

1 System.Reflection.MethodBase Class

```
2 [ILAsm]  
3 .class public abstract serializable MethodBase extends  
4 System.Reflection.MemberInfo  
  
5 [C#]  
6 public abstract class MethodBase: MemberInfo
```

7 Assembly Info:

- 8 • *Name:* mscorlib
- 9 • *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00]
- 10 • *Version:* 2.0.x.x
- 11 • *Attributes:*
 - 12 ○ CLSCompliantAttribute(true)

13 Summary

14 Provides information about methods and constructors.

15 Inherits From: System.Reflection.MemberInfo

16

17 **Library:** Reflection

18

19 **Thread Safety:** All public static members of this type are safe for multithreaded operations.
20 No instance members are guaranteed to be thread safe.

21

22 Description

23 [*Note:* MethodBase is used to represent method types.

24

25 The Base Class Library includes the following derived types:

- 26 • System.Reflection.MethodInfo
- 27 • System.Reflection.ConstructorInfo

28]

29

1 MethodBase() Constructor

```
2 [ILAsm]  
3 family rtspecialname specialname instance void .ctor()  
4 [C#]  
5 protected MethodBase()
```

6 Summary

7 Constructs a new instance of the `System.Reflection.MethodBase` class.

8

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

3 **MethodBase.GetGenericArguments() Method**

```
4 [ILAsm]  
5 .method public hidebysig virtual class System.Type[] GetGenericArguments()  
6 [C#]  
7 public virtual Type[] GetGenericArguments()
```

8 **Summary**

9 Returns an array of *System.Type* objects that represent the type arguments of a generic
10 method or the type parameters of a generic method definition.

11 **Return Value**

12 An array of *System.Type* objects that represent the type arguments of a generic method
13 or the type parameters of a generic method definition. Returns an empty array if the
14 current method is not a generic method.

15 **Description**

16 The default behavior, when not overridden in a derived class, is to throw
17 *System.NotSupportedException*. In other words, derived classes do not support
18 generics by default.

19
20 The elements of the returned array are in the order in which they appear in the list of
21 type parameters for the generic method.

- 22 • If the current method is a closed constructed method (that is, the
23 *System.Reflection.MethodBase.ContainsGenericParameters* property returns
24 false), the array returned by the
25 *System.Reflection.MethodBase.GetGenericArguments* method contains the types
26 that have been assigned to the generic type parameters of the generic method
27 definition.
- 28 • If the current method is a generic method definition, the array contains the type
29 parameters.
- 30 • If the current method is an open constructed method (that is, the
31 *System.Reflection.MethodBase.ContainsGenericParameters* property returns
32 true) in which specific types have been assigned to some type parameters and type
33 parameters of enclosing generic types have been assigned to other type parameters,
34 the array contains both types and type parameters. Use the
35 *System.Type.IsGenericParameter* property to tell them apart.

36 For a list of the invariant conditions for terms specific to generic methods, see the
37 *System.Reflection.MethodInfo.IsGenericMethod* property. For a list of the invariant

1 conditions for other terms used in generic reflection, see the `System.Type.IsGenericType`
2 property.

3 **Exceptions**

Exception	Condition
System.NotSupportedException	Default behavior when not overridden in a derived class.

4

5

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

3 **MethodBase.GetMethodFromHandle(System.R** 4 **untimeMethodHandle) Method**

```
5 [ILAsm]  
6 .method public hidebysig static class System.Reflection.MethodBase  
7 GetMethodFromHandle(valuetype System.RuntimeMethodHandle handle)  
8 [C#]  
9 public static MethodBase GetMethodFromHandle(RuntimeMethodHandle handle)
```

10 **Summary**

11 Gets method information by using the method's internal metadata representation
12 (handle).

13 **Parameters**

Parameter	Description
<i>handle</i>	The method's System.RuntimeMethodHandle handle.

