

# 1 System.Type Class

```
2 [ILAsm]  
3 .class public abstract serializable Type extends System.Object  
4 [C#]  
5 public abstract class Type: Object
```

## 6 Assembly Info:

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

## 12 Summary

13 Provides information about a type.

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

16  
17 **Library:** BCL

18  
19 **Thread Safety:** This type is safe for multithreaded operations.

## 21 Description

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

31  
32  
33  
34 A conforming CLI program which is written to run on only the Kernel profile cannot  
35 subclass System.Type. [*Note:* This only applies to conforming programs not conforming  
36 implementations.]

37  
38  
39  
40 A System.Type object that represents a type is unique; that is, two System.Type object  
41 references refer to the same object if and only if they represent the same type. This  
42 allows for comparison of System.Type objects using reference equality.

1  
2 [Note: An instance of `System.Type` can represent any one of the following types:

- 3 • Classes
- 4 • Value types
- 5 • Arrays
- 6 • Interfaces
- 7 • Pointers
- 8 • Enumerations
- 9 • Constructed generic types and generic type definitions
- 10 • Type arguments and type parameters of constructed generic types, generic type
- 11 definitions, and generic method definitions

12 The following table shows what members of a base class are returned by the methods that  
13 return members of types, such as `System.Type.GetConstructor` and  
14 `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.

15  
16  
17

1 For reflection, properties and events are hide-by-name-and-signature. If a property has  
2 both a get and a set accessor in the base class, but the derived class has only a get  
3 accessor, the derived class property hides the base class property, and the setter on the  
4 base class will not be accessible.

5

6 ]

7

8 The description of `System.Type.IsGenericType` contains definitions for some important  
9 terms.

10

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

### 3 Type.Delimiter Field

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

#### 8 Summary

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

#### 11 Description

12 This field is read-only.

13

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

### 3 **Type.EmptyTypes Field**

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

#### 8 **Summary**

9 Returns an empty array of type `System.Type`.

#### 10 **Description**

11 This field is read-only.

12  
13 The empty `System.Type` array returned by this field is used to specify that lookup  
14 methods in the `System.Type` class, such as `System.Type.GetMethod` and  
15 `System.Type.GetConstructor`, search for members that do not take parameters.  
16 [*Note:* For example, to locate the public instance constructor that takes no parameters,  
17 invoke `System.Type.GetConstructor (System.Reflection.BindingFlags.Public |`  
18 `System.Reflection.BindingFlags.Instance, null, System.Type.EmptyTypes, null).`]

19  
20

21

1 **The following member must be implemented if the Reflection library is present in the**  
2 **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 call will be made  
14 with the default value of a parameter as specified in the metadata. [*Note:* If the  
15 `Missing` field is specified for a parameter value and there is no default value for that  
16 parameter, a `System.ArgumentException` is thrown.]  
17  
18

19

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

## 3 **Type.Equals(System.Type) Method**

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

### 8 **Summary**

9 Determines if the underlying system type of the current `System.Type` is the same as the  
10 underlying system type of the specified `System.Type`.

### 11 **Parameters**

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

### 13 **Return Value**

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

# 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 A `System.Int32` containing the number of dimensions in the current `System.Type`.

## 10 Exceptions

Exception	Condition
<b>System.ArgumentException</b>	The current <code>System.Type</code> is not an array.

11

12

1 **The following member must be implemented if the Reflection library is present in the**  
2 **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 `System.Type.Attributes` property  
11 and returns the attributes specified for the type represented by the current instance.

#### 12 **Return Value**

13 A `System.Reflection.TypeAttributes` value that signifies the attributes of the type  
14 represented by the current instance.

#### 15 **Behaviors**

16 This property is read-only.

17  
18 This method returns a `System.Reflection.TypeAttributes` value that indicates the  
19 attributes set in the metadata of the type represented by the current instance.

#### 20 **Usage**

21 Use this property to determine the visibility, semantics, and layout format of the type  
22 represented by the current instance. Also use this property to determine if the type  
23 represented by the current instance has a special name.

24

25

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

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

```
[ILAsm]  
.method public hidebysig instance class System.Reflection.ConstructorInfo  
GetConstructor(valuetype System.Reflection.BindingFlags bindingAttr, class  
System.Reflection.Binder binder, class System.Type[] types, class  
System.Reflection.ParameterModifier[] modifiers)
```

```
[C#]  
public ConstructorInfo GetConstructor(BindingFlags bindingAttr, Binder  
binder, Type[] types, ParameterModifier[] modifiers)
```

## Summary

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

## Parameters

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

## Return Value

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

7 **Description**

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

- 10 • Specify either `System.Reflection.BindingFlags.Instance` or  
 11 `System.Reflection.BindingFlags.Static` to get a return value other than `null`.
- 12 • Specify `System.Reflection.BindingFlags.Public` to include public constructors in  
 13 the search.
- 14 • Specify `System.Reflection.BindingFlags.NonPublic` to include non-public  
 15 constructors (that is, private and protected constructors) in the search.

16 [Note: For more information, see `System.Reflection.BindingFlags`.]  
 17  
 18  
 19

20 If the current instance represents a generic type, this method returns the  
 21 `System.Reflection.ConstructorInfo` with the type parameters replaced by the  
 22 appropriate type arguments. If the current instance represents an unassigned type  
 23 parameter of a generic type or method, this method always returns `null`.

24 **Exceptions**

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

25

26 **Permissions**

Permission	Description
<b>System.Security.Permissions.ReflectionPermission</b>	Requires permission to retrieve information on non-public members of types in loaded assemblies. See <code>System.Security.Permissions</code> .

	ReflectionPermissionFlag.TypeInformation.
--	---

1

2

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

## 3 `Type.GetConstructor(System.Type[])` Method

```
4 [ILAsm]  
5 .method public hidebysig instance class System.Reflection.ConstructorInfo  
6 GetConstructor(class System.Type[] types)
```

```
7 [C#]  
8 public ConstructorInfo GetConstructor(Type[] types)
```

### 9 Summary

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

### 12 Parameters

Parameter	Description
<i>types</i>	An array of <code>System.Type</code> 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 <code>System.Type.EmptyTypes</code> to obtain a constructor that takes no parameters.

### 14 Return Value

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

20  
21 If the current instance represents a generic type, this method returns the  
22 `System.Reflection.ConstructorInfo` with the type parameters replaced by the  
23 appropriate type arguments. If the current instance represents an unassigned type  
24 parameter of a generic type or method, this method always returns `null`.

### 25 Description

26 This version of `System.Type.GetConstructor` is equivalent to  
27 `System.Type.GetConstructor(System.Reflection.BindingFlags.Public |`  
28 `System.Reflection.BindingFlags.Instance, null, types, null)`.

### 29 Exceptions

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

1

2 **Permissions**

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

3

4

1 **The following member must be implemented if the Reflection library is present in the**  
2 **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 represented by the  
11 current instance.

### 12 Return Value

13 An array of `System.Reflection.ConstructorInfo` objects that reflect the public  
14 constructors defined in the type represented by the current instance. If no public  
15 constructors are defined in the type represented by the current instance, or if the  
16 current instance represents an unassigned type parameter of a generic type or method,  
17 returns an empty array.

18  
19 If the current instance represents a generic type, this method returns the  
20 `System.Reflection.ConstructorInfo` objects with the type parameters replaced by  
21 the appropriate type arguments.

22  
23 If the current instance represents a generic type parameter, the  
24 `System.Type.GetConstructors` method returns an empty array.

### 25 Description

26 This version of `System.Type.GetConstructors` is equivalent to  
27 `System.Type.GetConstructors(System.Reflection.BindingFlags.Public |`  
28 `System.Reflection.BindingFlags.Instance)`.

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

## Type.GetConstructors(System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public hidebysig virtual abstract class
System.Reflection.ConstructorInfo[] GetConstructors(valuetype
System.Reflection.BindingFlags bindingAttr)

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

### Summary

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

### Parameters

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

### Return Value

An array of `System.Reflection.ConstructorInfo` objects that reflect the constructors that are defined in the type represented by the current instance and match the constraints of *bindingAttr*. If `System.Reflection.BindingFlags.NonPublic` and `System.Reflection.BindingFlags.Static` are specified, this array includes the type initializer if it is defined. If no constructors meeting the constraints of *bindingAttr* are defined in the type represented by the current instance, or if the current instance represents an unassigned type parameter of a generic type or method, returns an empty array. If the type represented by the current instance is contained in a loaded assembly, the constructors that match the specified criteria are not public, and the caller does not have sufficient permission, returns `null`.

If the current instance represents a generic type, this method returns the `System.Reflection.ConstructorInfo` objects with the type parameters replaced by the appropriate type arguments.

If the current instance represents a generic type parameter, the `System.Type.GetConstructors` method returns an empty array.

### Description

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

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

9 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
10  
11

## 12 Behaviors

13 As described above.  
14

## 15 Permissions

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

16

17

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

## 3 Type.GetDefaultMembers() Method

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

### 9 Summary

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

### 12 Return Value

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

### 16 Description

17 If the current instance represents a generic type, this method returns the  
18 `System.Reflection.MemberInfo` objects with the type parameters replaced by the  
19 appropriate type arguments.

20  
21 If the current instance represents an unassigned type parameter of a generic type or  
22 method, this method searches the members of the class constraint, or the members of  
23 `System.Object` if there is no class constraint; the members of all interface constraints;  
24 and the members of any interfaces inherited from class or interface constraints.

### 25 Behaviors

26 The members returned by this method have the  
27 `System.Reflection.DefaultMemberAttribute` attribute.

28

29

# 1 Type.GetElementType() Method

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

## 7 Summary

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

## 9 Return Value

10 A `System.Type` that represents the type used to create the current instance if the  
11 current instance represents an array, pointer, or an argument passed by reference.  
12 Otherwise, returns `null` if the current instance is not an array or a pointer, or is not  
13 passed by reference, or represents a generic type or a type parameter of a generic type  
14 or method.

## 15 Example

16 The following example demonstrates the `System.Type.GetElementType` method.

```
17 [C#]  
18  
19 using System;  
20 class TestType {  
21     public static void Main() {  
22         int[] array = {1,2,3};  
23         Type t = array.GetType();  
24         Type t2 = t.GetElementType();  
25         Console.WriteLine("{0} element type is {1}",array, t2.ToString());  
26  
27         TestType newMe = new TestType();  
28         t = newMe.GetType();  
29         t2 = t.GetElementType();  
30         Console.WriteLine("{0} element type is {1}", newMe, t2==null? "null":  
31 t2.ToString());  
32     }  
33 }
```

34 The output is

```
35  
36 System.Int32[] element type is System.Int32
```

```
37  
38  
39 TestType element type is null
```

40  
41

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

## Type.GetEvent(System.String) Method

```
[ILAsm]
.method public hidebysig instance class System.Reflection.EventInfo
GetEvent(string name)

[C#]
public EventInfo GetEvent(string name)
```

### Summary

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

### Parameters

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

### Return Value

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

If the current instance represents a generic type, this method returns the `System.Reflection.EventInfo` with the type parameters replaced by the appropriate type arguments.

### Description

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

The search for *name* is case-sensitive.

If the current instance represents an unassigned type parameter of a generic type or method, this method searches the events of the class constraint; the events of all interface constraints; and the events of any interfaces inherited from class or interface constraints.

### Exceptions

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

1

2

**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 <code>System.String</code> containing the name of the event to be returned.
<i>bindingAttr</i>	A bitwise combination of <code>System.Reflection.BindingFlags</code> 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.

If the current instance represents a generic type, this method returns the `System.Reflection.EventInfo` with the type parameters replaced by the appropriate type arguments.

### 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 public events in the  
2 search.

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

5 The following `System.Reflection.BindingFlags` value can be used to change how the  
6 search works:

7 • `System.Reflection.BindingFlags.DeclaredOnly` to search only the events  
8 declared on the type, not events that were simply inherited.

9 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
10  
11  
12

13 If the current instance represents an unassigned type parameter of a generic type or  
14 method, this method searches the events of the class constraint; the events of all interface  
15 constraints; and the events of any interfaces inherited from class or interface constraints.

## 16 Behaviors

17 As described above.  
18

## 19 Exceptions

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

20

## 21 Permissions

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

22

23

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

## 3 Type.GetEvents() Method

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

### 9 Summary

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

### 12 Return Value

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

16  
17 If the current instance represents a generic type, this method returns the  
18 `System.Reflection.EventInfo` objects with the type parameters replaced by the  
19 appropriate type arguments.

### 20 Description

21 If the current instance represents an unassigned type parameter of a generic type or  
22 method, this method searches the events of the class constraint; the events of all  
23 interface constraints; and the events of any interfaces inherited from class or interface  
24 constraints.

### 25 Behaviors

26 As described above.

### 28 Default

29 This version of `System.Type.GetEvents` is equivalent to  
30 `System.Type.GetEvents(System.Reflection.BindingFlags.Public |`  
31 `System.Reflection.BindingFlags.Static |`  
32 `System.Reflection.BindingFlags.Instance)`.

