

# 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

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

5  
6 The signature of each method in the invocation list is required to exactly match the  
7 signature specified by the delegate's type.

8  
9 When a delegate is invoked, the methods in the corresponding invocation list are  
10 invoked in the order in which they appear in that list. A delegate attempts to invoke  
11 every method in its invocation list, with duplicate methods being invoked once for each  
12 occurrence in that list.

13  
14 Delegates are immutable; once created, the invocation list of a delegate does not  
15 change. Combining operations, such as `System.Delegate.Combine` and  
16 `System.Delegate.Remove`, cannot alter existing delegates. Instead, such operations  
17 result in the return of either a new delegate that contains the results of the operation,  
18 an existing delegate, or the null value. [Note: A combining operation returns the null  
19 value when the result of the operation is an empty invocation list. A combining operation  
20 returns an existing delegate when the requested operation has no effect (for example, if  
21 an attempt is made to remove a nonexistent entry).]

22  
23  
24  
25 If an invoked method throws an exception, the method stops executing and the  
26 exception is passed back to the caller of the delegate. The delegate does not continue  
27 invoking methods from its invocation list. Catching the exception in the caller does not  
28 alter this behavior. It is possible that non-standard methods that implement combining  
29 operations allow the creation of delegates with different behavior. When this is the case,  
30 the non-standard methods are required to specify the behavior.

31  
32 When the signature of the methods invoked by a delegate includes a return value, the  
33 delegate returns the return value of the last element in the invocation list. When the  
34 signature includes a parameter that is passed by reference, the final value of the  
35 parameter is the result of every method in the invocation list executing sequentially and  
36 updating the parameter's value. [Note: For an example that demonstrates this behavior,  
37 see Example 2.]

## 40 Example

41 Example1:

42  
43 The following example creates two delegates. The first delegate invokes a static method,  
44 and the second invokes an instance method on a target object.

45  
46 [C#]

```
47 using System;  
48 public delegate string DelegatedMethod(string s);  
49 class MyClass {  
50     public static string StaticMethod(string s) {
```

```

1    return ("Static method Arg=" + s);
2    }
3    public string InstanceMethod(string s) {
4    return ("Instance method Arg=" + s);
5    }
6    }
7    class TestClass {
8    public static void Main() {
9    MyClass myInstance = new MyClass();
10   //Create delegates from delegate type DelegatedMethod.
11   DelegatedMethod delStatic = new DelegatedMethod(MyClass.StaticMethod);
12   DelegatedMethod delInstance = new
13   DelegatedMethod(myInstance.InstanceMethod);
14   //Invoke the methods referenced by the delegates.
15   Console.WriteLine (delStatic("Call 1"));
16   Console.WriteLine (delInstance ("Call 2"));
17   }
18   }
19   The output is
20
21   Static method Arg=Call 1
22
23
24   Instance method Arg=Call 2
25
26
27   Example2:
28
29   The following example shows the return value and the final value of a parameter that is
30   passed by reference to a delegate that invokes multiple methods.
31
32   [C#]
33   using System;
34   class MyClass {
35   public int Increment(ref int i) {
36   Console.WriteLine("Incrementing {0}",i);
37   return (i++);
38   }
39   public int Negate(ref int i){
40   Console.WriteLine("Negating {0}",i);
41   i = i * -1;
42   return i;
43   }
44   }
45
46   public delegate int DelegatedMethod(ref int i);
47   class TestClass {
48   public static void Main() {
49   MyClass myInstance = new MyClass();
50   DelegatedMethod delIncrementer = new
51   DelegatedMethod(myInstance.Increment);
52   DelegatedMethod delNegater = new DelegatedMethod(myInstance.Negate);
53   DelegatedMethod d = (DelegatedMethod) Delegate.Combine(delIncrementer,
54   delNegater);

```

```
1      int i = 1;
2      Console.WriteLine("Invoking delegate using ref value {0}",i);
3      int retvalue = d(ref i);
4      Console.WriteLine("After Invoking delegate i = {0} return value is {1}",i,
5 retvalue);
6  }
7  }
8  The output is
9
10 Invoking delegate using ref value 1
11
12
13 Incrementing 1
14
15
16 Negating 2
17
18
19 After Invoking delegate i = -2 return value is -2
20
21
```

