

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]
- Version: 1.0.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

1 corresponding target is **null**, otherwise the target is the instance on
2 which the method is to be called.

3
4 The signature of each method in the invocation list is required to
5 exactly match the signature specified by the delegate's type.

6
7 When a delegate is invoked, the methods in the corresponding
8 invocation list are invoked in the order in which they appear in that
9 list. A delegate attempts to invoke every method in its invocation list,
10 with duplicate methods being invoked once for each occurrence in that
11 list.

12
13 Delegates are immutable; once created, the invocation list of a
14 delegate does not change. Combining operations, such as
15 **System.Delegate.Combine** and **System.Delegate.Remove**, cannot
16 alter existing delegates. Instead, such operations result in the return
17 of either a new delegate that contains the results of the operation, an
18 existing delegate, or the null value. [*Note*: A combining operation
19 returns the null value when the result of the operation is an empty
20 invocation list. A combining operation returns an existing delegate
21 when the requested operation has no effect (for example, if an
22 attempt is made to remove a nonexistent entry).]

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

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

40 **Example**

41 42 **Example1:**

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

47
48 [C#]

```
49 using System;  
50 public delegate string DelegatedMethod(string s);
```

```

1      class MyClass {
2          public static string StaticMethod(string s) {
3              return ("Static method Arg=" + s);
4          }
5          public string InstanceMethod(string s) {
6              return ("Instance method Arg=" + s);
7          }
8      }
9      class TestClass {
10         public static void Main() {
11             MyClass myInstance = new MyClass();
12             //Create delegates from delegate type DelegatedMethod.
13             DelegatedMethod delStatic = new
14             DelegatedMethod(MyClass.StaticMethod);
15             DelegatedMethod delInstance = new
16             DelegatedMethod(myInstance.InstanceMethod);
17             //Invoke the methods referenced by the delegates.
18             Console.WriteLine (delStatic("Call 1"));
19             Console.WriteLine (delInstance ("Call 2"));
20         }
21     }

```

22 The output is

23
24 Static method Arg=Call 1

25
26
27 Instance method Arg=Call 2

28
29

30 **Example2:**

31
32 The following example shows the return value and the final value of a
33 parameter that is passed by reference to a delegate that invokes
34 multiple methods.

35
36 [C#]

37 using System;

```

1      class MyClass {
2          public int Increment(ref int i) {
3              Console.WriteLine("Incrementing {0}",i);
4              return (i++);
5          }
6          public int Negate(ref int i) {
7              Console.WriteLine("Negating {0}",i);
8              i = i * -1;
9              return i;
10         }
11     }
12
13     public delegate int DelegatedMethod(ref int i);
14     class TestClass {
15         public static void Main() {
16             MyClass myInstance = new MyClass();
17             DelegatedMethod delIncrementer = new
18             DelegatedMethod(myInstance.Increment);
19             DelegatedMethod delNegater = new
20             DelegatedMethod(myInstance.Negate);
21             DelegatedMethod d = (DelegatedMethod)
22             Delegate.Combine(delIncrementer, delNegater);
23             int i = 1;
24             Console.WriteLine("Invoking delegate using ref value
25 {0}",i);
26             int retvalue = d(ref i);
27             Console.WriteLine("After Invoking delegate i = {0}
28 return value is {1}",i, retvalue);
29         }
30     }

```

31 The output is

32
33 Invoking delegate using ref value 1

34
35
36 Incrementing 1

37
38
39 Negating 2

40
41

1 After Invoking delegate i = -2 return value is -2
2
3

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

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

11 Description

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

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

18 Behaviors

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

21 Default

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

30 How and When to Override

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

34

1 Delegate.Combine(System.Delegate, 2 System.Delegate) Method

```
3 [ILASM]  
4 .method public hidebysig static class System.Delegate  
5 Combine(class System.Delegate a, class System.Delegate b)  
  
6 [C#]  
7 public static Delegate Combine(Delegate a, Delegate b)
```

8 Summary

9 Concatenates the invocation lists of the specified delegates.

10 Parameters

11
12

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.