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

## Type.GetEvents(System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public hidebysig virtual abstract class
System.Reflection.EventInfo[] GetEvents(valuetype
System.Reflection.BindingFlags bindingAttr)

[C#]
public abstract EventInfo[] GetEvents(BindingFlags bindingAttr)
```

### Summary

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

### Parameters

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

### Return Value

An array of `System.Reflection.EventInfo` objects that reflect the events that are defined in the type represented by the current instance and match the constraints of *bindingAttr*. If no events match these constraints, returns an empty array. If 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 only public events.

If the current instance represents a generic type, this method returns the `System.Reflection.EventInfo` objects with the type parameters replaced by the appropriate type arguments.

### 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 public events in the  
2 search.
- 3 • Specify `System.Reflection.BindingFlags.NonPublic` to include non-public events  
4 (that is, private and protected events) in the search.

5 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
6  
7  
8

9 If the current instance represents an unassigned type parameter of a generic type or  
10 method, this method searches the events of the class constraint; the events of all interface  
11 constraints; and the events of any interfaces inherited from class or interface constraints.

## 12 Behaviors

13 As described above.

14

## 15 Permissions

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

16

17

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

### 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`.

If the current instance represents a generic type, this method returns the `System.Reflection.FieldInfo` with the type parameters replaced by the appropriate type arguments.

### 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 public fields in the  
2 search.

3 • Specify `System.Reflection.BindingFlags.NonPublic` to include non-public fields  
4 (that is, private and protected fields) in the search.

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

7 • `System.Reflection.BindingFlags.DeclaredOnly` to search only the fields declared  
8 in the type, not fields that were simply inherited.

9 • `System.Reflection.BindingFlags.IgnoreCase` to ignore the case of *name*.

10 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
11  
12  
13

14 If the current instance represents an unassigned type parameter of a generic type or  
15 method, this method searches the fields of the class constraint; the fields of all interface  
16 constraints; and the fields of any interfaces inherited from class or interface constraints.

## 17 Behaviors

18 As described above.  
19

## 20 Exceptions

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

21

## 22 Permissions

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

23

24

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

## 3 Type.GetField(System.String) Method

```
4 [ILAsm]  
5 .method public hidebysig instance class System.Reflection.FieldInfo  
6 GetField(string name)  
  
7 [C#]  
8 public FieldInfo GetField(string name)
```

### 9 Summary

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

### 12 Parameters

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

### 14 Return Value

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

21 If the current instance represents a generic type, this method returns the  
22 System.Reflection.FieldInfo with the type parameters replaced by the appropriate  
23 type arguments.

### 24 Description

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

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

32 If the current instance represents an unassigned type parameter of a generic type or  
33 method, this method searches the fields of the class constraint; the fields of all interface  
34 constraints; and the fields of any interfaces inherited from class or interface constraints.

### 35 Exceptions

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

1

2 **Permissions**

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

3

4

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

## 3 Type.GetFields() Method

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

### 9 Summary

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

### 12 Return Value

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

16  
17 If the current instance represents a generic type, this method returns the  
18 `System.Reflection.FieldInfo` objects with the type parameters replaced by the  
19 appropriate type arguments.

### 20 Description

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

25  
26 If the current instance represents an unassigned type parameter of a generic type or  
27 method, this method searches the fields of the class constraint; the fields of all interface  
28 constraints; and the fields of any interfaces inherited from class or interface constraints.

29

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

## Type.GetFields(System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public hidebysig virtual abstract class
System.Reflection.FieldInfo[] GetFields(valuetype
System.Reflection.BindingFlags bindingAttr)

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

### Summary

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

### Parameters

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

### Return Value

An array of `System.Reflection.FieldInfo` objects that reflect the fields that are defined in the type represented by the current instance and match the constraints of *bindingAttr*. If no fields match these constraints, returns an empty array. If 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 only public fields.

If the current instance represents a generic type, this method returns the `System.Reflection.FieldInfo` objects with the type parameters replaced by the appropriate type arguments.

### 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` in order to get a return value other than `null`.

1     • Specify `System.Reflection.BindingFlags.Public` to include public fields in the  
2 search.

3     • Specify `System.Reflection.BindingFlags.NonPublic` to include non-public fields  
4 (that is, private and protected fields) in the search.

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

7     • `System.Reflection.BindingFlags.DeclaredOnly` to search only the fields declared  
8 in the type, not fields that were simply inherited.

9 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
10  
11  
12

13 If the current instance represents an unassigned type parameter of a generic type or  
14 method, this method searches the fields of the class constraint; the fields of all interface  
15 constraints; and the fields of any interfaces inherited from class or interface constraints.

## 16 Behaviors

17     As described above.  
18

## 19 Permissions

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

20

21

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

## 3 **Type.GetGenericArguments() Method**

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

### 8 **Summary**

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

### 11 **Return Value**

12 An array of `System.Type` objects that represent the type arguments of a generic type or  
13 the type parameters of a generic type definition. Returns an empty array if the current  
14 type is not a generic type. The array elements are returned in the order in which they  
15 appear in the list of type arguments for the generic type.

### 16 **Description**

17 If the current type is a closed constructed type, the array returned by the  
18 `System.Type.GetGenericArguments` method contains the type arguments that are  
19 bound to the type parameters. If the current type is a generic type definition, the array  
20 contains the type parameters. If the current type is an open constructed type in which  
21 some of the type parameters are bound to specific types, the array contains both type  
22 arguments and type parameters.  
23

24 For a list of the invariant conditions for terms used in generic reflection, see the  
25 `System.Type.IsGenericType` property description.

### 26 **Example**

27 For an example of using this method, see the example for  
28 `System.Type.GenericParameterPosition`.

29

# 1 Type.GetGenericParameterConstraints() 2 Method

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

## 8 Summary

9 Returns an array of *System.Type* objects that represent the type constraints on the  
10 current generic type parameter.

## 11 Return Value

12 An array of *System.Type* objects that represent the type constraints on the current  
13 generic type parameter.

## 14 Description

15 Each constraint on a generic type parameter is expressed as a *System.Type* object. The  
16 first element of the array is the class constraint, if any. If a type parameter has no class  
17 constraint and no interface constraints, an empty array of *System.Type* is returned for  
18 that type parameter. Use *System.Reflection.GenericParameterAttributes* to get the  
19 special constraints.

20  
21 For a list of the invariant conditions for terms used in generic reflection, see the  
22 *System.Type.IsGenericType* property description.

## 23 Exceptions

Exception	Condition
<b>System.InvalidOperationException</b>	The current <i>System.Type</i> object is not a generic type parameter. That is, the <i>System.Type.IsGenericParameter</i> property returns false.

24

25

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

## Type.GetGenericTypeDefinition() Method

```
[ILAsm]
.method public hidebysig virtual class System.Type
GetGenericTypeDefinition()

[C#]
public virtual Type GetGenericTypeDefinition()
```

### Summary

Returns a `System.Type` object that represents a generic type from which the current type can be constructed.

### Return Value

A `System.Type` object representing a generic type from which the current type can be constructed.

### Description

If two constructed types are created from the same generic type definition, the `System.Type.GetGenericTypeDefinition` method returns the same `System.Type` object for both types.

If you call `System.Type.GetGenericTypeDefinition` on a `System.Type` object that already represents a generic type definition, it returns the current `System.Type`.

[*Note:* An array type whose element type is a generic type is not itself generic. Use `System.Type.IsGenericType` to determine whether a type is generic before calling `System.Type.GetGenericTypeDefinition`.]

For a list of the invariant conditions for terms used in generic reflection, see the `System.Type.IsGenericType` property description.

### Exceptions

Exception	Condition
<code>System.InvalidOperationException</code>	The current type is not a generic type. That is, <code>System.Type.HasGenericArguments</code> returns <code>false</code> .

### Example

1 For an example of using this method, see the example for  
2 `System.Type.MakeGenericType`.

3

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

## 10 Description

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

12

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

14

15

16

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

## Type.GetInterface(System.String, System.Boolean) Method

```
[ILAsm]
.method public hidebysig virtual abstract class System.Type
GetInterface(string name, bool ignoreCase)

[C#]
public abstract Type GetInterface(string name, bool ignoreCase)
```

### Summary

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

### 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.

### Return Value

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

### Description

If the current instance represents a generic type, this method returns the System.Type with the type parameters replaced by the appropriate type arguments.

If the current instance represents an unassigned type parameter of a generic type or method, this method searches the interface constraints and any interfaces inherited from class or interface constraints.

[Note: The *name* parameter cannot include type arguments.]

[Note: Even with the introduction of generics, this method continues to return only non-generic members. To get the generic ones, one must call System.Type.GetInterfaces, and filter them out.]

1  
2

### 3 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>name</i> is null.
<b>System.Reflection.AmbiguousMatchException</b>	The current instance represents a type that implements the same generic interface with different type arguments.

4  
5

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

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

```
4 [ILAsm]  
5 .method public hidebysig instance class System.Type GetInterface(string  
6 name )
```

```
7 [C#]  
8 public Type GetInterface(string name)
```

### 9 Summary

10 Searches for the interface with the specified name.

### 11 Parameters

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

### 13 Return Value

14 A System.Type object representing the interface with the specified name, implemented  
15 or inherited by the current System.Type, if found; otherwise, null.

### 16 Description

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

18  
19 If the current instance represents a generic type, this method returns the System.Type  
20 with the type parameters replaced by the appropriate type arguments.

21  
22 If the current instance represents an unassigned type parameter of a generic type or  
23 method, this method searches the interface constraints and any interfaces inherited  
24 from class or interface constraints.

25  
26 [Note: The *name* parameter cannot include type arguments.]  
27

28  
29  
30 [Note: Even with the introduction of generics, this method continues to return only non-  
31 generic members. To get the generic ones, one must call System.Type.GetInterfaces,  
32 and filter them out.]  
33  
34

### 35 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>name</i> is null.
<b>System.Reflection.AmbiguousMatchException</b>	The current instance represents a type that implements the same generic interface with different type arguments.

1

2

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

## 3 **Type.GetInterfaces() Method**

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

### 9 **Summary**

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

### 12 **Return Value**

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

### 16 **Description**

17 If the current instance represents a generic type, this method returns the *System.Type*  
18 objects with the type parameters replaced by the appropriate type arguments.

19  
20 If the current instance represents an unassigned type parameter of a generic type or  
21 method, this method searches the interface constraints and any interfaces inherited  
22 from class or interface constraints.

23  
24 [*Note:* Even with the introduction of generics, the overloads of  
25 *System.Type.GetInterface* continue to return only non-generic members. To get the  
26 generic ones, one must call *System.Type.GetInterfaces*, and filter them out.]  
27  
28

29

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

## Type.GetMember(System.String, System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public hidebysig virtual class System.Reflection.MemberInfo[]
GetMember(string name, valuetype System.Reflection.BindingFlags
bindingAttr)

[C#]
public virtual MemberInfo[] GetMember(string name, BindingFlags
bindingAttr)
```

### Summary

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

### Parameters

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

### Return Value

An array of `System.Reflection.MemberInfo` objects that reflect the members named *name*, are defined in the type represented by the current instance and match the constraints of *bindingAttr*. If no members match these constraints, returns an empty array. If the selected member is non-public, the type reflected 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 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 public members in the  
2 search.

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

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

7 • `System.Reflection.BindingFlags.DeclaredOnly` to search only the members  
8 declared in the type, not members that were simply inherited.

9 • `System.Reflection.BindingFlags.IgnoreCase` to ignore the case of *name*.

10 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
11  
12  
13

14 If the current instance represents a generic type, this method returns the  
15 `System.Reflection.MemberInfo` with the type parameters replaced by the appropriate type  
16 arguments.

17  
18 If the current instance represents an unassigned type parameter of a generic type or  
19 method, this method searches the members of the class constraint, or the members of  
20 `System.Object` if there is no class constraint; the members of all interface constraints; and  
21 the members of any interfaces inherited from class or interface constraints.  
22

23 [*Note:* The *name* parameter cannot include type arguments.]  
24  
25

## 26 Behaviors

27 As described above.  
28

## 29 Exceptions

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

30

## 31 Permissions

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

1

2

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

## Type.GetMember(System.String) Method

```
[ILAsm]  
.method public hidebysig instance class System.Reflection.MemberInfo[]  
GetMember(string name)
```

```
[C#]  
public MemberInfo[] GetMember(string name)
```

### Summary

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

### Parameters

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

### Return Value

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

### Description

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

The search for *name* is case-sensitive.

If the current instance represents a generic type, this method returns the `System.Reflection.MemberInfo` with the type parameters replaced by the appropriate type arguments.

If the current instance represents an unassigned type parameter of a generic type or method, this method searches the members of the class constraint, or the members of `System.Object` if there is no class constraint; the members of all interface constraints; and the members of any interfaces inherited from class or interface constraints.

1 [Note: The *name* parameter cannot include type arguments.]

2

3

#### 4 Exceptions

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

5

6

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

## Type.GetMembers(System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public hidebysig virtual abstract class
System.Reflection.MemberInfo[] GetMembers(valuetype
System.Reflection.BindingFlags bindingAttr)
```

```
[C#]
public abstract MemberInfo[] GetMembers(BindingFlags bindingAttr)
```

### Summary

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

### Parameters

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

### Return Value

An array of `System.Reflection.MemberInfo` objects that reflect the members defined in the type represented by the current instance that match the constraints of *bindingAttr*. If no members match these constraints, returns an empty array. If 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 only public members.

### 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.

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

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

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

7 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
8  
9

10  
11 If the current instance represents a generic type, this method returns the  
12 `System.Reflection.MemberInfo` objects with the type parameters replaced by the  
13 appropriate type arguments.  
14

15 If the current instance represents an unassigned type parameter of a generic type or  
16 method, this method searches the members of the class constraint, or the members of  
17 `System.Object` if there is no class constraint; the members of all interface constraints; and  
18 the members of any interfaces inherited from class or interface constraints.

## 19 Behaviors

20 As described above.  
21

## 22 Permissions

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

23

24

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

## 3 **Type.GetMembers() Method**

```
4 [ILAsm]  
5 .method public hidebysig instance class System.Reflection.MemberInfo[]  
6 GetMembers()
```

```
7 [C#]  
8 public MemberInfo[] GetMembers()
```

### 9 **Summary**

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

### 12 **Return Value**

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

### 16 **Description**

17 This version of `System.Type.GetMembers` is equivalent to  
18 `System.Type.GetMembers(System.Reflection.BindingFlags.Public |`  
19 `System.Reflection.BindingFlags.Static |`  
20 `System.Reflection.BindingFlags.Instance)`.

21  
22 If the current instance represents a generic type, this method returns the  
23 `System.Reflection.MemberInfo` objects with the type parameters replaced by the  
24 appropriate type arguments.

25  
26 If the current instance represents an unassigned type parameter of a generic type or  
27 method, this method searches the members of the class constraint, or the members of  
28 `System.Object` if there is no class constraint; the members of all interface constraints;  
29 and the members of any interfaces inherited from class or interface constraints.

30

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

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

