
```
.class public abstract serializable Delegate extends System.Object
implements System.ICloneable
```

[C#]
```
public abstract class Delegate: ICloneable
```

Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 2.0.x.x
- *Attributes:*
    - CLSCompliantAttribute(true)

Implements:

- **System.ICloneable**

Summary

A class used to create types that invoke methods.

Inherits From: System.Object

Library: BCL

Description

Delegate types derive from the `System.Delegate` class. The declaration of a delegate type establishes a contract that specifies the signature of one or more methods. [*Note:* For an example of a delegate type declaration, see the examples at the end of this topic.]

Delegate types are implicitly sealed: it is not permissible to derive a new type from a delegate type. [*Note:* The `System.Delegate` class is not considered a delegate type; it is a class used to derive delegate types.]

[*Note:* For information on subclassing the Delegate class, see Partition II of the CLI Specification.]

A delegate is an instance of a delegate type. A non-null delegate references an *invocation list*, which is made up of one or more entries. Each entry consists of a pair of values: a non-null method, and a corresponding object, called the *target*. If the method is static, the corresponding target is `null`, otherwise the target is the instance on which the method is to be called.

The signature of each method in the invocation list is required to exactly match the signature specified by the delegate's type.

When a delegate is invoked, the methods in the corresponding invocation list are invoked in the order in which they appear in that list. A delegate attempts to invoke every method in its invocation list, with duplicate methods being invoked once for each occurrence in that list.

Delegates are immutable; once created, the invocation list of a delegate does not change. Combining operations, such as `System.Delegate.Combine` and `System.Delegate.Remove`, cannot alter existing delegates. Instead, such operations result in the return of either a new delegate that contains the results of the operation, an existing delegate, or the null value. [*Note:* A combining operation returns the null value when the result of the operation is an empty invocation list. A combining operation returns an existing delegate when the requested operation has no effect (for example, if an attempt is made to remove a nonexistent entry).]

If an invoked method throws an exception, the method stops executing and the exception is passed back to the caller of the delegate. The delegate does not continue invoking methods from its invocation list. Catching the exception in the caller does not alter this behavior. It is possible that non-standard methods that implement combining operations allow the creation of delegates with different behavior. When this is the case, the non-standard methods are required to specify the behavior.

When the signature of the methods invoked by a delegate includes a return value, the delegate returns the return value of the last element in the invocation list. When the signature includes a parameter that is passed by reference, the final value of the parameter is the result of every method in the invocation list executing sequentially and updating the parameter's value. [*Note:* For an example that demonstrates this behavior, see Example 2.]

Example

`Example1:`

The following example creates two delegates. The first delegate invokes a static method, and the second invokes an instance method on a target object.

[C#]

```
using System;
public delegate string DelegatedMethod(string s);
class MyClass {
 public static string StaticMethod(string s) {
 return ("Static method Arg=" + s);
 }
 public string InstanceMethod(string s) {
 return ("Instance method Arg=" + s);
 }
}
class TestClass {
 public static void Main() {
 MyClass myInstance = new MyClass();
 //Create delegates from delegate type DelegatedMethod.
 DelegatedMethod delStatic = new DelegatedMethod(MyClass.StaticMethod);
 DelegatedMethod delInstance = new
DelegatedMethod(myInstance.InstanceMethod);
 //Invoke the methods referenced by the delegates.
 Console.WriteLine (delStatic("Call 1"));
 Console.WriteLine (delInstance ("Call 2"));
 }
}
```

The output is

```
Static method Arg=Call 1


Instance method Arg=Call 2
```

Example2:

The following example shows the return value and the final value of a parameter that is passed by reference to a delegate that invokes multiple methods.