14

15 **Return Value**

16 A System.Reflection.MethodBase object containing information about the method.

17 **Description**

18 The handles are valid only in the application domain in which they were obtained.

19

1 MethodBase.GetParameters() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual abstract class  
4 System.Reflection.ParameterInfo[] GetParameters()  
  
5 [C#]  
6 public abstract ParameterInfo[] GetParameters()
```

7 Summary

8 Returns the parameters of the method or constructor reflected by the current instance.

9 Return Value

10 An array of `System.Reflection.ParameterInfo` objects that contain information that
11 matches the signature of the method or constructor reflected by the current instance.

12 Behaviors

13 As described above.

14

15

1 MethodBase.Invoke(System.Object, 2 System.Reflection.BindingFlags, 3 System.Reflection.Binder, System.Object[], 4 System.Globalization.CultureInfo) Method

```
5 [ILAsm]  
6 .method public hidebysig virtual abstract object Invoke(object obj,  
7 valuetype System.Reflection.BindingFlags invokeAttr, class  
8 System.Reflection.Binder binder, object[] parameters, class  
9 System.Globalization.CultureInfo culture)  
10  
11 [C#]  
12 public abstract object Invoke(object obj, BindingFlags invokeAttr, Binder  
binder, object[] parameters, CultureInfo culture)
```

13 Summary

14 Invokes the method or constructor reflected by the current instance as determined by
15 the specified arguments.

16 Parameters

Parameter	Description
<i>obj</i>	An instance of the type that contains the method reflected by the current instance. If the method is static, <i>obj</i> is ignored. For non-static methods, <i>obj</i> is an instance of a class that inherits or declares the method.
<i>invokeAttr</i>	A System.Reflection.BindingFlags value that controls the binding process.
<i>binder</i>	An object that enables the binding, coercion of argument types, invocation of members, and retrieval of MemberInfo objects via reflection. If <i>binder</i> is null, the default binder is used.
<i>parameters</i>	An array of objects that match the number, order and type of the parameters for the constructor or method reflected by the current instance. If the member reflected by the current instance takes no parameters, specify either an array with zero elements or null. [Note: Any object in this array that is not explicitly initialized with a value will contain the default value for that object type. For reference-type elements, this value is null. For value-type elements, this value is 0, 0.0, or false, depending on the specific element type. If the method or constructor reflected by the current instance is static, this parameter is ignored.]

<i>culture</i>	The only defined value for this parameter is <code>null</code> .

1

2 **Return Value**

3 A `System.Object` that contains the return value of the invoked method, or a re-
 4 initialized object if a constructor was invoked.

5 **Description**

6 Optional parameters can not be omitted in calls to
 7 `System.Reflection.MethodBase.Invoke`.

8 **Exceptions**

Exception	Condition
System.ArgumentException	The types of the elements of <i>parameters</i> do not match the types of the parameters accepted by the constructor or method reflected by the current instance, under the constraints of the default binder.
System.Reflection.TargetException	The constructor or method reflected by the current instance is non-static, and <i>obj</i> is <code>null</code> or is of a type that does not implement the member reflected by the current instance.
System.Reflection.TargetInvocationException	The method reflected by the current instance threw an exception.
System.Reflection.TargetParameterCountException	<i>parameters.Length</i> does not equal the number of parameters required by the contract of the constructor or method reflected by the current instance.
System.MemberAccessException	The caller does not have permission to execute the method or constructor.
System.InvalidOperationException	The type that declares the method is an open generic type. That is, <code>System.Type.ContainsGenericParameters</code> returns <code>true</code> for the declaring type.

1

2 Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to invoke non-public members of loaded assemblies. See <code>System.Security.Permissions.ReflectionPermissionFlag.MemberAccess</code> .

3

4

1 `MethodBase.Invoke(System.Object,` 2 `System.Object[])` Method

