

System.Type Class

```
[ILASM]
.class public abstract serializable Type extends
System.Object

[C#]
public abstract class Type: Object
```

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)

Summary

Provides information about a type.

Inherits From: System.Object [*Note:* When implementing the Reflection library, this type inherits from System.Reflection.MemberInfo.]

Library: BCL

Thread Safety: This type is safe for multithreaded operations.

Description

The **System.Type** class is abstract, as is the **System.Reflection.MemberInfo** class and its subclasses **System.Reflection.FieldInfo**, **System.Reflection.PropertyInfo**, **System.Reflection.MethodBase**, and **System.Reflection.EventInfo**. **System.Reflection.ConstructorInfo** and **System.Reflection.MethodInfo** are subclasses of **System.Reflection.MethodBase**. The runtime provides non-public implementations of these classes. [*Note:* For example, **System.Type.GetMethod** is typed as returning a **System.Reflection.MethodInfo** object. The returned object is actually an instance of the non-public runtime type that implements **System.Reflection.MethodInfo**.]

A conforming CLI program which is written to run on only the Kernel profile cannot subclass **System.Type**. [*Note:* This only applies to conforming programs not conforming implementations.]

1 A **System.Type** object that represents a type is unique; that is, two
2 **System.Type** object references refer to the same object if and only if
3 they represent the same type. This allows for comparison of
4 **System.Type** objects using reference equality.

5
6 [Note: An instance of **System.Type** can represent any one of the
7 following types:

- 8 • Classes
- 9 • Value types
- 10 • Arrays
- 11 • Interfaces
- 12 • Pointers
- 13 • Enumerations

14 The following table shows what members of a base class are returned
15 by the methods that return members of types, such as
16 **System.Type.GetConstructor** and **System.Type.GetMethod**.

Member Type	Static	Non-Static
Constructor	No	No
Field	No	Yes. A field is always hide-by-name-and-signature.
Event	Not applicable	The common type system rule is that the inheritance of an event is the same as that of the accessors that implement the event. Reflection treats events as hide-by-name-and-signature.
Method	No	Yes. A method (both virtual and non-virtual) can be hide-by-name or hide-by-name-and-signature.
Nested Type	No	No
Property	Not applicable	The common type system rule is that the inheritance is the same as that of the accessors that implement the property. Reflection treats properties as hide-by-name-and-signature.

17
18
19
20 For reflection, properties and events are hide-by-name-and-signature.

1 If a property has both a get and a set accessor in the base class, but
2 the derived class has only a get accessor, the derived class property
3 hides the base class property, and the setter on the base class will not
4 be accessible.]

5

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

3 Type() Constructor

```
4 [ILASM]  
5 family rtspecialname specialname instance void .ctor()  
6 [C#]  
7 protected Type()
```

8 Summary

9 Constructs a new instance of the **System.Type** class.

10

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

3 Type.Delimiter Field

```
4 [ILASM]  
5 .field public static initOnly valuetype System.Char  
6 Delimiter  
7 [C#]  
8 public static readonly char Delimiter
```

9 Summary

10 Specifies the character that separates elements in the fully qualified
11 name of a **System.Type**.

12 Description

13 This field is read-only.

14

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

3 Type.EmptyTypes Field

```
4 [ILASM]  
5 .field public static initOnly class System.Type[]  
6 EmptyTypes  
7 [C#]  
8 public static readonly Type[] EmptyTypes
```

9 Summary

10 Returns an empty array of type **System.Type**.

11 Description

12 This field is read-only.

13
14 The empty **System.Type** array returned by this field is used to specify
15 that lookup methods in the **System.Type** class, such as
16 **System.Type.GetMethod** and **System.Type.GetConstructor**,
17 search for members that do not take parameters. [*Note:* For example,
18 to locate the public instance constructor that takes no parameters,
19 invoke **System.Type.GetConstructor**
20 (**System.Reflection.BindingFlags.Public** |
21 **System.Reflection.BindingFlags.Instance**, **null**,
22 **System.Type.EmptyTypes**, **null**).]

23

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

3 Type.Missing Field

```
4 [ILASM]  
5 .field public static initOnly object Missing  
6 [C#]  
7 public static readonly object Missing
```

8 Summary

9 Represents a missing value in the **System.Type** information.

10 Description

11 This field is read-only.

12
13 Use the **Missing** field for invocation through reflection to ensure that a
14 call will be made with the default value of a parameter as specified in
15 the metadata. [*Note:* If the **Missing** field is specified for a parameter
16 value and there is no default value for that parameter, a
17 **System.ArgumentException** is thrown.]

18

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

3 Type.Equals(System.Type) Method

```
4 [ILASM]  
5 .method public hidebysig instance bool Equals(class  
6 System.Type o)  
  
7 [C#]  
8 public bool Equals(Type o)
```

9 Summary

10 Determines if the underlying system type of the current **System.Type**
11 is the same as the underlying system type of the specified
12 **System.Type**.

13 Parameters

14
15

Parameter	Description
<i>o</i>	The System.Type whose underlying system type is to be compared with the underlying system type of the current System.Type .

16
17
18

17 Return Value

19 **true** if the underlying system type of *o* is the same as the underlying
20 system type of the current **System.Type**; otherwise, **false**.

21

1 Type.GetArrayRank() Method

```
2 [ILASM]  
3 .method public hidebysig virtual int32 GetArrayRank()  
4 [C#]  
5 public virtual int GetArrayRank()
```

6 Summary

7 Returns the number of dimensions in the current **System.Type**.

8 Return Value

9

10 A **System.Int32** containing the number of dimensions in the current
11 **System.Type**.

12 Exceptions

13

14

Exception	Condition
System.ArgumentException	The current System.Type is not an array.

15

16

17

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

3 **Type.GetAttributeFlagsImpl()** Method

```
4 [ILASM]  
5 .method family hidebysig virtual abstract valuetype  
6 System.Reflection.TypeAttributes GetAttributeFlagsImpl()  
  
7 [C#]  
8 protected abstract TypeAttributes GetAttributeFlagsImpl()
```

9 **Summary**

10 When overridden in a derived type implements the
11 **System.Type.Attributes** property and returns the attributes specified
12 for the type represented by the current instance.

13 **Return Value**

15 A **System.Reflection.TypeAttributes** value that signifies the
16 attributes of the type represented by the current instance.

17 **Behaviors**

18 This property is read-only.

19
20 This method returns a **System.Reflection.TypeAttributes** value that
21 indicates the attributes set in the metadata of the type represented by
22 the current instance.

23 **Usage**

24 Use this property to determine the visibility, semantics, and layout
25 format of the type represented by the current. Also use this property
26 to determine if the type represented by the current instance has a
27 special name.

28

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

3 **Type.GetConstructor(System.Reflection.Bi**
4 **ndingFlags, System.Reflection.Binder,**
5 **System.Type[],**
6 **System.Reflection.ParameterModifier[])**
7 **Method**

```
8 [ILASM]  
9 .method public hidebysig instance class  
10 System.Reflection.ConstructorInfo GetConstructor(valuetype  
11 System.Reflection.BindingFlags bindingAttr, class  
12 System.Reflection.Binder binder, class System.Type[] types,  
13 class System.Reflection.ParameterModifier[] modifiers)  
  
14 [C#]  
15 public ConstructorInfo GetConstructor(BindingFlags  
16 bindingAttr, Binder binder, Type[] types,  
17 ParameterModifier[] modifiers)
```

18 **Summary**

19 Returns a constructor defined in the type represented by the current
20 instance. The parameters of the constructor match the specified
21 argument types and modifiers, under the specified binding constraints.

22 **Parameters**

23
24

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null.
<i>binder</i>	A System.Reflection.Binder object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specify null to use the System.Type.DefaultBinder .
<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the constructor to be returned.
<i>modifiers</i>	The only defined value for this parameter is null .

1
2
3

Return Value

4
5
6
7
8
9
10
11

A **System.Reflection.ConstructorInfo** object that reflects the constructor that matches the specified criteria. If an exact match does not exist, *binder* will attempt to coerce the parameter types specified in *types* to select a match. If *binder* is unable to select a match, returns **null**. If the type represented by the current instance is contained in a loaded assembly, the constructor that matches the specified criteria is not public, and the caller does not have sufficient permissions, returns **null**.

Description

13
14

The following **System.Reflection.BindingFlags** are used to define which constructors to include in the search:

15
16
17

- Specify either **System.Reflection.BindingFlags.Instance** or **System.Reflection.BindingFlags.Static** to get a return value other than **null**.

18
19

- Specify **System.Reflection.BindingFlags.Public** to include public constructors in the search.

20
21
22

- Specify **System.Reflection.BindingFlags.NonPublic** to include non-public constructors (that is, private and protected constructors) in the search.

23

[Note: For more information, see **System.Reflection.BindingFlags**.]

24
25
26

Exceptions

Exception	Condition
System.ArgumentNullException	<i>types</i> is null , or at least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

27
28
29
30

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

1
2
3

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

3 **Type.GetConstructor(System.Type[])** 4 **Method**

```
5 [ILASM]  
6 .method public hidebysig instance class  
7 System.Reflection.ConstructorInfo GetConstructor(class  
8 System.Type[] types)  
9 [C#]  
10 public ConstructorInfo GetConstructor(Type[] types)
```

11 **Summary**

12 Returns a public instance constructor defined in the type represented
13 by the current instance. The parameters of the constructor match the
14 specified argument types.

15 **Parameters**

Parameter	Description
<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the constructor to be returned. Specify System.Type.EmptyTypes to obtain a constructor that takes no parameters.

18 **Return Value**

19 A **System.Reflection.ConstructorInfo** object representing the public
20 instance constructor whose parameters match exactly the types in
21 *types*, if found; otherwise, **null**. If the type represented by the current
22 instance is contained in a loaded assembly, the constructor that
23 matches the specified criteria is not public, and the caller does not
24 have sufficient permissions, returns **null**.

25 **Description**

26 This version of **System.Type.GetConstructor** is equivalent to
27 **System.Type.GetConstructor(System.Reflection.BindingFlags.P**
28 **ublic | System.Reflection.BindingFlags.Instance, null, types,**
29 **null)**.

1 **Exceptions**

2

3

Exception	Condition
System.ArgumentNullException	<i>types</i> is null , or at least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

4

5 **Permissions**

6

7

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

8

9

10

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

3 **Type.GetConstructors(System.Reflection.** 4 **BindingFlags) Method**

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class  
7 System.Reflection.ConstructorInfo[]  
8 GetConstructors(valuetype System.Reflection.BindingFlags  
9 bindingAttr)
```

```
10 [C#]  
11 public abstract ConstructorInfo[]  
12 GetConstructors(BindingFlags bindingAttr)
```

13 **Summary**

14 Returns an array of constructors defined in the type represented by
15 the current instance, under the specified binding constraints.

16 **Parameters**

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

19 **Return Value**

20 An array of **System.Reflection.ConstructorInfo** objects that reflect
21 the constructors that are defined in the type represented by the
22 current instance and match the constraints of *bindingAttr*. If
23 **System.Reflection.BindingFlags.NonPublic** and
24 **System.Reflection.BindingFlags.Static** are specified, this array
25 includes the type initializer if it is defined. If no constructors meeting
26 the constraints of *bindingAttr* are defined in the type represented by
27 the current instance, returns an empty array. If the type represented
28 by the current instance is contained in a loaded assembly, the
29 constructors that match the specified criteria are not public, and the
30 caller does not have sufficient permission, returns **null**.

31 **Description**

1 The following **System.Reflection.BindingFlags** are used to define
2 which constructors to include in the search:

- 3 • Specify either **System.Reflection.BindingFlags.Instance** or
4 **System.Reflection.BindingFlags.Static** to get a return value
5 other than **null**.
- 6 • Specify **System.Reflection.BindingFlags.Public** to include
7 public constructors in the search.
- 8 • Specify **System.Reflection.BindingFlags.NonPublic** to
9 include non-public constructors (that is, private and protected
10 constructors) in the search.

11 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

12 Behaviors

13 As described above.

14 Permissions

15

16

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

17

18

19

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

3 **Type.GetConstructors() Method**

```
4 [ILASM]  
5 .method public hidebysig instance class  
6 System.Reflection.ConstructorInfo[] GetConstructors()  
  
7 [C#]  
8 public ConstructorInfo[] GetConstructors()
```

9 **Summary**

10 Returns an array of the public constructors defined in the type
11 represented by the current instance.

12 **Return Value**

13

14 An array of **System.Reflection.ConstructorInfo** objects that reflect
15 the public constructors defined in the type represented by the current
16 instance. If no public constructors are defined in the type represented
17 by the current instance, returns an empty array.

18 **Description**

19 This version of **System.Type.GetConstructors** is equivalent to
20 **System.Type.GetConstructors(System.Reflection.BindingFlags.P**
21 **ublic | System.Reflection.BindingFlags.Instance)**.

22

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

3 **Type.DefaultMembers() Method**

```
4 [ILASM]  
5 .method public hidebysig virtual class  
6 System.Reflection.MemberInfo[] GetDefaultMembers()  
  
7 [C#]  
8 public virtual MemberInfo[] GetDefaultMembers()
```

9 **Summary**

10 Returns an array of **System.Reflection.MemberInfo** objects that
11 reflect the default members defined in the type represented by the
12 current instance.

13 **Return Value**

14
15 An array of **System.Reflection.MemberInfo** objects reflecting the
16 default members of the type represented by the current instance. If
17 the type represented by the current instance does not have any
18 default members, returns an empty array.

19 **Behaviors**

20 The members returned by this method have the
21 **System.Reflection.DefaultMemberAttribute** attribute.

22

1 Type.GetElementType() Method

```
2 [ILASM]
3 .method public hidebysig virtual abstract class System.Type
4 GetElementType()
5
6 [C#]
7 public abstract Type GetElementType()
```

7 Summary

8 Returns the element type of the current **System.Type**.

9 Return Value

10

11 A **System.Type** that represents the type used to create the current
12 instance if the current instance represents an array, pointer, or an
13 argument passed by reference. Otherwise, returns **null**.

14 Example

15

16 The following example demonstrates the
17 **System.Type.GetElementType** method.

18

19

```
20 using System;
21 class TestType {
22     public static void Main() {
23         int[] array = {1,2,3};
24         Type t = array.GetType();
25         Type t2 = t.GetElementType();
26         Console.WriteLine("{0} element type is {1}",array,
27 t2.ToString());
28
29         TestType newMe = new TestType();
30         t = newMe.GetType();
31         t2 = t.GetElementType();
32         Console.WriteLine("{0} element type is {1}", newMe,
33 t2==null? "null": t2.ToString());
34     }
35 }
```

36 The output is

37

38

39

```
System.Int32[] element type is System.Int32
```

1
2 TestType element type is null
3
4

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

Type.GetEvent(System.String, System.Reflection.BindingFlags) Method

```
[ILASM]
.method public hidebysig virtual abstract class
System.Reflection.EventInfo GetEvent(string name, valuetype
System.Reflection.BindingFlags bindingAttr)

[C#]
public abstract EventInfo GetEvent(string name,
BindingFlags bindingAttr)
```

Summary

Returns a **System.Reflection.EventInfo** object reflecting the event that has the specified name, is defined in the type represented by the current instance, and matches the specified binding constraints.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the event to be returned.
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

Return Value

A **System.Reflection.EventInfo** object reflecting the event that is named *name*, is defined in the type represented by the current instance, and matches the constraints of *bindingAttr*. If an event matching these criteria is not found, returns **null**. If the event is not public, the current instance represents a type from a loaded assembly, and the caller does not have sufficient permission, returns **null**.

Description

The following **System.Reflection.BindingFlags** are used to define which events to include in the search:

- Specify either **System.Reflection.BindingFlags.Instance** or **System.Reflection.BindingFlags.Static** to get a return value other than **null**.

- 1 • Specify **System.Reflection.BindingFlags.Public** to include
2 public events in the search.
- 3 • Specify **System.Reflection.BindingFlags.NonPublic** to
4 include non-public events(that is, private and protected events)
5 in the search.
- 6 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
7 to include to include static members declared in ancestors in
8 the search.

9 The following **System.Reflection.BindingFlags** value can be used to
10 change how the search works:

- 11 • **System.Reflection.BindingFlags.DeclaredOnly** to search
12 only the events declared on the type, not events that were
13 simply inherited.

14 [Note: For more information, see **System.Reflection.BindingFlags.**]

15 **Behaviors**

16 As described above.

17 **Exceptions**

18
19

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .

20

21 **Permissions**

22
23

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

24
25
26

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

3 Type.GetEvent(System.String) Method

```
4 [ILASM]  
5 .method public hidebysig instance class  
6 System.Reflection.EventInfo GetEvent(string name)  
  
7 [C#]  
8 public EventInfo GetEvent(string name)
```

9 Summary

10 Returns a **System.Reflection.EventInfo** object reflecting the public
11 event that has the specified name and is defined in the type
12 represented by the current instance.

13 Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the public event to be returned.

16 Return Value

19 A **System.Reflection.EventInfo** object reflecting the public event
20 that is named *name* and is defined in the type represented by the
21 current instance, if found; otherwise, **null**.

22 Description

23 This version of **System.Type.GetEvent** is equivalent to
24 **System.Type.GetEvent**(*name*,
25 **System.Reflection.BindingFlags.Static** |
26 **System.Reflection.BindingFlags.Instance** |
27 **System.Reflection.BindingFlags.Public**).

28 The search for *name* is case-sensitive.
29

30 Exceptions

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .

1
2
3

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

3 Type.GetEvents(System.Reflection.Binding 4 gFlags) Method

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class  
7 System.Reflection.EventInfo[] GetEvents(valuetype  
8 System.Reflection.BindingFlags bindingAttr)  
9  
10 [C#]  
11 public abstract EventInfo[] GetEvents(BindingFlags  
bindingAttr)
```

12 Summary

13 Returns an array of **System.Reflection.EventInfo** objects that
14 reflect the events that are defined in the type represented by the
15 current instance and match the specified binding constraints.