13
14
15

14 Return Value

16 A delegate, or **null**.

17
18

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

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

19
20
21

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

22 Description

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

24
25

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.

26
27
28

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

3
4 Then, `D4 = Combine(D2, D1)` results in D4's having an invocation list
5 of `E2 + E1`.

6
7 Then, `D5 = Combine(D3, D4)` results in D5's having an invocation list
8 of `E1 + E2 + E2 + E1`.

9
10 [*Note:* The invocation list of the returned delegate may contain
11 duplicate methods.]

12
13 **System.Delegate.Combine** is useful for creating event handlers that
14 call multiple methods each time an event occurs.]

15 Exceptions

16
17

Exception	Condition
System.ArgumentException	<i>a</i> and <i>b</i> are not of the same type.

18
19
20

1 Delegate.Combine(System.Delegate[])

2 Method

```
3 [ILASM]  
4 .method public hidebysig static class System.Delegate  
5 Combine(class System.Delegate[] delegates)  
  
6 [C#]  
7 public static Delegate Combine(Delegate[] delegates)
```

8 Summary

9 Concatenates the invocation lists of the specified delegates.

10 Parameters

11
12

Parameter	Description
<i>delegates</i>	An array of delegates of the exact same type.

13

14 Return Value

15

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

17 Description

18 The invocation list of the returned delegate is constructed by
19 concatenating the invocation lists of the delegates in *delegates*, in
20 increasing subscript order. For example, consider the following
21 situation, in which the elements of *delegates* have the following
22 invocation lists (where E_n represents an entry in an invocation list, and
23 null represents an empty invocation list): [0] = E_1 , [1] = null, [2] =
24 $E_2 + E_3$, and [3] = $E_4 + E_5 + E_6$. When these elements are
25 combined, the resulting delegate contains the invocation list $E_1 + E_2 +$
26 $E_3 + E_4 + E_5 + E_6$.

27

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

29

30 [Note: The invocation list of the returned delegate may contain
31 duplicate methods.]

32 Exceptions

33

34

Exception	Condition
-----------	-----------

1
2
3

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

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

3 Delegate.CreateDelegate(System.Type, 4 System.Object, System.String) Method

```
5 [ILASM]  
6 .method public hidebysig static class System.Delegate  
7 CreateDelegate(class System.Type type, object target,  
8 string method)  
9 [C#]  
10 public static Delegate CreateDelegate(Type type, object  
11 target, string method)
```

12 Summary

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

15 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> .

18 Return Value

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

22 Description

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

27 Exceptions

Exception	Condition
-----------	-----------

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

1
2
3
4

Permissions

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

5
6
7

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

1
2
3
4

Permissions

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

5
6
7

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
System.ArgumentNullException	<i>type</i> or <i>method</i> is null .
System.ArgumentException	<i>type</i> does not derive from System.Delegate .

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

1
2
3
4

Permissions

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

5
6
7

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

3 Delegate.DynamicInvoke(System.Object[] 4) Method

```
5 [ILASM]  
6 .method public hidebysig instance object  
7 DynamicInvoke(class System.Object[] args)  
8 [C#]  
9 public object DynamicInvoke(object[] args)
```

10 Summary

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

13 Parameters

14
15

Parameter	Description
<i>args</i>	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.

16
17
18

Return Value

19 The **System.Object** returned by the last method in the invocation list
20 of the current instance.

21 Exceptions

22
23

Exception	Condition
System.ArgumentException	The type of one or more elements in <i>args</i> 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 <i>args</i> 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 <i>args</i> 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.

1
2
3

1 Delegate.Equals(System.Object) Method

```
2 [ILASM]  
3 .method public hidebysig virtual bool Equals(object obj)  
4 [C#]  
5 public override bool Equals(object obj)
```

6 Summary

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

9 Parameters

10
11

Parameter	Description
<i>obj</i>	The System.Object to compare with the current instance.

12
13
14

Return Value

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

16 Description

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

22
23
24
25
26

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.