```
[ILAsm]  
.method public final hidebysig virtual class System.Reflection.MethodInfo  
GetMethod(string name, valuetype System.Reflection.BindingFlags  
bindingAttr, class System.Reflection.Binder binder, class System.Type[]  
types, class System.Reflection.ParameterModifier[] modifiers)
```

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

## Summary

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

## Parameters

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

## Return Value

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

## 7 **Description**

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

- 10 • Specify either `System.Reflection.BindingFlags.Instance` or  
11 `System.Reflection.BindingFlags.Static` to get a return value other than `null`.
- 12 • Specify `System.Reflection.BindingFlags.Public` to include public members in the  
13 search.
- 14 • Specify `System.Reflection.BindingFlags.NonPublic` to include non-public  
15 members (that is, private and protected members) in the search.

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

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

21 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
22  
23  
24

25 If the current instance represents a generic type, this method returns the  
26 `System.Reflection.MethodInfo` with the type parameters replaced by the appropriate type  
27 arguments.  
28

29 If the current instance represents an unassigned type parameter of a generic type or  
30 method, this method searches the methods of the class constraint, or the methods of  
31 `System.Object` if there is no class constraint; the methods of all interface constraints; and  
32 the methods of any interfaces inherited from class or interface constraints.  
33

34 [*Note:* The *name* parameter cannot include type arguments.]  
35  
36

## 37 **Exceptions**

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

1

2 Permissions

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

3

4

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

### 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`.
- 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.

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

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

6 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
7  
8  
9

10 This version of `System.Type.GetMethod` is equivalent to `System.Type.GetMethod(name,`  
11 `bindingAttr, null, null, null)`.

12  
13 If the current instance represents a generic type, this method returns the  
14 `System.Reflection.MethodInfo` with the type parameters replaced by the appropriate type  
15 arguments.

16  
17 If the current instance represents an unassigned type parameter of a generic type or  
18 method, this method searches the methods of the class constraint, or the methods of  
19 `System.Object` if there is no class constraint; the methods of all interface constraints; and  
20 the methods of any interfaces inherited from class or interface constraints.

21  
22 [*Note:* The *name* parameter cannot include type arguments.]  
23  
24

## 25 Exceptions

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

26

## 27 Permissions

Permission	Description
<b>System.Security.Permissions.ReflectionPermission</b>	Requires permission to retrieve information on non-public members of types in loaded assemblies. See <code>System.Security.Permissions</code> .

	ReflectionPermissionFlag.TypeInformation.
--	---

1

2

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

## 3 Type.GetMethod(System.String) Method

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

### 9 Summary

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

### 12 Parameters

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

### 14 Return Value

15 A `System.Reflection.MethodInfo` object reflecting the public method that is defined in  
16 the type represented by the current instance and has the specified name, if found;  
17 otherwise, `null`.

### 18 Description

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

20  
21 This version of `System.Type.GetMethod` is equivalent to  
22 `System.Type.GetMethod(name, System.Reflection.BindingFlags.Public |`  
23 `System.Reflection.BindingFlags.Static |`  
24 `System.Reflection.BindingFlags.Instance, null, null, null)`.

25  
26 If the current instance represents a generic type, this method returns the  
27 `System.Reflection.MethodInfo` with the type parameters replaced by the appropriate  
28 type arguments.

29  
30 If the current instance represents an unassigned type parameter of a generic type or  
31 method, this method searches the methods of the class constraint, or the methods of  
32 `System.Object` if there is no class constraint; the methods of all interface constraints;  
33 and the methods of any interfaces inherited from class or interface constraints.

34  
35 [*Note:* The *name* parameter cannot include type arguments.]

1  
2

### 3 Exceptions

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

4  
5

**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 <code>System.String</code> containing the name of the public method to be returned.
<i>types</i>	An array of <code>System.Type</code> 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)`.

If the current instance represents a generic type, this method returns the `System.Reflection.MethodInfo` with the type parameters replaced by the appropriate type arguments.

1 If the current instance represents an unassigned type parameter of a generic type or  
2 method, this method searches the methods of the class constraint, or the methods of  
3 `System.Object` if there is no class constraint; the methods of all interface constraints;  
4 and the methods of any interfaces inherited from class or interface constraints.

5  
6 [Note: The *name* parameter cannot include type arguments.]  
7  
8

## 9 Exceptions

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

10

11

**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 <code>System.String</code> containing the name of the public method to be returned.
<i>types</i>	An array of <code>System.Type</code> 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 <code>null</code> .

## 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

The default binder does not process *modifier*.

The search for *name* is case-sensitive.

1 This version of `System.Type.GetMethod` is equivalent to `System.Type.GetMethod (`  
 2 `name, System.Reflection.BindingFlags.Public`  
 3 `|System.Reflection.BindingFlags.Static`  
 4 `|System.Reflection.BindingFlags.Instance, null, types, modifiers)`.

5  
 6 If the current instance represents a generic type, this method returns the  
 7 `System.Reflection.MethodInfo` with the type parameters replaced by the appropriate  
 8 type arguments.

9  
 10 If the current instance represents an unassigned type parameter of a generic type or  
 11 method, this method searches the methods of the class constraint, or the methods of  
 12 `System.Object` if there is no class constraint; the methods of all interface constraints;  
 13 and the methods of any interfaces inherited from class or interface constraints.

14  
 15 [Note: The *name* parameter cannot include type arguments.]  
 16  
 17

18 **Exceptions**

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

19

20

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

## Type.GetMethods(System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public hidebysig virtual abstract class
System.Reflection.MethodInfo[] GetMethods(valuetype
System.Reflection.BindingFlags bindingAttr)

[C#]
public abstract MethodInfo[] GetMethods(BindingFlags bindingAttr)
```

### Summary

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

### Parameters

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

### Return Value

An array of `System.Reflection.MethodInfo` objects reflecting the methods defined in the type represented by the current instance that match the constraints of *bindingAttr*. If no such methods found, returns an empty array. If the type represented 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 only public methods.

### 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.

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

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

5 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
6  
7  
8

9 If the current instance represents a generic type, this method returns the  
10 `System.Reflection.MethodInfo` objects with the type parameters replaced by the  
11 appropriate type arguments.

12  
13 If the current instance represents an unassigned type parameter of a generic type or  
14 method, this method searches the methods of the class constraint, or the methods of  
15 `System.Object` if there is no class constraint; the methods of all interface constraints; and  
16 the methods of any interfaces inherited from class or interface constraints.

## 17 Behaviors

18 As described above.

19

## 20 Permissions

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

21

22

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

## 3 Type.GetMethods() Method

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

### 9 Summary

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

### 11 Return Value

12 An array of `System.Reflection.MethodInfo` objects reflecting the public methods  
13 defined in the type represented by the current instance. If no methods are found,  
14 returns an empty array.

### 15 Description

16 This version of `System.Type.GetMethods` is equivalent to `System.Type.GetMethods(  
17 System.Reflection.BindingFlags.Instance |  
18 System.Reflection.BindingFlags.Static |  
19 System.Reflection.BindingFlags.Public)`.

20  
21 If the current instance represents a generic type, this method returns the  
22 `System.Reflection.MethodInfo` objects with the type parameters replaced by the  
23 appropriate type arguments.

24  
25 If the current instance represents an unassigned type parameter of a generic type or  
26 method, this method searches the methods of the class constraint, or the methods of  
27 `System.Object` if there is no class constraint; the methods of all interface constraints;  
28 and the methods of any interfaces inherited from class or interface constraints.

29

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

## 3 Type.GetNestedType(System.String) Method

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

### 9 Summary

10 Returns the public nested type defined in the type represented by the current instance

### 11 Parameters

Parameter	Description
<i>name</i>	A <code>System.String</code> containing the name of the public 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 <code>typeA.GetNestedType("B").</code> ]

### 13 Return Value

14 A `System.Type` object representing the public nested type with the specified name, if  
15 found; otherwise, `null`.

### 16 Description

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

18  
19 Use the simple name of the nested class for *name*; do not qualify it with the name of the  
20 outer class. CLS rules require a naming pattern for nested types; see Partition I.

21  
22 If the current instance represents an unassigned type parameter of a generic type or  
23 method definition, this method does not search the nested types of the class constraint.

24  
25 [*Note:* The *name* parameter cannot include type arguments. For example, passing  
26 "MyGenericNestedType<int>" to this method searches for a nested type with the text  
27 name "MyGenericNestedType<int>", rather than for a nested type named  
28 MyGenericNestedType that has one generic argument of type `int`.]

29  
30 [*Note:* If the nested type is generic, what is returned is always a generic type  
31 definition.]

1  
2 For information on constructing nested generic types from their generic type definitions,  
3 see the `System.Type.MakeGenericType(System.Type[])` method.

4 **Exceptions**

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

5

6

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

## Type.GetNestedType(System.String, System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public hidebysig virtual abstract class System.Type
GetNestedType(string name, valuetype System.Reflection.BindingFlags
bindingAttr)

[C#]
public abstract Type GetNestedType(string name, BindingFlags bindingAttr)
```

### Summary

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

### Parameters

Parameter	Description
<i>name</i>	A <code>System.String</code> 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 <code>typeA.GetNestedType("B").</code> ]
<i>bindingAttr</i>	A bitwise combination of <code>System.Reflection.BindingFlags</code> values that control the binding process. If zero is specified, this method returns <code>null</code> .

### Return Value

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

1 • Specify `System.Reflection.BindingFlags.Public` to include public members in the  
2 search.

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

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

7 • `System.Reflection.BindingFlags.DeclaredOnly` to search only the members  
8 declared in the type, not members that were simply inherited.

9 • `System.Reflection.BindingFlags.IgnoreCase` to ignore the case of *name*.

10 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
11  
12  
13

14 If the current instance represents an unassigned type parameter of a generic type or  
15 method, this method searches the nested types of the class constraint.  
16

17 [*Note:* The *name* parameter cannot include type arguments.]  
18  
19

## 20 Exceptions

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

21

## 22 Permissions

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

23

24

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

### 3 **Type.GetNestedTypes() Method**

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

#### 8 **Summary**

9 Returns all the public types nested within the current `System.Type`.

#### 10 **Return Value**

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

#### 14 **Description**

15 This version of `System.Type.GetNestedTypes` is equivalent to  
16 `System.Type.GetNestedTypes(System.Reflection.BindingFlags.Public)`.

17  
18 If the current instance represents an unassigned type parameter of a generic type or  
19 method, this method searches the nested types of the class constraint.

20

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

## Type.GetNestedTypes(System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public hidebysig virtual abstract class System.Type[]
GetNestedTypes(valuetype System.Reflection.BindingFlags bindingAttr)

[C#]
public abstract Type[] GetNestedTypes(BindingFlags bindingAttr)
```

### Summary

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

### 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.

### Return Value

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

### Description

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

- 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.

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

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

3 *[Note: For more information, see `System.Reflection.BindingFlags`.]*  
4  
5  
6

7 If the current instance represents an unassigned type parameter of a generic type or  
8 method, this method searches the nested types of the class constraint.

9 **Permissions**

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

10

11

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

## 3 Type.GetProperties(System.Reflection.Binding 4 gFlags) 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 bindingAttr)
```

### 11 Summary

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

### 15 Parameters

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

### 17 Return Value

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

### 24 Description

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

- 27 • Specify either `System.Reflection.BindingFlags.Instance` or  
28 `System.Reflection.BindingFlags.Static` to get a return value other than `null`.
- 29 • Specify `System.Reflection.BindingFlags.Public` to include public members in the  
30 search.

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

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

8  
9  
10

If the current instance represents a generic type, this method returns the `System.Reflection.PropertyInfo` objects with the type parameters replaced by the appropriate type arguments.

If the current instance represents an unassigned type parameter of a generic type or method, this method searches the properties of the class constraint; the properties of all interface constraints; and the properties of any interfaces inherited from class or interface constraints.

## Behaviors

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

22

## Permissions

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

24

25

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

## 3 **Type.GetProperties() Method**

```
4 [ILAsm]  
5 .method public hidebysig instance class System.Reflection.PropertyInfo[]  
6 GetProperties()
```

```
7 [C#]  
8 public PropertyInfo[] GetProperties()
```

### 9 **Summary**

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

### 12 **Return Value**

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

### 16 **Description**

17 This version of `System.Type.GetProperties` is equivalent to  
18 `System.Type.GetProperties( System.Reflection.BindingFlags.Instance |`  
19 `System.Reflection.BindingFlags.Static |`  
20 `System.Reflection.BindingFlags.Public )`.

21 A property is considered by reflection to be `public` if it has at least one accessor that is  
22 `public`. Otherwise, the property is considered to be not `public`.

23 If the current instance represents a generic type, this method returns the  
24 `System.Reflection.PropertyInfo` objects with the type parameters replaced by the  
25 appropriate type arguments.

26 If the current instance represents an unassigned type parameter of a generic type or  
27 method, this method searches the properties of the class constraint; the properties of all  
28 interface constraints; and the properties of any interfaces inherited from class or  
29 interface constraints.

33

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

## Type.GetProperty(System.String, System.Type, System.Type[]) Method

```
[ILAsm]
.method public hidebysig instance class System.Reflection.PropertyInfo
GetProperty(string name, class System.Type returnType, class System.Type[]
types)

[C#]
public PropertyInfo GetProperty(string name, Type returnType, Type[]
types)
```

### Summary

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

### Parameters

Parameter	Description
<i>name</i>	A <code>System.String</code> containing the name of the public property to be returned.
<i>returnType</i>	A <code>System.Type</code> object that represents the type of the public property to be returned.
<i>types</i>	An array of <code>System.Type</code> 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 <code>System.Type.EmptyTypes</code> for a property that is not indexed.

### Return Value

A `System.Reflection.PropertyInfo` object reflecting the public property defined in the type represented by the current instance that matches the specified criteria. 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, returnTypes, types, null)`.

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

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

8  
9 If the current instance represents a generic type, this method returns the  
10 `System.Reflection.PropertyInfo` with the type parameters replaced by the  
11 appropriate type arguments.

12  
13 If the current instance represents an unassigned type parameter of a generic type or  
14 method, this method searches the properties of the class constraint; the properties of all  
15 interface constraints; and the properties of any interfaces inherited from class or  
16 interface constraints.

### 17 Exceptions

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

18

19

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

## Type.GetProperty(System.String, System.Type[]) Method

```
[ILAsm]
.method public hidebysig instance class System.Reflection.PropertyInfo
GetProperty(string name, class System.Type[] types)

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

### Summary

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

### Parameters

Parameter	Description
<i>name</i>	A <code>System.String</code> containing the name of the public property to be returned.
<i>types</i>	An array of <code>System.Type</code> 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 <code>System.Type.EmptyTypes</code> to obtain a property that is not indexed.

### Return Value

A `System.Reflection.PropertyInfo` object reflecting the public property defined on the type represented by the current instance that matches the specified criteria. 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, types, null)`.

The search for *name* is case-sensitive.