16 Parameters

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null.

19 Return Value

20 An array of **System.Reflection.EventInfo** objects that reflect the
21 events that are defined in the type represented by the current instance
22 and match the constraints of *bindingAttr*. If no events match these
23 constraints, returns an empty array. If the type reflected by the
24 current instance is from a loaded assembly and the caller does not
25 have permission to reflect on non-public objects in loaded assemblies,
26 returns only public events.
27
28

29 Description

30 The following **System.Reflection.BindingFlags** are used to define
31 which events to include in the search:

- 32 • Specify either **System.Reflection.BindingFlags.Instance** or
33 **System.Reflection.BindingFlags.Static** to get a return value
34 other than **null**.

1
2

- Specify **System.Reflection.BindingFlags.Public** to include public events in the search.

3
4
5

- Specify **System.Reflection.BindingFlags.NonPublic** to include non-public events (that is, private and protected events) in the search.

6

[*Note:* For more information, see **System.Reflection.BindingFlags.**]

7

Behaviors

8

As described above.

9

Permissions

10

11

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

12

13

14

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

3 **Type.GetEvents()** Method

```
4 [ILASM]  
5 .method public hidebysig virtual class  
6 System.Reflection.EventInfo[] GetEvents()  
  
7 [C#]  
8 public virtual EventInfo[] GetEvents()
```

9 **Summary**

10 Returns an array of **System.Reflection.EventInfo** objects that
11 reflect the public events defined in the type represented by the current
12 instance.

13 **Return Value**

14
15 An array of **System.Reflection.EventInfo** objects that reflect the
16 public events defined in the type represented by the current instance.
17 If no public events are defined in the type represented by the current
18 instance, returns an empty array.

19 **Behaviors**

20 As described above.

21 **Default**

22 This version of **System.Type.GetEvents** is equivalent to
23 **System.Type.GetEvents(System.Reflection.BindingFlags.Public**
24 **| System.Reflection.BindingFlags.Static |**
25 **System.Reflection.BindingFlags.Instance).**

26

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

Type.GetField(System.String, System.Reflection.BindingFlags) Method

```
[ILASM]
.method public hidebysig virtual abstract class
System.Reflection.FieldInfo GetField(string name, valuetype
System.Reflection.BindingFlags bindingAttr)

[C#]
public abstract FieldInfo GetField(string name,
BindingFlags bindingAttr)
```

Summary

Returns a **System.Reflection.FieldInfo** object reflecting the field that has the specified name, is defined in the type represented by the current instance, and matches the specified binding constraints.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the field to be returned.
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

Return Value

A **System.Reflection.FieldInfo** object reflecting the field that is named *name*, is defined in the type represented by the current instance, and matches the constraints of *bindingAttr*. If a field matching these criteria cannot be found, returns **null**. If the field is not public, the current type is from a loaded assembly, and the caller does not have sufficient permission, returns **null**.

Description

The following **System.Reflection.BindingFlags** are used to define which fields to include in the search:

- Specify either **System.Reflection.BindingFlags.Instance** or **System.Reflection.BindingFlags.Static** to get a return value other than **null**.

- 1 • Specify **System.Reflection.BindingFlags.Public** to include
2 public fields in the search.
- 3 • Specify **System.Reflection.BindingFlags.NonPublic** to
4 include non-public fields (that is, private and protected fields)
5 in the search.
- 6 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
7 to include static members declared in ancestors in the search.

8 The following **System.Reflection.BindingFlags** values can be used
9 to change how the search works:

- 10 • **System.Reflection.BindingFlags.DeclaredOnly** to search
11 only the fields declared in the type, not fields that were simply
12 inherited.
- 13 • **System.Reflection.BindingFlags.IgnoreCase** to ignore the
14 case of *name*.

15 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

16 Behaviors

17 As described above.

18 Exceptions

19
20

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .

21
22
23
24

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

25
26
27

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

Type.GetField(System.String) Method

```
[ILASM]
.method public hidebysig instance class
System.Reflection.FieldInfo GetField(string name)

[C#]
public FieldInfo GetField(string name)
```

Summary

Returns a **System.Reflection.FieldInfo** object reflecting the field that has the specified name and is defined in the type represented by the current instance.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the field to be returned.

Return Value

A **System.Reflection.FieldInfo** object reflecting the field that is named *name* and is defined in the type represented by the current instance, if found; otherwise, **null**. If the selected field is non-public, the type represented by the current instance is from a loaded assembly and the caller does not have sufficient permission to reflect on non-public objects in loaded assemblies, returns **null**.

Description

The search for *name* is case-sensitive.

This version of **System.Type.GetField** is equivalent to **System.Type.GetField(*name*, System.Reflection.BindingFlags.Public | System.Reflection.BindingFlags.Static | System.Reflection.BindingFlags.Instance)**.

Exceptions

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

1
2
3
4

Permissions

System.ArgumentNullException	<i>name</i> is null .
-------------------------------------	------------------------------

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

5
6
7

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

3 Type.GetFields(System.Reflection.Binding 4 Flags) Method

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class  
7 System.Reflection.FieldInfo[] GetFields(valuetype  
8 System.Reflection.BindingFlags bindingAttr)
```

```
9 [C#]  
10 public abstract FieldInfo[] GetFields(BindingFlags  
11 bindingAttr)
```

12 Summary

13 Returns an array of **System.Reflection.FieldInfo** objects that reflect
14 the fields that are defined in the type represented by the current
15 instance and match the specified binding constraints.

16 Parameters

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

19 Return Value

20 An array of **System.Reflection.FieldInfo** objects that reflect the
21 fields that are defined in the type represented by the current instance
22 and match the constraints of *bindingAttr*. If no fields match these
23 constraints, returns an empty array. If the type represented by the
24 current instance is from a loaded assembly and the caller does not
25 have sufficient permission to reflect on non-public objects in loaded
26 assemblies, returns only public fields.
27
28

29 Description

30 The following **System.Reflection.BindingFlags** are used to define
31 which fields to include in the search:

- 1 • Specify either **System.Reflection.BindingFlags.Instance** or
2 **System.Reflection.BindingFlags.Static** in order to get a
3 return value other than **null**.
- 4 • Specify **System.Reflection.BindingFlags.Public** to include
5 public fields in the search.
- 6 • Specify **System.Reflection.BindingFlags.NonPublic** to
7 include non-public fields (that is, private and protected fields)
8 in the search.
- 9 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
10 to include static members declared in ancestors in the search.

11 The following **System.Reflection.BindingFlags** values can be used
12 to change how the search works:

- 13 • **System.Reflection.BindingFlags.DeclaredOnly** to search
14 only the fields declared in the type, not fields that were simply
15 inherited.

16 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

17 **Behaviors**

18 As described above.

19 **Permissions**

20
21

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of a type in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

22
23
24

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

3 **Type.GetFields() Method**

```
4 [ILASM]  
5 .method public hidebysig instance class  
6 System.Reflection.FieldInfo[] GetFields()  
  
7 [C#]  
8 public FieldInfo[] GetFields()
```

9 **Summary**

10 Returns an array of **System.Reflection.FieldInfo** objects that reflect
11 the public fields defined in the type represented by the current
12 instance.

13 **Return Value**

14
15 An array of **System.Reflection.FieldInfo** objects that reflect the
16 public fields defined in the type represented by the current instance. If
17 no public fields are defined in the type represented by the current
18 instance, returns an empty array.

19 **Description**

20 This version of **System.Type.GetFields** is equivalent to
21 **System.Type.GetFields(System.Reflection.BindingFlags.Instance |**
22 **System.Reflection.BindingFlags.Static |**
23 **System.Reflection.BindingFlags.Public)**.

24

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

3 Type.GetInterface(System.String, 4 System.Boolean) Method

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class System.Type  
7 GetInterface(string name, bool ignoreCase)  
  
8 [C#]  
9 public abstract Type GetInterface(string name, bool  
10 ignoreCase)
```

11 Summary

12 Returns the specified interface, specifying whether to do a case-
13 sensitive search.

14 Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the interface to return.
<i>ignoreCase</i>	A System.Boolean where true indicates that the name search is to be done case-insensitively, and false performs a case-sensitive search.

18 Return Value

20 A **System.Type** object representing the interface with the specified
21 name, implemented or inherited by the type represented by the
22 instance, if found; otherwise, **null**.

23 Exceptions

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .

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

3 **Type.GetInterface(System.String) Method**

```
4 [ILASM]  
5 .method public hidebysig instance class System.Type  
6 GetInterface(string name)  
  
7 [C#]  
8 public Type GetInterface(string name)
```

9 **Summary**

10 Searches for the interface with the specified name.

11 **Parameters**

12
13

Parameter	Description
<i>name</i>	A System.String containing the name of the interface to get.

14
15
16

15 **Return Value**

17 A **System.Type** object representing the interface with the specified
18 name, implemented or inherited by the current **System.Type**, if
19 found; otherwise, **null**.

20 **Description**

21 The search for *name* is case-sensitive.

22 **Exceptions**

23
24

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .

25
26
27

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

3 **Type.GetInterfaces() Method**

4 `[ILASM]`
5 `.method public hidebysig virtual abstract class`
6 `System.Type[] GetInterfaces()`

7 `[C#]`
8 `public abstract Type[] GetInterfaces()`

9 **Summary**

10 Returns all interfaces implemented or inherited by the type
11 represented by the current instance.

12 **Return Value**

13

14 An array of **System.Type** objects representing the interfaces
15 implemented or inherited by the type represented by the current
16 instance. If no interfaces are found, returns an empty **System.Type**
17 array.

18

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

3 Type.GetMember(System.String, 4 System.Reflection.BindingFlags) Method

```
5 [ILASM]  
6 .method public hidebysig virtual class  
7 System.Reflection.MemberInfo[] GetMember(string name,  
8 valuetype System.Reflection.BindingFlags bindingAttr)  
9  
10 [C#]  
11 public virtual MemberInfo[] GetMember(string name,  
BindingFlags bindingAttr)
```

12 Summary

13 Returns an array of **System.Reflection.MemberInfo** objects that
14 reflect the members defined in the type represented by the current
15 instance that have the specified name and match the specified binding
16 constraints.

17 Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the member to be returned.
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

21 Return Value

23 An array of **System.Reflection.MemberInfo** objects that reflect the
24 members named *name*, are defined in the type represented by the
25 current instance and match the constraints of *bindingAttr*. If no
26 members match these constraints, returns an empty array. If the
27 selected member is non-public, the type reflected by the current
28 instance is from a loaded assembly and the caller does not have
29 sufficient permission to reflect on non-public objects in loaded
30 assemblies, returns **null**.

31 Description

32 The following **System.Reflection.BindingFlags** are used to define
33 which members to include in the search:

- 1 • Specify either **System.Reflection.BindingFlags.Instance** or
2 **System.Reflection.BindingFlags.Static** to get a return value
3 other than **null**.
- 4 • Specify **System.Reflection.BindingFlags.Public** to include
5 public members in the search.
- 6 • Specify **System.Reflection.BindingFlags.NonPublic** to
7 include non-public members (that is, private and protected
8 members) in the search.
- 9 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
10 to include static members declared in ancestors in the search.

11 The following **System.Reflection.BindingFlags** values can be used
12 to change how the search works:

- 13 • **System.Reflection.BindingFlags.DeclaredOnly** to search
14 only the members declared in the type, not members that were
15 simply inherited.
- 16 • **System.Reflection.BindingFlags.IgnoreCase** to ignore the
17 case of *name*.

18 [Note: For more information, see **System.Reflection.BindingFlags**.]

19 **Behaviors**

20 As described above.

21 **Exceptions**

22
23

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .

24
25
26
27

25 **Permissions**

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

28
29
30

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

3 **Type.GetMember(System.String) Method**

```
4 [ILASM]  
5 .method public hidebysig instance class  
6 System.Reflection.MemberInfo[] GetMember(string name)  
  
7 [C#]  
8 public MemberInfo[] GetMember(string name)
```

9 **Summary**

10 Returns an array of **System.Reflection.MemberInfo** objects that
11 reflect the public members that have the specified name and are
12 defined in the type represented by the current instance.

13 **Parameters**

Parameter	Description
<i>name</i>	A System.String containing the name of the members to be returned.

16 **Return Value**

17
18
19 An array of **System.Reflection.MemberInfo** objects that reflect the
20 public members that are named *name* and are defined in the type
21 represented by the current instance. If no public members with the
22 specified name are defined in the type represented by the current
23 instance, returns an empty array.

24 **Description**

25 This version of **System.Type.GetMember** is equivalent to
26 **System.Type.GetMember(*name*,
27 System.Reflection.BindingFlags.Static |
28 System.Reflection.BindingFlags.Instance |
29 System.Reflection.BindingFlags.Public)**.

30 The search for *name* is case-sensitive.
31

32 **Exceptions**

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

1
2
3

System.ArgumentNullException	<i>name</i> is null .
-------------------------------------	------------------------------

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

3 Type.GetMembers(System.Reflection.Bind 4 ingFlags) Method

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class  
7 System.Reflection.MemberInfo[] GetMembers(valuetype  
8 System.Reflection.BindingFlags bindingAttr)  
9  
10 [C#]  
11 public abstract MemberInfo[] GetMembers(BindingFlags  
bindingAttr)
```

12 Summary

13 Returns an array of **System.Reflection.MemberInfo** objects that
14 reflect the members that are defined in the type represented by the
15 current instance and match the specified binding constraints.

16 Parameters

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

19 Return Value

20 An array of **System.Reflection.MemberInfo** objects that reflect the
21 members defined in the type represented by the current instance that
22 match the constraints of *bindingAttr*. If no members match these
23 constraints, returns an empty array. If the type represented by the
24 current instance is from a loaded assembly and the caller does not
25 have sufficient permission to reflect on non-public objects in loaded
26 assemblies, returns only public members.
27
28

29 Description

30 The following **System.Reflection.BindingFlags** are used to define
31 which members to include in the search:

- 1 • Specify either **System.Reflection.BindingFlags.Instance** or
2 **System.Reflection.BindingFlags.Static** to get a return value
3 other than **null**.
- 4 • Specify **System.Reflection.BindingFlags.Public** to include
5 public members in the search.
- 6 • Specify **System.Reflection.BindingFlags.NonPublic** to
7 include non-public members (that is, private and protected
8 members) in the search.
- 9 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
10 to include static members declared in ancestors in the search.

11 The following **System.Reflection.BindingFlags** values can be used
12 to change how the search works:

- 13 • **System.Reflection.BindingFlags.DeclaredOnly** to search
14 only the members declared in the type, not members that were
15 simply inherited.

16 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

17 **Behaviors**

18 As described above.

19 **Permissions**

20
21

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

22
23
24

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

3 **Type.GetMembers()** Method

```
4 [ILASM]  
5 .method public hidebysig instance class  
6 System.Reflection.MemberInfo[] GetMembers()  
  
7 [C#]  
8 public MemberInfo[] GetMembers()
```

9 **Summary**

10 Returns an array of **System.Reflection.MemberInfo** objects that
11 reflect the public members defined in the type represented by the
12 current instance.

13 **Return Value**

14
15 An array of **System.Reflection.MemberInfo** objects that reflect the
16 public members defined in the type represented by the current
17 instance. If no public members are defined in the type represented by
18 the current instance, returns an empty array.

19 **Description**

20 This version of **System.Type.GetMembers** is equivalent to
21 **System.Type.GetMembers(System.Reflection.BindingFlags.Public |**
22 **System.Reflection.BindingFlags.Static |**
23 **System.Reflection.BindingFlags.Instance).**

24

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

Type.GetMethod(System.String, System.Reflection.BindingFlags) Method

```
[ILASM]
.method public final hidebysig virtual class
System.Reflection.MethodInfo GetMethod(string name,
valuetype System.Reflection.BindingFlags bindingAttr)