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

28

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

10 A **System.Int32** containing the hash code for this instance.

11 Description

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

13

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

15

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

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

13 Behaviors

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

18 How and When to Override

19 Override **System.Delegate.GetInvocationList** when subclassing
20 Delegate.

21

1 Delegate.op_Equality(System.Delegate, 2 System.Delegate) Method

```
3 [ILASM]  
4 .method public hidebysig static specialname bool  
5 op_Equality(class System.Delegate d1, class System.Delegate  
6 d2)  
7  
8 [C#]  
9 public static bool operator ==(Delegate d1, Delegate d2)
```

9 Summary

10 Determines whether the specified delegates are equal.

11 Parameters

12
13

Parameter	Description
<i>d1</i>	The first delegate to compare.
<i>d2</i>	The second delegate to compare.

14
15
16

15 Return Value

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

18 Description

19 [Note: See **System.Delegate.Equals.**]
20

1 Delegate.op_Inequality(System.Delegate, 2 System.Delegate) Method

```
3 [ILASM]  
4 .method public hidebysig static specialname bool  
5 op_Inequality(class System.Delegate d1, class  
6 System.Delegate d2)  
  
7 [C#]  
8 public static bool operator !=(Delegate d1, Delegate d2)
```

9 Summary

10 Determines whether the specified Delegates are not equal.

11 Parameters

12
13

Parameter	Description
<i>d1</i>	The first delegate to compare.
<i>d2</i>	The second delegate to compare.

14
15
16

Return Value

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

18 Description

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

20

1 Delegate.Remove(System.Delegate, 2 System.Delegate) Method

```
3 [ILASM]  
4 .method public hidebysig static class System.Delegate  
5 Remove(class System.Delegate source, class System.Delegate  
6 value)  
  
7 [C#]  
8 public static Delegate Remove(Delegate source, Delegate  
9 value)
```

10 Summary

11 Removes the invocation list of a **System.Delegate** from the
12 invocation list of another delegate.

13 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> .

16 Return Value

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

20
21 If *source* and *value* are not **null**, are not equal, and the invocation list
22 of *value* is contained in the invocation list of *source*, returns a new
23 delegate with the invocation list of *value* removed from the invocation
24 list of *source*.

25
26 If the invocation lists of *source* and *value* are equal, returns **null**.

27
28 If the invocation list of *value* is not found in the invocation list of
29 *source*, returns *source*.

30
31 The following table describes the value returned when *source* or *value*
32 is **null**.

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

1

2 Description

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

7 Example

8

9 The following example demonstrates the **System.Delegate.Remove**
10 method.

11
12

```
[C#]
```

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

```

1         DelegatedMethod [] remove5 = {delInstance,
2 delInstance};
3         Delegate source = Delegate.Combine(sourceArray);
4         // Display invocation list of source
5         TestClass.WriteDelegate("source", source);
6         //Test 1: value occurs in source twice.
7         Delegate value1 = Delegate.Combine(remove1);
8         Delegate result1 = Delegate.Remove(source, value1);
9         TestClass.WriteDelegate("value1", value1);
10        if (result1==null) {
11            Console.WriteLine("removal test 1 result is null");
12        } else {
13            TestClass.WriteDelegate("result1", result1);
14        }
15        //Test 2: value matches the middle two elements of
16 source.
17        Delegate value2 = Delegate.Combine(remove2);
18        Delegate result2 = Delegate.Remove(source, value2);
19        TestClass.WriteDelegate("value2", value2);
20        if (result2==null) {
21            Console.WriteLine("removal test 2 result2 is
22 null");
23        } else {
24            TestClass.WriteDelegate("result2", result2);
25        }
26        //Test 3: value matches the last two elements of
27 source.
28        Delegate value3 = Delegate.Combine(remove3);
29        Delegate result3 = Delegate.Remove(source, value3);
30        TestClass.WriteDelegate("value3", value3);
31        if (result3==null) {
32            Console.WriteLine("removal test 3 result3 is
33 null");
34        } else {
35            TestClass.WriteDelegate("result3", result3);
36        }
37        //Test 4: value matches the first two elements of
38 source.
39        Delegate value4 = Delegate.Combine(remove4);
40        Delegate result4 = Delegate.Remove(source, value4);
41        TestClass.WriteDelegate("value4", value4);
42        if (result4==null) {
43            Console.WriteLine("removal test 4 result4 is
44 null");
45        } else {
46            TestClass.WriteDelegate("result4", result4);
47        }
48        //Test 5: value does not occur in source.
49        Delegate value5 = Delegate.Combine(remove5);
50        Delegate result5 = Delegate.Remove(source, value5);
51        TestClass.WriteDelegate("value5", value5);
52        if (result5==null) {
53            Console.WriteLine("removal test 5 result5 is
54 null");
55        } else {
56            TestClass.WriteDelegate("result5", result5);
57        }