Different programming languages use different syntax to specify indexed properties. Internally, this property is referred to by the name "Item" in the metadata. Therefore, any attempt to retrieve an indexed property using reflection is required to specify this

1 internal name in order for the `PropertyInfo` to be returned correctly.  
2  
3 If the current instance represents a generic type, this method returns the  
4 `System.Reflection.PropertyInfo` with the type parameters replaced by the  
5 appropriate type arguments.  
6  
7 If the current instance represents an unassigned type parameter of a generic type or  
8 method, this method searches the properties of the class constraint; the properties of all  
9 interface constraints; and the properties of any interfaces inherited from class or  
10 interface constraints.

11 **Exceptions**

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

12

13

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

## Type.GetProperty(System.String, System.Type) Method

```
[ILAsm]
.method public hidebysig instance class System.Reflection.PropertyInfo
GetProperty(string name, class System.Type returnType)

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

### Summary

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

### Parameters

Parameter	Description
<i>name</i>	A <code>System.String</code> containing the name of the property to be returned.
<i>returnType</i>	A <code>System.Type</code> object that represents the type 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 matches the specified criteria. 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, returnType, null, null)`.

The search for *name* is case-sensitive.

If the current instance represents a generic type, this method returns the `System.Reflection.PropertyInfo` with the type parameters replaced by the appropriate type arguments.

If the current instance represents an unassigned type parameter of a generic type or method, this method searches the properties of the class constraint; the properties of all

1 interface constraints; and the properties of any interfaces inherited from class or  
2 interface constraints.

3 **Exceptions**

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

4

5

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

## 3 Type.GetProperty(System.String) Method

```
4 [ILAsm]  
5 .method public hidebysig instance class System.Reflection.PropertyInfo  
6 GetProperty(string name)
```

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

### 9 Summary

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

### 12 Parameters

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

### 14 Return Value

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

### 18 Description

19 This version of `System.Type.GetProperty` is equivalent to  
20 `System.Type.GetPropertyImpl(name, System.Reflection.BindingFlags.Static |`  
21 `System.Reflection.BindingFlags.Instance |`  
22 `System.Reflection.BindingFlags.Public, null, null, null, null)`.

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

25  
26 If the current instance represents a generic type, this method returns the  
27 `System.Reflection.PropertyInfo` with the type parameters replaced by the  
28 appropriate type arguments.

29  
30 If the current instance represents an unassigned type parameter of a generic type or  
31 method, this method searches the properties of the class constraint; the properties of all  
32 interface constraints; and the properties of any interfaces inherited from class or  
33 interface constraints.

### 34 Exceptions

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

1

2

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

## Type.GetProperty(System.String, System.Reflection.BindingFlags) Method

```
[ILAsm]
.method public final hidebysig virtual class
System.Reflection.PropertyInfo GetProperty(string name, valuetype
System.Reflection.BindingFlags bindingAttr)

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

### Summary

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

### Return Value

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

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

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

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

8 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
9

10  
11

12 This version of `System.Type.GetProperty` is equivalent to  
13 `System.Type.GetPropertyImpl(name, bindingAttr, null, null, null, null)`.

14

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

16

17 If the current instance represents a generic type, this method returns the  
18 `System.Reflection.PropertyInfo` with the type parameters replaced by the appropriate  
19 type arguments.

20

21 If the current instance represents an unassigned type parameter of a generic type or  
22 method, this method searches the properties of the class constraint; the properties of all  
23 interface constraints; and the properties of any interfaces inherited from class or interface  
24 constraints.

## 25 Exceptions

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

26

## 27 Permissions

Permission	Description
<b>System.Security.Permissions.ReflectionPermission</b>	Requires permission to retrieve information on non-public members of types in loaded assemblies. See <code>System.Security.Permissions</code> .

	ReflectionPermissionFlag.TypeInformation.
--	---

1

2

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

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

```
[ILAsm]  
.method public final hidebysig virtual class  
System.Reflection.PropertyInfo GetProperty(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#]  
public PropertyInfo GetProperty(string name, BindingFlags bindingAttr,  
Binder binder, Type returnType, Type[] types, ParameterModifier[]  
modifiers)
```

## Summary

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 <code>System.String</code> containing the name of the property to be returned.
<i>bindingAttr</i>	A bitwise combination of <code>System.Reflection.BindingFlags</code> values that control the binding process. If zero is specified, this method returns <code>null</code> .
<i>binder</i>	A <code>System.Reflection.Binder</code> object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specify <code>null</code> to use the <code>System.Type.DefaultBinder</code> .
<i>returnType</i>	A <code>System.Type</code> object that represents the type of the property to be returned.
<i>types</i>	An array of <code>System.Type</code> 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 <code>System.Type.EmptyTypes</code> to obtain a property that is not indexed.
<i>modifiers</i>	The only defined value for this parameter is <code>null</code> .

1

2 **Return Value**

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

8 **Description**

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

- 11 • Specify either `System.Reflection.BindingFlags.Instance` or  
 12 `System.Reflection.BindingFlags.Static` to get a return value other than `null`.
- 13 • Specify `System.Reflection.BindingFlags.Public` to include public members in the  
 14 search.
- 15 • Specify `System.Reflection.BindingFlags.NonPublic` to include non-public  
 16 members (that is, private and protected members) in the search.

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

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

22 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
 23  
 24  
 25

26 This version of `System.Type.GetProperty` is equivalent to  
 27 `System.Type.GetPropertyImpl(name, bindingAttr, binder, returnType, types, modifiers)`.  
 28

29 Different programming languages use different syntax to specify indexed properties.  
 30 Internally, this property is referred to by the name "Item" in the metadata. Therefore, any  
 31 attempt to retrieve an indexed property using reflection is required to specify this internal  
 32 name in order for the `PropertyInfo` to be returned correctly.  
 33

1 If the current instance represents a generic type, this method returns the  
2 `System.Reflection.PropertyInfo` with the type parameters replaced by the appropriate  
3 type arguments.

4  
5 If the current instance represents an unassigned type parameter of a generic type or  
6 method, this method searches the properties of the class constraint; the properties of all  
7 interface constraints; and the properties of any interfaces inherited from class or interface  
8 constraints.

9 **Exceptions**

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

10

11 **Permissions**

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

12

13

**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 <code>System.String</code> containing the name of the property to be returned.
<i>bindingAttr</i>	A bitwise combination of <code>System.Reflection.BindingFlags</code> values that control the binding process. If zero is specified, this method returns <code>null</code> .
<i>binder</i>	A <code>System.Reflection.Binder</code> object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specify <code>null</code> to use the <code>System.Type.DefaultBinder</code> .
<i>returnType</i>	A <code>System.Type</code> object that represents the type of the property to be returned.
<i>types</i>	An array of <code>System.Type</code> 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 <code>System.Type.EmptyTypes</code> to obtain a property that is not indexed.
<i>modifiers</i>	The only defined value for this parameter is <code>null</code> .

1

## 2 Return Value

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

## 8 Description

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

- 11 • Specify either `System.Reflection.BindingFlags.Instance` or  
 12 `System.Reflection.BindingFlags.Static` to get a return value other than `null`.
- 13 • Specify `System.Reflection.BindingFlags.Public` to include public members in the  
 14 search.
- 15 • Specify `System.Reflection.BindingFlags.NonPublic` to include non-public  
 16 members (that is, private and protected members) in the search.

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

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

22 [*Note:* For more information, see `System.Reflection.BindingFlags`.]  
 23  
 24

## 25 Behaviors

26 Different programming languages use different syntax to specify indexed properties.  
 27 Internally, this property is referred to by the name "Item" in the metadata. Therefore,  
 28 any attempt to retrieve an indexed property using reflection is required to specify this  
 29 internal name in order for the `PropertyInfo` to be returned correctly.

1

## 2 Exceptions

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

3

## 4 Permissions

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

5

6

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

## Type.GetType(System.String, System.Boolean, System.Boolean) Method

```
[ILAsm]
.method public hidebysig static class System.Type GetType(string typeName,
bool throwOnError, bool ignoreCase)

[C#]
public static Type GetType(string typeName, bool throwOnError, bool
ignoreCase)
```

### Summary

Returns the `System.Type` with the specified name, optionally performing a case-insensitive search and optionally throwing an exception if an error occurs while loading the `System.Type`.

### Parameters

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

### Return Value

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

### Description

*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, the assembly name, and generic type arguments.]

1  
 2 If *typeName* includes only the name of the `System.Type`, this method searches in the  
 3 calling object's assembly, then in the `mscorlib.dll` assembly. If *typeName* is fully  
 4 qualified with the partial or complete assembly name, this method searches in the  
 5 specified assembly.

6  
 7 [Note:

8  
 9 The following table shows calls to `GetType` for various types. (Some long strings have  
 10 been wrapped to fit in the right column.)

<b>To Get this Type</b>	<b>Use this String</b>
An unmanaged pointer to <code>MyType</code>	<code>Type.GetType("MyType*")</code>
An unmanaged pointer to a pointer to <code>MyType</code>	<code>Type.GetType("MyType**")</code>
A managed pointer or reference to <code>MyType</code>	<code>Type.GetType("MyType&amp;")</code> Note that unlike pointers, references are limited to one level.
A parent class and a nested class	<code>Type.GetType("MyParentClass+MyNestedClass")</code>
A one-dimensional array with a lower bound of 0	<code>Type.GetType("MyArray[ ]")</code>
A one-dimensional array with an unknown lower	<code>Type.GetType("MyArray[*]")</code>

bound	
An n-dimensional array	A comma (,) inside the brackets a total of n-1 times. For example, <code>System.Object[ , , ]</code> represents a three-dimensional <code>Object</code> array.
A two-dimensional array's array	<code>Type.GetType("MyArray[ ][ ]")</code>
A rectangular two-dimensional array with unknown lower bounds	<code>Type.GetType("MyArray[ *, * ]")</code> Or <code>Type.GetType("MyArray[ , ]")</code>
A generic type with one type argument	<code>Type.GetType("MyGenericType[MyType]")</code>
A generic type with two type arguments	<code>Type.GetType("MyGenericType[MyType, AnotherType]")</code>
A generic type with two assembly-qualified type arguments	<code>Type.GetType("MyGenericType[[MyType, MyAssembly], [AnotherType, AnotherAssembly]]")</code>
An assembly-qualified generic type with an assembly-	<code>Type.GetType("MyGenericType[[MyType, MyAssembly]], MyGenericTypeAssembly")</code>

qualified type argument	
A generic type whose type argument is a generic type with two type arguments	<code>Type.GetType("MyGenericType[AnotherGenericType [MyType,AnotherType]]")</code>

1  
2 ]

### 3 Exceptions

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

### 4 5 Permissions

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

6  
7

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

## Type.GetType(System.String, System.Boolean) Method

```
[ILAsm]
.method public hidebysig static class System.Type GetType(string typeName,
bool throwOnError)
[C#]
public static Type GetType(string typeName, bool throwOnError)
```

### Summary

Returns the `System.Type` with the specified name, optionally throwing an exception if an error occurs while loading the `System.Type`.

### Parameters

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

### Return Value

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

### Description

This method is equivalent to `System.Type.GetType(name, throwOnError, false)`.

*typeName* can be a simple type name, a fully qualified name, or a complex name that includes an assembly name specification. 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:* `System.Type.AssemblyQualifiedName` can return a fully qualified type name including nested types, the assembly name, and generic type arguments. For complete details, see `System.Type.GetType(System.String, System.Boolean,`

1 System.Boolean).]

2

3

#### 4 Exceptions

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

5

#### 6 Permissions

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

7

8

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

## Type.GetType(System.String) Method

```
[ILAsm]
.method public hidebysig static class System.Type GetType(string typeName)

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

### Summary

Returns the `System.Type` with the specified name.

### Parameters

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

### Return Value

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

### Description

This method is equivalent to `System.Type.GetType(name, false, false)`.

*typeName* can be a simple type name, a type name that includes a namespace, or a complex name that includes an assembly name specification. 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:* `System.Type.AssemblyQualifiedName` can return a fully qualified type name including nested types, the assembly name, and generic type arguments. For complete details, see `System.Type.GetType(System.String, System.Boolean, System.Boolean)`.]

### Exceptions

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

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

1

2 **Permissions**

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

3

4

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

## 3 **Type.GetTypeArray(System.Object[]) Method**

```
4 [ILAsm]  
5 .method public hidebysig static class System.Type[] GetTypeArray(object[]  
6 args)
```

```
7 [C#]  
8 public static Type[] GetTypeArray(object[] args)
```

### 9 **Summary**

10 Returns the types of the objects in the specified array.

### 11 **Parameters**

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

### 12 **Return Value**

13 An array of *System.Type* objects representing the types of the corresponding elements  
14 in *args*. If a requested type is not public and the caller does not have permission to  
15 reflect non-public objects outside the current assembly, the corresponding element in  
16 the array returned by this method will be *null*.  
17

### 18 **Exceptions**

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

### 19 **Permissions**

Permission	Description
<b>System.Security.Permissions.ReflectionPermission</b>	Requires permission to retrieve information on non-public members of types in loaded assemblies. See <i>System.Security.Permissions</i> .

	ReflectionPermissionFlag.TypeInformation.
--	---

1

2

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

## 3 Type.GetTypeFromHandle(System.RuntimeType 4 peHandle) Method

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

### 10 Summary

11 Gets the System.Type referenced by the specified type handle.

### 12 Parameters

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

### 14 Return Value

15 The System.Type referenced by the specified System.RuntimeTypeHandle.

### 16 Description

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

### 18 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>handle</i> is null.
<b>System.Security.SecurityException</b>	The requested type is non-public and outside the current assembly, and the caller does not have the required permission.
<b>System.Reflection.TargetInvocationException</b>	A type initializer was invoked and threw an exception.

### 20 Permissions

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

1

2

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

### 3 **Type.GetTypeHandle(System.Object) Method**

```
4 [ILAsm]  
5 .method public hidebysig static valuetype System.RuntimeTypeHandle  
6 GetTypeHandle(object o)  
  
7 [C#]  
8 public static RuntimeTypeHandle GetTypeHandle(object o)
```

#### 9 **Summary**

10 Returns the handle for the System.Type of the specified object.

#### 11 **Parameters**

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

12

#### 13 **Return Value**

14 The System.RuntimeTypeHandle for the System.Type of the specified System.Object.

#### 15 **Description**

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

17

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

## 3 `Type.HasElementTypeImpl()` Method

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

### 8 **Summary**

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

### 13 **Return Value**

14 true if the `System.Type` is an array, a pointer, or is passed by reference; otherwise,  
15 false.

### 16 **Description**

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

23

**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.Globalization.CultureInfo) Method**

```
[ILAsm]
.method public hidebysig instance object InvokeMember(string name,
valuetype System.Reflection.BindingFlags invokeAttr, class
System.Reflection.Binder binder, object target, object[] args, class
System.Globalization.CultureInfo culture)

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

## Summary

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

## 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>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 <code>System.Object</code> on which to invoke the member that matches the other specified criteria. If the matching member is <code>static</code> , 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 <code>null</code> for a member that has no parameters.
<i>culture</i>	The only defined value for this parameter is <code>null</code> .