[C#]
public MethodInfo GetMethod(string name, BindingFlags
bindingAttr)
```

Summary

Returns a **System.Reflection.MethodInfo** object that reflects the method that has the specified name and is defined in the type represented by the current instance.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the method to be returned.
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

Return Value

A **System.Reflection.MethodInfo** object that reflects the method that is defined in the type represented by the current instance and matches the specified criteria, if found; otherwise, **null**.

Description

The following **System.Reflection.BindingFlags** are used to define which members to include in the search:

- Specify either **System.Reflection.BindingFlags.Instance** or **System.Reflection.BindingFlags.Static** to get a return value other than **null**.

- 1 • Specify **System.Reflection.BindingFlags.Public** to include
2 public members in the search.
- 3 • Specify **System.Reflection.BindingFlags.NonPublic** to
4 include non-public members (that is, private and protected
5 members) in the search.
- 6 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
7 to include static members declared in ancestors in the search.

8 The following **System.Reflection.BindingFlags** values can be used
9 to change how the search works:

- 10 • **System.Reflection.BindingFlags.DeclaredOnly** to search
11 only the members declared in the type, not members that were
12 simply inherited.
- 13 • **System.Reflection.BindingFlags.IgnoreCase** to ignore the
14 case of *name*.

15 [Note: For more information, see **System.Reflection.BindingFlags**.]

16 This version of **System.Type.GetMethod** is equivalent to
17 **System.Type.GetMethod**(*name*, *bindingAttr*, **null**, **null**, **null**).
18

19 Exceptions

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one method matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> is null .

23 Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

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

3 **Type.GetMethod(System.String,**
4 **System.Reflection.BindingFlags,**
5 **System.Reflection.Binder, System.Type[],**
6 **System.Reflection.ParameterModifier[])**
7 **Method**

```
8 [ILASM]
9 .method public final hidebysig virtual class
10 System.Reflection.MethodInfo GetMethod(string name,
11 valuetype System.Reflection.BindingFlags bindingAttr, class
12 System.Reflection.Binder binder, class System.Type[] types,
13 class System.Reflection.ParameterModifier[] modifiers)

14 [C#]
15 public MethodInfo GetMethod(string name, BindingFlags
16 bindingAttr, Binder binder, Type[] types,
17 ParameterModifier[] modifiers)
```

18 Summary

19 Returns a **System.Reflection.MethodInfo** object that reflects the
20 method that matches the specified criteria and is defined in the type
21 represented by the current instance.

22 Parameters

23
24

Parameter	Description
<i>name</i>	A System.String containing the name of the method to be returned.
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .
<i>binder</i>	A System.Reflection.Binder object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specify null to use the System.Type.DefaultBinder .
<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the method to be returned.

modifiers The only defined value for this parameter is **null**.

Return Value

A **System.Reflection.MethodInfo** object that reflects the method defined in the type represented by the current instance that matches the specified criteria. If no method matching the specified criteria is found, returns **null**. If the selected method is non-public, the type reflected by the current instance is from a loaded assembly, and the caller does not have permission to reflect on non-public objects in loaded assemblies, returns **null**.

Description

The following **System.Reflection.BindingFlags** are used to define which members to include in the search:

- Specify either **System.Reflection.BindingFlags.Instance** or **System.Reflection.BindingFlags.Static** to get a return value other than **null**.
- Specify **System.Reflection.BindingFlags.Public** to include public members in the search.
- Specify **System.Reflection.BindingFlags.NonPublic** to include non-public members (that is, private and protected members) in the search.
- Specify **System.Reflection.BindingFlags.FlattenHierarchy** to include static members declared in ancestors in the search.

The following **System.Reflection.BindingFlags** values can be used to change how the search works:

- **System.Reflection.BindingFlags.DeclaredOnly** to search only the members declared in the type, not members that were simply inherited.
- **System.Reflection.BindingFlags.IgnoreCase** to ignore the case of *name*.

[Note: For more information, see **System.Reflection.BindingFlags**.]

Exceptions

Exception	Condition
System.Reflection.	More than one method matching the

AmbiguousMatchException	specified criteria was found.
System.ArgumentNullException	<i>name</i> or <i>types</i> is null . -or- At least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

1
2
3
4

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

5
6
7

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

Type.GetMethod(System.String, System.Type[], System.Reflection.ParameterModifier[]) Method

```
[ILASM]
.method public hidebysig instance class
System.Reflection.MethodInfo GetMethod(string name, class
System.Type[] types, class
System.Reflection.ParameterModifier[] modifiers)

[C#]
public MethodInfo GetMethod(string name, Type[] types,
ParameterModifier[] modifiers)
```

Summary

Returns a **System.Reflection.MethodInfo** object that reflects the public method that has the specified name and is defined in the type represented by the current instance.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the public method to be returned.
<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the method to be returned.
<i>modifiers</i>	The only defined value for this parameter is null .

Return Value

A **System.Reflection.MethodInfo** object reflecting the public method that is defined in the type represented by the current instance and matches the specified criteria, if found; otherwise, **null**.

Description

1 The default binder does not process *modifier*.
 2
 3 The search for *name* is case-sensitive.
 4
 5 This version of **System.Type.GetMethod** is equivalent to
 6 **System.Type.GetMethod** (*name*,
 7 **System.Reflection.BindingFlags.Public**
 8 | **System.Reflection.BindingFlags.Static**
 9 | **System.Reflection.BindingFlags.Instance**, **null**, *types*, *modifiers*).

10 **Exceptions**

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one method matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> or <i>types</i> is null . -or- At least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

13
 14
 15

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

Type.GetMethod(System.String, System.Type[]) Method

```
[ILASM]
.method public hidebysig instance class
System.Reflection.MethodInfo GetMethod(string name, class
System.Type[] types)

[C#]
public MethodInfo GetMethod(string name, Type[] types)
```

Summary

Returns a **System.Reflection.MethodInfo** object that reflects the public method defined in the type represented by the current instance that has the specified name and parameter information.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the public method to be returned.
<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the method to be returned.

Return Value

A **System.Reflection.MethodInfo** object reflecting the public method defined in the type represented by the current instance that matches the specified criteria. If no public method matching the specified criteria is found, returns **null**.

Description

The search for *name* is case-sensitive.

This version of **System.Type.GetMethod** is equivalent to **System.Type.GetMethod(name, System.Reflection.BindingFlags.Public | System.Reflection.BindingFlags.Static | System.Reflection.BindingFlags.Instance, null, types, null)**.

1 Exceptions
2
3

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one method matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> or <i>types</i> is null . -or- At least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

4
5
6

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

3 Type.GetMethod(System.String) Method

```
4 [ILASM]  
5 .method public hidebysig instance class  
6 System.Reflection.MethodInfo GetMethod(string name)  
  
7 [C#]  
8 public MethodInfo GetMethod(string name)
```

9 Summary

10 Returns a **System.Reflection.MethodInfo** object that reflects the
11 public method that has the specified name and is defined in the type
12 represented by the current instance.

13 Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the public method to be returned.

17 Return Value

19 A **System.Reflection.MethodInfo** object reflecting the public
20 method that is defined in the type represented by the current instance
21 and has the specified name, if found; otherwise, **null**.

22 Description

23 The search for *name* is case-sensitive.

24
25 This version of **System.Type.GetMethod** is equivalent to
26 **System.Type.GetMethod**(*name*,
27 **System.Reflection.BindingFlags.Public** |
28 **System.Reflection.BindingFlags.Static** |
29 **System.Reflection.BindingFlags.Instance**, **null**, **null**, **null**).

30 Exceptions

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one method matching the specified criteria was found.

1
2
3

System.ArgumentNullException	<i>name</i> is null .
-------------------------------------	------------------------------

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

3 Type.GetMethods(System.Reflection.BindingFlags) Method 4

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class  
7 System.Reflection.MethodInfo[] GetMethods(valuetype  
8 System.Reflection.BindingFlags bindingAttr)  
9  
10 [C#]  
11 public abstract MethodInfo[] GetMethods(BindingFlags  
bindingAttr)
```

12 Summary

13 Returns an array of **System.Reflection.MethodInfo** objects that
14 reflect the methods defined in the type represented by the current
15 instance that match the specified binding constraints.

16 Parameters

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

19 Return Value

20 An array of **System.Reflection.MethodInfo** objects reflecting the
21 methods defined in the type represented by the current instance that
22 match the constraints of *bindingAttr*. If no such methods found,
23 returns an empty array. If the type represented by the current
24 instance is from a loaded assembly and the caller does not have
25 permission to reflect on non-public objects in loaded assemblies,
26 returns only public methods.
27
28

29 Description

30 The following **System.Reflection.BindingFlags** are used to define
31 which members to include in the search:

- 1 • Specify either **System.Reflection.BindingFlags.Instance** or
2 **System.Reflection.BindingFlags.Static** to get a return value
3 other than **null**.
- 4 • Specify **System.Reflection.BindingFlags.Public** to include
5 public members in the search.
- 6 • Specify **System.Reflection.BindingFlags.NonPublic** to
7 include non-public members (that is, private and protected
8 members) in the search.
- 9 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
10 to include static members declared in ancestors in the search.

11 The following **System.Reflection.BindingFlags** values can be used
12 to change how the search works:

- 13 • **System.Reflection.BindingFlags.DeclaredOnly** to search
14 only the members declared in the type, not members that were
15 simply inherited.

16 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

17 **Behaviors**

18 As described above.

19 **Permissions**

20
21

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

22
23
24

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

3 **Type.GetMethods()** Method

```
4 [ILASM]  
5 .method public hidebysig instance class  
6 System.Reflection.MethodInfo[] GetMethods()  
  
7 [C#]  
8 public MethodInfo[] GetMethods()
```

9 **Summary**

10 Returns the public methods defined in the type represented by the
11 current instance.

12 **Return Value**

13

14 An array of **System.Reflection.MethodInfo** objects reflecting the
15 public methods defined in the type represented by the current instance
16 under the constraints of *bindingAttr*. If no methods matching the
17 constraints are found, returns an empty array.

18 **Description**

19 This version of **System.Type.GetMethods** is equivalent to
20 **System.Type.GetMethods(System.Reflection.BindingFlags.Instance |**
21 **System.Reflection.BindingFlags.Static |**
22 **System.Reflection.BindingFlags.Public)**.

23

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

3 Type.GetNestedType(System.String, 4 System.Reflection.BindingFlags) Method

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class System.Type  
7 GetNestedType(string name, valuetype  
8 System.Reflection.BindingFlags bindingAttr)  
9  
10 [C#]  
11 public abstract Type GetNestedType(string name,  
BindingFlags bindingAttr)
```

12 Summary

13 Returns a nested types defined in the type represented by the current
14 instance that match the specified binding constraints.

15 Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the nested type to return. Specify the unqualified name of the nested type. [<i>Note:</i> For example, for a type B nested within A, if typeA represents the type object for A, the correct invocation is typeA.GetNestedType("B").]
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

19 Return Value

21 A **System.Type** object representing the nested type that matches the
22 specified criteria, if found; otherwise, **null**. If the selected nested type
23 is non-public, the current instance represents a type contained in a
24 loaded assembly and the caller does not have sufficient permissions,
25 returns **null**.

26 Description

27 The following **System.Reflection.BindingFlags** are used to define
28 which members to include in the search:

- 1 • Specify either **System.Reflection.BindingFlags.Instance** or
2 **System.Reflection.BindingFlags.Static** to get a return value
3 other than **null**.
- 4 • Specify **System.Reflection.BindingFlags.Public** to include
5 public members in the search.
- 6 • Specify **System.Reflection.BindingFlags.NonPublic** to
7 include non-public members (that is, private and protected
8 members) in the search.
- 9 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
10 to include static members declared in ancestors in the search.

11 The following **System.Reflection.BindingFlags** values can be used
12 to change how the search works:

- 13 • **System.Reflection.BindingFlags.DeclaredOnly** to search
14 only the members declared in the type, not members that were
15 simply inherited.
- 16 • **System.Reflection.BindingFlags.IgnoreCase** to ignore the
17 case of *name*.

18 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

19 **Exceptions**

20
21

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .

22
23
24
25

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

26
27
28

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

3 Type.GetNestedType(System.String) 4 Method

```
5 [ILASM]  
6 .method public hidebysig instance class System.Type  
7 GetNestedType(string name)  
8 [C#]  
9 public Type GetNestedType(string name)
```

10 Summary

11 Returns the public nested type defined in the type represented by the
12 current instance

13 Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the public nested type to return. Specify the unqualified name of the nested type. [Note: For example, for a type B nested within A, if typeA represents the type object for A, the correct invocation is typeA.GetNestedType("B").]

17 Return Value

19 A **System.Type** object representing the public nested type with the
20 specified name, if found; otherwise, **null**.

21 Description

22 The search for *name* is case-sensitive.

23
24 This version of **System.Type.GetNestedTypes** is equivalent to
25 **System.Type.GetNestedTypes**(*name*,
26 **System.Reflection.BindingFlags.Public**).

27 Exceptions

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .

1
2
3

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

3 **Type.GetNestedTypes(System.Reflection.** 4 **BindingFlags) Method**

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class  
7 System.Type[] GetNestedTypes(valuetype  
8 System.Reflection.BindingFlags bindingAttr)  
9  
10 [C#]  
11 public abstract Type[] GetNestedTypes(BindingFlags  
bindingAttr)
```

12 **Summary**

13 Returns an array containing the nested types defined in the type
14 represented by the current instance that match the specified binding
15 constraints.

16 **Parameters**

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

19 **Return Value**

20 An array of **System.Type** objects representing all types nested within
21 the type represented by the current instance that match the specified
22 binding constraints, if any. Otherwise, returns an empty **System.Type**
23 array. If the type reflected by the current instance is contained in a
24 loaded assembly, the type that matches the specified criteria is not
25 public, and the caller does not have sufficient permission, returns only
26 public types.
27
28

29 **Description**

30 The following **System.Reflection.BindingFlags** are used to define
31 which members to include in the search:

- 32 • Specify **System.Reflection.BindingFlags.Public** to include
33 public members in the search.

- 1 • Specify **System.Reflection.BindingFlags.NonPublic** to
2 include non-public members (that is, private and protected
3 members) in the search.
- 4 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
5 to include static members declared in ancestors in the search.

6 The following **System.Reflection.BindingFlags** values can be used
7 to change how the search works:

- 8 • **System.Reflection.BindingFlags.DeclaredOnly** to search
9 only the members declared in the type, not members that were
10 simply inherited.

11 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

12 Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

15
16
17

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

3 **Type.GetNestedTypes() Method**

```
4 [ILASM]  
5 .method public hidebysig instance class System.Type[]  
6 GetNestedTypes()  
  
7 [C#]  
8 public Type[] GetNestedTypes()
```

9 **Summary**

10 Returns all the public types nested within the current **System.Type**.

11 **Return Value**

12

13 An array of **System.Type** objects representing all public types nested
14 within the type represented by the current instance, if any. Otherwise,
15 returns an empty **System.Type** array.

16 **Description**

17 This version of **System.Type.GetNestedTypes** is equivalent to
18 **System.Type.GetNestedTypes(System.Reflection.BindingFlags.P**
19 **ublic)**.

20

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

3 Type.GetProperties(System.Reflection.Bind 4 ingFlags) Method

```
5 [ILASM]  
6 .method public hidebysig virtual abstract class  
7 System.Reflection.PropertyInfo[] GetProperties(valuetype  
8 System.Reflection.BindingFlags bindingAttr)  
  
9 [C#]  
10 public abstract PropertyInfo[] GetProperties(BindingFlags  
11 bindingAttr)
```

12 Summary

13 Returns an array of **System.Reflection.PropertyInfo** objects that
14 reflect the properties defined for the type represented by the current
15 instance that match the specified binding constraints.

16 Parameters

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

19 Return Value

20 An array of **System.Reflection.PropertyInfo** objects that reflect the
21 properties defined in the type represented by the current instance and
22 match the constraints of *bindingAttr*. If no matching properties are
23 found, returns an empty array. If the type represented by the current
24 instance is from a loaded assembly and the caller does not have
25 permission to reflect on non-public objects in loaded assemblies,
26 returns only public properties.
27
28

29 Description

30 The following **System.Reflection.BindingFlags** are used to define
31 which members to include in the search:

- 32 • Specify either **System.Reflection.BindingFlags.Instance** or
33 **System.Reflection.BindingFlags.Static** to get a return value
34 other than **null**.

- 1 • Specify **System.Reflection.BindingFlags.Public** to include
2 public members in the search.
- 3 • Specify **System.Reflection.BindingFlags.NonPublic** to
4 include non-public members (that is, private and protected
5 members) in the search.
- 6 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
7 to include static members declared in ancestors in the search.

8 The following **System.Reflection.BindingFlags** values can be used
9 to change how the search works:

- 10 • **System.Reflection.BindingFlags.DeclaredOnly** to search
11 only the members declared in the type, not members that were
12 simply inherited.

13 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

14 Behaviors

15 A property is considered by reflection to be **public** if it has at least one
16 accessor that is **public**. Otherwise, the property is not **public**.

17 Permissions

18
19

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

20
21
22

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

3 **Type.GetProperties() Method**

```
4 [ILASM]  
5 .method public hidebysig instance class  
6 System.Reflection.PropertyInfo[] GetProperties()  
  
7 [C#]  
8 public PropertyInfo[] GetProperties()
```

9 **Summary**

10 Returns an array of **System.Reflection.PropertyInfo** objects that
11 reflect the public properties defined in the type represented by the
12 current instance.

13 **Return Value**

14
15 An array of **System.Reflection.PropertyInfo** objects that reflect the
16 public properties defined in the type represented by the current
17 instance. If no public properties are found, returns an empty array.

18 **Description**

19 This version of **System.Type.GetProperties** is equivalent to
20 **System.Type.GetProperties(System.Reflection.BindingFlags.Ins**
21 **tance | System.Reflection.BindingFlags.Static |**
22 **System.Reflection.BindingFlags.Public)**.

23
24 A property is considered by reflection to be **public** if it has at least one
25 accessor that is **public**. Otherwise, the property is considered to be
26 not **public**.