[C#]

```
using System;
class MyClass {
 public int Increment(ref int i) {
   Console.WriteLine("Incrementing {0}",i);
   return (i++);
 }
 public int Negate(ref int i) {
   Console.WriteLine("Negating {0}",i);
   i = i * -1;
   return i;
 }
}

public delegate int DelegatedMethod(ref int i);
class TestClass {
 public static void Main() {
   MyClass myInstance = new MyClass();
   DelegatedMethod delIncrementer = new
DelegatedMethod(myInstance.Increment);
   DelegatedMethod delNegater = new DelegatedMethod(myInstance.Negate);
   DelegatedMethod d = (DelegatedMethod)
Delegate.Combine(delIncrementer, delNegater);
   int i = 1;
   Console.WriteLine("Invoking delegate using ref value {0}",i);
   int retvalue = d(ref i);
   Console.WriteLine("After Invoking delegate i = {0} return value is
{1}",i, retvalue);
  }
}
```

The output is

```
Invoking delegate using ref value 1


Incrementing 1


Negating 2


After Invoking delegate i = -2 return value is -2
```

Delegate.Clone() Method

[ILAsm]
```
.method public hidebysig virtual object Clone()
```

[C#]
```
public virtual object Clone()
```

Summary

Creates a copy of the current instance.

Return Value

A `System.Object` that is a copy of the current instance.

Description

The `System.Delegate.Clone` method creates a new instance of the same type as the current instance and then copies the contents of each of the current instance's non-static fields.

[*Note:* This method is implemented to support the `System.ICloneable` interface.]

Behaviors

The returned object must have the exact same type and invocation list as the current instance.

Default

The default implementation of the `System.Delegate.Clone` method creates a new instance, which is the exact same type as the current instance, and then copies the contents of each of the current instance's non-static fields. If the field is a value type, a bit-by-bit copy of the field is performed. If the field is a reference type, the object referenced by the field is not copied; instead, the returned object contains a copy of the reference. This behavior is identical to `System.Object.MemberwiseClone`.

How and When to Override

Subclasses of `System.Delegate` should override `System.Delegate.Clone` to customize the way in which copies of the subclass are constructed.

Delegate.Combine(System.Delegate, System.Delegate) Method

```
[ILAsm]
.method public hidebysig static class System.Delegate Combine(class
System.Delegate a, class System.Delegate b)

[C#]
public static Delegate Combine(Delegate a, Delegate b)
```

Summary

Concatenates the invocation lists of the specified delegates.

Parameters

Parameter	Description
a	The delegate whose invocation list will be first in the invocation list of the new delegate.
b	The delegate whose invocation list will be last in the invocation list of the new delegate.

Return Value

A delegate, or `null`.

The following table describes the value returned when *a* or *b* is `null`.

a	b	Return Value
null	null	null
null	non-null	*b*
non-null	null	*a*

When *a* and *b* are non-null, this method returns a new delegate with the concatenated invocation lists of *a* and *b*.

Description

Unless *a* or *b* is `null`, *a* and *b* are required to be the exact same type.

Consider the following situation, in which D1, D2, D3, D4, and D5 are delegate instances of the same type, D1's invocation list has one entry, E1, and D2's invocation list has one entry, E2.

Then, D3 = Combine(D1, D2) results in D3's having an invocation list of E1 + E2.

Then, D4 = Combine(D2, D1) results in D4's having an invocation list of E2 + E1.

Then, D5 = Combine(D3, D4) results in D5's having an invocation list of E1 + E2 + E2 + E1.

[*Note:* The invocation list of the returned delegate can contain duplicate methods.

`System.Delegate.Combine` is useful for creating event handlers that call multiple methods each time an event occurs.

]

Exceptions

Exception	**Condition**
System.ArgumentException	*a* and *b* are not `null` and not of the same type.

Delegate.Combine(System.Delegate[]) Method

```
[ILAsm]
.method public hidebysig static class System.Delegate Combine(class
System.Delegate[] delegates)

[C#]
public static Delegate Combine(Delegate[] delegates)
```

Summary

Concatenates the invocation lists of the specified delegates.

Parameters

Parameter	Description
delegates	An array of delegates of the exact same type.

Return Value

A new delegate, or `null` if *delegates* is `null` or has only `null` elements.

Description

The invocation list of the returned delegate is constructed by concatenating the invocation lists of the delegates in *delegates*, in increasing subscript order. For example, consider the following situation, in which the elements of delegates have the following invocation lists (where En represents an entry in an invocation list, and null represents an empty invocation list): [0] = E1, [1] = null, [2] = E2 + E3, and [3] = E4 + E5 + E6. When these elements are combined, the resulting delegate contains the invocation list E1 + E2 + E3 + E4 + E5 + E6.