1

## 2 Return Value

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

## 6 Description

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

## 10 Exceptions

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

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

	<p><code>System.Reflection.BindingFlags.SetField</code> Or  <code>System.Reflection.BindingFlags.SetProperty</code>.</p> <p>-or-</p> <p><i>invokeAttr</i> contains  <code>System.Reflection.BindingFlags.SetField</code> and  <i>args</i> has more than one element.</p>
<b>System.MissingFieldException</b>	A field or property matching the specified criteria was not found.
<b>System.MissingMethodException</b>	<p>A method matching the specified criteria was not found.</p> <p>-or-</p> <p>The current instance object represents a type that contains open type parameters (that is, <code>System.Type.ContainsGenericParameters</code> returns true).</p>
<b>System.MethodAccessException</b>	The requested member is non-public and the caller does not have the required permission.
<b>System.Reflection.TargetException</b>	The member matching the specified criteria cannot be invoked on <i>target</i> .
<b>System.Reflection.TargetInvocationException</b>	The member matching the specified criteria threw an exception.
<b>System.Reflection.AmbiguousMatchException</b>	More than one member matches the specified criteria.

1

2 **Example**

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

8 **Permissions**

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

1

2

**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[]) Method

```
[ILAsm]
.method public hidebysig instance object InvokeMember(string name,
valuetype System.Reflection.BindingFlags invokeAttr, class
System.Reflection.Binder binder, object target, object[] args)

[C#]
public object InvokeMember(string name, BindingFlags invokeAttr, Binder
binder, object target, object[] args)
```

### Summary

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

### Parameters

Parameter	Description
<i>name</i>	A <code>System.String</code> 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 <code>System.String.Empty</code> to invoke that member. [Note: For more information on default members, see <code>System.Reflection.DefaultMemberAttribute</code> .]
<i>invokeAttr</i>	A bitwise combination of <code>System.Reflection.BindingFlags</code> values that control the binding process. If zero is specified, <code>System.Reflection.BindingFlags.Public</code>   <code>System.Reflection.BindingFlags.Instance</code> is used by default.
<i>binder</i>	A <code>System.Reflection.Binder</code> object that defines a set of properties and enables the binding, coercion of argument types, and invocation of members using reflection. Specify <code>null</code> to use the <code>System.Type.DefaultBinder</code> .
<i>target</i>	A <code>System.Object</code> on which to invoke the member that matches the other specified criteria. If the matching member is <code>static</code> , 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 <code>null</code> for a member that has no parameters.
-------------	--

1

2 **Return Value**

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

6 **Description**

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

10 [*Note:* For a demonstration of the use of `System.Type.InvokeMember`, see the example  
 11 for `System.Type.InvokeMember(System.String, System.Reflection.BindingFlags,  
 12 System.Reflection.Binder, System.Object, System.Object[],  
 13 System.Reflection.ParameterModifier[], System.Globalization.CultureInfo,  
 14 System.String[])`.]  
 15  
 16

17 **Exceptions**

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

	<p><i>invokeAttr</i> contains  <code>System.Reflection.BindingFlags.InvokeMethod</code>  and at least one of  <code>System.Reflection.BindingFlags.SetField</code> or  <code>System.Reflection.BindingFlags.SetProperty</code>.</p> <p>-or-</p> <p><i>invokeAttr</i> contains  <code>System.Reflection.BindingFlags.SetField</code> and  <i>args</i> has more than one element.</p>
<b>System.MissingFieldException</b>	A field or property matching the specified criteria was not found.
<b>System.MissingMethodException</b>	<p>A method matching the specified criteria cannot be found.</p> <p>-or-</p> <p>The current instance object represents a type that contains open type parameters (that is, <code>System.Type.ContainsGenericParameters</code> returns true).</p>
<b>System.MethodAccessException</b>	The requested member is non-public and the caller does not have the required permission.
<b>System.Reflection.TargetException</b>	The member matching the specified criteria cannot be invoked on <i>target</i> .
<b>System.Reflection.TargetInvocationException</b>	The member matching the specified criteria threw an exception.
<b>System.Reflection.AmbiguousMatchException</b>	More than one member matches the specified criteria.

- 1
- 2 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

**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, object[] args, class
System.Reflection.ParameterModifier[] modifiers, class
System.Globalization.CultureInfo culture, 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>binder</i>	A <code>System.Reflection.Binder</code> that defines a set of properties, and enables the binding, coercion of argument types, and invocation of members using reflection. Specify <code>null</code> to use <code>System.Type.DefaultBinder</code> .
<i>target</i>	A <code>System.Object</code> on which to invoke the member that matches the other specified criteria. If the matching member is <code>static</code> , 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 <code>null</code> . If <i>namedParameters</i> is not <code>null</code> , 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 <code>null</code> for a member that takes no parameters.
<i>modifiers</i>	The only defined value for this parameter is <code>null</code> .
<i>culture</i>	The only defined value for this parameter is <code>null</code> .
<i>namedParameters</i>	An array of <code>System.String</code> 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 <code>null</code> for a member that takes no parameters. Specify <code>null</code> to have this parameter ignored.

1

## 2 Return Value

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

## 6 Description

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

9

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

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

4  
5 The following `System.Reflection.BindingFlags` are used to define which members to  
6 include in the search:

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

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

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

18 [Note: For more information, see `System.Reflection.BindingFlags`.]  
19  
20

## 21 Behaviors

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

25  
26 A member will be found only if the number of parameters in the member declaration  
27 equals the number of arguments in the *args* array (unless default arguments are defined  
28 on the member). Also, The type of each argument is required to be convertible by the  
29 binder to the type of the parameter.

30  
31 It is required that the caller specify values for *invokeAttr* as follows:

Action	BindingFlags
Invoke a constructor.	<code>System.Reflection.BindingFlags.CreateInstance</code> . 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.	<code>System.Reflection.BindingFlags.InvokeMethod</code> . This flag if not valid with <code>System.Reflection.BindingFlags.CreateInstance</code> , <code>System.Reflection.BindingFlags.SetField</code> , or

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

1  
2  
3  
4  
5  
6

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

## 7 Usage

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

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

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

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

```
23  
24 typeof(C).InvokeMember("F", BindingFlags.SetField, null, C, new Object{new
```

1 String[]{"a","z","c","d"}, null, null, null);

2  
3

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

6

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

9

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

## 11 Exceptions

Exception	Condition
<b>System.ArgumentNullException</b>	<i>name</i> is null.
<b>System.ArgumentException</b>	<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 *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  
*args* has more than one element.

-or-

	<p><code>namedParameters.Length &gt; args.Length.</code></p> <p>-or-</p> <p>At least one element in <code>namedParameters</code> is <code>null</code>.</p> <p>-or-</p> <p>At least one element in <code>args</code> is not assignment-compatible with the corresponding parameter in <code>namedParameters</code>.</p>
<b>System.MissingFieldException</b>	A field or property matching the specified criteria was not found.
<b>System.MissingMethodException</b>	<p>A method matching the specified criteria cannot be found.</p> <p>-or-</p> <p>The current instance object represents a type that contains open type parameters (that is, <code>System.Type.ContainsGenericParameters</code> returns <code>true</code>).</p>
<b>System.MethodAccessException</b>	The requested member is non-public and the caller does not have the required permission.
<b>System.Reflection.TargetException</b>	The member matching the specified criteria cannot be invoked on <code>target</code> .
<b>System.Reflection.TargetInvocationException</b>	The member matching the specified criteria threw an exception.
<b>System.Reflection.AmbiguousMatchException</b>	More than one member matches the specified criteria.

1  
2  
3  
4  
5  
6

## Example

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.

```

1
2     [C#]

3 using System;
4 using System.Reflection;
5
6 class InvokeMemberExample
7 {
8     static void Main(string[] args)
9     {
10         // Create the parameter arrays that will
11         // be passed to InvokeMember.
12         char[] cAry =
13         new char[] {'A',' ','s','t','r','i','n','g'};
14         object[] oAry = new object[] {cAry, 0, cAry.Length};
15
16         Type t = typeof(string);
17
18         // Invoke the constructor of a string.
19         string str =
20             (string)t.InvokeMember(null, BindingFlags.Instance |
21             BindingFlags.Public | BindingFlags.CreateInstance, null,
22             null, oAry, null, null, null);
23         Console.WriteLine("The string is \"{0}\".", str);
24
25         // Access a property of the string.
26         int i =
27             (int) t.InvokeMember("Length", BindingFlags.Instance |
28             BindingFlags.Public | BindingFlags.GetProperty, null,
29             str, null, null, null, null);
30         Console.WriteLine("The length of the string is {0}.", i);
31
32         // Invoke a method on the string.
33         string newStr = "new ";
34         object[] oAry2 = new Object[] {2, newStr};
35         str = (string) t.InvokeMember("Insert", BindingFlags.Instance |
36             BindingFlags.Public | BindingFlags.InvokeMethod, null, str,
37             oAry2, null, null, null);
38         Console.WriteLine("The modified string is \"{0}\".", str);
39
40         // Access a field of the string.
41         str = (string) t.InvokeMember("Empty", BindingFlags.Static |
42             BindingFlags.Public | BindingFlags.GetField, null, str,
43             null);
44         Console.WriteLine("The empty string is \"{0}\".", str);
45
46     }
47 }

```

48 The output is

49  
50 The string is "A string".

51  
52  
53 The length of the string is 8.

54

1  
2 The modified string is "A new string"

3  
4  
5 The empty string is "".

6  
7 **Permissions**

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

8  
9

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

## 3 Type.IsArrayImpl() Method

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

### 8 Summary

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

### 12 Return Value

13 true if the `System.Type` is an array; otherwise, false.

### 14 Description

15 An instance of the `System.Array` class is required to return false because it is an  
16 object, not an array.

### 17 Behaviors

18 As described above.

19

20

# 1 Type.IsAssignableFrom(System.Type)

## 2 Method

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

### 8 Summary

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

### 11 Parameters

Parameter	Description
<code>c</code>	The <code>System.Type</code> to compare with the current <code>System.Type</code> .

### 12 Return Value

- 14 `false` if `c` is a null reference.  
15  
16 `true` if one or more of the following statements are true; otherwise `false`.
- 17 • If `c` and the current `System.Type` represent the same type.
  - 18 • If the current `System.Type` is in the inheritance hierarchy of `c`.
  - 19 • If the current `System.Type` is an interface and `c` supports that interface.
  - 20 • If `c` is a generic type parameter and the current instance represents one of the  
21 constraints of `c`.

### 22 Description

23 [Note: A generic type definition is not assignable from a closed constructed type.  
24 ]  
25 ]

### 26 Example

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

```

1
2     [C#]

3 using System;
4 class ArrayTypeTest {
5     public static void Main() {
6         int i = 1;
7         int [] array10 = new int [10];
8         int [] array2 = new int[2];
9         int [,]array22 = new int[2,2];
10        int [,]array24 = new int[2,4];
11        int [,,]array333 = new int[3,3,3];
12        Type array10Type = array10.GetType();
13        Type array2Type = array2.GetType();
14        Type array22Type = array22.GetType();
15        Type array24Type = array24.GetType();
16        Type array333Type = array333.GetType();
17
18        // If X and Y are not both arrays, then false
19        Console.WriteLine("int[2] is assignable from int? {0} ",
20 array2Type.IsAssignableFrom(i.GetType()));
21        // If X and Y have same type and rank, then true.
22        Console.WriteLine("int[2] is assignable from int[10]? {0} ",
23 array2Type.IsAssignableFrom(array10Type));
24        Console.WriteLine("int[2,2] is assignable from int[2,4]? {0}",
25 array22Type.IsAssignableFrom(array24Type));
26        Console.WriteLine("int[2,4] is assignable from int[2,2]? {0}",
27 array24Type.IsAssignableFrom(array22Type));
28        Console.WriteLine("");
29        // If X and Y do not have the same rank, then false.
30        Console.WriteLine("int[2,2] is assignable from int[10]? {0}",
31 array22Type.IsAssignableFrom(array10Type));
32        Console.WriteLine("int[2,2] is assignable from int[3,3,3]? {0}",
33 array22Type.IsAssignableFrom(array333Type));
34        Console.WriteLine("int[3,3,3] is assignable from int[2,2]? {0}",
35 array333Type.IsAssignableFrom(array22Type));
36    }
37 }
38 The output is
39
40 int[2] is assignable from int? False
41
42
43 int[2] is assignable from int[10]? True
44
45
46 int[2,2] is assignable from int[2,4]? True
47
48
49 int[2,4] is assignable from int[2,2]? True
50
51
52 int[2,2] is assignable from int[10]? False
53

```

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

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

## 3 **Type.IsByRefImpl()** Method

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

### 8 **Summary**

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

### 11 **Return Value**

12 `true` if the `System.Type` is passed by reference; otherwise, `false`.

### 13 **Behaviors**

14 As described above.

15

16

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

### 3 Type.IsCOMObjectImpl() Method

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

#### 8 Summary

9 Reserved.

#### 10 Return Value

11 false

#### 12 Description

13 This abstract method is required to be present for legacy implementations. Conforming  
14 implementations are permitted to throw the `System.NotSupportedException` as their  
15 implementation.

16

# 1 Type.IsInstanceOfType(System.Object)

## 2 Method

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

### 7 Summary

8 Determines whether the specified object is an instance of the current `System.Type`.

### 9 Parameters

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

10

### 11 Return Value

12 `true` if either of the following statements is true; otherwise `false`.

- 13 • If the current `System.Type` is in the inheritance hierarchy of `o`.
- 14 • If the current `System.Type` is an interface and `o` supports that interface.

15 If `o` is a null reference or if the current instance is an open generic type (that is,  
16 `System.Type.ContainsGenericParameters` returns `true`) returns `false`.

### 17 Description

18 As described above.

19

20 [*Note:* A constructed type is not an instance of its generic type definition.