```

```
1         //Test 6: value exactly matches source.
2         Delegate result6 = Delegate.Remove(source, source);
3         TestClass.WriteDelegate("value=source", source);
4         if (result6==null) {
5             Console.WriteLine("removal test 6 result6 is
6 null");
7         } else {
8
9             TestClass.WriteDelegate("result6", result6);
10        }
11    }
12 }
```

13 The output is

14 Invocation list targets for source:

15

16

17 MyClass

18

19

20 MyClass2

21

22

23

24 MyClass2

25

26

27 MyClass

28

29

30 Invocation list targets for value1:

31

32

```
1 MyClass
2
3
4 Invocation list targets for result1:
5
6
7 MyClass
8
9
10 MyClass2
11
12
13 MyClass2
14
15
16 Invocation list targets for value2:
17
18
19 MyClass2
20
21
22 MyClass2
23
24
25 Invocation list targets for result2:
26
27
28 MyClass
29
30
31 MyClass
32
33
34 Invocation list targets for value3:
35
36
37 MyClass2
38
39
40 MyClass
41
42
43 Invocation list targets for result3:
44
45
46 MyClass
47
48
49 MyClass2
50
51
52 Invocation list targets for value4:
53
54
55 MyClass
56
57
```

```
1      MyClass2
2
3
4      Invocation list targets for result4:
5
6
7      MyClass2
8
9
10     MyClass
11
12
13     Invocation list targets for value5:
14
15
16     MyClass
17
18
19     MyClass
20
21
22     Invocation list targets for result5:
23
24
25     MyClass
26
27
28     MyClass2
29
30
31     MyClass2
32
33
34     MyClass
35
36
37     Invocation list targets for value=source:
38
39
40     MyClass
41
42
43     MyClass2
44
45
46     MyClass2
47
48
49     MyClass
50
51
52     removal test 6 result6 is null
53
54
```

1 Delegate.RemoveAll(System.Delegate, 2 System.Delegate) Method

```
3 [ILASM]  
4 .method public hidebysig static class System.Delegate  
5 RemoveAll(class System.Delegate source, class  
6 System.Delegate value)  
  
7 [C#]  
8 public static Delegate RemoveAll(Delegate source, Delegate  
9 value)
```

10 Summary

11 Removes all matching occurrences of the invocation list of a
12 **System.Delegate** from the invocation list of another delegate.

13 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> .

16 Return Value

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

20
21 If *source* and *value* are not **null**, are not equal, and the invocation list
22 of *value* is contained in the invocation list of *source*, returns a new
23 delegate with all matching occurrences of the invocation list of *value*
24 removed from the invocation list of *source*.

25
26 If the invocation lists of *source* and *value* are equal, or if *source*
27 contains only a succession of invocation lists equal to *value*, returns
28 **null**.

29
30 If the invocation list of *value* is not found in the invocation list of
31 *source*, returns *source*.

32
33 The following table describes the value returned when *source* or *value*
34 is **null**.

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

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

1

2 **Description**

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

7

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 .