Null elements in *delegates* are not included in the returned delegate.

[*Note:* The invocation list of the returned delegate can contain duplicate methods.]

Exceptions

Exception	Condition
System.ArgumentException	The non-`null` delegates in *delegates* are not of the

	same type.

The following member must be implemented if the Reflection library is present in the implementation.

Delegate.CreateDelegate(System.Type, System.Object, System.String) Method

```
[ILAsm]
.method public hidebysig static class System.Delegate
CreateDelegate(class System.Type type, object target, string method)

[C#]
public static Delegate CreateDelegate(Type type, object target,
string method)
```

Summary

Returns a new delegate with the specified target and instance method as its invocation list.

Parameters

Parameter	Description
type	The `System.Type` of the delegate to return. This `System.Type` is required to derive from `System.Delegate`.
target	An instance of an object that implements *method*.
method	A `System.String` containing the name of the instance method to be invoke on *target*.

Return Value

A `System.Delegate` of type *type* that invokes *method* on *target*.

Description

[*Note:* This method is used to dynamically create delegates that invoke instance methods. To create a delegate that invokes static methods, see `System.Delegate.CreateDelegate`(`System.Type`, `System.Type`, `System.String`).]

Exceptions

Exception	Condition

System.ArgumentNullException	*type, target, or method* is `null`.
System.ArgumentException	*type* does not derive from `System.Delegate`. -or- *method* is not an instance method. -or- *target* does not implement *method*.
System.MethodAccessException	The caller does not have the required permission.

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to access type information. See `System.Security.Permissions.ReflectionPermissionFlag.MemberAccess`

The following member must be implemented if the Reflection library is present in the implementation.

Delegate.CreateDelegate(System.Type, System.Type, System.String) Method

```
[ILAsm]
.method public hidebysig static class System.Delegate
CreateDelegate(class System.Type type, class System.Type target,
string method)

[C#]
public static Delegate CreateDelegate(Type type, Type target, string
method)
```

Summary

Returns a new delegate with the specified static method as its invocation list.

Parameters

Parameter	Description
type	The `System.Type` of delegate to return. This `System.Type` is required to derive from `System.Delegate`.
target	A `System.Type` representing the class that implements *method*.
method	A `System.String` containing the name of the static method implemented by *target*.

Return Value

A `System.Delegate` of type *type* that invokes *method.*

Description

[*Note:* This method is used to dynamically create delegates that invoke static methods. To create a delegate that invokes instance methods, see `System.Delegate.CreateDelegate`(`System.Type`, `System.Object`, `System.String`).]

Exceptions

Exception	Condition

System.ArgumentNullException	*type, target, or method* is `null`.
System.ArgumentException	*type* does not derive from `System.Delegate`. -or- *method* is not a static method. -or- *target* does not implement *method*.
System.MethodAccessException	The caller does not have the required permission.

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to access type information. See `System.Security.Permissions.ReflectionPermissionFlag.MemberAccess`

The following member must be implemented if the Reflection library is present in the implementation.

Delegate.CreateDelegate(System.Type, System.Reflection.MethodInfo) Method

```
[ILAsm]
.method public hidebysig static class System.Delegate
CreateDelegate(class System.Type type, class
System.Reflection.MethodInfo method)

[C#]
public static Delegate CreateDelegate(Type type, MethodInfo method)
```

Summary

Returns a new delegate with the specified static method as its invocation list.

Parameters

Parameter	Description
type	The `System.Type` of `System.Delegate` to return. This `System.Type` is required to derive from `System.Delegate`.
method	A `System.Reflection.MethodInfo` that reflects a static method.

Return Value

A `System.Delegate` of type *type* that invokes *method.*

Description

[*Note:* This method is used to dynamically create delegates that invoke static methods. To create a delegate that invokes instance methods, see `System.Delegate.CreateDelegate(System.Type, System.Object, System.String)`.]

Exceptions

Exception	Condition
System.ArgumentNullException	*type* or *method* is `null`.
System.ArgumentException	*type* does not derive from `System.Delegate`.