21

22 ]

### 23 Behaviors

24 As described above.

25

### 26 Example

1 The following example demonstrates the System.Type.IsInstanceOfType method.

2

3 [C#]

```
4 using System;
5 public interface IFoo { }
6 public class MyClass: IFoo {}
7 public class MyDerivedClass: MyClass {}
8 class IsInstanceTest {
9     public static void Main() {
10         Type ifooType=typeof(IFoo);
11         MyClass mc = new MyClass();
12         Type mcType = mc.GetType();
13         MyClass mdc = new MyDerivedClass();
14         Type mdcType = mdc.GetType();
15         int [] array = new int [10];
16         Type arrayType = typeof(Array);
17         Console.WriteLine("int[] is instance of Array? {0}",
18 arrayType.IsInstanceOfType(array));
19         Console.WriteLine("myclass instance is instance of MyClass? {0}",
20 mcType.IsInstanceOfType(mc));
21         Console.WriteLine("myderivedclass instance is instance of MyClass? {0}",
22 mcType.IsInstanceOfType(mdc));
23         Console.WriteLine("myclass instance is instance of IFoo? {0}",
24 ifooType.IsInstanceOfType(mc));
25         Console.WriteLine("myderivedclass instance is instance of IFoo? {0}",
26 ifooType.IsInstanceOfType(mdc));
27     }
28 }
```

29 The output is

30

31 int[] is instance of Array? True

32

33

34 myclass instance is instance of MyClass? True

35

36

37 myderivedclass instance is instance of MyClass? True

38

39

40 myclass instance is instance of IFoo? True

41

42

43 myderivedclass instance is instance of IFoo? True

44

45

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

## 3 Type.IsPointerImpl() Method

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

### 8 Summary

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

### 11 Return Value

12 true if the `System.Type` is a pointer; otherwise, false.

### 13 Behaviors

14 As described above.

15

16

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

## 3 Type.IsPrimitiveImpl() Method

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

### 8 Summary

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

### 11 Return Value

12 true if the `System.Type` is one of the primitive types; otherwise, false.

### 13 Behaviors

14 This method returns true if the underlying type of the current instance is one of the  
15 following: `System.Boolean`, `System.Byte`, `System.SByte`, `System.Int16`,  
16 `System.UInt16`, `System.Int32`, `System.UInt32`, `System.Int64`, `System.UInt64`,  
17 `System.Char`, `System.Double`, and `System.Single`.

# 1 Type.IsSubclassOf(System.Type) Method

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

## 6 Summary

7 Determines whether the current `System.Type` derives from the specified `System.Type`.

## 8 Parameters

Parameter	Description
<code>c</code>	The <code>System.Type</code> to compare with the current <code>System.Type</code> .

## 9 Return Value

11 true if `c` and the current `System.Type` represent classes, and the class represented by  
12 the current `System.Type` derives from the class represented by `c`; otherwise false.  
13 Returns false if `c` and the current `System.Type` represent the same class.

## 14 Description

15 Interfaces are not considered.

16  
17 If the current instance represents an unassigned type parameter of a generic type or  
18 method, it derives from its class constraint, or from `System.Object` if it has no class  
19 constraint.

## 20 Example

21 The following example demonstrates the `System.Type.IsSubclassOf` method.

```
22 [C#]  
23  
24 using System;  
25 public interface IFoo { }  
26 public interface IBar:IFoo{}  
27 public class MyClass: IFoo {}  
28 public class MyDerivedClass: MyClass {}  
29 class IsSubclassTest {  
30     public static void Main() {  
31         Type ifooType = typeof(IFoo);  
32         Type ibarType = typeof(Bar);  
33         MyClass mc = new MyClass();  
34         Type mcType = mc.GetType();
```

```

1 MyClass mdc = new MyDerivedClass();
2 Type mdcType = mdc.GetType();
3 int [] array = new int [10];
4 Type arrayOfIntsType = array.GetType();
5 Type arrayType = typeof(Array);
6
7 Console.WriteLine("Array is subclass of int[]? {0}",
8 arrayType.IsSubclassOf(arrayOfIntsType));
9 Console.WriteLine("int [] is subclass of Array? {0}",
10 arrayOfIntsType.IsSubclassOf(arrayType));
11 Console.WriteLine("IFoo is subclass of IBar? {0}",
12 ifooType.IsSubclassOf(ibarType));
13 Console.WriteLine("myclass is subclass of MyClass? {0}",
14 mcType.IsSubclassOf(mcType));
15 Console.WriteLine("myderivedclass is subclass of MyClass? {0}",
16 mdcType.IsSubclassOf(mcType));
17 Console.WriteLine("IBar is subclass of IFoo? {0}",
18 ibarType.IsSubclassOf(ifooType));
19 }
20 }
21 The output is
22
23 Array is subclass of int[]? False
24
25
26 int [] is subclass of Array? True
27
28
29 IFoo is subclass of IBar? False
30
31
32 myclass is subclass of MyClass? False
33
34
35 myderivedclass is subclass of MyClass? True
36
37
38 IBar is subclass of IFoo? False
39
40

```

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

## 3 Type.MakeArrayType() Method

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

### 9 Summary

10 Returns a *System.Type* object representing a one-dimensional array type whose  
11 element type is the current type, with a lower bound of zero.

### 12 Return Value

13 A *System.Type* object representing a one-dimensional array type whose element type is  
14 the current type, with a lower bound of zero.

### 15 Description

16 This method provides a way to generate an array type with any possible element type,  
17 including generic types.

18

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

## 3 `Type.MakeArrayType(System.Int32)` Method

```
4 [ILAsm]  
5 .method public hidebysig virtual instance class System.Type  
6 MakeArrayType(int32 rank)  
  
7 [C#]  
8 public virtual Type MakeArrayType(int rank)
```

### 9 Summary

10 Returns a `System.Type` object representing an array of the current type, with the  
11 specified number of dimensions.

### 12 Parameters

Parameter	Description
<i>rank</i>	The number of dimensions for the array.

### 14 Return Value

15 A `System.Type` object representing an array of the current type, with the specified  
16 number of dimensions.

### 17 Description

18 This method provides a way to generate an array with any possible element type,  
19 including generic types.

### 20 Exceptions

Exception	Condition
<code>System.IndexOutOfRangeException</code>	<i>rank</i> is invalid (being less than 1, for example).

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

## 3 Type.MakeByRefType() Method

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

### 9 Summary

10 Returns a `System.Type` object that represents the current type when passed as a byref  
11 parameter.

### 12 Return Value

13 A `System.Type` object that represents the current type when passed as a byref  
14 parameter.

### 15 Description

16 This method provides a way to generate a byref type for any type.

17  
18 [*Note:* Using ilasm syntax, if the current `System.Type` object represents `int32`, this  
19 method returns a `System.Type` object representing `int32&`.]  
20  
21  
22

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

## Type.MakeGenericType(System.Type[]) Method

```
[ILAsm]  
.method public hidebysig virtual class System.Type MakeGenericType(class  
System.Type[] typeArguments)  
  
[C#]  
public override Type MakeGenericType(System.Type[] typeArguments)
```

### Summary

Substitutes the elements of an array of types for the type parameters of the current generic type definition, and returns a `System.Type` object representing the resulting constructed type.

The current type shall be a generic type definition.

### Parameters

Parameter	Description
<i>typeArguments</i>	An array of types to be substituted for the type parameters of the current generic type definition.

### Return Value

A `System.Type` representing the constructed type formed by substituting the elements of *typeArguments* for the type parameters of the current generic type definition.

### Description

This method allows you to write code that assigns specific types to the type parameters of a generic type definition, thus creating a `System.Type` object that represents a particular constructed type. You can use this `System.Type` object to create runtime instances of the constructed type.

The `System.Type` object returned by this method is the same as that obtained by calling the `System.Object.GetType` method of the resulting constructed type, or the `System.Object.GetType` method of any constructed type that was created from the same generic type using the same type arguments.

[Note: An array type whose element type is a generic type is not itself a generic type. Thus, you cannot call this method to bind an array type. To bind a type argument to this type, call `System.Type.GetElementType` to obtain the generic type, then this method to bind the type argument to the generic type, and, finally, `System.Type.MakeArrayType`

1 to create the array type.]

2  
3  
4

5 For a list of the invariant conditions for terms used in generic reflection, see the  
6 `System.Type.IsGenericType` property description.

## 7 Exceptions

Exception	Condition
<b>System.ArgumentException</b>	The number of elements in <i>typeArguments</i> is not the same as the number of type parameters of the current generic type definition.  -or-  An element of <i>typeArguments</i> does not satisfy the constraints specified for the corresponding type parameter of the current generic type definition.
<b>System.ArgumentNullException</b>	<i>typeArguments</i> is null.  -or-  An element of <i>typeArguments</i> is null.
<b>System.InvalidOperationException</b>	The current type does not represent the definition of a generic type. That is, <code>System.Type.IsGenericTypeDefinition</code> returns false.

8

## 9 Example

10 The following example uses `System.Type.GetType` and `System.Type.MakeGenericType`  
11 to create a constructed type from the generic  
12 `System.Collections.Generic.Dictionary` type. The constructed type represents a  
13 `System.Collections.Generic.Dictionary` of Test objects with string keys.

14  
15

```
16 using System;  
17 using System.Reflection;  
18 using System.Collections.Generic;  
19  
20 public class Test  
21 {
```

```

1      public static void Main()
2      {
3          Console.WriteLine("\n--- Create a constructed type from the
4 generic Dictionary type.");
5
6          // Create a type object representing the generic Dictionary
7          // type.
8          Type generic =
9 Type.GetType("System.Collections.Generic.Dictionary");
10
11         DisplayTypeInfo(generic);
12
13         // Create an array of types to substitute for the type
14         // parameters of Dictionary. The key is of type string, and
15         // the type to be contained in the Dictionary is Test.
16         Type[] typeArgs = { typeof(string), typeof(Test) };
17         Type constructed = generic.MakeGenericType(typeArgs);
18
19         DisplayTypeInfo(constructed);
20
21         // Compare the type objects obtained above to type objects
22         // obtained using typeof() and GetGenericTypeDefinition().
23         Console.WriteLine("\n--- Compare types obtained by different
24 methods:");
25
26         Type t = typeof(Dictionary<string, Test>);
27
28         Console.WriteLine("\tAre the constructed types equal? {0}", t
29 == constructed);
30         Console.WriteLine("\tAre the generic types equal? {0}",
31 t.GetGenericTypeDefinition() == generic);
32     }
33
34     private static void DisplayTypeInfo(Type t)
35     {
36         Console.WriteLine("\n{0}", t);
37         Console.WriteLine("\tIs this a generic type definition? {0}",
38 t.IsGenericTypeDefinition);
39         Console.WriteLine("\tDoes it have generic type arguments?
40 {0}", t.HasGenericArguments);
41
42         Type[] typeArguments = t.GetGenericArguments();
43
44         Console.WriteLine("\tList type arguments ({0}):",
45 typeArguments.Length);
46         foreach (Type tParam in typeArguments)
47         {
48             Console.WriteLine("\t\t{0}", tParam);
49         }
50     }
51 }
52
53 /* This example produces the following output:
54 --- Create a constructed type from the generic Dictionary type.
55 System.Collections.Generic.Dictionary[KeyType,ValueType]

```

```
1         Is this a generic type definition? True
2         Does it have generic type arguments? True
3         List type arguments (2):
4             K
5             V
6
7     System.Collections.Generic.Dictionary[System.String, Test]
8         Is this a generic type definition? False
9         Does it have generic type arguments? True
10        List type arguments (2):
11            System.String
12            Test
13
14    --- Compare types obtained by different methods:
15        Are the constructed types equal? True
16        Are the generic types equal? True
17    */
18
```

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

## 3 **Type.MakePointerType() Method**

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

### 9 **Summary**

10 Returns a `System.Type` object that represents the type of an unmanaged pointer to the  
11 current type.

### 12 **Return Value**

13 A `System.Type` object that represents the type of an unmanaged pointer to the current  
14 type.

### 15 **Description**

16 This method provides a way to generate an unmanaged pointer type for types computed  
17 at runtime.

18  
19 [*Note:* Using `ilasm` syntax, if the current `System.Type` object represents `int32`, this  
20 method returns a `System.Type` object representing `int32*`.]  
21  
22

23

# 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 Returns `System.Type.FullName`.

## 10 Description

11 [*Note:* This method overrides `System.Object.ToString`.]  
12  
13  
14

15 If the current instance represents a generic type, the type and its type arguments are  
16 qualified by namespace and by nested type, but not by assembly. If the current instance  
17 represents an unassigned type parameter of a generic type or method, this method  
18 returns the unqualified name of the type parameter.

19

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 { public hidebysig  
6 virtual abstract specialname class System.Reflection.Assembly  
7 get_Assembly() }  
8 [C#]  
9 public abstract Assembly Assembly { get; }
```

### 10 Summary

11 Gets the `System.Reflection.Assembly` in which the type is declared. For generic types,  
12 gets the `System.Reflection.Assembly` that contains the generic type definition.

### 13 Property Value

14 A `System.Reflection.Assembly` instance that describes the assembly containing the  
15 current type. For generic types, the instance describes the assembly that contains the  
16 definition of the generic type.

### 17 Description

18 If the current instance represents a generic type, this property returns the assembly in  
19 which the type was defined. For example, suppose you create an assembly named  
20 `MyGenerics.dll` that contains a class named `MyGenericStack<T>`. If you create an  
21 instance of `MyGenericStack<int>` in another assembly, the `System.Type.Assembly`  
22 property for the constructed type returns a `System.Reflection.Assembly` that  
23 represents `MyGenerics.dll`.

24  
25 Similarly, if the current instance represents a generic parameter `T`, this property returns  
26 the assembly that contains the generic type definition that defines `T`.

### 27 Behaviors

28 This property is read-only.

29

30

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

## Type.AssemblyQualifiedName Property

```
[ILAsm]
.property string AssemblyQualifiedName { public hidebysig virtual abstract
specialname string get_AssemblyQualifiedName() }

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

### Summary

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

### Property Value

The assembly-qualified name of the `System.Type`, including the name of the assembly from which the `System.Type` was loaded. If the current `System.Type` object represents a generic parameter, this property returns `null`.

### Behaviors

This property is read-only.