# 1 Delegate.Clone() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual object Clone()  
  
4 [C#]  
5 public virtual object Clone()
```

## 6 Summary

7 Creates a copy of the current instance.

## 8 Return Value

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

## 10 Description

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

14  
15 *[Note: This method is implemented to support the `System.ICloneable` interface.]*  
16  
17

## 18 Behaviors

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

## 22 Default

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

## 30 How and When to Override

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

1

2

# 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(Delagate a, Delagate b)
```

## Summary

Concatenates the invocation lists of the specified delegates.

## Parameters

Parameter	Description
<i>a</i>	The delegate whose invocation list will be first in the invocation list of the new delegate.
<i>b</i>	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`.

<b>a</b>	<b>b</b>	<b>Return Value</b>
null	null	null
null	non-null	<i>b</i>
non-null	null	<i>a</i>

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
<b>System.ArgumentException</b>	<i>a</i> and <i>b</i> are not <code>null</code> 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(Delagate[] delegates)
```

### Summary

Concatenates the invocation lists of the specified delegates.

### Parameters

Parameter	Description
<i>delegates</i>	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  $E_n$  represents an entry in an invocation list, and null represents an empty invocation list):  $[0] = E_1$ ,  $[1] = \text{null}$ ,  $[2] = E_2 + E_3$ , and  $[3] = E_4 + E_5 + E_6$ . When these elements are combined, the resulting delegate contains the invocation list  $E_1 + E_2 + E_3 + E_4 + E_5 + E_6$ .

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
<b>System.ArgumentException</b>	The non-null delegates in <i>delegates</i> are not of the same type.

1

2

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

### 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
-----------	-----------

<b>System.ArgumentNullException</b>	<i>type, target, or method</i> is null.
<b>System.ArgumentException</b>	<i>type</i> does not derive from <code>System.Delegate</code> . -or- <i>method</i> is not an instance method. -or- <i>target</i> does not implement <i>method</i> .
<b>System.MethodAccessException</b>	The caller does not have the required permission.

1

## 2 Permissions

Permission	Description
<b>System.Security.Permissions.ReflectionPermission</b>	Requires permission to access type information. See <code>System.Security.Permissions.ReflectionPermissionFlag.MemberAccess</code>

3

4

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

### 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
-----------	-----------

<b>System.ArgumentNullException</b>	<i>type, target, or method</i> is null.
<b>System.ArgumentException</b>	<i>type</i> does not derive from <code>System.Delegate</code> . -or- <i>method</i> is not a static method. -or- <i>target</i> does not implement <i>method</i> .
<b>System.MethodAccessException</b>	The caller does not have the required permission.

1

## 2 Permissions

Permission	Description
<b>System.Security.Permissions.ReflectionPermission</b>	Requires permission to access type information. See <code>System.Security.Permissions.ReflectionPermissionFlag.MemberAccess</code>

3

4

**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
<i>type</i>	The System.Type of System.Delegate to return. This System.Type is required to derive from System.Delegate.
<i>method</i>	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
<b>System.ArgumentNullException</b>	<i>type</i> or <i>method</i> is null.
<b>System.ArgumentException</b>	<i>type</i> does not derive from System.Delegate.

	-or-  <i>method</i> does not reflect a static method.
<b>System.InvalidProgramException</b>	The Invoke method of the <i>type</i> delegate was not found.
<b>System.MethodAccessException</b>	The caller does not have the required permission.

1

## 2 Permissions

Permission	Description
<b>System.Security.Permissions.ReflectionPermission</b>	Requires permission to access type information. See <code>System.Security.Permissions.ReflectionPermissionFlag.MemberAccess</code>

3

4



**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
<i>args</i>	An array of <code>System.Object</code> instances that are to be passed to the methods in the invocation list of the current instance. Specify <code>null</code> 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
<b>System.ArgumentException</b>	The type of one or more elements in <i>args</i> is invalid as a parameter to the methods implemented by the current instance.
<b>System.MethodAccessException</b>	The caller does not have the required permissions.  -or-  The number, order or type of parameters listed in <i>args</i> is invalid.

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

1

2

# 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
<i>obj</i>	The <code>System.Object</code> 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`.]

# 1 Delegate.GetHashCode() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual int32 GetHashCode()  
  
4 [C#]  
5 public override int GetHashCode()
```