	-or- *method* does not reflect a static method.
System.InvalidProgramException	The `Invoke` method of the *type* delegate was not found.
System.MethodAccessException	The caller does not have the required permission.

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to access type information. See `System.Security.Permissions.ReflectionPermissionFlag.MemberAccess`

The following member must be implemented if the Reflection library is present in the implementation.

Delegate.DynamicInvoke(System.Object[]) Method

```
[ILAsm]
.method public hidebysig instance object DynamicInvoke(object[]
args)

[C#]
public object DynamicInvoke(object[] args)
```

Summary

Causes a delegate to invoke the methods in its invocation list using the specified arguments.

Parameters

Parameter	Description
args	An array of `System.Object` instances that are to be passed to the methods in the invocation list of the current instance. Specify `null` if the methods invoked by the current instance do not take arguments.

Return Value

The `System.Object` returned by the last method in the invocation list of the current instance.

Exceptions

Exception	Condition
System.ArgumentException	The type of one or more elements in *args* is invalid as a parameter to the methods implemented by the current instance.
System.MethodAccessException	The caller does not have the required permissions. -or- The number, order or type of parameters

	listed in *args* is invalid.
System.Reflection.TargetException	A method in the invocation list of the current instance is an instance method and its target object is `null`. -or- A method in the invocation list of the current instance was invoked on a target object or a class that does not implement it.
System.Reflection. TargetParamterCountException	The number of elements in *args* is not equal to the number of parameters required by the methods invoked by the current instance.
System.Reflection. TargetInvocationException	A method in the invocation list of the current instance threw an exception.

Delegate.Equals(System.Object) Method

```
[ILAsm]
.method public hidebysig virtual bool Equals(object obj)

[C#]
public override bool Equals(object obj)
```

Summary

Determines whether the specified object is equal to the current instance.

Parameters

Parameter	Description
obj	The `System.Object` to compare with the current instance.

Return Value

`true` if *obj* is equal to the current instance, otherwise `false`.

Description

Two delegates are equal if they are not null and are of the exact same type, their invocation lists contain the same number of elements, and every element in the invocation list of the first delegate is equal to the element in the corresponding position in the invocation list of the second delegate.

Two invocation list elements are equal if they invoke the same instance method on the same target instance, or they invoke the same static method.

[*Note:* This method overrides `System.Object.Equals`.]

Delegate.GetHashCode() Method

```
[ILAsm]
.method public hidebysig virtual int32 GetHashCode()

[C#]
public override int GetHashCode()
```

Summary

Generates a hash code for the current instance.

Return Value

A `System.Int32` containing the hash code for this instance.

Description

The algorithm used to generate the hash code is unspecified.

[*Note:* This method overrides `System.Object.GetHashCode`.]

Delegate.GetInvocationList() Method

[ILAsm]

```
.method public hidebysig virtual class System.Delegate[]
GetInvocationList()
```

[C#]

```
public virtual Delegate[] GetInvocationList()
```

Summary

Returns the invocation list of the current delegate.

Return Value

An ordered set of `System.Delegate` instances whose invocation lists collectively match those of the current delegate.

Behaviors

The array contains a set of delegates, each having an invocation list of one entry. Invoking these delegates sequentially, in the order in which they appear in the array, produces the same results as invoking the current delegate.

How and When to Override

Override `System.Delegate.GetInvocationList` when subclassing Delegate.

Delegate.op_Equality(System.Delegate, System.Delegate) Method

[ILAsm]
```
.method public hidebysig static specialname bool op_Equality(class
System.Delegate d1, class System.Delegate d2)
```

[C#]
```
public static bool operator ==(Delegate d1, Delegate d2)
```

Summary

Determines whether the specified delegates are equal.

Parameters

Parameter	**Description**
d1	The first delegate to compare.
d2	The second delegate to compare.

Return Value

`true` if *d1*.Equals(*d2*) returns `true`; otherwise, `false`.

Description

[*Note:* See `System.Delegate.Equals`.]

Delegate.op_Inequality(System.Delegate, System.Delegate) Method

[ILAsm]

```
.method public hidebysig static specialname bool op_Inequality(class
System.Delegate d1, class System.Delegate d2)
```

[C#]

```
public static bool operator !=(Delegate d1, Delegate d2)
```

Summary

Determines whether the specified Delegates are not equal.