27

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

3 **Type.GetProperty(System.String,**
4 **System.Reflection.BindingFlags,**
5 **System.Reflection.Binder, System.Type,**
6 **System.Type[],**
7 **System.Reflection.ParameterModifier[])**
8 **Method**

```
9 [ILASM]  
10 .method public final hidebysig virtual class  
11 System.Reflection.PropertyInfo GetProperty(string name,  
12 valueType System.Reflection.BindingFlags bindingAttr, class  
13 System.Reflection.Binder binder, class System.Type  
14 returnType, class System.Type[] types, class  
15 System.Reflection.ParameterModifier[] modifiers)  
  
16 [C#]  
17 public PropertyInfo GetProperty(string name, BindingFlags  
18 bindingAttr, Binder binder, Type returnType, Type[] types,  
19 ParameterModifier[] modifiers)
```

20 **Summary**

21 Returns a **System.Reflection.PropertyInfo** object that reflects the
22 property defined in the type represented by the current instance that
23 matches the specified search criteria.

24 **Parameters**

Parameter	Description
<i>name</i>	A System.String containing the name of the property to be returned.
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .
<i>binder</i>	A System.Reflection.Binder object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specify null to use the System.Type.DefaultBinder .
<i>returnType</i>	A System.Type object that represents the type of the property to be returned.

<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the indexer to be returned. Specify System.Type.EmptyTypes to obtain a property that is not indexed.
<i>modifiers</i>	The only defined value for this parameter is null .

1
2
3

Return Value

4 A **System.Reflection.PropertyInfo** object reflecting the property
5 that is defined in the type represented by the current instance and
6 matches the specified criteria. If no matching property is found,
7 returns **null**. If the type reflected by the current instance is contained
8 in a loaded assembly, the property that matches the specified criteria
9 is not public, and the caller does not have sufficient permission,
10 returns **null**.

Description

12 The following **System.Reflection.BindingFlags** are used to define
13 which members to include in the search:

- 14 • Specify either **System.Reflection.BindingFlags.Instance** or
15 **System.Reflection.BindingFlags.Static** to get a return value
16 other than **null**.
- 17 • Specify **System.Reflection.BindingFlags.Public** to include
18 public members in the search.
- 19 • Specify **System.Reflection.BindingFlags.NonPublic** to
20 include non-public members (that is, private and protected
21 members) in the search.
- 22 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
23 to include static members declared in ancestors in the search.

24 The following **System.Reflection.BindingFlags** values can be used
25 to change how the search works:

- 26 • **System.Reflection.BindingFlags.DeclaredOnly** to search
27 only the members declared in the type, not members that were
28 simply inherited.
- 29 • **System.Reflection.BindingFlags.IgnoreCase** to ignore the
30 case of *name*.

31 [*Note:* For more information, see **System.Reflection.BindingFlags**.]

32
33 This version of **System.Type.GetProperty** is equivalent to

1 **System.Type.GetPropertyImpl**(*name*, *bindingAttr*, *binder*,
2 *returnType*, *types*, *modifiers*).

3
4 The search for *name* is case-sensitive.

5
6 Different programming languages use different syntax to specify
7 indexed properties. Internally, this property is referred to by the name
8 "Item" in the metadata. Therefore, any attempt to retrieve an indexed
9 property using reflection is required to specify this internal name in
10 order for the **PropertyInfo** to be returned correctly.

11 Exceptions

12
13

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one property matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> or <i>types</i> is null , or at least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

14

15 Permissions

16
17

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

18

19

20

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

3 Type.GetProperty(System.String, 4 System.Reflection.BindingFlags) Method

```
5 [ILASM]  
6 .method public final hidebysig virtual class  
7 System.Reflection.PropertyInfo GetProperty(string name,  
8 valuetype System.Reflection.BindingFlags bindingAttr)  
9  
10 [C#]  
11 public PropertyInfo GetProperty(string name, BindingFlags  
bindingAttr)
```

12 Summary

13 Returns a **System.Reflection.PropertyInfo** object that reflects the
14 property defined in the type represented by the current instance that
15 matches the specified search criteria.

16 Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the property to be returned.
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .

19 Return Value

20
21
22 A **System.Reflection.PropertyInfo** object reflecting the property
23 defined in the type represented by the current instance that matches
24 the specified criteria. If no matching property is found, returns **null**. If
25 the type reflected by the current instance is contained in a loaded
26 assembly, the property that matches the specified criteria is not
27 public, and the caller does not have sufficient permission, returns **null**.

28 Description

29 The following **System.Reflection.BindingFlags** are used to define
30 which members to include in the search:

- 1 • Specify either **System.Reflection.BindingFlags.Instance** or
2 **System.Reflection.BindingFlags.Static** to get a return value
3 other than **null**.
- 4 • Specify **System.Reflection.BindingFlags.Public** to include
5 public members in the search.
- 6 • Specify **System.Reflection.BindingFlags.NonPublic** to
7 include non-public members (that is, private and protected
8 members) in the search.
- 9 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
10 to include static members declared in ancestors in the search.

11 The following **System.Reflection.BindingFlags** values can be used
12 to change how the search works:

- 13 • **System.Reflection.BindingFlags.DeclaredOnly** to search
14 only the members declared in the type, not members that were
15 simply inherited.
- 16 • **System.Reflection.BindingFlags.IgnoreCase** to ignore the
17 case of *name*.

18 [*Note:* For more information, see **System.Reflection.BindingFlags.**]

19 This version of **System.Type.GetProperty** is equivalent to
20 **System.Type.GetPropertyImpl**(*name, bindingAttr, null, null, null,*
21 **null**).

22 The search for *name* is case-sensitive.

25 Exceptions

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one property matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> is null .

28 Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

1
2
3

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

3 **Type.GetProperty(System.String,** 4 **System.Type, System.Type[]) Method**

```
5 [ILASM]  
6 .method public hidebysig instance class  
7 System.Reflection.PropertyInfo GetProperty(string name,  
8 class System.Type returnType, class System.Type[] types)  
9 [C#]  
10 public PropertyInfo GetProperty(string name, Type  
11 returnType, Type[] types)
```

12 **Summary**

13 Returns a **System.Reflection.PropertyInfo** object that reflects the
14 public property defined in the type represented by the current instance
15 that matches the specified search criteria.

16 **Parameters**

Parameter	Description
<i>name</i>	A System.String containing the name of the public property to be returned.
<i>returnType</i>	A System.Type object that represents the type of the public property to be returned.
<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the indexer to be returned. Specify System.Type.EmptyTypes for a property that is not indexed.

19 **Return Value**

20 A **System.Reflection.PropertyInfo** object reflecting the public
21 property defined in the type represented by the current instance that
22 matches the specified criteria. If no matching property is found,
23 returns **null**.
24
25

26 **Description**

27 This version of **System.Type.GetProperty** is equivalent to
28 **System.Type.GetPropertyImpl(name,**

1 **System.Reflection.BindingFlags.Static** |
2 **System.Reflection.BindingFlags.Instance** |
3 **System.Reflection.BindingFlags.Public**, **null**, *returnTypes*, *types*,
4 **null**).

5
6 The search for *name* is case-sensitive.

7
8 Different programming languages use different syntax to specify
9 indexed properties. Internally, this property is referred to by the name
10 "Item" in the metadata. Therefore, any attempt to retrieve an indexed
11 property using reflection is required to specify this internal name in
12 order for the **PropertyInfo** to be returned correctly.

13 Exceptions

14
15

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one property matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> or <i>types</i> is null , or at least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

16
17
18

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

3 **Type.GetProperty(System.String,** 4 **System.Type[]) Method**

```
5 [ILASM]  
6 .method public hidebysig instance class  
7 System.Reflection.PropertyInfo GetProperty(string name,  
8 class System.Type[] types)  
9 [C#]  
10 public PropertyInfo GetProperty(string name, Type[] types)
```

11 **Summary**

12 Returns a **System.Reflection.PropertyInfo** object that reflects the
13 public property defined in the type represented by the current instance
14 that matches the specified search criteria.

15 **Parameters**

Parameter	Description
<i>name</i>	A System.String containing the name of the public property to be returned.
<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the indexer to be returned. Specify System.Type.EmptyTypes to obtain a property that is not indexed.

19 **Return Value**

21 A **System.Reflection.PropertyInfo** object reflecting the public
22 property defined on the type represented by the current instance that
23 matches the specified criteria. If no matching property is found,
24 returns **null**.

25 **Description**

26 This version of **System.Type.GetProperty** is equivalent to
27 **System.Type.GetPropertyImpl**(*name*,
28 **System.Reflection.BindingFlags.Static** |
29 **System.Reflection.BindingFlags.Instance** |
30 **System.Reflection.BindingFlags.Public**, *null*, *null*, *types*, *null*).
31

1 The search for *name* is case-sensitive.

2

3 Different programming languages use different syntax to specify
4 indexed properties. Internally, this property is referred to by the name
5 "Item" in the metadata. Therefore, any attempt to retrieve an indexed
6 property using reflection is required to specify this internal name in
7 order for the **PropertyInfo** to be returned correctly.

8 **Exceptions**

9

10

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one property matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> or <i>types</i> is null , or at least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

11

12

13

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

3 **Type.GetProperty(System.String,** 4 **System.Type) Method**

```
5 [ILASM]  
6 .method public hidebysig instance class  
7 System.Reflection.PropertyInfo GetProperty(string name,  
8 class System.Type returnType)
```

```
9 [C#]  
10 public PropertyInfo GetProperty(string name, Type  
11 returnType)
```

12 **Summary**

13 Returns a **System.Reflection.PropertyInfo** object that reflects the
14 public property defined in the type represented by the current instance
15 that matches the specified search criteria.

16 **Parameters**

Parameter	Description
<i>name</i>	A System.String containing the name of the property to be returned.
<i>returnType</i>	A System.Type object that represents the type of the property to be returned.

19 **Return Value**

20
21
22 A **System.Reflection.PropertyInfo** object reflecting the public
23 property defined on the type represented by the current instance that
24 matches the specified criteria. If no matching property is found,
25 returns **null**.

26 **Description**

27 This version of **System.Type.GetProperty** is equivalent to
28 **System.Type.GetPropertyImpl(name,**
29 **System.Reflection.BindingFlags.Static |**
30 **System.Reflection.BindingFlags.Instance |**
31 **System.Reflection.BindingFlags.Public, null, returnType, null,**
32 **null)**.

33
34 The search for *name* is case-sensitive.

1 **Exceptions**

2

3

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one property matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> is null .

4

5

6

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

Type.GetProperty(System.String) Method

```
[ILASM]
.method public hidebysig instance class
System.Reflection.PropertyInfo GetProperty(string name)

[C#]
public PropertyInfo GetProperty(string name)
```

Summary

Returns a **System.Reflection.PropertyInfo** object that reflects the public property defined in the type represented by the current instance that has the specified name.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the property to be returned.

Return Value

A **System.Reflection.PropertyInfo** object reflecting the public property defined on the type represented by the current instance that has the specified name. If no matching property is found, returns **null**.

Description

This version of **System.Type.GetProperty** is equivalent to **System.Type.GetPropertyImpl(*name*, System.Reflection.BindingFlags.Static | System.Reflection.BindingFlags.Instance | System.Reflection.BindingFlags.Public, null, null, null, null)**.

The search for *name* is case-sensitive.

Exceptions

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one property matching the specified criteria was found.

1
2
3

System.ArgumentNullException	<i>name</i> is null .
-------------------------------------	------------------------------

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

**Type.GetPropertyImpl(System.String,
System.Reflection.BindingFlags,
System.Reflection.Binder, System.Type,
System.Type[],
System.Reflection.ParameterModifier[])
Method**

```
[ILASM]
.method family hidebysig virtual abstract class
System.Reflection.PropertyInfo GetPropertyImpl(string name,
valuetype System.Reflection.BindingFlags bindingAttr, class
System.Reflection.Binder binder, class System.Type
returnType, class System.Type[] types, class
System.Reflection.ParameterModifier[] modifiers)

[C#]
protected abstract PropertyInfo GetPropertyImpl(string
name, BindingFlags bindingAttr, Binder binder, Type
returnType, Type[] types, ParameterModifier[] modifiers)
```

Summary

When overridden in a derived class implements the **System.Type.GetProperty** method and returns a **System.Reflection.PropertyInfo** object that reflects the property defined in the type represented by the current instance that matches the specified search criteria.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the property to be returned.
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, this method returns null .
<i>binder</i>	A System.Reflection.Binder object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specify null to use the System.Type.DefaultBinder .

<i>returnType</i>	A System.Type object that represents the type of the property to be returned.
<i>types</i>	An array of System.Type objects. The elements in the array are of the same number, in the same order, and represent the same types as the parameters for the indexer to be returned. Specify System.Type.EmptyTypes to obtain a property that is not indexed.
<i>modifiers</i>	The only defined value for this parameter is null .

1
2
3

Return Value

4
5
6
7
8
9

A **System.Reflection.PropertyInfo** object representing the property that matches the specified search criteria, if found; otherwise, **null**. If the type reflected by the current instance is from a loaded assembly, the matching property is not public, and the caller does not have permission to reflect on non-public objects in loaded assemblies, returns **null**.

Description

11
12

The following **System.Reflection.BindingFlags** are used to define which members to include in the search:

13
14
15

- Specify either **System.Reflection.BindingFlags.Instance** or **System.Reflection.BindingFlags.Static** to get a return value other than **null**.

16
17

- Specify **System.Reflection.BindingFlags.Public** to include public members in the search.

18
19
20

- Specify **System.Reflection.BindingFlags.NonPublic** to include non-public members (that is, private and protected members) in the search.

21
22

- Specify **System.Reflection.BindingFlags.FlattenHierarchy** to include static members declared in ancestors in the search.

23
24

The following **System.Reflection.BindingFlags** values can be used to change how the search works:

25
26
27

- **System.Reflection.BindingFlags.DeclaredOnly** to search only the members declared in the type, not members that were simply inherited.

28
29

- **System.Reflection.BindingFlags.IgnoreCase** to ignore the case of *name*.

30

[*Note:* For more information, see **System.Reflection.BindingFlags**.]

1 **Behaviors**

2 Different programming languages use different syntax to specify
3 indexed properties. Internally, this property is referred to by the name
4 "Item" in the metadata. Therefore, any attempt to retrieve an indexed
5 property using reflection is required to specify this internal name in
6 order for the **PropertyInfo** to be returned correctly.

7 **Exceptions**

8
9

Exception	Condition
System.Reflection.AmbiguousMatchException	More than one property matching the specified criteria was found.
System.ArgumentNullException	<i>name</i> or <i>types</i> is null , or at least one of the elements in <i>types</i> is null .
System.ArgumentException	<i>types</i> has more than one dimension.

10
11
12
13

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

14
15
16

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

3 **Type.GetType(System.String,** 4 **System.Boolean, System.Boolean) Method**

```
5 [ILASM]  
6 .method public hidebysig static class System.Type  
7 GetType(string typeName, bool throwOnError, bool  
8 ignoreCase)  
9 [C#]  
10 public static Type GetType(string typeName, bool  
11 throwOnError, bool ignoreCase)
```

12 **Summary**

13 Returns the **System.Type** with the specified name, optionally
14 performing a case-sensitive search and throwing an exception if an
15 error occurs while loading the **System.Type**.

16 **Parameters**

Parameter	Description
<i>typeName</i>	A System.String containing the name of the System.Type to return.
<i>throwOnError</i>	A System.Boolean . Specify true to throw a System.TypeLoadException if an error occurs while loading the System.Type . Specify false to ignore errors while loading the System.Type .
<i>ignoreCase</i>	A System.Boolean . Specify true to perform a case-insensitive search for <i>typeName</i> . Specify false to perform a case-sensitive search for <i>typeName</i> .

19 **Return Value**

20 The **System.Type** with the specified name, if found; otherwise, **null**.
21 If the requested type is non-public and the caller does not have
22 permission to reflect non-public objects outside the current assembly,
23 this method returns **null**.
24
25

26 **Description**

1
2
3
4
5
6
7
8
9
10
11
12
13

typeName can be a simple type name, a fully qualified name, or a complex name that includes an assembly name. [Note: **System.Type.AssemblyQualifiedName** returns a fully qualified type name including nested types and the assembly name.]

If *typeName* includes only the name of the **System.Type**, this method searches in the calling object's assembly, then in the mscorlib.dll assembly. If *typeName* is fully qualified with the partial or complete assembly name, this method searches in the specified assembly.

[Note:

The following table shows calls to **GetType** for various types.

To Get	Use
An unmanaged pointer to MyType	Type.GetType("MyType*")
An unmanaged pointer to a pointer to MyType	Type.GetType("MyType**")
A managed pointer or reference to MyType	Type.GetType("MyType&"). Note that unlike pointers, references are limited to one level.
A parent class and a nested class	Type.GetType("MyParentClass+MyNestedClass")
A one-dimensional array with a lower bound of 0	Type.GetType("MyArray[]")
A one-dimensional array with an unknown lower bound	Type.GetType("MyArray[*]")
An n-dimensional array	A comma (,) inside the brackets a total of n-1 times. For example, System.Object[, ,] represents a three-dimensional Object array.
A two-dimensional array's array	Type.GetType("MyArray[][]")
A rectangular two-dimensional array with unknown lower bounds	Type.GetType("MyArray[*,*]") or Type.GetType("MyArray[,]")

14
15
16
17
18

]

Exceptions

Exception	Condition
System.ArgumentNullException	<i>typeName</i> is null .
System.Reflection.TargetInvocationException	A type initializer was invoked and threw an exception.
System.TypeLoadException	<i>throwOnError</i> is true and an error was encountered while loading the selected System.Type .