## 6 Summary

7 Generates a hash code for the current instance.

## 8 Return Value

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

## 10 Description

11 The algorithm used to generate the hash code is unspecified.

12  
13 [*Note:* This method overrides `System.Object.GetHashCode`.]  
14  
15  
16

# 1 Delegate.GetInvocationList() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual class System.Delegate[]  
4 GetInvocationList()  
  
5 [C#]  
6 public virtual Delegate[] GetInvocationList()
```

## 7 Summary

8 Returns the invocation list of the current delegate.

## 9 Return Value

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

## 12 Behaviors

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

## 17 How and When to Override

18 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
<i>d1</i>	The first delegate to compare.
<i>d2</i>	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
<i>d1</i>	The first delegate to compare.
<i>d2</i>	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(Delagate source, Delagate value)
```

## Summary

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

## Parameters

Parameter	Description
<i>source</i>	The delegate from which to remove the invocation list of <i>value</i> .
<i>value</i>	The delegate that supplies the invocation list to remove from <i>source</i> .

## 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`.

	<i>value</i>	Return value
<code>null</code>	<code>null</code>	<code>null</code>
<code>null</code>	non-null	<code>null</code>
non-null	<code>null</code>	<i>source</i>



1

## 2 Description

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

## 6 Example

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

8

9 [C#]

```
10 using System;
11 class MyClass {
12     public string InstanceMethod(string s) {
13         return ("Instance String " + s);
14     }
15 }
16 class MyClass2 {
17     public string InstanceMethod2(string s) {
18         return ("Instance String2 " + s);
19     }
20 }
21 public delegate string DelegatedMethod(string s);
22
23 class TestClass {
24     public static void WriteDelegate (string label, Delegate d) {
25         Console.WriteLine("Invocation list targets for {0}:",label);
26         foreach(Delegate x in d.GetInvocationList())
27             Console.WriteLine("{0}",x.Target);
28     }
29
30     public static void Main() {
31         MyClass myInstance = new MyClass();
32         DelegatedMethod delInstance = new
33         DelegatedMethod(myInstance.InstanceMethod);
34         MyClass2 myInstance2 = new MyClass2();
35         DelegatedMethod delInstance2 = new
36         DelegatedMethod(myInstance2.InstanceMethod2);
37         DelegatedMethod [] sourceArray = {delInstance, delInstance2,
38         delInstance2, delInstance};
39         DelegatedMethod [] remove1 = {delInstance};
40         DelegatedMethod [] remove2 = {delInstance2, delInstance2};
41         DelegatedMethod [] remove3 = {delInstance2, delInstance};
42         DelegatedMethod [] remove4 = {delInstance, delInstance2};
43         DelegatedMethod [] remove5 = {delInstance, delInstance};
44         Delegate source = Delegate.Combine(sourceArray);
45         // Display invocation list of source
46         TestClass.WriteDelegate("source", source);
47         //Test 1: value occurs in source twice.
48         Delegate value1 = Delegate.Combine(remove1);
49         Delegate result1 = Delegate.Remove(source, value1);
50         TestClass.WriteDelegate("value1", value1);
```

```

1      if (result1==null) {
2          Console.WriteLine("removal test 1 result is null");
3      } else {
4          TestClass.WriteDelegate("result1", result1);
5      }
6      //Test 2: value matches the middle two elements of source.
7      Delegate value2 = Delegate.Combine(remove2);
8      Delegate result2 = Delegate.Remove(source, value2);
9      TestClass.WriteDelegate("value2", value2);
10     if (result2==null) {
11         Console.WriteLine("removal test 2 result2 is null");
12     } else {
13         TestClass.WriteDelegate("result2", result2);
14     }
15     //Test 3: value matches the last two elements of source.
16     Delegate value3 = Delegate.Combine(remove3);
17     Delegate result3 = Delegate.Remove(source, value3);
18     TestClass.WriteDelegate("value3", value3);
19     if (result3==null) {
20         Console.WriteLine("removal test 3 result3 is null");
21     } else {
22         TestClass.WriteDelegate("result3", result3);
23     }
24     //Test 4: value matches the first two elements of source.
25     Delegate value4 = Delegate.Combine(remove4);
26     Delegate result4 = Delegate.Remove(source, value4);
27     TestClass.WriteDelegate("value4", value4);
28     if (result4==null) {
29         Console.WriteLine("removal test 4 result4 is null");
30     } else {
31         TestClass.WriteDelegate("result4", result4);
32     }
33     //Test 5: value does not occur in source.
34     Delegate value5 = Delegate.Combine(remove5);
35     Delegate result5 = Delegate.Remove(source, value5);
36     TestClass.WriteDelegate("value5", value5);
37     if (result5==null) {
38         Console.WriteLine("removal test 5 result5 is null");
39     } else {
40         TestClass.WriteDelegate("result5", result5);
41     }
42     //Test 6: value exactly matches source.
43     Delegate result6 = Delegate.Remove(source, source);
44     TestClass.WriteDelegate("value=source", source);
45     if (result6==null) {
46         Console.WriteLine("removal test 6 result6 is null");
47     } else {
48
49         TestClass.WriteDelegate("result6", result6);
50     }
51 }
52 }