Parameters

Parameter	Description
d1	The first delegate to compare.
d2	The second delegate to compare.

Return Value

`true` if *d1*.Equals(*d2*) returns `false`; otherwise, `false`.

Description

[*Note:* See `System.Delegate.Equals`.]

Delegate.Remove(System.Delegate, System.Delegate) Method

[ILAsm]
```
.method public hidebysig static class System.Delegate Remove(class
System.Delegate source, class System.Delegate value)
```

[C#]
```
public static Delegate Remove(Delegate source, Delegate value)
```

Summary

Removes the invocation list of a `System.Delegate` from the invocation list of another delegate.

Parameters

Parameter	Description
source	The delegate from which to remove the invocation list of *value*.
value	The delegate that supplies the invocation list to remove from *source*.

Return Value

Returns a new delegate, *source*, or `null`.

If *source* and *value* are not `null`, are not equal, and the invocation list of *value* is contained in the invocation list of source, returns a new delegate with the invocation list of *value* removed from the invocation list of *source*.

If the invocation lists of *source* and *value* are equal, returns `null`.

If the invocation list of *value* is not found in the invocation list of *source*, returns *source*.

The following table describes the value returned when *source* or *value* is `null`.

	value	**Return value**
null	null	null
null	non-null	null
non-null	null	*source*

Description

The invocation list of *value* is required to be an exact match of a contiguous set of elements in the invocation list of *source*. If the invocation list of *value* occurs more than once in the invocation list of *source*, the last occurrence is removed.

Example

The following example demonstrates the `System.Delegate.Remove` method.

[C#]

```
using System;
class MyClass {
    public string InstanceMethod(string s) {
    return ("Instance String " + s);
    }
}
class MyClass2 {
    public string InstanceMethod2(string s) {
    return ("Instance String2 " + s);
    }
}
public delegate string DelegatedMethod(string s);

class TestClass {
    public static void WriteDelegate (string label, Delegate d) {
    Console.WriteLine("Invocation list targets for {0}:",label);
    foreach(Delegate x in d.GetInvocationList())
        Console.WriteLine("{0}",x.Target);
    }

    public static void Main() {
    MyClass myInstance = new MyClass();
    DelegatedMethod delInstance = new
DelegatedMethod(myInstance.InstanceMethod);
    MyClass2 myInstance2 = new MyClass2();
    DelegatedMethod delInstance2 = new
DelegatedMethod(myInstance2.InstanceMethod2);
    DelegatedMethod [] sourceArray = {delInstance, delInstance2,
delInstance2, delInstance};
    DelegatedMethod [] remove1 = {delInstance};
    DelegatedMethod [] remove2 = {delInstance2, delInstance2};
    DelegatedMethod [] remove3 = {delInstance2, delInstance};
    DelegatedMethod [] remove4 = {delInstance, delInstance2};
    DelegatedMethod [] remove5 = {delInstance, delInstance};
    Delegate source = Delegate.Combine(sourceArray);
    // Display invocation list of source
    TestClass.WriteDelegate("source", source);
    //Test 1: value occurs in source twice.
    Delegate value1 = Delegate.Combine(remove1);
    Delegate result1 = Delegate.Remove(source, value1);
    TestClass.WriteDelegate("value1", value1);
    if (result1==null) {
        Console.WriteLine("removal test 1 result is null");
    } else {
        TestClass.WriteDelegate("result1", result1);
    }
    //Test 2: value matches the middle two elements of source.
    Delegate value2 = Delegate.Combine(remove2);
    Delegate result2 = Delegate.Remove(source, value2);
    TestClass.WriteDelegate("value2", value2);
    if (result2==null) {
        Console.WriteLine("removal test 2 result2 is null");
    } else {
        TestClass.WriteDelegate("result2", result2);
    }
    //Test 3: value matches the last two elements of source.
    Delegate value3 = Delegate.Combine(remove3);
    Delegate result3 = Delegate.Remove(source, value3);
    TestClass.WriteDelegate("value3", value3);
    if (result3==null) {
        Console.WriteLine("removal test 3 result3 is null");
    } else {
        TestClass.WriteDelegate("result3", result3);
    }
    //Test 4: value matches the first two elements of source.
    Delegate value4 = Delegate.Combine(remove4);
    Delegate result4 = Delegate.Remove(source, value4);
    TestClass.WriteDelegate("value4", value4);
    if (result4==null) {
        Console.WriteLine("removal test 4 result4 is null");
    } else {
        TestClass.WriteDelegate("result4", result4);
    }
    //Test 5: value does not occur in source.
    Delegate value5 = Delegate.Combine(remove5);
    Delegate result5 = Delegate.Remove(source, value5);
    TestClass.WriteDelegate("value5", value5);
    if (result5==null) {
        Console.WriteLine("removal test 5 result5 is null");
    } else {
        TestClass.WriteDelegate("result5", result5);
    }
    //Test 6: value exactly matches source.
    Delegate result6 = Delegate.Remove(source, source);
    TestClass.WriteDelegate("value=source", source);
    if (result6==null) {
        Console.WriteLine("removal test 6 result6 is null");
    } else {

        TestClass.WriteDelegate("result6", result6);
    }
}
}
```