```
3 [ILAsm]  
4 .method public hidebysig instance object Invoke(object obj, object[]  
5 parameters)  
6 [C#]  
7 public object Invoke(object obj, object[] parameters)
```

8 **Summary**

9 Invokes the method or constructor reflected by the current instance on the specified
10 object and using the specified arguments.

11 **Parameters**

Parameter	Description
<i>obj</i>	An instance of a type that contains the constructor or method reflected by the current instance. If the member is static, <i>obj</i> is ignored. For non-static methods, <i>obj</i> is an instance of a class that inherits or declares the method.
<i>parameters</i>	An array objects that match the number, order and type of the parameters for the constructor or method reflected by the current instance. If the member reflected by the current instance takes no parameters, specify either an array with zero elements or <code>null</code> . [Note: Any object in this array that is not explicitly initialized with a value will contain the default value for that object type. For reference-type elements, this value is <code>null</code> . For value-type elements, this value is 0, 0.0, or <code>false</code> , depending on the specific element type. If the method or constructor reflected by the current instance is <code>static</code> , this parameter is ignored.]

12

13 **Return Value**

14 A `System.Object` that contains the return value of the invoked method, or a re-
15 initialized object if a constructor was invoked.

16 **Description**

17 This version of `System.Reflection.MethodBase.Invoke` is equivalent to
18 `System.Reflection.MethodBase.Invoke(obj, (BindingFlags)0, null, parameters,`
19 `null)`.

1
 2 Optional parameters cannot be omitted in calls to
 3 `System.Reflection.MethodBase.Invoke`.

4 **Exceptions**

Exception	Condition
System.ArgumentException	The types of the elements of <i>parameters</i> do not match the types of the parameters accepted by the constructor or method reflected by the current instance, under the constraints of the default binder.
System.Reflection.TargetException	The constructor or method reflected by the current instance is non-static and <i>obj</i> is <code>null</code> , or is of a type that does not implement the member reflected by the current instance.
System.Reflection.TargetInvocationException	The constructor or method reflected by the current instance threw an exception.
System.Reflection.TargetParameterCountException	<i>parameters.Length</i> does not equal the number of parameters required by the contract of the member reflected by the current instance.
System.MemberAccessException	The caller does not have permission to execute the method or constructor.
System.InvalidOperationException	The type that declares the method is an open generic type. That is, <code>System.Type.ContainsGenericParameters</code> returns <code>true</code> for the declaring type.

5
 6 **Permissions**

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to invoke non-public members of loaded assemblies. See <code>System.Security.Permissions.ReflectionPermissionFlag.MemberAccess</code> .

7
 8

1 MethodBase.Attributes Property

```
2 [ILAsm]  
3 .property valuetype System.Reflection.MethodAttributes Attributes { public  
4 hidebysig virtual abstract specialname valuetype  
5 System.Reflection.MethodAttributes get_Attributes() }  
6 [C#]  
7 public abstract MethodAttributes Attributes { get; }
```

8 Summary

9 Gets the attributes of the method reflected by the current instance.

10 Property Value

11 A `System.Reflection.MethodAttributes` value that signifies the attributes of the
12 method reflected by the current instance.

13 Behaviors

14 This property is read-only.

15
16 This property gets a `System.Reflection.MethodAttributes` value that indicates the
17 attributes set in the metadata of the method reflected by the current instance.

18 Usage

19 Use this property to determine the accessibility, layout, and semantics of the constructor
20 or method reflected by the current instance. Also use this property to determine if the
21 member reflected by the current instance is implemented in native code or has a special
22 name.

23

24 Example

25 The following example demonstrates using this property to obtain the attributes of three
26 methods.

```
27 [C#]  
28  
29 using System;  
30 using System.Reflection;  
31  
32 abstract class MyBaseClass  
33 {  
34  
35     abstract public void MyPublicInstanceMethod();  
36  
37 }
```

```

1
2 class MyDerivedClass: MyBaseClass
3 {
4
5     public override void MyPublicInstanceMethod() {}
6     private static void MyPrivateStaticMethod() {}
7
8 }
9
10 class MethodAttributesExample
11 {
12
13     static void PrintMethodAttributes(Type t)
14     {
15
16         string str;
17         MethodInfo[] miAry = t.GetMethods( BindingFlags.Static |
18             BindingFlags.Instance | BindingFlags.Public |
19             BindingFlags.NonPublic | BindingFlags.DeclaredOnly );
20         foreach (MethodInfo mi in miAry)
21         {
22
23             Console.WriteLine("Method {0} is: ", mi.Name);
24             str = ((mi.Attributes & MethodAttributes.Static) != 0) ?
25                 "Static": "Instance";
26             Console.Write(str + " ");
27             str = ((mi.Attributes & MethodAttributes.Public) != 0) ?
28                 "Public": "Not-Public";
29             Console.Write(str + " ");
30             str = ((mi.Attributes & MethodAttributes.HideBySig) != 0) ?
31                 "HideBySig": "Hide-by-name";
32             Console.Write(str + " ");
33             str = ((mi.Attributes & MethodAttributes.Abstract) != 0) ?
34                 "Abstract": String.Empty;
35             Console.WriteLine(str);
36
37         }
38     }
39 }
40
41 public static void Main()
42 {
43
44     PrintMethodAttributes(typeof(MyBaseClass));
45     PrintMethodAttributes(typeof(MyDerivedClass));
46
47 }
48
49 }
50
51 The output is
52
53 Method MyPublicInstanceMethod is:
54
55 Instance Public HideBySig Abstract
56

```

1
2
3 Method MyPublicInstanceMethod is:
4
5
6 Instance Public HideBySig
7
8
9 Method MyPrivateStaticMethod is:
10
11
12 Static Not-Public HideBySig
13
14

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

3 **MethodBase.ContainsGenericParameters** 4 **Property**