```

The output is

Invocation list targets for source:

```
1
2 MyClass
3
4
5 MyClass2
6
7
8 MyClass2
9
10
11 MyClass
12
13
14 Invocation list targets for value1:
15
16
17 MyClass
18
19
20 Invocation list targets for result1:
21
22
23 MyClass
24
25
26 MyClass2
27
28
29 MyClass2
30
31
32 Invocation list targets for value2:
33
34
35 MyClass2
36
37
38 MyClass2
39
40
41 Invocation list targets for result2:
42
43
44 MyClass
45
46
47 MyClass
48
49
```

```
1  Invocation list targets for value3:
2
3
4  MyClass2
5
6
7  MyClass
8
9
10 Invocation list targets for result3:
11
12
13 MyClass
14
15
16 MyClass2
17
18
19 Invocation list targets for value4:
20
21
22 MyClass
23
24
25 MyClass2
26
27
28 Invocation list targets for result4:
29
30
31 MyClass2
32
33
34 MyClass
35
36
37 Invocation list targets for value5:
38
39
40 MyClass
41
42
43 MyClass
44
45
46 Invocation list targets for result5:
47
48
49 MyClass
```

```
1
2
3 MyClass2
4
5
6 MyClass2
7
8
9 MyClass
10
11
12 Invocation list targets for value=source:
13
14
15 MyClass
16
17
18 MyClass2
19
20
21 MyClass2
22
23
24 MyClass
25
26
27 removal test 6 result6 is null
28
29
```

# 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(Delagate source, Delagate value)
```

## Summary

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

## Parameters

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

## 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`.

	<i>value</i>	Return value
<code>null</code>	<code>null</code>	<code>null</code>
<code>null</code>	non-null	<code>null</code>

non-null	null	<i>source</i>
----------	------	---------------

1

## 2 **Description**

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

6

**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
<b>System.MemberAccessException</b>	The caller does not have the required permissions.

### Permissions

Permission	Description
<b>System.Security.Permissions.ReflectionPermission</b>	Requires permission to access type information. See <code>System.Security.Permissions.ReflectionPermissionFlag.TypeInformation</code> .



# 1 Delegate.Target Property

```
2 [ILAsm]
3 .property object Target { public hidebysig specialname instance object
4 get_Target() }

5 [C#]
6 public object Target { get; }
```

## 7 Summary

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

## 9 Property Value

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

## 11 Description

12 This property is read-only.

13  
14 If the delegate invokes only static methods, this property returns null. If the delegate  
15 invokes one or more instance methods, this property returns the target of the last  
16 instance method/target pair in the invocation list.

## 17 Example

18 Example 1:

19  
20 The following example gets the System.Delegate.Target property values for two  
21 delegates. The first delegate invokes a static method, and the second invokes an  
22 instance method.

