1 System.Action<-T1,-T2> Delegate

2 [ILAsm]
3 `.class public sealed System.Action`2<-T1,-T2> extends`
4 `System.MulticastDelegate`

5 [C#]
6 `public delegate void Action<in T1,in T2>(T1 arg1, T2 arg2);`

7 **Assembly Info:**

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

13 **Summary**

14 Encapsulates a method that has two parameters and does not return a value.

15 **Parameters**

Parameter	Description
arg1	The first parameter of the method that this delegate encapsulates.
arg2	The second parameter of the method that this delegate encapsulates.

16
17 **Inherits From: System.MulticastDelegate**
18
19 **Library:** BCL
20
21 **Description**

22 You can use the `System.Action`2<T1,T2>` delegate to pass a method as a parameter
23 without explicitly declaring a custom delegate. The encapsulated method must
24 correspond to the method signature that is defined by this delegate. This means that the
25 encapsulated method must have two parameters that are both passed to it by value,
26 and it must not return a value. Typically, such a method is used to perform an
27 operation.
28
29 [*Note:* To reference a method that has two parameters and returns a value, use the
30 generic `System.Func`3<T1,T2,TResult>` delegate instead.
31
32]
33
34 When you use the `System.Action`2<T1,T2>` delegate, you do not have to explicitly

1   define a delegate that encapsulates a method with two parameters.
2
3   You can also use the `System.Action`2<T1,T2>` delegate with anonymous methods in
4   C#. (For an introduction to anonymous methods, see the C# standard.)

5