1
2
3
4

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

5
6
7

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

3 **Type.GetType(System.String,** 4 **System.Boolean) Method**

```
5 [ILASM]  
6 .method public hidebysig static class System.Type  
7 GetType(string typeName, bool throwOnError)  
  
8 [C#]  
9 public static Type GetType(string typeName, bool  
10 throwOnError)
```

11 **Summary**

12 Returns the **System.Type** with the specified name, optionally
13 throwing an exception if an error occurs while loading the
14 **System.Type**.

15 **Parameters**

Parameter	Description
<i>typeName</i>	A System.String containing the case-sensitive name of the System.Type to return.
<i>throwOnError</i>	A System.Boolean . Specify true to throw a System.TypeLoadException if an error occurs while loading the System.Type . Specify false to ignore errors while loading the System.Type .

18 19 **Return Value**

20
21 The **System.Type** with the specified name, if found; otherwise, **null**.
22 If the requested type is non-public and the caller does not have
23 permission to reflect non-public objects outside the current assembly,
24 this method returns **null**.

25 **Description**

26 This method is equivalent to **System.Type.GetType(name,**
27 **throwOnError, false)**.

28
29 *typeName* can be a simple type name, a fully qualified name, or a
30 complex name that includes an assembly name specification. If
31 *typeName* includes only the name of the **System.Type**, this method

1 searches in the calling object's assembly, then in the mscorlib.dll
2 assembly. If *typeName* is fully qualified with the partial or complete
3 assembly name, this method searches in the specified assembly.

4
5 [Note: **System.Type.AssemblyQualifiedName** can return a fully
6 qualified type name including nested types and the assembly name.
7 For complete details, see **System.Type.GetType(System.String,**
8 **System.Boolean, System.Boolean).**]

9 **Exceptions**

10
11

Exception	Condition
System.ArgumentNullException	<i>typeName</i> is null .
System.Reflection.TargetInvocationException	A type initializer was invoked and threw an exception.
System.TypeLoadException	<i>throwOnError</i> is true and an error was encountered while loading the System.Type .

12
13
14
15

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public objects. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation

16
17
18

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

3 Type.GetType(System.String) Method

```
4 [ILASM]  
5 .method public hidebysig static class System.Type  
6 GetType(string typeName)
```

```
7 [C#]  
8 public static Type GetType(string typeName)
```

9 Summary

10 Returns the **System.Type** with the specified name.

11 Parameters

Parameter	Description
<i>typeName</i>	A System.String containing the case-sensitive name of the System.Type to return.

15 Return Value

17 The **System.Type** with the specified name, if found; otherwise, **null**.
18 If the requested type is non-public and the caller does not have
19 permission to reflect non-public objects outside the current assembly,
20 this method returns **null**.

21 Description

22 This method is equivalent to **System.Type.GetType(name, false,**
23 **false)**.

24
25 *typeName* can be a simple type name, a type name that includes a
26 namespace, or a complex name that includes an assembly name
27 specification. If *typeName* includes only the name of the
28 **System.Type**, this method searches in the calling object's assembly,
29 then in the mscorlib.dll assembly. If *typeName* is fully qualified with
30 the partial or complete assembly name, this method searches in the
31 specified assembly.

32
33 [Note: **System.Type.AssemblyQualifiedName** can return a fully
34 qualified type name including nested types and the assembly name.
35 For complete details, see **System.Type.GetType(System.String,**
36 **System.Boolean, System.Boolean)**.]

1 **Exceptions**

2

3

Exception	Condition
System.ArgumentNullException	<i>typeName</i> is null .
System.Reflection.TargetInvocationException	A type initializer was invoked and threw an exception.

4

5 **Permissions**

6

7

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

8

9

10

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

Type.GetTypeArray(System.Object[]) Method

```
[ILASM]  
.method public hidebysig static class System.Type[]  
GetTypeArray(class System.Object[] args)  
  
[C#]  
public static Type[] GetTypeArray(object[] args)
```

Summary

Returns the types of the objects in the specified array.

Parameters

Parameter	Description
<i>args</i>	An array of objects whose types are to be returned.

Return Value

An array of **System.Type** objects representing the types of the corresponding elements in *args*. If a requested type is not public and the caller does not have permission to reflect non-public objects outside the current assembly, the corresponding element in the array returned by this method will be **null**.

Exceptions

Exception	Condition
System.ArgumentNullException	<i>args</i> is null .
System.Reflection.TargetInvocationException	The type initializers were invoked and at least one threw an exception.

Permissions

Permission	Description
System.Security.Permissions	Requires permission to retrieve information on

1
2
3

ReflectionPermission	non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .
-----------------------------	---

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

3 Type.GetTypeFromHandle(System.Runtime 4 eTypeHandle) Method

```
5 [ILASM]  
6 .method public hidebysig static class System.Type  
7 GetTypeFromHandle(valuetype System.RuntimeTypeHandle  
8 handle)  
9 [C#]  
10 public static Type GetTypeFromHandle(RuntimeTypeHandle  
11 handle)
```

12 Summary

13 Gets the **System.Type** referenced by the specified type handle.

14 Parameters

Parameter	Description
<i>handle</i>	The System.RuntimeTypeHandle object that refers to the desired System.Type .

18 Return Value

20 The **System.Type** referenced by the specified
21 **System.RuntimeTypeHandle**.

22 Description

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

25 Exceptions

Exception	Condition
System.ArgumentNullException	<i>handle</i> is null .
System.Security.SecurityException	The requested type is non-public and outside the current assembly, and the caller does not have the required permission.

1
2
3
4

Permissions

System.Reflection.TargetInvocationException	A type initializer was invoked and threw an exception.
--	--

5
6
7

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public objects. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation

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

3 **Type.GetHandle(System.Object)** 4 **Method**

```
5 [ILASM]  
6 .method public hidebysig static valuetype  
7 System.RuntimeTypeHandle GetHandle(object o)  
8 [C#]  
9 public static RuntimeTypeHandle GetHandle(object o)
```

10 **Summary**

11 Returns the handle for the **System.Type** of the specified object.

12 **Parameters**

13
14

Parameter	Description
<i>o</i>	The object for which to get the type handle.

15
16
17

16 **Return Value**

18 The **System.RuntimeTypeHandle** for the **System.Type** of the
19 specified **System.Object**.

20 **Description**

21 The handle is valid only in the application domain in which it was
22 obtained.

23

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

3 **Type.HasElementTypeImpl()** Method

```
4 [ILASM]  
5 .method family hidebysig virtual abstract bool  
6 HasElementTypeImpl()  
  
7 [C#]  
8 protected abstract bool HasElementTypeImpl()
```

9 **Summary**

10 When overridden in a derived class, implements the
11 **System.Type.HasElementType** property and determines whether
12 the current **System.Type** encompasses or refers to another type; that
13 is, whether the current **System.Type** is an array, a pointer, or is
14 passed by reference.

15 **Return Value**

16

17 **true** if the **System.Type** is an array, a pointer, or is passed by
18 reference; otherwise, **false**.

19 **Description**

20 [*Note:* For example, **System.Type.GetType**
21 ("System.Int32[]").HasElementTypeImpl returns **true**, but
22 **System.Type.GetType** ("System.Int32").HasElementTypeImpl
23 returns **false**. **System.Type.HasElementTypeImpl** also returns **true**
24 for "System.Int32*" and "System.Int32&".]

25

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

Type.InvokeMember(System.String, System.Reflection.BindingFlags, System.Reflection.Binder, System.Object, System.Object[], System.Reflection.ParameterModifier[], System.Globalization.CultureInfo, System.String[]) Method

```
[ILASM]
.method public hidebysig virtual abstract object
InvokeMember(string name, valuetype
System.Reflection.BindingFlags invokeAttr, class
System.Reflection.Binder binder, object target, class
System.Object[] args, class
System.Reflection.ParameterModifier[] modifiers, class
System.Globalization.CultureInfo culture, class
System.String[] namedParameters)

[C#]
public abstract object InvokeMember(string name,
BindingFlags invokeAttr, Binder binder, object target,
object[] args, ParameterModifier[] modifiers, CultureInfo
culture, string[] namedParameters)
```

Summary

Invokes or accesses a member defined on the type represented by the current instance that matches the specified binding criteria.

Parameters

Parameter	Description
<i>name</i>	A System.String containing the name of the constructor or method to invoke, or property or field to access. If the type represented by the current instance has a default member, specify System.String.Empty to invoke that member. [Note: For more information on default members, see System.Reflection.DefaultMemberAttribute .]
<i>invokeAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, System.Reflection.BindingFlags.Public

	System.Reflection.BindingFlags.Instance is used by default.
<i>target</i>	A System.Object on which to invoke the member that matches the other specified criteria. If the matching member is static , this parameter is ignored.
<i>args</i>	An array of objects containing the arguments to pass to the member to be invoked. The elements of this array are of the same number and in the same order by assignment-compatible type as specified by the contract of the member to be bound if and only if <i>nameParameters</i> is null . If <i>namedParameters</i> is not null , the order of the elements in <i>args</i> corresponds to the order of the parameters specified in <i>namedParameters</i> . Specify an empty array or null for a member that takes no parameters.
<i>modifiers</i>	The only defined value for this parameter is null .
<i>culture</i>	The only defined value for this parameter is null .
<i>namedParameters</i>	An array of System.String objects containing the names of the parameters to which the values in <i>args</i> are passed. These names are processed in a case-sensitive manner and have a one-to-one correspondence with the elements of <i>args</i> . Specify an empty array or null for a member that takes no parameters. Specify null to have this parameter ignored.

1
2
3

Return Value

4
5
6

A **System.Object** containing the return value of the invoked or accessed member. If the member does not have a return value, returns a **System.Object** containing **System.Void**.

7

Description

8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23

System.Type.InvokeMember calls a constructor or a method, gets or sets a property, gets or sets a field, or gets or sets an element of an array.

The binder finds all of the matching members. These members are found based upon the type of binding specified by *InvokeAttr*. The **System.Reflection.Binder.BindToMethod** is responsible for selecting the method to be invoked. The default binder selects the most specific match. The set of members is then filtered by name, number of arguments, and a set of search modifiers defined in the binder. After the member is selected, it is invoked or accessed.

Accessibility is checked at that point. Access restrictions are ignored for fully trusted code; that is, private constructors, methods, fields, and properties can be accessed and invoked via reflection whenever the code is fully trusted.

1 The following **System.Reflection.BindingFlags** are used to define
2 which members to include in the search:

- 3 • Specify either **System.Reflection.BindingFlags.Instance** or
4 **System.Reflection.BindingFlags.Static** to get a return value
5 other than **null**.
- 6 • Specify **System.Reflection.BindingFlags.Public** to include
7 public members in the search.
- 8 • Specify **System.Reflection.BindingFlags.NonPublic** to
9 include non-public members (that is, private and protected
10 members) in the search.
- 11 • Specify **System.Reflection.BindingFlags.FlattenHierarchy**
12 to include static members declared in ancestors in the search.

13 The following **System.Reflection.BindingFlags** values can be used
14 to change how the search works:

- 15 • **System.Reflection.BindingFlags.DeclaredOnly** to search
16 only the members declared in the type, not members that were
17 simply inherited.
- 18 • **System.Reflection.BindingFlags.IgnoreCase** to ignore the
19 case of *name*.

20 [Note: For more information, see **System.Reflection.BindingFlags**.]

21 Behaviors

22 Each parameter in the *namedParameters* array is assigned the value in
23 the corresponding element in the *args* array. If the length of *args* is
24 greater than the length of *namedParameters*, the remaining argument
25 values are passed in order.

26
27 A member will be found only if the number of parameters in the
28 member declaration equals the number of arguments in the *args* array
29 (unless default arguments are defined on the member). Also, The type
30 of each argument is required to be convertible by the binder to the
31 type of the parameter.

32
33 It is required that the caller specify values for *bindingAttr* as follows:

Action	BindingFlags
Invoke a constructor.	System.Reflection.BindingFlags.CreateInstance . This flag is not valid with the other flags in this table. If this flag is specified, <i>name</i> is ignored.
Invoke a method.	System.Reflection.BindingFlags.InvokeMethod . This flag if not valid with Svstem.Reflection.BindinaFlaas.CreateInstance .

	System.Reflection.BindingFlags.SetField , or System.Reflection.BindingFlags.SetProperty .
Define a field value.	System.Reflection.BindingFlags.SetField . This flag is not valid with System.Reflection.BindingFlags.CreateInstance , System.Reflection.BindingFlags.InvokeMethod , or System.Reflection.BindingFlags.GetField .
Return a field value.	System.Reflection.BindingFlags.GetField . This flag is not valid with System.Reflection.BindingFlags.CreateInstance , System.Reflection.BindingFlags.InvokeMethod , or System.Reflection.BindingFlags.SetField .
Set a property.	System.Reflection.BindingFlags.SetProperty . This flag is not valid with System.Reflection.BindingFlags.CreateInstance , System.Reflection.BindingFlags.InvokeMethod , or System.Reflection.BindingFlags.GetProperty .
Get a property.	System.Reflection.BindingFlags.GetProperty . This flag is not valid with System.Reflection.BindingFlags.CreateInstance , System.Reflection.BindingFlags.InvokeMethod , or System.Reflection.BindingFlags.SetProperty .

1
2
3
4

[Note: For more information, see **System.Reflection.BindingFlags**.]

5 Usage

6 **System.Type.InvokeMember** can be used to invoke methods with
7 parameters that have default values. To bind to these methods,
8 **System.Reflection.BindingFlags.OptionalParamBinding** must be
9 specified. For a parameter that has a default value, the caller can
10 supply a value or supply **System.Type.Missing** to use the default
11 value.

12
13 **System.Type.InvokeMember** can be used to set a field to a
14 particular value by specifying
15 **System.Reflection.BindingFlags.SetField**. For example, to set a
16 public instance field named F on class C, where F is a string, the value
17 is set using the following statement:

```
18 typeof(C).InvokeMember("F", BindingFlags.SetField, null, C,  
19 new Object{ "strings new value"}, null, null, null);
```

20
21
22 A string array F can be initialized as follows:

```
23  
24 typeof(C).InvokeMember("F", BindingFlags.SetField, null, C,  
25 new Object{new String[]{"a","z","c","d"}, null, null,  
26 null});
```

1
2
3
4
5
6
7
8
9
10
11
12
13

Use **System.Type.InvokeMember** to set the value of an element in an array by specifying the index of the value and the new value for the element as follows:

```
typeof(C).InvokeMember("F", BindingFlags.SetField, null, C, new Object{1, "b"}, null, null, null);
```

The preceding statement changes "z" in array F to "b".

Exceptions

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .
System.ArgumentException	<i>args</i> has more than one dimension.
	-or-
	<i>invokeAttr</i> is not a valid System.Reflection.BindingFlags value.
	-or-
	The member to be invoked is a constructor and System.Reflection.BindingFlags.CreateInstance is not specified in <i>invokeAttr</i> .
	-or-
	The member to be invoked is a method that is not a type initializer or instance constructor, and System.Reflection.BindingFlags.InvokeMethod is not specified in <i>invokeAttr</i> .
-or-	
The member to be accessed is a field, and neither System.Reflection.BindingFlags.GetField nor System.Reflection.BindingFlags.SetField is specified in <i>invokeAttr</i> .	
-or-	
The member to be accessed is a property, and neither System.Reflection.BindingFlags.GetProperty nor System.Reflection.BindingFlags.SetProperty is specified in <i>invokeAttr</i> .	

-or-

invokeAttr contains

System.Reflection.BindingFlags.CreateInstance
and at least one of

System.Reflection.BindingFlags.InvokeMethod,
System.Reflection.BindingFlags.GetField,
System.Reflection.BindingFlags.SetField,
System.Reflection.BindingFlags.GetProperty, or
System.Reflection.BindingFlags.SetProperty.

-or-

invokeAttr contains both

System.Reflection.BindingFlags.GetField and
System.Reflection.BindingFlags.SetField.

-or-

invokeAttr contains both

System.Reflection.BindingFlags.GetProperty
and
System.Reflection.BindingFlags.SetProperty.

-or-

invokeAttr contains

System.Reflection.BindingFlags.InvokeMethod
and at least one of

System.Reflection.BindingFlags.SetField or
System.Reflection.BindingFlags.SetProperty.

-or-

invokeAttr contains

System.Reflection.BindingFlags.SetField and
args has more than one element.

-or-

namedParameters.Length > *args.Length*.

-or-

At least one element in *namedParameters* is **null**.

-or-

At least one element in *args* is not assignment-
compatible with the corresponding parameter in

	<i>namedParameters</i> .
System.MissingFieldException	A field or property matching the specified criteria was not found.
System.MissingMethodException	A method matching the specified criteria cannot be found.
System.MethodAccessException	The requested member is non-public and the caller does not have the required permission.
System.Reflection.TargetException	The member matching the specified criteria cannot be invoked on <i>target</i> .
System.Reflection.TargetInvocationException	The member matching the specified criteria threw an exception.
System.Reflection.AmbiguousMatchException	More than one member matches the specified criteria.