```
23 [C#]
24
25 using System;
26 public delegate string DelegatedMethod(string s);
27 class MyClass {
28     public static string StaticMethod(string s) {
29         return ("Static method Arg=" + s);
30     }
31     public string InstanceMethod(string s) {
32         return ("Instance method Arg=" + s);
33     }
34 }
35 class TestClass {
36     public static void Main() {
37         MyClass myInstance = new MyClass();
38         //Create delegates from delegate type DelegatedMethod.
39         DelegatedMethod delStatic = new DelegatedMethod(MyClass.StaticMethod);
40         DelegatedMethod delInstance = new
41         DelegatedMethod(myInstance.InstanceMethod);
```

```

1      object t = delStatic.Target;
2      Console.WriteLine ("Static target is {0}", t==null ? "null":t);
3      t = delInstance.Target;
4      Console.WriteLine ("Instance target is {0}", t==null ? "null":t);
5      }
6  }
7  The output is
8
9  Static target is null
10
11
12  Instance target is MyClass
13
14
15  Example 2:
16
17  The following example gets the System.Delegate.Target property value for three
18  delegates created using instance methods, static methods, and a combination of the two.
19
20  [C#]
21  using System;
22  class MyClass {
23      public static string StaticMethod(string s) {
24          return ("Static String " + s);
25      }
26      public string InstanceMethod(string s) {
27          return ("Instance String " + s);
28      }
29  }
30  class MyClass2 {
31      public static string StaticMethod2(string s) {
32          return ("Static String2 " + s);
33      }
34      public string InstanceMethod2(string s) {
35          return ("Instance String2 " + s);
36      }
37  }
38  public delegate string DelegatedMethod(string s);
39
40  class TestClass {
41      public static void Main() {
42          DelegatedMethod delStatic = new DelegatedMethod(MyClass.StaticMethod);
43          DelegatedMethod delStatic2 = new DelegatedMethod(MyClass2.StaticMethod2);
44
45          MyClass myInstance = new MyClass();
46          DelegatedMethod delInstance = new
47  DelegatedMethod(myInstance.InstanceMethod);
48
49          MyClass2 myInstance2 = new MyClass2();
50          DelegatedMethod delInstance2 = new
51  DelegatedMethod(myInstance2.InstanceMethod2);
52
53      Delegate d = Delegate.Combine(delStatic, delInstance );

```

```

1      Delegate e = Delegate.Combine(delInstance,delInstance2);
2      Delegate f = Delegate.Combine(delStatic, delStatic2 );
3      if (d!=null) {
4          Console.WriteLine("Combined 1 static, 1 instance, same class:");
5          Console.WriteLine("target...{0}", d.Target == null ? "null":
6 d.Target);
7          foreach(Delegate x in d.GetInvocationList())
8              Console.WriteLine("invoke element target: {0}",x.Target);
9      }
10     }
11     Console.WriteLine("");
12     if (e!=null) {
13         Console.WriteLine("Combined 2 instance methods, different classes:");
14         Console.WriteLine("target...{0}", e.Target == null ? "null":
15 e.Target);
16         foreach(Delegate x in e.GetInvocationList())
17             Console.WriteLine("invoke element target: {0}",x.Target);
18     }
19     Console.WriteLine("");
20     if (f!=null) {
21         Console.WriteLine("Combined 2 static methods, different classes:");
22         Console.WriteLine("target...{0}", f.Target == null ? "null":
23 f.Target);
24         foreach(Delegate x in f.GetInvocationList())
25             Console.WriteLine("invoke element target: {0}",x.Target);
26     }
27 }
28 }
29 }

```

30 The output is

```

31
32 Combined 1 static, 1 instance, same class:
33
34
35 target...MyClass
36
37
38 invoke element target:
39
40
41 invoke element target: MyClass
42
43
44 Combined 2 instance methods, different classes:
45
46
47 target...MyClass2
48
49
50 invoke element target: MyClass
51
52
53 invoke element target: MyClass2

```

```
1
2
3 Combined 2 static methods, different classes:
4
5
6 target...null
7
8
9 invoke element target:
10
11
12 invoke element target:
13
14
```