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

Delimiter	Meaning
Backslash (\)	Escape character.
Comma (,)	Precedes the Assembly name.
Plus sign (+)	Precedes a nested class.
Period (.)	Denotes namespace identifiers.
Square brackets ([])	After a type name, denotes an array of that type. -or- For a generic type, encloses the entire generic type argument list. -or-

	Within a type argument list, encloses an assembly-qualified type.
--	---

1  
2 [Note: For example, the fully qualified name for a class might look like this:

3  
4 TopNamespace.SubNameSpace.ContainingClass+NestedClass,MyAssembly

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

9  
10 TopNamespace.Sub\+Namespace.ContainingClass+NestedClass,MyAssembly

11  
12 A "++" becomes "\+\"+", and a "\" becomes "\\\".

13  
14 ]

15  
16 Type names are permitted to include trailing characters that denote additional  
17 information about the type, such as whether the type is a reference type, a pointer type  
18 or an array type. To retrieve the type name without these trailing characters, use  
19 `t.GetElementType().ToString()`, where *t* is the type.

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

24  
25 Generic arguments of generic types are themselves fully qualified. For example, the  
26 output from the following C# program, if compiled to an assembly called Try64

```
27 using System;  
28 using System.Reflection;  
29  
30 class MyTest {  
31     public static void Main(String[] args) {  
32         Type b = typeof(B<string,object>);  
33         Console.WriteLine(b.AssemblyQualifiedName);  
34     }  
35 }  
36 public class B<T,U> { }  
37 is as follows:
```

```
38 B`2[[System.String, mscorlib, Version=2.0.3600.0, Culture=neutral,  
39 PublicKeyToken=b77a5c561934e089],[System.Object, mscorlib,  
40 Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b77a5c561934e089]],  
41 Try64, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null
```

## 42 Usage

43 The name returned by this method can be persisted and later used to load the  
44 `System.Type`. To search for and load a `System.Type`, use `System.Type.GetType` either  
45 with the type name only or with the assembly qualified type name.

1 `System.Type.GetType` with the type name only will look for the `System.Type` in the  
2 caller's assembly and then in the `System` assembly. `System.Type.GetType` with the  
3 assembly qualified type name will look for the `System.Type` in any assembly.

4

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

## 3 Type.Attributes Property

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

### 10 Summary

11 Gets the attributes associated with the type represented by the current instance.

### 12 Property Value

13 A System.Reflection.TypeAttributes object representing the attribute set of the  
14 System.Type.

### 15 Description

16 This property is read-only.

17  
18 If the current instance represents a generic type, this property returns the attributes of  
19 the generic type definition.

20  
21 If the current instance represents a generic type parameter T, the  
22 System.Reflection.TypeAttributes returned by this property includes  
23 System.Reflection.TypeAttributes.Abstract,  
24 System.Reflection.TypeAttributes.AnsiClass,  
25 System.Reflection.TypeAttributes.AutoLayout,  
26 System.Reflection.TypeAttributes.Class,  
27 System.Reflection.TypeAttributes.Public, and  
28 System.Reflection.TypeAttributes.Sealed. These are arbitrary choices which have  
29 no meaning in the context of a type parameter.

30

# 1 Type.BaseType Property

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

## 7 Summary

8 Gets the base `System.Type` of the current `System.Type`.

## 9 Property Value

10 A `System.Type` object representing the type from which the current `System.Type`  
11 directly inherits, or `null` if the current `System.Type` represents the `System.Object`  
12 class.

## 13 Description

14 The base type is the type from which the current type directly inherits. `System.Object`  
15 is the only type that does not have a base type, therefore `null` is returned as the base  
16 type of `System.Object`.

17  
18 Interfaces inherit from `System.Object` and from zero or more base interfaces;  
19 therefore, the base type of an interface is considered to be `System.Object`.

20  
21 If the current instance represents a constructed generic type, the base type reflects the  
22 generic arguments.

23  
24 If the current instance represents an unassigned type parameter,  
25 `System.Type.BaseType` returns the base class type constraint declared for that  
26 parameter, or `System.Object` if no base class type constraint was declared.

## 27 Behaviors

28 This property is read-only.

29

## 30 Example

31 The following example demonstrates using the `System.Type.BaseType` property.

```
32 [C#]
33
34 using System;
35 class TestType {
36     public static void Main() {
37         Type t = typeof(int);
```

```
1 Console.WriteLine("{0} inherits from {1}", t,t.BaseType);
2 }
3 }
4 The output is
5
6 System.Int32 inherits from System.ValueType
7
```

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

## Type.ContainsGenericParameters Property

```
[ILAsm]
.property bool ContainsGenericParameters { public hidebysig virtual
specialname bool get_ContainsGenericParameters() }

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

### Summary

Gets a value that indicates whether a `System.Type` object contains unassigned generic parameters.

### Property Value

true if a `System.Type` object contains unassigned generic parameters; otherwise false.

### Description

In order to create an instance of a generic type, there must be no generic type definitions or open constructed types in the type arguments. For other constructed types, such as arrays and managed pointers, the types from which they are constructed must be able to be instantiated. If the `System.Type.ContainsGenericParameters` property returns `true`, the type cannot be instantiated.

The `System.Type.ContainsGenericParameters` property searches recursively for type parameters. For example, it returns `true` for an array whose element type is `A<T>`, even though the array type itself is not generic. Contrast this with the behavior of the `System.Type.IsGenericType` property, which returns `false` for arrays.

For a set of example classes and a table showing the values of the `System.Type.ContainsGenericParameters` property, see the `System.Type.IsGenericType` property description.

### Behaviors

This property is read-only.

### Example

For an example of using this method, see the example for `System.Type.GenericParameterPosition`.

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

## 3 Type.DeclaringMethod Property

```
4 [ILAsm]  
5 .property System.Reflection.MethodBase DeclaringMethod { public hidebysig  
6 virtual specialname System.Reflection.MethodBase get_DeclaringMethod() }  
7 [C#]  
8 public virtual MethodBase DeclaringMethod { get; }
```

### 9 Summary

10 If the current `System.Type` represents a type parameter of a generic method, gets a  
11 `System.Reflection.MethodInfo` that represents the declaring method.

### 12 Property Value

13 If the current `System.Type` represents a type parameter of a generic method, a  
14 `System.Reflection.MethodInfo` that represents the declaring method; otherwise `null`.

### 15 Description

16 The declaring method is a generic method definition. That is, if  
17 `System.Type.DeclaringMethod` does not return `null`, then  
18 `DeclaringMethod.IsGenericMethodDefinition` returns `true`.

19  
20 The `System.Type.DeclaringType` and `System.Type.DeclaringMethod` properties  
21 identify the generic type definition or generic method definition where the generic type  
22 parameter was originally defined:

- 23 • If the `System.Type.DeclaringMethod` property returns a  
24 `System.Reflection.MethodBase`, that `System.Reflection.MethodBase` represents  
25 a generic method definition, and the current `System.Type` object represents a type  
26 parameter of that generic method definition.
- 27 • If the `System.Type.DeclaringMethod` property returns a `null`, then the  
28 `System.Type.DeclaringType` property always returns a `System.Type` object  
29 representing a generic type definition, and the current `System.Type` object  
30 represents a type parameter of that generic type definition.

31 For a list of the invariant conditions for terms used in generic reflection, see the  
32 `System.Type.IsGenericType` property description.

### 33 Behaviors

34 This property is read-only.

35

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

## Type.DeclaringType Property

```
[ILAsm]
.property class System.Type DeclaringType { public hidebysig virtual
specialname class System.Type get_DeclaringType() }

[C#]
public virtual Type DeclaringType { get; }
```

### Summary

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

### Property Value

The `System.Type` object for the class that declares the type represented by the current instance. If the type is a nested type, this property returns the enclosing type; otherwise, returns the current instance.

### Description

[*Note:* This property implements the abstract property inherited from `System.Reflection.MemberInfo`.]

If the current `System.Type` represents a type parameter of a generic type or method definition, the `System.Type.DeclaringType` and `System.Type.DeclaringMethod` properties identify the generic type definition or generic method definition where the generic type parameter was originally defined:

- If the `System.Type.DeclaringMethod` property returns a `System.Reflection.MethodBase`, that `System.Reflection.MethodBase` represents a generic method definition, and the current `System.Type` object represents a type parameter of that generic method definition.
- If the `System.Type.DeclaringMethod` property returns a null, then the `System.Type.DeclaringType` property always returns a `System.Type` object representing a generic type definition, and the current `System.Type` object represents a type parameter of that generic type definition.

For a type parameter of a generic method, this property returns the type that contains the generic method definition.

**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 can 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 abstract specialname
4 string get_FullName() }

5 [C#]
6 public abstract string FullName { get; }
```

## 7 Summary

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

## 9 Property Value

10 A `System.String` containing the fully qualified name of the `System.Type`.

## 11 Description

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

13  
14  
15  
16 If the current instance represents a generic type, the type arguments in the string  
17 returned by `System.Type.FullName` are qualified by their assembly, version, and so on,  
18 even though the string representation of the generic type itself is not qualified by  
19 assembly. Thus, `t.FullName + ", " + t.Assembly.FullName` produces the same  
20 result as `t.AssemblyQualifiedName`, as with types that are not generic.

21  
22 If the current instance represents an unassigned type parameter of a generic type, this  
23 property returns `null`.

## 24 Behaviors

25 This property is read-only.

26

## 27 Example

28 The following example demonstrates using the `System.Type.FullName` property.

```
29 [C#]
30
31 using System;
32 class TestType {
33     public static void Main() {
34         Type t = typeof(Array);
35         Console.WriteLine("Full name of Array type is {0}",t.FullName);
36     }
37 }
```

- 1 The output is
- 2
- 3 Full name of Array type is System.Array
- 4

# 1 Type.GenericParameterAttributes Property

```
2 [ILAsm]
3 .property valuetype System.Reflection.GenericParameterAttributes
4 GenericParameterAttributes { public hidebysig specialname virtual instance
5 valuetype System.Reflection.GenericParameterAttributes
6 get_GenericParameterAttributes () }
7
8 [C#]
9 public virtual GenericParameterAttributes GenericParameterAttributes {
  get; }
```

## 10 Summary

11 Gets a combination of `System.Reflection.GenericParameterAttributes` flags that  
12 describe the variance and special constraints of the current generic type parameter.

## 13 Property Value

14 A `System.Reflection.GenericParameterAttributes` value that describes the variance  
15 and special constraints of the current generic type parameter.

## 16 Description

17 This property is read-only.

18  
19 The value of this property contains flags that describe whether the current generic type  
20 parameter is variant, and flags that describe any special constraints. Use  
21 `System.GenericParameterAttributes.VarianceMask` to select the variance flags, and  
22 `System.GenericParameterAttributes.SpecialConstraintMask` to select the  
23 constraint flags. Use `System.Reflection.GetGenericParameterConstraints` to get the  
24 type constraints.

25  
26 For a list of the invariant conditions for terms used in generic reflection, see the  
27 `System.Type.IsGenericType` property description.

## 28 Exceptions

Exception	Condition
<b>System.InvalidOperationException</b>	The current <code>System.Type</code> object is not a generic type parameter. That is, the <code>System.Type.IsGenericParameter</code> property returns <code>false</code> .

29

30

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

## Type.GenericParameterPosition Property

```
[ILAsm]
.property int GenericParameterPosition { public hidebysig virtual
specialname int get_GenericParameterPosition() }

[C#]
public virtual int GenericParameterPosition { get; }
```

### Summary

For a `System.Type` object that represents a type parameter of a generic type or generic method, gets the position of the type parameter in the type parameter list of the generic type or generic method.

### Property Value

A zero-based integer representing the position of a type parameter in the type parameter list of the generic type or generic method that declared the parameter.

### Description

This read-only property returns the position of a type parameter in the parameter list of the generic type definition or generic method definition where the type parameter was originally defined. The `System.Type.DeclaringType` and `System.Type.DeclaringMethod` properties identify the generic type or method definition:

- If the `System.Type.DeclaringMethod` property returns a `System.Reflection.MethodBase`, that `System.Reflection.MethodBase` represents a generic method definition, and the current `System.Type` object represents a type parameter of that generic method definition.
- If the `System.Type.DeclaringMethod` property returns a null, then the `System.Type.DeclaringType` property always returns a `System.Type` object representing a generic type definition, and the current `System.Type` object represents a type parameter of that generic type definition.

To provide the correct context for the value of the `System.Type.GenericParameterPosition` property, it is necessary to identify the generic type or method a type parameter belongs to. For example, consider the return value of the generic method `GetSomething` in the following C# code:

```
public class B<T, U> { }
public class A<V>
{
    public B<V, X> GetSomething<X>()
    {
```

```

1         return new Base<V, X>();
2     }
3 }

```

4 The type returned by `GetSomething` depends on the type arguments supplied to class `A` and `GetSomething` itself. You can obtain a `System.Reflection.MethodInfo` for `GetSomething` and from that you can obtain the return type. When you examine the type parameters of the return type, `System.Type.GenericParameterPosition` returns zero for both. The position of `v` is zero because `v` is the first type parameter in the type parameter list for class `A`. The position of `x` is zero because `x` is the first type parameter in the type parameter list for `GetSomething`.

11  
12 *[Note: Calling the `System.Type.GenericParameterPosition` property causes an exception if the current `System.Type` does not represent a type parameter. When you examine the type arguments of an open constructed type, use the `System.Type.IsGenericParameter` property to tell which are type parameters and which are types. The `System.Type.IsGenericParameter` property returns `true` for a type parameter; you can then use the `System.Type.GenericParameterPosition` method to obtain its position, and the `System.Type.DeclaringMethod` and `System.Type.DeclaringType` properties to determine the generic method or type definition that defines it.*

```
21 ]
```

22 **Exceptions**

Exception	Condition
<b>System.InvalidOperationException</b>	The current type does not represent a type parameter. That is, <code>System.Type.IsGenericParameter</code> returns <code>false</code> .

23

24 **Example**

25 The following example defines a generic class with two type parameters, and a generic class that derives from it. The base class of the derived type has one unbound type parameter and one type parameter bound to `System.Int32`. The example displays information about these generic classes, including the positions reported by `System.Type.GenericParameterPosition`.