1

2

Example

3

4

5

6

7

8

9

The following example demonstrates the use of **System.Type.InvokeMember** to construct a **System.String**, obtain its **System.String.Length** property, invoke **System.String.Insert** on it, and then set its value using the **System.String.Empty** field.

[C#]

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

```
using System;
using System.Reflection;

class InvokeMemberExample
{
    static void Main(string[] args)
    {
        // Create the parameter arrays that will
        // be passed to InvokeMember.
        char[] cAry =
            new char[] {'A', ' ', 's', 't', 'r', 'i', 'n', 'g'};
        object[] oAry = new object[] {cAry, 0, cAry.Length};

        Type t = typeof(string);

        // Invoke the constructor of a string.
        string str =
            (string)t.InvokeMember(null, BindingFlags.Instance
            |
            BindingFlags.Public | BindingFlags.CreateInstance,
            null,
            null, oAry, null, null, null);
        Console.WriteLine("The string is \"{0}\".", str);

        // Access a property of the string.
        int i =
```

```

1         (int) t.InvokeMember("Length",
2 BindingFlags.Instance |
3         BindingFlags.Public | BindingFlags.GetProperty,
4 null,
5         str, null, null, null, null);
6         Console.WriteLine("The length of the string is {0}.",
7 i);
8
9         // Invoke a method on the string.
10        string newStr = "new ";
11        object[] oAry2 = new Object[] {2, newStr};
12        str = (string) t.InvokeMember("Insert",
13 BindingFlags.Instance |
14        BindingFlags.Public | BindingFlags.InvokeMethod,
15 null, str,
16        oAry2, null, null, null);
17        Console.WriteLine("The modified string is \"{0}\".",
18 str);
19
20        // Access a field of the string.
21        str = (string) t.InvokeMember("Empty",
22 BindingFlags.Static |
23        BindingFlags.Public | BindingFlags.GetField, null,
24 str,
25        null);
26        Console.WriteLine("The empty string is \"{0}\".",
27 str);
28
29    }
30 }

```

```

31 The output is
32
33 The string is "A string".
34
35
36 The length of the string is 8.
37
38
39 The modified string is "A new string"
40
41

```

1 The empty string is "".

2

3 **Permissions**

4

5

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

6

7

8

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

3 **Type.InvokeMember(System.String,**
4 **System.Reflection.BindingFlags,**
5 **System.Reflection.Binder, System.Object,**
6 **System.Object[],**
7 **System.Globalization.CultureInfo) Method**

```
8 [ILASM]  
9 .method public hidebysig instance object  
10 InvokeMember(string name, valuetype  
11 System.Reflection.BindingFlags invokeAttr, class  
12 System.Reflection.Binder binder, object target, class  
13 System.Object[] args, class  
14 System.Globalization.CultureInfo culture)  
  
15 [C#]  
16 public object InvokeMember(string name, BindingFlags  
17 invokeAttr, Binder binder, object target, object[] args,  
18 CultureInfo culture)
```

19 Summary

20 Invokes the specified member, using the specified binding constraints
21 and matching the specified argument list and culture.

22 Parameters

23
24

Parameter	Description
<i>name</i>	A System.String containing the name of the constructor or method to invoke, or property or field to access. If the type represented by the current instance has a default member, specify System.String.Empty to invoke that member. [Note: For more information on default members, see System.Reflection.DefaultMemberAttribute .]
<i>invokeAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, System.Reflection.BindingFlags.Public System.Reflection.BindingFlags.Instance is used by default.
<i>binder</i>	A System.Reflection.Binder object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specifv null to use the

	System.Type.DefaultBinder.
<i>target</i>	A System.Object on which to invoke the member that matches the other specified criteria. If the matching member is static , this parameter is ignored.
<i>args</i>	An array of objects containing the arguments to pass to the member to be invoked. The elements of this array are of the same number and in the same order by assignment-compatible type as specified by the contract of the member to be bound. Specify an empty array or null for a member that has no parameters.
<i>culture</i>	The only defined value for this parameter is null .

1

2 **Return Value**

3

4 A **System.Object** containing the return value of the invoked member.
 5 If the invoked member does not have a return value, returns a
 6 **System.Object** containing **System.Void**.

7 **Description**

8 This version of **System.Type.InvokeMember** is equivalent to
 9 **System.Type.InvokeMember**(*name*, *invokeAttr*, *binder*, *target*,
 10 *args*, **null**, *culture*, **null**).

11 **Exceptions**

12

13

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .
System.ArgumentException	<p><i>args</i> has more than one dimension.</p> <p>-or-</p> <p><i>invokeAttr</i> is not a valid System.Reflection.BindingFlags value.</p> <p>-or-</p> <p>The member to be invoked is a constructor and System.Reflection.BindingFlags.CreateInstance not specified in <i>invokeAttr</i>.</p> <p>-or-</p> <p>The member to be invoked is a method that is not a type initializer or instance constructor. and</p>

System.Reflection.BindingFlags.InvokeMethod not specified in *invokeAttr*.

-or-

The member to be accessed is a field, and neither **System.Reflection.BindingFlags.GetField** nor **System.Reflection.BindingFlags.SetField** is specified in *invokeAttr*.

-or-

The member to be accessed is a property, and neither **System.Reflection.BindingFlags.GetProperty** nor **System.Reflection.BindingFlags.SetProperty** is specified in *invokeAttr*.

-or-

invokeAttr contains **System.Reflection.BindingFlags.CreateInstance** and at least one of **System.Reflection.BindingFlags.InvokeMethod**, **System.Reflection.BindingFlags.GetField**, **System.Reflection.BindingFlags.SetField**, **System.Reflection.BindingFlags.GetProperty**, or **System.Reflection.BindingFlags.SetProperty**.

-or-

invokeAttr contains both **System.Reflection.BindingFlags.GetField** and **System.Reflection.BindingFlags.SetField**.

-or-

invokeAttr contains both **System.Reflection.BindingFlags.GetProperty** and **System.Reflection.BindingFlags.SetProperty**.

-or-

invokeAttr contains **System.Reflection.BindingFlags.InvokeMethod** and at least one of **System.Reflection.BindingFlags.SetField** or **System.Reflection.BindingFlags.SetProperty**.

-or-

invokeAttr contains **System.Reflection.BindingFlags.SetField** and at least one of **System.Reflection.BindingFlags.InvokeMethod** or **System.Reflection.BindingFlags.SetProperty**.

	has more than one element.
System.MissingFieldException	A field or property matching the specified criteria was not found.
System.MissingMethodException	A method matching the specified criteria was not found.
System.MethodAccessException	The requested member is non-public and the caller does not have the required permission.
System.Reflection.TargetException	The member matching the specified criteria cannot be invoked on <i>target</i> .
System.Reflection.TargetInvocationException	The member matching the specified criteria threw an exception.
System.Reflection.AmbiguousMatchException	More than one member matches the specified criteria.

1

2 **Example**

3

4 For an example that demonstrates **System.Type.InvokeMember**,
5 see **System.Type.InvokeMember(System.String,**
6 **System.Reflection.BindingFlags, System.Reflection.Binder,**
7 **System.Object, System.Object[],**
8 **System.Reflection.ParameterModifier[],**
9 **System.Globalization.CultureInfo, System.String[]).**

10 **Permissions**

11

12

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

13

14

15

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

3 **Type.InvokeMember(System.String,**
4 **System.Reflection.BindingFlags,**
5 **System.Reflection.Binder, System.Object,**
6 **System.Object[]) Method**

```
7 [ILASM]  
8 .method public hidebysig instance object  
9 InvokeMember(string name, valuetype  
10 System.Reflection.BindingFlags invokeAttr, class  
11 System.Reflection.Binder binder, object target, class  
12 System.Object[] args)  
  
13 [C#]  
14 public object InvokeMember(string name, BindingFlags  
15 invokeAttr, Binder binder, object target, object[] args)
```

16 **Summary**

17 Invokes the specified member, using the specified binding constraints
18 and matching the specified argument list.

19 **Parameters**

20
21

Parameter	Description
<i>name</i>	A System.String containing the name of the constructor or method to invoke, or property or field to access. If the type represented by the current instance has a default member, specify System.String.Empty to invoke that member. [Note: For more information on default members, see System.Reflection.DefaultMemberAttribute .]
<i>invokeAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. If zero is specified, System.Reflection.BindingFlags.Public System.Reflection.BindingFlags.Instance is used by default.
<i>binder</i>	A System.Reflection.Binder object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specify null to use the System.Type.DefaultBinder .
<i>target</i>	A System.Object on which to invoke the member that matches the other specified criteria. If the matching member is static , this parameter is ignored.

<i>args</i>	An array of objects containing the arguments to pass to the member to be invoked. The elements of this array are of the same number and in the same order by assignment-compatible type as specified by the contract of the member to be bound. Specify an empty array or null for a member that has no parameters.
-------------	--

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21

Return Value

A **System.Object** containing the return value of the invoked member. If the invoked member does not have a return value, returns a **System.Object** containing **System.Void**.

Description

This version of **System.Type.InvokeMember** is equivalent to **System.Type.InvokeMember(name, invokeAttr, binder, target, args, null, null, null)**.

[Note: For a demonstration of the use of **System.Type.InvokeMember**, see the example for **System.Type.InvokeMember(System.String, System.Reflection.BindingFlags, System.Reflection.Binder, System.Object, System.Object[], System.Reflection.ParameterModifier[], System.Globalization.CultureInfo, System.String[]).**]

Exceptions

Exception	Condition
System.ArgumentNullException	<i>name</i> is null .
System.ArgumentException	<i>args</i> has more than one dimension. -or- <i>invokeAttr</i> is not a valid System.Reflection.BindingFlags value. -or- The member to be invoked is a constructor and System.Reflection.BindingFlags.CreateInstance is not specified in <i>invokeAttr</i> . -or- The member to be invoked is a method that is not a type initializer or instance constructor. and

System.Reflection.BindingFlags.InvokeMethod is not specified in *invokeAttr*.

-or-

The member to be accessed is a field, and neither **System.Reflection.BindingFlags.GetField** nor **System.Reflection.BindingFlags.SetField** is specified in *invokeAttr*.

-or-

The member to be accessed is a property, and neither **System.Reflection.BindingFlags.GetProperty** nor **System.Reflection.BindingFlags.SetProperty** is specified in *invokeAttr*.

-or-

invokeAttr contains

System.Reflection.BindingFlags.CreateInstance and at least one of **System.Reflection.BindingFlags.InvokeMethod**, **System.Reflection.BindingFlags.GetField**, **System.Reflection.BindingFlags.SetField**, **System.Reflection.BindingFlags.GetProperty**, or **System.Reflection.BindingFlags.SetProperty**.

-or-

invokeAttr contains both

System.Reflection.BindingFlags.GetField and **System.Reflection.BindingFlags.SetField**.

-or-

invokeAttr contains both

System.Reflection.BindingFlags.GetProperty and **System.Reflection.BindingFlags.SetProperty**.

-or-

invokeAttr contains

System.Reflection.BindingFlags.InvokeMethod and at least one of **System.Reflection.BindingFlags.SetField** or **System.Reflection.BindingFlags.SetProperty**.

-or-

	<i>invokeAttr</i> contains System.Reflection.BindingFlags.SetField and <i>args</i> has more than one element.
System.MissingFieldException	A field or property matching the specified criteria was not found.
System.MissingMethodException	A method matching the specified criteria cannot be found.
System.MethodAccessException	The requested member is non-public and the caller does not have the required permission.
System.Reflection.TargetException	The member matching the specified criteria cannot be invoked on <i>target</i> .
System.Reflection.TargetInvocationException	The member matching the specified criteria threw an exception.
System.Reflection.AmbiguousMatchException	More than one member matches the specified criteria.

1
2
3
4

Permissions

Permission	Description
System.Security.Permissions.ReflectionPermission	Requires permission to retrieve information on non-public members of types in loaded assemblies. See System.Security.Permissions.ReflectionPermissionFlag.TypeInformation .

5
6
7

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

3 **Type.IsArrayImpl()** Method

```
4 [ILASM]  
5 .method family hidebysig virtual abstract bool  
6 IsArrayImpl()  
  
7 [C#]  
8 protected abstract bool IsArrayImpl()
```

9 **Summary**

10 When overridden in a derived class implements the
11 **System.Type.IsArray** property returning a **System.Boolean** value
12 that indicates whether the type represented by the current instance is
13 an array.

14 **Return Value**

15

16 **true** if the **System.Type** is an array; otherwise, **false**.

17 **Description**

18 An instance of the **System.Array** class is required to return **false**
19 because it is an object, not an array.

20 **Behaviors**

21 As described above.

22

1 Type.IsAssignableFrom(System.Type) 2 Method

```
3 [ILASM]  
4 .method public hidebysig virtual bool  
5 IsAssignableFrom(class System.Type c)  
  
6 [C#]  
7 public virtual bool IsAssignableFrom(Type c)
```

8 Summary

9 Determines whether an instance of the current **System.Type** can be
10 assigned from an instance of the specified **System.Type**.

11 Parameters

12
13

Parameter	Description
c	The System.Type to compare with the current System.Type .

14
15
16

Return Value

17 **false** if *c* is a null reference.

18
19
20

true if one or more of the following statements are true; otherwise
false.

- 21 • If *c* and the current **System.Type** represent the same type.
- 22 • If the current **System.Type** is in the inheritance hierarchy of *c*.
- 23 • If the current **System.Type** is an interface and *c* supports that
24 interface.

25 Example

26

27 The following example demonstrates the
28 **System.Type.IsAssignableFrom** method using arrays.

29
30

```
[C#]  
  
31 using System;  
32 class ArrayTypeTest {  
33     public static void Main() {  
34         int i = 1;  
35         int [] array10 = new int [10];
```

```

1      int [] array2 = new int[2];
2      int [,]array22 = new int[2,2];
3      int [,]array24 = new int[2,4];
4      int [,,]array333 = new int[3,3,3];
5      Type array10Type = array10.GetType();
6      Type array2Type = array2.GetType();
7      Type array22Type = array22.GetType();
8      Type array24Type = array24.GetType();
9      Type array333Type = array333.GetType();
10
11     // If X and Y are not both arrays, then false
12     Console.WriteLine("int[2] is assignable from int? {0} ",
13 array2Type.IsAssignableFrom(i.GetType()));
14     // If X and Y have same type and rank, then true.
15     Console.WriteLine("int[2] is assignable from int[10]? {0}
16 ", array2Type.IsAssignableFrom(array10Type));
17     Console.WriteLine("int[2,2] is assignable from int[2,4]?
18 {0}", array22Type.IsAssignableFrom(array24Type));
19     Console.WriteLine("int[2,4] is assignable from int[2,2]?
20 {0}", array24Type.IsAssignableFrom(array22Type));
21     Console.WriteLine("");
22     // If X and Y do not have the same rank, then false.
23     Console.WriteLine("int[2,2] is assignable from int[10]?
24 {0}", array22Type.IsAssignableFrom(array10Type));
25     Console.WriteLine("int[2,2] is assignable from int[3,3,3]?
26 {0}", array22Type.IsAssignableFrom(array333Type));
27     Console.WriteLine("int[3,3,3] is assignable from int[2,2]?
28 {0}", array333Type.IsAssignableFrom(array22Type));
29     }
30 }

```

31 The output is

```

32
33 int[2] is assignable from int? False
34
35
36 int[2] is assignable from int[10]? True
37
38
39 int[2,2] is assignable from int[2,4]? True
40
41
42 int[2,4] is assignable from int[2,2]? True

```

```
1
2
3   int[2,2] is assignable from int[10]? False
4
5
6   int[2,2] is assignable from int[3,3,3]? False
7
8
9   int[3,3,3] is assignable from int[2,2]? False
10
11
```

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

3 **Type.IsByRefImpl()** Method

```
4 [ILASM]  
5 .method family hidebysig virtual abstract bool  
6 IsByRefImpl()  
  
7 [C#]  
8 protected abstract bool IsByRefImpl()
```

9 **Summary**

10 When overridden in a derived class, implements the
11 **System.Type.IsByRef** property and determines whether the
12 **System.Type** is passed by reference.

13 **Return Value**

14

15 **true** if the **System.Type** is passed by reference; otherwise, **false**.

16 **Behaviors**

17 As described above.

18

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

3 **Type.IsCOMObjectImpl()** Method

```
4 [ILASM]  
5 .method family hidebysig virtual abstract bool  
6 IsCOMObjectImpl()  
  
7 [C#]  
8 protected abstract bool IsCOMObjectImpl()
```

9 **Summary**

10 Reserved.

11 **Return Value**

12

13 **false**

14 **Description**

15 This abstract method is required to be present for legacy
16 implementations. Conforming implementations are permitted to throw
17 the **System.NotSupportedException** as their implementation.

18

1 Type.IsInstanceOfType(System.Object)

2 Method

```
3 [ILASM]
4 .method public hidebysig virtual bool
5 IsInstanceOfType(object o)
6
7 [C#]
8 public virtual bool IsInstanceOfType(object o)
```

8 Summary

9 Determines whether the specified object is an instance of the current
10 **System.Type**.

11 Parameters

12
13

Parameter	Description
<i>o</i>	The object to compare with the current System.Type .