```
5 [ILAsm]  
6 .property bool ContainsGenericParameters { public hidebysig virtual  
7 specialname bool get_ContainsGenericParameters() }  
8 [C#]  
9 public virtual bool ContainsGenericParameters { get; }
```

10 **Summary**

11 Gets a value that indicates whether a generic method contains unassigned generic type
12 parameters.

13 **Property Value**

14 true if the current method contains unassigned generic type parameters; otherwise
15 false.

16 **Description**

17 The default behavior, when not overridden in a derived class, is to return `false`. In
18 other words, by default, derived classes do not support generics.

19
20 In order to invoke a generic method, there must be no generic type definitions or open
21 constructed types in the type arguments of the method itself, or in any enclosing types.
22 If the `System.Reflection.MethodBase.ContainsGenericParameters` property returns
23 `true`, the method cannot be invoked.

24
25 The `System.Reflection.MethodBase.ContainsGenericParameters` property searches
26 recursively for type parameters. For example, it returns `true` for any method in an open
27 type `A<T>`, even though the method itself is not generic. Contrast this with the behavior
28 of the `System.Reflection.MethodBase.IsGenericMethod` property, which returns
29 `false` for such a method.

30
31 For a list of the invariant conditions for terms specific to generic methods, see the
32 `System.Reflection.MethodInfo.IsGenericMethod` property. For a list of the invariant
33 conditions for other terms used in generic reflection, see the
34 `System.Type.IsGenericType` property.

35 **Behaviors**

36 This property is read-only.

37

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

MethodBase.IsGenericMethod Property

```
[ILAsm]
.property bool IsGenericMethod { public hidebysig virtual specialname bool
get_IsGenericMethod() }

[C#]
public virtual bool IsGenericMethod { get; }
```

Summary

Gets a value that indicates whether the current object is a generic method.

Property Value

true if the current object is a generic method; otherwise false.

Description

The default behavior, when not overridden in a derived class, is to return false. In other words, by default, derived classes do not support generics.

Use this property to determine whether the current `System.Reflection.MethodBase` object represents a generic method. Use the `System.Reflection.MethodBase.ContainsGenericParameters` property to determine whether the current `System.Reflection.MethodBase` object represents an open constructed method or a closed constructed method.

For a list of the invariant conditions for terms specific to generic methods, see the `System.Reflection.MethodInfo.IsGenericMethod` property. For a list of the invariant conditions for other terms used in generic reflection, see the `System.Type.IsGenericType` property.

Behaviors

This property is read-only.

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

MethodBase.IsGenericMethodDefinition Property

```
[ILAsm]
.property bool IsGenericMethodDefinition { public hidebysig virtual
specialname bool get_IsGenericMethodDefinition() }

[C#]
public virtual bool IsGenericMethodDefinition { get; }
```

Summary

Gets a value that indicates whether the current `System.Reflection.MethodBase` represents a definition of a generic method.

Property Value

`true` if the current `System.Reflection.MethodBase` object represents the definition of a generic method; otherwise `false`.

Description

The default behavior, when not overridden in a derived class, is to return `false`. In other words, by default, derived classes do not support generics.

If the current `System.Reflection.MethodBase` represents a generic method definition, then:

- `System.Reflection.MethodBase.IsGenericMethodDefinition` returns `true`.
- For each `System.Type` object in the array returned by the `System.Reflection.MethodBase.GetGenericArguments` method: The `System.Type.IsGenericParameter` property returns `true`; the `System.Type.DeclaringMethod` returns the current instance; and the `System.Type.GenericParameterPosition` property is the same as the position of the `System.Type` object in the array.

For a list of the invariant conditions for terms specific to generic methods, see the `System.Reflection.MethodInfo.IsGenericMethod` property. For a list of the invariant conditions for other terms used in generic reflection, see the `System.Type.IsGenericType` property.

Behaviors

This property is read-only.