The output is

```
Invocation list targets for source:


MyClass


MyClass2
```

```
MyClass2

MyClass

Invocation list targets for value1:

MyClass

Invocation list targets for result1:

MyClass

MyClass2

MyClass2

Invocation list targets for value2:

MyClass2

MyClass2

Invocation list targets for result2:

MyClass

MyClass

Invocation list targets for value3:

MyClass2

MyClass

Invocation list targets for result3:

MyClass
```

```
MyClass2

Invocation list targets for value4:

MyClass

MyClass2

Invocation list targets for result4:

MyClass2

MyClass

Invocation list targets for value5:

MyClass

MyClass

Invocation list targets for result5:

MyClass

MyClass2

MyClass2

MyClass

Invocation list targets for value=source:

MyClass

MyClass2

MyClass2
```

```
MyClass

removal test 6 result6 is null
```

Delegate.RemoveAll(System.Delegate, System.Delegate) Method

```
[ILAsm]
.method public hidebysig static class System.Delegate
RemoveAll(class System.Delegate source, class System.Delegate value)

[C#]
public static Delegate RemoveAll(Delegate source, Delegate value)
```

Summary

Removes all matching occurrences of the invocation list of a `System.Delegate` from the invocation list of another delegate.

Parameters

Parameter	Description
source	The delegate from which to remove all matching occurrences of the invocation list of *value*.
value	The delegate that supplies the invocation list to remove from *source*.

Return Value

Returns a new delegate, *source*, or `null`.

If *source* and *value* are not `null`, are not equal, and the invocation list of *value* is contained in the invocation list of source, returns a new delegate with all matching occurrences of the invocation list of *value* removed from the invocation list of *source*.

If the invocation lists of *source* and *value* are equal, or if *source* contains only a succession of invocation lists equal to *value*, returns `null`.

If the invocation list of *value* is not found in the invocation list of *source*, returns *source*.

The following table describes the value returned when *source* or *value* is `null`.

	value	**Return value**
null	null	null
null	non-null	null
non-null	null	*source*

Description

The invocation list of *value* is required to be an exact match of a contiguous set of elements in the invocation list of *source*. If the invocation list of *value* occurs more than once in the invocation list of *source*, all occurrences are removed.

The following member must be implemented if the Reflection library is present in the implementation.

Delegate.Method Property

```
[ILAsm]
.property class System.Reflection.MethodInfo Method { public
hidebysig specialname instance class System.Reflection.MethodInfo
get_Method() }

[C#]
public MethodInfo Method { get; }
```

Summary

Gets the last method in a delegate's invocation list.

Property Value

A `System.Reflection.MethodInfo`.

Description

This property is read-only.

Exceptions

Exception	Condition
System.MemberAccessException	The caller does not have the required permissions.

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to access type information. See `System.Security.Permissions.ReflectionPermissionFlag.TypeInformation`.