1 Delegate.Target Property

```
2 [ILASM]
3 .property object Target { public hidebysig specialname
4 instance object get_Target() }
5 [C#]
6 public object Target { get; }
```

7 Summary

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

10 Property Value

11

12 A **System.Object** instance, or **null** if the delegate invokes only static
13 methods.

14 Description

15 This property is read-only.

16

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

21 Example

22

23 Example 1:

24

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

28

29

```
30 using System;
31 public delegate string DelegatedMethod(string s);
32 class MyClass {
33     public static string StaticMethod(string s) {
34         return ("Static method Arg=" + s);
35     }
36     public string InstanceMethod(string s) {
37         return ("Instance method Arg=" + s);
38     }
39 }
40 class TestClass {
```

```

1     public static void Main() {
2         MyClass myInstance = new MyClass();
3         //Create delegates from delegate type
4     DelegatedMethod.
5         DelegatedMethod delStatic = new
6     DelegatedMethod(MyClass.StaticMethod);
7         DelegatedMethod delInstance = new
8     DelegatedMethod(myInstance.InstanceMethod);
9         object t = delStatic.Target;
10        Console.WriteLine ("Static target is {0}", t==null ?
11    "null":t);
12        t = delInstance.Target;
13        Console.WriteLine ("Instance target is {0}", t==null ?
14    "null":t);
15    }
16 }

```

17 The output is

```

18
19 Static target is null
20

```

```

21
22 Instance target is MyClass
23

```

24
25 Example 2:

26
27 The following example gets the **System.Delegate.Target** property
28 value for three delegates created using instance methods, static
29 methods, and a combination of the two.

30
31 [C#]

```

32 using System;
33 class MyClass {
34     public static string StaticMethod(string s) {
35         return ("Static String " + s);
36     }
37     public string InstanceMethod(string s) {

```

```

1     return ("Instance String " + s);
2     }
3 }
4 class MyClass2 {
5     public static string StaticMethod2(string s) {
6         return ("Static String2 " + s);
7     }
8     public string InstanceMethod2(string s) {
9         return ("Instance String2 " + s);
10    }
11 }
12 public delegate string DelegatedMethod(string s);
13
14 class TestClass {
15     public static void Main() {
16         DelegatedMethod delStatic = new
17         DelegatedMethod(MyClass.StaticMethod);
18         DelegatedMethod delStatic2 = new
19         DelegatedMethod(MyClass2.StaticMethod2);
20
21         MyClass myInstance = new MyClass();
22         DelegatedMethod delInstance = new
23         DelegatedMethod(myInstance.InstanceMethod);
24
25         MyClass2 myInstance2 = new MyClass2();
26         DelegatedMethod delInstance2 = new
27         DelegatedMethod(myInstance2.InstanceMethod2);
28
29         Delegate d = Delegate.Combine(delStatic, delInstance);
30         Delegate e =
31         Delegate.Combine(delInstance, delInstance2);
32         Delegate f = Delegate.Combine(delStatic, delStatic2);
33         if (d!=null) {
34             Console.WriteLine("Combined 1 static, 1 instance,
35 same class:");
36             Console.WriteLine("target...{0}", d.Target == null
37 ? "null": d.Target);
38             foreach(Delegate x in d.GetInvocationList())
39                 Console.WriteLine("invoke element target:
40 {0}",x.Target);
41         }
42     }
43     Console.WriteLine("");
44     if (e!=null) {
45         Console.WriteLine("Combined 2 instance methods,
46 different classes:");
47         Console.WriteLine("target...{0}", e.Target == null
48 ? "null": e.Target);
49         foreach(Delegate x in e.GetInvocationList())
50             Console.WriteLine("invoke element target:
51 {0}",x.Target);
52     }
53     Console.WriteLine("");
54     if (f!=null) {
55         Console.WriteLine("Combined 2 static methods,
56 different classes:");

```

```
1         Console.WriteLine("target...{0}", f.Target == null
2 ? "null": f.Target);
3         foreach(Delegate x in f.GetInvocationList())
4             Console.WriteLine("invoke element target:
5 {0}",x.Target);
6     }
7
8     }
9 }
```

10 The output is

11
12 Combined 1 static, 1 instance, same class:

13
14
15 target...MyClass

16
17
18 invoke element target:

19
20
21 invoke element target: MyClass

22
23
24 Combined 2 instance methods, different classes:

25
26
27 target...MyClass2

28
29
30 invoke element target: MyClass

```
1
2
3   invoke element target: MyClass2
4
5
6   Combined 2 static methods, different classes:
7
8
9   target...null
10
11
12  invoke element target:
13
14
15  invoke element target:
16
17
```