14

15 Return Value

16

17 **true** if either of the following statements is true; otherwise **false**.

- 18 • If the current **System.Type** is in the inheritance hierarchy of *o*.
- 19 • If the current **System.Type** is an interface and *o* supports that
20 interface.

21 If *o* is a null reference, returns **false**.

22 Behaviors

23 As described above.

24 Example

25

26 The following example demonstrates the
27 **System.Type.IsInstanceOfType** method.

28

29

```
[C#]
```

```
30 using System;
31 public interface IFoo { }
32 public class MyClass: IFoo {}
33 public class MyDerivedClass: MyClass {}
```

```

1      class IsInstanceTest {
2          public static void Main() {
3              Type ifooType=typeof(IFoo);
4              MyClass mc = new MyClass();
5              Type mcType = mc.GetType();
6              MyClass mdc = new MyDerivedClass();
7              Type mdcType = mdc.GetType();
8              int [] array = new int [10];
9              Type arrayType = typeof(Array);
10             Console.WriteLine("int[] is instance of Array? {0}",
11 arrayType.IsInstanceOfType(array));
12             Console.WriteLine("myclass instance is instance of
13 MyClass? {0}", mcType.IsInstanceOfType(mc));
14             Console.WriteLine("myderivedclass instance is instance of
15 MyClass? {0}", mcType.IsInstanceOfType(mdc));
16             Console.WriteLine("myclass instance is instance of IFoo?
17 {0}", ifooType.IsInstanceOfType(mc));
18             Console.WriteLine("myderivedclass instance is instance of
19 IFoo? {0}", ifooType.IsInstanceOfType(mdc));
20         }
21     }

```

22 The output is

```

23
24 int[] is instance of Array? True
25
26
27 myclass instance is instance of MyClass? True
28
29
30 myderivedclass instance is instance of MyClass? True
31
32
33 myclass instance is instance of IFoo? True
34
35
36 myderivedclass instance is instance of IFoo? True
37

```


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

3 **Type.IsPointerImpl()** Method

```
4 [ILASM]  
5 .method family hidebysig virtual abstract bool  
6 IsPointerImpl()  
  
7 [C#]  
8 protected abstract bool IsPointerImpl()
```

9 **Summary**

10 When overridden in a derived class, implements the
11 **System.Type.IsPointer** property and determines whether the
12 **System.Type** is a pointer.

13 **Return Value**

14

15 **true** if the **System.Type** is a pointer; otherwise, **false**.

16 **Behaviors**

17 As described above.

18

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

3 **Type.IsPrimitiveImpl()** Method

```
4 [ILASM]  
5 .method family hidebysig virtual abstract bool  
6 IsPrimitiveImpl()  
  
7 [C#]  
8 protected abstract bool IsPrimitiveImpl()
```

9 **Summary**

10 When overridden in a derived class, implements the
11 **System.Type.IsPrimitive** property and determines whether the
12 **System.Type** is one of the primitive types.

13 **Return Value**

15 **true** if the **System.Type** is one of the primitive types; otherwise,
16 **false**.

17 **Behaviors**

18 This method returns **true** if the underlying type of the current instance
19 is one of the following: **System.Boolean**, **System.Byte**,
20 **System.SByte**, **System.Int16**, **System.UInt16**, **System.Int32**,
21 **System.UInt32**, **System.Int64**, **System.UInt64**, **System.Char**,
22 **System.Double**, and **System.Single**.

23

1 Type.IsSubclassOf(System.Type) Method

```
2 [ILASM]
3 .method public hidebysig virtual bool IsSubclassOf(class
4 System.Type c)
5
6 [C#]
7 public virtual bool IsSubclassOf(Type c)
```

7 Summary

8 Determines whether the current **System.Type** derives from the
9 specified **System.Type**.

10 Parameters

11
12

Parameter	Description
c	The System.Type to compare with the current System.Type .

13
14
15

14 Return Value

16 **true** if *c* and the current **System.Type** represent classes, and the
17 class represented by the current **System.Type** derives from the class
18 represented by *c*; otherwise **false**. Returns **false** if *c* and the current
19 **System.Type** represent the same class.

20 Example

21

22 The following example demonstrates the **System.Type.IsSubclassOf**
23 method.

24
25

```
26 [C#]
27 using System;
28 public interface IFoo { }
29 public interface IBar:IFoo{}
30 public class MyClass: IFoo {}
31 public class MyDerivedClass: MyClass {}
32 class IsSubclassTest {
33     public static void Main() {
34         Type ifooType = typeof(IFoo);
35         Type ibarType = typeof(IBar);
36         MyClass mc = new MyClass();
37         Type mcType = mc.GetType();
38         MyDerivedClass mdc = new MyDerivedClass();
39         Type mdcType = mdc.GetType();
40         int [] array = new int [10];
41         Type arrayOfIntsType = array.GetType();
```

```
1     Type arrayType = typeof(Array);
2
3     Console.WriteLine("Array is subclass of int[]? {0}",
4 arrayType.IsSubclassOf(arrayOfIntsType));
5     Console.WriteLine("int [] is subclass of Array? {0}",
6 arrayOfIntsType.IsSubclassOf(arrayType));
7     Console.WriteLine("IFoo is subclass of IBar? {0}",
8 ifooType.IsSubclassOf(ibarType));
9     Console.WriteLine("myclass is subclass of MyClass? {0}",
10 mcType.IsSubclassOf(mcType));
11     Console.WriteLine("myderivedclass is subclass of MyClass?
12 {0}", mdcType.IsSubclassOf(mcType));
13     }
14 }
```

15 The output is

```
16
17     Array is subclass of int[]? False
18
19
20     int [] is subclass of Array? True
21
22
23     IFoo is subclass of IBar? False
24
25
26     myclass is subclass of MyClass? False
27
28
29     myderivedclass is subclass of MyClass? True
30
```

31

1 Type.ToString() Method

```
2 [ILASM]  
3 .method public hidebysig virtual string ToString()  
4 [C#]  
5 public override string ToString()
```

6 Summary

7 Returns a **System.String** representation of the current **System.Type**.

8 Return Value

9

10 Returns **System.Type.FullName**.

11 Description

12 [*Note:* This method overrides **System.Object.ToString**.]

13

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

3 Type.Assembly Property

```
4 [ILASM]  
5 .property class System.Reflection.Assembly Assembly {  
6 public hidebysig virtual abstract specialname class  
7 System.Reflection.Assembly get_Assembly() }
```

```
8 [C#]  
9 public abstract Assembly Assembly { get; }
```

10 Summary

11 Gets the **System.Reflection.Assembly** that the type is declared in.

12 Property Value

13

14 A **System.Reflection.Assembly** instance that describes assembly
15 containing the current type.

16 Behaviors

17 This property is read-only.

18

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

3 Type.AssemblyQualifiedName Property

```
4 [ILASM]  
5 .property string AssemblyQualifiedName { public hidebyseq  
6 virtual abstract specialname string  
7 get_AssemblyQualifiedName() }
```

```
8 [C#]  
9 public abstract string AssemblyQualifiedName { get; }
```

10 Summary

11 Gets the fully qualified name of the type represented by the current
12 instance including the name of the assembly from which the
13 **System.Type** was loaded.

14 Property Value

15
16 A **System.String** containing the fully qualified name of the type
17 represented by the current instance, including the name of the
18 assembly from which the **System.Type** was loaded.

19 Behaviors

20 This property is read-only.

21
22 Compilers emit the simple name of a nested class, and reflection
23 constructs a mangled name when queried, in accordance with the
24 following conventions.

Delimiter	Meaning
Backslash (\)	Escape character.
Comma (,)	Precedes the Assembly name.
Plus sign (+)	Precedes a nested class.
Period (.)	Denotes namespace identifiers.

25
26 *[Note: For example, the fully qualified name for a class might look like*
27 *this:*
28

1 TopNamespace.SubNameSpace.ContainingClass+NestedClass,MyAsse
2 mbly

3
4 If the namespace were TopNamespace.Sub+Namespace, then the
5 string would have to precede the plus sign (+) with an escape
6 character (\) to prevent it from being interpreted as a nesting
7 separator. Reflection emits this string as follows:

8
9 TopNamespace.Sub\+Namespace.ContainingClass+NestedClass,MyAss
10 embly

11
12 A "++" becomes "\\+\\+", and a "\" becomes "\\".] Type names are
13 permitted to include trailing characters that denote additional
14 information about the type, such as whether the type is a reference
15 type, a pointer type or an array type. To retrieve the type name
16 without these trailing characters, use
17 `t.GetElementType().ToString()`, where *t* is the type.

18
19 Spaces are significant in all type name components except the
20 assembly name. In the assembly name, spaces before the ',' separator
21 are significant, but spaces after the ',' separator are ignored.

22 Usage

23 The name returned by this method can be persisted and later used to
24 load the **System.Type**. To search for and load a **System.Type**, use
25 **System.Type.GetType** either with the type name only or with the
26 assembly qualified type name. **System.Type.GetType** with the type
27 name only will look for the **System.Type** in the caller's assembly and
28 then in the System assembly. **System.Type.GetType** with the
29 assembly qualified type name will look for the **System.Type** in any
30 assembly.

31

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

3 Type.Attributes Property

```
4 [ILASM]  
5 .property valuetype System.Reflection.TypeAttributes  
6 Attributes { public hidebysig specialname instance  
7 valuetype System.Reflection.TypeAttributes get_Attributes()  
8 }  
9 [C#]  
10 public TypeAttributes Attributes { get; }
```

11 Summary

12 Gets the attributes associated with the type represented by the current
13 instance.

14 Property Value

15

16 A **System.Reflection.TypeAttributes** object representing the
17 attribute set of the **System.Type**.

18 Description

19 This property is read-only.

20

1 Type.BaseType Property

```
2 [ILASM]
3 .property class System.Type BaseType { public hidebysig
4 virtual abstract specialname class System.Type
5 get_BaseType() }
6
7 [C#]
8 public abstract Type BaseType { get; }
```

8 Summary

9 Gets the base **System.Type** of the current **System.Type**.

10 Property Value

11

12 A **System.Type** object representing the type from which the current
13 **System.Type** directly inherits, or **null** if the current **System.Type**
14 represents the **System.Object** class.

15 Behaviors

16 This property is read-only.

17 Example

18

19 The following example demonstrates using the
20 **System.Type.BaseType** property.

21
22

```
23 [C#]
24 using System;
25 class TestType {
26     public static void Main() {
27         Type t = typeof(int);
28         Console.WriteLine("{0} inherits from {1}", t,t.BaseType);
29     }
30 }
```

30 The output is

31

32 System.Int32 inherits from System.ValueType

33

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

3 Type.DeclaringType Property

```
4 [ILASM]  
5 .property class System.Type DeclaringType { public  
6 hidebysig virtual specialname class System.Type  
7 get_DeclaringType() }
```

```
8 [C#]  
9 public override Type DeclaringType { get; }
```

10 Summary

11 Gets the type that declares the type represented by the current
12 instance.

13 Property Value

14
15 The **System.Type** object for the class that declares the type
16 represented by the current instance. If the type is a nested type, this
17 property returns the enclosing type; otherwise, returns the current
18 instance.

19 Description

20 [*Note:* This property overrides
21 **System.Reflection.MemberInfo.DeclaringType.**]

22 Example

23
24 The following example demonstrates the
25 **System.Type.DeclaringType** property.

```
26 [C#]  
27  
28 using System;  
29 using System.Reflection;  
30  
31 public abstract class DeclaringTypeTest {  
32     public abstract class MyClassA {  
33         public abstract int m();  
34     }  
35     public abstract class MyClassB: MyClassA {  
36     }  
37     public static void Main() {  
38         Console.WriteLine("Declaring type of m is {0}",  
39             typeof(MyClassB).GetMethod("m").DeclaringType);  
40     }  
}
```

```
1     }  
2
```

```
3     The output is
```

```
4
```

```
5     Declaring type of m is DeclaringTypeTest+MyClassA
```

```
6
```

```
7
```

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

Type.DefaultBinder Property

```
[ILASM]
.property class System.Reflection.Binder DefaultBinder {
public hidebysig static specialname class
System.Reflection.Binder get_DefaultBinder() }

[C#]
public static Binder DefaultBinder { get; }
```

Summary

Gets the default binder used by the system.

Property Value

The default **System.Reflection.Binder** used by the system.

Description

This property is read-only.

Reflection models the accessibility rules of the common type system. For example, if the caller is in the same assembly, the caller does not need special permissions for internal members. Otherwise, the caller needs **System.Security.Permissions.ReflectionPermission**. This is consistent with lookup of members that are protected, private, and so on.

[Note: The general principle is that **System.Reflection.Binder.ChangeType** typically performs only widening coercions, which never lose data. An example of a widening coercion is coercing a value that is a 32-bit signed integer to a value that is a 64-bit signed integer. This is distinguished from a narrowing coercion, which may lose data. An example of a narrowing coercion is coercing a 64-bit signed integer to a 32-bit signed integer.]

The following table lists the coercions performed by the default binder's implementation of **ChangeType**.

Source Type	Target Type
Any type	Its base type.
Any type	The interface it implements.
Char	Unt16, UInt32, Int32, UInt64, Int64, Single, Double
Byte	Char, Unt16, Int16, UInt32, Int32, UInt64, Int64, Single, Double

SByte	Int16, Int32, Int64, Single, Double
UInt16	UInt32, Int32, UInt64, Int64, Single, Double
Int16	Int32, Int64, Single, Double
UInt32	UInt64, Int64, Single, Double
Int32	Int64, Single, Double
UInt64	Single, Double
Int64	Single, Double
Single	Double
Non-reference	By-reference.

1

2

1 Type.FullName Property

```
2 [ILASM]
3 .property string FullName { public hidebysig virtual
4 abstract specialname string get_FullName() }
5
6 [C#]
7 public abstract string FullName { get; }
```

7 Summary

8 Gets the fully qualified name of the type represented by the current
9 instance.

10 Property Value

11

12 A **System.String** containing the fully qualified name of the
13 **System.Type**.

14 Description

15 [*Note:* For example, the fully qualified name of the C# string type is
16 "System.String".]

17 Behaviors

18 This property is read-only.

19 Example

20

21 The following example demonstrates using the
22 **System.Type.FullName** property.

23
24

```
25 [C#]
26 using System;
27 class TestType {
28     public static void Main() {
29         Type t = typeof(Array);
30         Console.WriteLine("Full name of Array type is
31 {0}", t.FullName);
32     }
33 }
```

33 The output is

34

35 Full name of Array type is System.Array

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

3 Type.HasElementType Property

```
4 [ILASM]  
5 .property bool HasElementType { public hidebysig  
6 specialname instance bool get_HasElementType() }  
7 [C#]  
8 public bool HasElementType { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the type
11 represented by the current instance encompasses or refers to another
12 type; that is, whether the current **System.Type** is an array, a pointer,
13 or is passed by reference.

14 Property Value

16 **true** if the **System.Type** is an array, a pointer, or is passed by
17 reference; otherwise, **false**.

18 Description

19 This property is read-only.

20
21 [Note: For example, **System.Type.GetType("System.Int32**
22 **[]").HasElementType** returns **true**, but
23 **System.Type.GetType("System.Int32 ").HasElementType** returns
24 **false**. **System.Type.HasElementType** also returns **true** for "Int32*" and "Int32&".]

26

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

3 Type.IsAbstract Property

```
4 [ILASM]  
5 .property bool IsAbstract { public hidebysig specialname  
6 instance bool get_IsAbstract() }  
  
7 [C#]  
8 public bool IsAbstract { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the type
11 represented by the current instance is abstract and is required to be
12 overridden.

13 Property Value

14

15 **true** if the **System.Type** is abstract; otherwise, **false**.

16 Description

17 This property is read-only.

18

1 Type.IsArray Property

```
2 [ILASM]
3 .property bool IsArray { public hidebysig specialname
4 instance bool get_IsArray() }
5
6 [C#]
7 public bool IsArray { get; }
```

7 Summary

8 Gets a **System.Boolean** value that indicates whether the current
9 **System.Type** represents an array.

10 Property Value

11

12 **true** if the current **System.Type** represents an array; otherwise
13 **false**.

14 Description

15 This property is read-only.

16

17 This property returns **true** for an array of objects, but not for the
18 **System.Array** type itself, which is a class.

19 Example

20

21 The following example demonstrates using the **System.Type.IsArray**
22 property.

23

24

```
25 using System;
26 class TestType {
27     public static void Main() {
28         int [] array = {1,2,3,4};
29         Type at = typeof(Array);
30         Type t = array.GetType();
31         Console.WriteLine("Type is {0}. IsArray? {1}", at,
32 at.IsArray);
33         Console.WriteLine("Type is {0}. IsArray? {1}", t,
34 t.IsArray);
35     }
36 }
```

37 The output is

38

```
1      Type is System.Array. IsArray? False
2
3
4      Type is System.Int32[]. IsArray? True
5
6
```

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

3 Type.IsAutoLayout Property

```
4 [ILASM]  
5 .property bool IsAutoLayout { public hidebysig specialname  
6 instance bool get_IsAutoLayout() }  
  
7 [C#]  
8 public bool IsAutoLayout { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the type layout
11 attribute **System.Reflection.TypeAttributes.AutoLayout** is
12 specified for the **System.Type**.

13 Property Value

14

15 **true** if the type layout attribute
16 **System.Reflection.TypeAttributes.AutoLayout** is specified for the
17 current **System.Type**; otherwise, **false**.

18 Description

19 This property is read-only.

20

21 [*Note:* The **System.Reflection.TypeAttributes.AutoLayout**
22 attribute specifies that the system selects the layout the objects of the
23 type. Types marked with this attribute indicate that the system will
24 choose the appropriate way to lay out the type; any layout information
25 that may have been specified is ignored.]

26

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

3 Type.IsByRef Property

```
4 [ILASM]  
5 .property bool IsByRef { public hidebysig specialname  
6 instance bool get_IsByRef() }  
7 [C#]  
8 public bool IsByRef { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the **System.Type** is
11 passed by reference.