Delegate.Target Property

[ILAsm]

```
.property object Target { public hidebysig specialname instance
object get_Target() }
```

[C#]

```
public object Target { get; }
```

Summary

Gets the last object upon which a delegate invokes an instance method.

Property Value

A `System.Object` instance, or `null` if the delegate invokes only static methods.

Description

This property is read-only.

If the delegate invokes only static methods, this property returns `null`. If the delegate invokes one or more instance methods, this property returns the target of the last instance method/target pair in the invocation list.

Example

Example 1:

The following example gets the `System.Delegate.Target` property values for two delegates. The first delegate invokes a static method, and the second invokes an instance method.

[C#]

```
using System;
public delegate string DelegatedMethod(string s);
class MyClass {
  public static string StaticMethod(string s) {
    return ("Static method Arg=" + s);
  }
  public string InstanceMethod(string s) {
    return ("Instance method Arg=" + s);
  }
}
class TestClass {
  public static void Main() {
    MyClass myInstance = new MyClass();
     //Create  delegates from delegate type DelegatedMethod.
    DelegatedMethod delStatic = new
DelegatedMethod(MyClass.StaticMethod);
    DelegatedMethod delInstance = new
DelegatedMethod(myInstance.InstanceMethod);
    object t = delStatic.Target;
    Console.WriteLine ("Static target is {0}", t==null ? "null":t);
    t = delInstance.Target;
    Console.WriteLine ("Instance target is {0}", t==null ? "null":t);
    }
}
```

The output is

```
Static target is null


Instance target is MyClass
```

Example 2:

The following example gets the `System.Delegate.Target` property value for three delegates created using instance methods, static methods, and a combination of the two.

[C#]

```
using System;
class MyClass {
  public static string StaticMethod(string s) {
    return ("Static String " + s);
  }
  public string InstanceMethod(string s) {
    return ("Instance String " + s);
  }
}
class MyClass2 {
  public static string StaticMethod2(string s) {
    return ("Static String2 " + s);
  }
  public string InstanceMethod2(string s) {
    return ("Instance String2 " + s);
  }
}
public delegate string DelegatedMethod(string s);

class TestClass {
    public static void Main() {
    DelegatedMethod delStatic = new
DelegatedMethod(MyClass.StaticMethod);
    DelegatedMethod delStatic2 = new
DelegatedMethod(MyClass2.StaticMethod2);

    MyClass myInstance = new MyClass();
    DelegatedMethod delInstance = new
DelegatedMethod(myInstance.InstanceMethod);

    MyClass2 myInstance2 = new MyClass2();
    DelegatedMethod delInstance2 = new
DelegatedMethod(myInstance2.InstanceMethod2);

    Delegate d = Delegate.Combine(delStatic, delInstance );
    Delegate e = Delegate.Combine(delInstance,delInstance2);
    Delegate f = Delegate.Combine(delStatic, delStatic2 );
    if (d!=null) {
        Console.WriteLine("Combined 1 static, 1 instance, same
class:");
        Console.WriteLine("target...{0}", d.Target == null ? "null":
d.Target);
        foreach(Delegate x in d.GetInvocationList())
            Console.WriteLine("invoke element target: {0}",x.Target);

    }
    Console.WriteLine("");
    if (e!=null) {
        Console.WriteLine("Combined 2 instance methods, different
classes:");
        Console.WriteLine("target...{0}", e.Target == null ? "null":
e.Target);
        foreach(Delegate x in e.GetInvocationList())
            Console.WriteLine("invoke element target: {0}",x.Target);
    }
    Console.WriteLine("");
    if (f!=null) {
        Console.WriteLine("Combined 2 static methods, different
classes:");
        Console.WriteLine("target...{0}", f.Target == null ? "null":
f.Target);
        foreach(Delegate x in f.GetInvocationList())
            Console.WriteLine("invoke element target: {0}",x.Target);
    }

    }
}
```

The output is

```
Combined 1 static, 1 instance, same class:


target...MyClass


invoke element target:


invoke element target: MyClass


Combined 2 instance methods, different classes:


target...MyClass2


invoke element target: MyClass
invoke element target: MyClass2

Combined 2 static methods, different classes:

target...null

invoke element target:

invoke element target:
```