12 Property Value

13

14 **true** if the **System.Type** is passed by reference; otherwise, **false**.

15 Description

16 This property is read-only.

17

1 Type.IsClass Property

```
2 [ILASM]
3 .property bool IsClass { public hidebysig specialname
4 instance bool get_IsClass() }
5
6 [C#]
7 public bool IsClass { get; }
```

7 Summary

8 Gets a **System.Boolean** value that indicates whether the current
9 **System.Type** represents a class.

10 Property Value

11

12 **true** if the current **System.Type** represents a class; otherwise **false**.

13 Description

14 This property is read-only.

15

16 Note that this property returns **true** for **System.Type** instances
17 representing **System.Enum** and **System.ValueType**.

18

1 Type.IsEnum Property

```
2 [ILASM]
3 .property bool IsEnum { public hidebysig specialname
4 instance bool get_IsEnum() }
5
6 [C#]
7 public bool IsEnum { get; }
```

7 Summary

8 Gets a **System.Boolean** value that indicates whether the current
9 **System.Type** represents an enumeration.

10 Property Value

11

12 **true** if the current **System.Type** represents an enumeration;
13 otherwise **false**.

14 Description

15 This property is read-only.

16

17 This property returns **true** for an enumeration, but not for the
18 **System.Enum** type itself, which is a class.

19 Example

20

21 The following example demonstrates using the **System.Type.IsEnum**
22 property.

23

24

```
25 using System;
26 public enum Color {
27 Red, Blue, Green
28 }
29 class TestType {
30 public static void Main() {
31 Type colorType = typeof(Color);
32 Type enumType = typeof(Enum);
33 Console.WriteLine("Color is enum ? {0}",
34 colorType.IsEnum);
35 Console.WriteLine("Color is valueType? {0}",
36 colorType.IsValueType);
37 Console.WriteLine("Enum is enum Type? {0}",
38 enumType.IsEnum);
39 Console.WriteLine("Enum is value? {0}",
40 enumType.IsValueType);
41 }
42 }
```

```
1      The output is
2
3      Color is enum ? True
4
5
6      Color is valueType? True
7
8
9      Enum is enum Type? False
10
11
12     Enum is value? False
13
```

14

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

3 **Type.IsExplicitLayout Property**

```
4 [ILASM]  
5 .property bool IsExplicitLayout { public hideby sig  
6 specialname instance bool get_IsExplicitLayout() }  
7 [C#]  
8 public bool IsExplicitLayout { get; }
```

9 **Summary**

10 Gets a **System.Boolean** value indicating whether the type layout
11 attribute **System.Reflection.TypeAttributes.ExplicitLayout** is
12 specified for the **System.Type**.

13 **Property Value**

14
15 **true** if the type layout attribute
16 **System.Reflection.TypeAttributes.ExplicitLayout** is specified for
17 the current **System.Type**; otherwise, **false**.

18 **Description**

19 This property is read-only.

20
21 [Note: Types marked with the
22 **System.Reflection.TypeAttributes.ExplicitLayout** attribute cause
23 the system to ignore field sequence and to use the explicit layout rules
24 provided, in the form of field offsets, overall class size and alignment.]

25

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

3 Type.IsImport Property

```
4 [ILASM]  
5 .property bool IsImport { public hidebysig specialname  
6 instance bool get_IsImport() }  
7 [C#]  
8 public bool IsImport { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the **System.Type**
11 was imported from another class.

12 Property Value

13

14 **true** if the **System.Type** was imported from another class; otherwise,
15 **false**.

16 Description

17 This property is read-only.

18

1 Type.IsInterface Property

```
2 [ILASM]
3 .property bool IsInterface { public hidebysig specialname
4 instance bool get_IsInterface() }
5
6 [C#]
7 public bool IsInterface { get; }
```

7 Summary

8 Gets a **System.Boolean** value that indicates whether the current
9 **System.Type** represents an interface.

10 Property Value

11

12 **true** if the current **System.Type** represents an interface; otherwise
13 **false**.

14 Description

15 This property is read-only.

16

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

3 Type.IsLayoutSequential Property

```
4 [ILASM]  
5 .property bool IsLayoutSequential { public hidebysig  
6 specialname instance bool get_IsLayoutSequential() }  
  
7 [C#]  
8 public bool IsLayoutSequential { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the type layout
11 attribute **System.Reflection.TypeAttributes.SequentialLayout** is
12 specified for the **System.Type**.

13 Property Value

14
15 **true** if the type layout attribute
16 **System.Reflection.TypeAttributes.SequentialLayout** is specified
17 for the current **System.Type**; otherwise, **false**.

18 Description

19 This property is read-only.

20
21 [*Note:* The **System.Reflection.TypeAttributes.SequentialLayout**
22 attribute is used to indicate that the system is to preserve field order
23 as emitted, but otherwise the specific offsets are calculated based on
24 the type of the field; these may be shifted by explicit offset, padding,
25 or alignment information.]

26

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

3 Type.IsMarshalByRef Property

```
4 [ILASM]  
5 .property bool IsMarshalByRef { public hidebysig  
6 specialname instance bool get_IsMarshalByRef() }  
7 [C#]  
8 public bool IsMarshalByRef { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the current type is
11 marshaled by reference.

12 Property Value

13

14 **true** if the **System.Type** is marshaled by reference; otherwise, **false**.

15 Description

16 This property is read-only.

17

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

3 **Type.IsNestedAssembly** Property

```
4 [ILASM]  
5 .property bool IsNestedAssembly { public hidebysig  
6 specialname instance bool get_IsNestedAssembly() }  
7 [C#]  
8 public bool IsNestedAssembly { get; }
```

9 **Summary**

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** is nested and visible only within its own assembly.

12 **Property Value**

13

14 **true** if the **System.Type** is nested and visible only within its own
15 assembly; otherwise, **false**.

16 **Description**

17 This property is read-only.

18

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

3 **Type.IsNestedFamANDAssem Property**

```
4 [ILASM]  
5 .property bool IsNestedFamANDAssem { public hideby sig  
6 specialname instance bool get_IsNestedFamANDAssem() }  
7 [C#]  
8 public bool IsNestedFamANDAssem { get; }
```

9 **Summary**

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** is nested and visible only to classes that belong to both
12 its own family and its own assembly.

13 **Property Value**

15 **true** if the **System.Type** is nested and visible only to classes that
16 belong to both its own family and its own assembly; otherwise, **false**.

17 **Description**

18 This property is read-only.

19
20 A **System.Type** object's family is defined as all objects of the exact
21 same **System.Type** and of its subclasses.

22

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

3 **Type.IsNestedFamily Property**

```
4 [ILASM]  
5 .property bool IsNestedFamily { public hidebysig  
6 specialname instance bool get_IsNestedFamily() }  
7 [C#]  
8 public bool IsNestedFamily { get; }
```

9 **Summary**

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** is nested and visible only within its own family.

12 **Property Value**

13

14 **true** if the **System.Type** is nested and visible only within its own
15 family; otherwise, **false**.

16 **Description**

17 This property is read-only.

18

19 A **System.Type** object's family is defined as all objects of the exact
20 same **System.Type** and of its subclasses.

21

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

3 **Type.IsNestedFamORAssem Property**

```
4 [ILASM]  
5 .property bool IsNestedFamORAssem { public hidebysig  
6 specialname instance bool get_IsNestedFamORAssem() }  
7 [C#]  
8 public bool IsNestedFamORAssem { get; }
```

9 **Summary**

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** is nested and visible only to classes that belong to
12 either its own family or to its own assembly.

13 **Property Value**

15 **true** if the **System.Type** is nested and visible only to classes that
16 belong to its own family or to its own assembly; otherwise, **false**.

17 **Description**

18 This property is read-only.

19
20 A **System.Type** object's family is defined as all objects of the exact
21 same **System.Type** and of its subclasses.

22

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

3 **Type.IsNestedPrivate** Property

```
4 [ILASM]  
5 .property bool IsNestedPrivate { public hideby sig  
6 specialname instance bool get_IsNestedPrivate() }  
7 [C#]  
8 public bool IsNestedPrivate { get; }
```

9 **Summary**

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** is nested and declared private.

12 **Property Value**

13

14 **true** if the **System.Type** is nested and declared private; otherwise,
15 **false**.

16 **Description**

17 This property is read-only.

18

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

3 Type.IsNestedPublic Property

```
4 [ILASM]  
5 .property bool IsNestedPublic { public hidebysig  
6 specialname instance bool get_IsNestedPublic() }  
7 [C#]  
8 public bool IsNestedPublic { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** is a public nested class.

12 Property Value

13

14 **true** if the class is nested and declared public; otherwise, **false**.

15 Description

16 This property is read-only.

17

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

3 Type.IsNotPublic Property

```
4 [ILASM]  
5 .property bool IsNotPublic { public hidebysig specialname  
6 instance bool get_IsNotPublic() }  
7 [C#]  
8 public bool IsNotPublic { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the top-level
11 **System.Type** is not declared public.

12 Property Value

13

14 **true** if the top-level **System.Type** is not declared public; otherwise,
15 **false**.

16 Description

17 This property is read-only.

18

1 Type.IsPointer Property

```
2 [ILASM]  
3 .property bool IsPointer { public hidebysig specialname  
4 instance bool get_IsPointer() }  
5 [C#]  
6 public bool IsPointer { get; }
```

7 Summary

8 Gets a **System.Boolean** value that indicates whether the current
9 **System.Type** represents a pointer.

10 Property Value

11

12 This property is read-only.

13

14 **true** if the current **System.Type** represents a pointer; otherwise
15 **false**.

16 Description

17 This property is read-only.

18

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

3 Type.IsPrimitive Property

```
4 [ILASM]  
5 .property bool IsPrimitive { public hidebysig specialname  
6 instance bool get_IsPrimitive() }  
7 [C#]  
8 public bool IsPrimitive { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** is one of the primitive types.

12 Property Value

13

14 **true** if the **System.Type** is one of the primitive types; otherwise,
15 **false**.

16 Description

17 This property is read-only.

18

19 The primitive types are **System.Boolean**, **System.Byte**,
20 **System.SByte**, **System.Int16**, **System.UInt16**, **System.Int32**,
21 **System.UInt32**, **System.Int64**, **System.UInt64**, **System.Char**,
22 **System.Double**, and **System.Single**.

23

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

3 Type.IsPublic Property

```
4 [ILASM]  
5 .property bool IsPublic { public hidebysig specialname  
6 instance bool get_IsPublic() }  
7 [C#]  
8 public bool IsPublic { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the top-level
11 **System.Type** is declared public.

12 Property Value

13

14 **true** if the top-level **System.Type** is declared public; otherwise,
15 **false**.

16 Description

17 This property is read-only.

18

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

3 Type.IsSealed Property

```
4 [ILASM]  
5 .property bool IsSealed { public hidebysig specialname  
6 instance bool get_IsSealed() }  
  
7 [C#]  
8 public bool IsSealed { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** is declared sealed.

12 Property Value

13

14 **true** if the **System.Type** is declared sealed; otherwise, **false**.

15

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

3 Type.IsSpecialName Property

```
4 [ILASM]  
5 .property bool IsSpecialName { public hidebysig specialname  
6 instance bool get_IsSpecialName() }  
7 [C#]  
8 public bool IsSpecialName { get; }
```

9 Summary

10 Gets a **System.Boolean** value indicating whether the current
11 **System.Type** has a name that requires special handling.

12 Property Value

13

14 **true** if the **System.Type** has a name that requires special handling;
15 otherwise, **false**.

16 Description

17 This property is read-only.

18

19 [*Note:* Names that begin with or contain an underscore character (`_`)
20 are examples of type names that might require special treatment by
21 some tools.]

22

1 Type.IsValueType Property

```
2 [ILASM]  
3 .property bool IsValueType { public hidebysig specialname  
4 instance bool get_IsValueType() }  
5 [C#]  
6 public bool IsValueType { get; }
```

7 Summary

8 Gets a **System.Boolean** value that indicates whether the current
9 **System.Type** represents a value type.

10 Property Value

11

12 **true** if the current **System.Type** represents a value type (structure);
13 otherwise **false**.

14 Description

15 This property is read-only.

16

17 This property returns true for enumerations, but not for the
18 **System.Enum** type itself, which is a class. [*Note:* For an example that
19 demonstrates this behavior, see **System.Type.IsEnum**.]

20

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

3 Type.Module Property

```
4 [ILASM]  
5 .property class System.Reflection.Module Module { public  
6 hidebysig virtual abstract specialname class  
7 System.Reflection.Module get_Module() }  
  
8 [C#]  
9 public abstract Module Module { get; }
```

10 Summary

11 Gets the module in which the current **System.Type** is defined.

12 Property Value

13

14 A **System.Reflection.Module** that reflects the module in which the
15 current **System.Type** is defined.

16 Behaviors

17 This property is read-only.

18

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

3 Type.Namespace Property

```
4 [ILASM]  
5 .property string Namespace { public hidebysig virtual  
6 abstract specialname string get_Namespace() }  
7 [C#]  
8 public abstract string Namespace { get; }
```

9 Summary

10 Gets the namespace of the **System.Type**.

11 Property Value

12

13 A **System.String** containing the namespace of the current
14 **System.Type**.

15 Description

16 [*Note: A namespace is a logical design-time naming convenience, used*
17 *mainly to define scope in an application and organize classes and other*
18 *types in a hierarchical structure. From the viewpoint of the system,*
19 *there are no namespaces.*]

20 Behaviors

21 This property is read-only.

22

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

Type.ReflectedType Property

```
[ILASM]
.property class System.Type ReflectedType { public
hidebysig virtual specialname class System.Type
get_ReflectedType() }

[C#]
public override Type ReflectedType { get; }
```

Summary

Gets the type that was used to obtain the current instance.

Property Value

The **Type** object through which the current instance was obtained.

Description

This property is read-only.

[*Note:* This property overrides **System.Reflection.MemberInfo.ReflectedType**.]

Example

The following example demonstrates the **System.Type.ReflectedType** property. Although the method *m* is declared in **MyClassA**, its reflected type is obtained from **MyClassB**.

```
[C#]
using System;
using System.Reflection;
public abstract class ReflectedTypeTest {
    public abstract class MyClassA {
        public abstract int m();
    }
    public abstract class MyClassB: MyClassA {
    }
    public static void Main(string[] args) {
        Console.WriteLine("Reflected type of m is {0}",
            typeof(MyClassB).GetMethod("m").ReflectedType);
    }
}
```

1
2
3
4

The output is

Reflected type of m is ReflectedTypeTest+MyClassB

5

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

3 Type.TypeHandle Property

```
4 [ILASM]  
5 .property valuetype System.RuntimeTypeHandle TypeHandle {  
6 public hidebysig virtual abstract specialname valuetype  
7 System.RuntimeTypeHandle get_TypeHandle() }
```

```
8 [C#]  
9 public abstract RuntimeTypeHandle TypeHandle { get; }
```

10 Summary

11 Gets the handle for the current **System.Type**.

12 Property Value

13

14 The **System.RuntimeTypeHandle** for the current **System.Type**.

15 Description

16 This property is read-only.

17

18 The **System.RuntimeTypeHandle** encapsulates a pointer to an
19 internal data structure that represents the type. This handle is unique
20 during the process lifetime. The handle is valid only in the application
21 domain in which it was obtained.

22

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

3 Type.TypeInitializer Property

```
4 [ILASM]  
5 .property class System.Reflection.ConstructorInfo  
6 TypeInitializer { public hidebysig specialname instance  
7 class System.Reflection.ConstructorInfo  
8 get_TypeInitializer() }  
9 [C#]  
10 public ConstructorInfo TypeInitializer { get; }
```

11 Summary

12 Gets the initializer for the type represented by the current instance.

13 Property Value

14

15 A **System.Reflection.ConstructorInfo** containing the name of the
16 static constructor for the type represented by the current instance

17 Description

18 This property is read-only.

19

20 [Note: Type initializers are available through
21 **System.Type.GetMember**, **System.Type.GetMembers**,
22 **System.Type.FindMembers**, and **System.Type.GetConstructors**.]

23

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

3 **Type.UnderlyingSystemType** Property

```
4 [ILASM]  
5 .property class System.Type UnderlyingSystemType { public  
6 hidebysig virtual abstract specialname class System.Type  
7 get_UnderlyingSystemType() }
```

```
8 [C#]  
9 public abstract Type UnderlyingSystemType { get; }
```

10 **Summary**

11 Returns the system-supplied type that represents the current type.

12 **Property Value**

13

14 The underlying system type for the **System.Type**.

15 **Description**

16 This property is read-only.

17 **Behaviors**

18 As described above.

19