30  
31

```
31 [C#]
```

```

32 using System;
33 using System.Reflection;
34 using System.Collections.Generic;
35 // Define a base class with two type parameters.
36 public class Base<T, U> { }
37
38 // Define a derived class. The derived class inherits from a constructed
39 // class that meets the following criteria:

```

```

1 // (1) Its generic type definition is Base<T, U>.
2 // (2) It specifies int for the first type parameter.
3 // (3) For the second type parameter, it uses the same type that is used
4 //     for the type parameter of the derived class.
5 // Thus, the derived class is a generic type with one type parameter, but
6 // its base class is an open constructed type with one type argument and
7 // one type parameter.
8 public class Derived<V>: Base<int,V> { }
9
10 public class Test
11 {
12     public static void Main()
13     {
14         Console.WriteLine("\n--- Display a generic type and the open
15 constructed");
16         Console.WriteLine("    type from which it is derived.");
17
18         // Create a Type object representing the generic type Derived.
19         //
20         Type derivedType = Type.GetType("Derived");
21
22         DisplayGenericTypeInfo(derivedType);
23
24         // Display its open constructed base type.
25         DisplayGenericTypeInfo(derivedType.BaseType);
26     }
27
28     private static void DisplayGenericTypeInfo(Type t)
29     {
30         Console.WriteLine("\n{0}", t);
31         Console.WriteLine("\tIs this a generic type definition? {0}",
32 t.IsGenericTypeDefinition);
33         Console.WriteLine("\tDoes it have generic arguments? {0}",
34 t.HasGenericArguments);
35         Console.WriteLine("\tDoes it have unbound generic parameters?
36 {0}", t.ContainsGenericParameters);
37         if (t.HasGenericArguments)
38         {
39             // If the type is a generic type definition or a
40             // constructed type, display the type arguments.
41             //
42             Type[] typeArguments = t.GetGenericArguments();
43
44             Console.WriteLine("\tList type arguments ({0}):",
45 typeArguments.Length);
46             foreach (Type tParam in typeArguments)
47             {
48                 // IsGenericParameter is true only for generic
49                 type
50                 // parameters.
51                 //
52                 if (tParam.IsGenericParameter)
53                 {
54                     Console.WriteLine("\t\t{0} (unbound -
55 parameter position {1})", tParam, tParam.GenericParameterPosition);
56                 }
57                 else

```

```

1           {
2           Console.WriteLine("\t\t{0}", tParam);
3           }
4       }
5   }
6   else
7   {
8       Console.WriteLine("\tThis is not a generic or
9 constructed type.");
10  }
11 }
12 }
13
14 /* This example produces the following output:
15
16 --- Display a generic type and the open constructed
17     type from which it is derived.
18
19 Derived[V]
20     Is this a generic type definition? True
21     Does it have generic arguments? True
22     Does it have unbound generic parameters? True
23     List type arguments (1):
24         V (unbound - parameter position 0)
25
26 Base[System.Int32, V]
27     Is this a generic type definition? False
28     Does it have generic arguments? True
29     Does it have unbound generic parameters? True
30     List type arguments (2):
31         System.Int32
32         V (unbound - parameter position 0)
33 */
34

```

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

## 3 Type.HasElementType Property

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

### 9 Summary

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

### 13 Property Value

14 true if the `System.Type` is an array, a pointer, or is passed by reference; otherwise,  
15 false.

### 16 Description

17 This property is read-only.

18  
19 *[Note: For example, `System.Type.GetType("System.Int32 []").HasElementType`  
20 returns true, but `System.Type.GetType("System.Int32 ").HasElementType` returns  
21 false. `System.Type.HasElementType` also returns true for "Int32\*" and "Int32&".]*

22  
23  
24  
25 If the current instance represents a generic type, or a type parameter of a generic type  
26 or method, this property returns false.

27

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

## 3 Type.IsAbstract Property

```
4 [ILAsm]  
5 .property bool IsAbstract { public hidebysig specialname instance bool  
6 get_IsAbstract() }
```

```
7 [C#]  
8 public bool IsAbstract { get; }
```

### 9 Summary

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

### 12 Property Value

13 `true` if the `System.Type` is abstract; otherwise, `false`.

### 14 Description

15 This property is read-only.

16  
17 If the current instance represents an unassigned type parameter of a generic type, this  
18 property always returns `true`. This is because it is not possible to create an instance of a  
19 generic type parameter.

20

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

## 3 Type.IsAnsiClass Property

```
4 [ILAsm]  
5 .property bool IsAnsiClass { public hidebysig specialname virtual instance  
6 bool get_IsAnsiClass() }  
  
7 [C#]  
8 public virtual bool IsAnsiClass { get; }
```

### 9 Summary

10 Indicates whether the type attribute `System.Reflection.TypeAttributes.AnsiClass`  
11 is selected for the current type.

### 12 Property Value

13 `true` if the type attribute `System.Reflection.TypeAttributes.AnsiClass` is selected for  
14 the current type; otherwise, `false`.

### 15 Description

16 This property is read-only.

17  
18 If the current `System.Type` represents a generic type, this property applies to the  
19 definition of the type. If the current `System.Type` represents a type parameter of a  
20 generic type or method, this property always returns `false`.

21

# 1 Type.IsArray Property

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

## 7 Summary

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

## 10 Property Value

11 true if the current `System.Type` represents an array; otherwise false.

## 12 Description

13 This property is read-only.

14  
15 This property returns `true` for an array of objects, but not for the `System.Array` type  
16 itself, which is a class.

17  
18 If the current instance represents a generic type, or a type parameter of a generic type  
19 or method, this property returns `false`.

## 20 Example

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

```
22 [C#]
23
24 using System;
25 class TestType {
26     public static void Main() {
27         int [] array = {1,2,3,4};
28         Type at = typeof(Array);
29         Type t = array.GetType();
30         Console.WriteLine("Type is {0}. IsArray? {1}", at, at.IsArray);
31         Console.WriteLine("Type is {0}. IsArray? {1}", t, t.IsArray);
32     }
33 }
```

34 The output is

```
35
36 Type is System.Array. IsArray? False
37
38
39 Type is System.Int32[]. IsArray? True
40
```



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

## 3 Type.IsAutoClass Property

```
4 [ILAsm]  
5 .property bool IsAutoClass { public hidebysig specialname virtual instance  
6 bool get_IsAutoClass() }  
  
7 [C#]  
8 public virtual bool IsAutoClass { get; }
```

### 9 Summary

10 Indicates whether the type attribute `System.Reflection.TypeAttributes.AutoClass`  
11 is selected for the current type.

### 12 Property Value

13 `true` if the type attribute `System.Reflection.TypeAttributes.AutoClass` is selected for  
14 the current type; otherwise, `false`.

### 15 Description

16 This property is read-only.

17  
18 If the current `System.Type` represents a generic type, this property applies to the  
19 definition of the type. If the current `System.Type` represents a type parameter of a  
20 generic type or method, this property always returns `false`.

21

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

## 3 Type.IsAutoLayout Property

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

### 9 Summary

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

### 12 Property Value

13 true if the type layout attribute System.Reflection.TypeAttributes.AutoLayout is  
14 specified for the current System.Type; otherwise, false.

### 15 Description

16 This property is read-only.

17  
18 If the current instance represents a generic type, this property applies to the definition  
19 of the type. If the current instance represents an unassigned type parameter of a  
20 generic type or method, this property always returns false.

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

26  
27 ]

28

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

### 3 **Type.IsByRef Property**

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

#### 9 **Summary**

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

#### 12 **Property Value**

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

#### 14 **Description**

15 This property is read-only.

16

# 1 Type.IsClass Property

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

## 7 Summary

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

## 10 Property Value

11 `true` if the current `System.Type` represents a class; otherwise `false`.

## 12 Description

13 This property is read-only.

14

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

17

18 If the current instance represents a generic type, this property returns `true` if the  
19 generic type definition is a class definition (that is, it does not define an interface or a  
20 value type).

21

22 If the current instance represents an unassigned type parameter of a generic type or  
23 method, this property always returns `false`.

24

# 1 Type.IsEnum Property

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

## 7 Summary

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

## 10 Property Value

11 `true` if the current `System.Type` represents an enumeration; otherwise `false`.

## 12 Description

13 This property is read-only.

14  
15 This property returns `true` for an enumeration, but not for the `System.Enum` type itself,  
16 which is a class.

17  
18 If the current instance represents a generic type, this property applies to the definition  
19 of the type. If the current instance represents an unassigned type parameter of a  
20 generic type or method, this property always returns `false`.

## 21 Example

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

```
23
24 [C#]
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}", colorType.IsEnum);
34 Console.WriteLine("Color is valueType? {0}", colorType.IsValueType);
35 Console.WriteLine("Enum is enum Type? {0}", enumType.IsEnum);
36 Console.WriteLine("Enum is value? {0}", enumType.IsValueType);
37 }
38 }
```

39 The output is

```
40
41 Color is enum ? True
```

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

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

## 3 Type.IsExplicitLayout Property

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

### 9 Summary

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

### 13 Property Value

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

### 16 Description

17 This property is read-only.

18  
19 [*Note:* Types marked with the System.Reflection.TypeAttributes.ExplicitLayout  
20 attribute cause the system to ignore field sequence and to use the explicit layout rules  
21 provided, in the form of field offsets, overall class size and alignment.

22 ]

23  
24 If the current instance represents a generic type, this property applies to the definition  
25 of the type. If the current instance represents an unassigned type parameter of a  
26 generic type or method, this property always returns false.

28

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

## 3 Type.IsGenericParameter Property

```
4 [ILAsm]  
5 .property bool IsGenericParameter { public virtual hidebysig specialname  
6 instance bool get_IsGenericParameter() }  
  
7 [C#]  
8 public virtual bool IsGenericParameter { get; }
```

### 9 Summary

10 Gets a value that indicates whether the current type represents a type parameter of a  
11 generic type or method.

### 12 Property Value

13 true if the current object represents a type parameter of a generic type or method; otherwise  
14 false.

### 15 Description

16 This property is read-only.

17  
18 Use this property to distinguish between type parameters and type arguments. When  
19 you call `System.Type.GetGenericArguments` to obtain the type arguments of a generic  
20 type, some elements of the array might be specific types (type arguments) and others  
21 might be type parameters. `System.Type.IsGenericParameter` returns false for the  
22 types and true for the type parameters.

23  
24 For a list of the invariant conditions for terms used in generic reflection, see the  
25 `System.Type.IsGenericType` property description.

### 26 Example

27 For an example of using this method, see the example for  
28 `System.Type.GenericParameterPosition`.

29

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

## Type.IsGenericType Property

```
[ILAsm]
.property bool IsGenericType { public hidebysig virtual specialname bool
get_IsGenericType() }

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

### Summary

Gets a value that indicates whether the current type has type arguments, and is therefore a generic type.

### Property Value

true if the current type has type arguments; otherwise false.

### Description

Use this property to determine whether a `System.Type` object represents a generic type. Use the `System.Type.ContainsGenericParameters` property to determine whether a `System.Type` object represents an open constructed type or a closed constructed type.

[*Note:* The `System.Type.HasGenericArguments` property returns false if the immediate type is not generic.]

The following table summarizes the invariant conditions for common terms used in generic reflection.

Term	Invariant
generic type definition	<p>The <code>System.Type.IsGenericTypeDefinition</code> property is true.</p> <p>Defines a generic type. A constructed type is created by calling the <code>System.Type.MakeGenericType(System.Type[])</code> method on a <code>System.Type</code> object that represents a generic type definition, and specifying an array of type arguments.</p> <p><code>System.Type.MakeGenericType(System.Type[])</code> can be called only on generic type definitions.</p> <p>Any generic type definition is a generic type, but the converse is not true.</p>

generic type	<p>The <code>System.Type.IsGenericType</code> property is true.</p> <p>Can be a generic type definition, an open constructed type, or a closed constructed type.</p> <p>Note that an array type whose element type is generic is not itself a generic type. The same is true of a <code>System.Type</code> object representing a pointer to a generic type.</p>
open constructed type	<p>The <code>System.Type.ContainsGenericParameters</code> property is true.</p> <p>It is not possible to create an instance of an open constructed type.</p> <p>Note that not all open constructed types are generic, such as an array type whose element type is a generic type definition.</p>
closed constructed type	<p>The <code>System.Type.ContainsGenericParameters</code> property is false.</p> <p>When examined recursively, the type has no unassigned generic parameters. The containing type or method has no generic type parameters, and, recursively, no type arguments have unassigned generic type parameters.</p>
generic type parameter	<p>The <code>System.Type.IsGenericParameter</code> property is true.</p> <p>In a generic type definition, a placeholder for a type that will be assigned later.</p>
generic type argument	<p>Can be any type, including a generic type parameter.</p> <p>Type arguments are specified as an array of <code>System.Type</code> objects passed to the <code>System.Type.MakeGenericType(System.Type[])</code> method when creating a constructed generic type. If instances of the resulting type are to be created, the <code>System.Type.ContainsGenericParameters</code> property must be false for all the type arguments.</p>

1  
2

3 **Behaviors**

4     This property is read-only.

5 **Example**

1 For an example of using this method, see the example for  
2 `System.Type.MakeGenericType`.

3

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

## 3 **Type.IsGenericTypeDefinition** Property

```
4 [ILAsm]  
5 .property virtual bool IsGenericTypeDefinition { public hideby sig  
6 specialname instance bool get_IsGenericTypeDefinition() }  
7 [C#]  
8 public virtual bool IsGenericTypeDefinition { get; }
```

### 9 **Summary**

10 Gets a value that indicates whether the current object represents the definition of a  
11 generic type, or whether one or more of its type parameters has been specified.

### 12 **Property Value**

13 `true` if the current object represents the definition of a generic type, none of whose type  
14 parameters have been bound to specific types; otherwise `false`.

### 15 **Description**

16 This property is read-only.

17  
18 Use this property to determine whether type arguments have been specified for any of  
19 the type parameters of a generic type. If type arguments have been specified (that is,  
20 bound to the corresponding type parameters), this property returns `false`.

21  
22 For a list of the invariant conditions for terms used in generic reflection, see the  
23 `System.Type.IsGenericType` property description.

24  
25 [*Note:* An open generic type can have type parameters even if types have been  
26 specified for its type parameters.

27  
28 ]

### 29 **Example**

30 For an example of using this method, see the example for  
31 `System.Type.MakeGenericType..`

32

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

## 3 Type.IsImport Property

```
4 [ILAsm]  
5 .property bool IsImport { public hidebysig specialname instance bool  
6 get_IsImport() }
```

```
7 [C#]  
8 public bool IsImport { get; }
```

### 9 Summary

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

### 12 Property Value

13 `true` if the `System.Type` was imported from another class; otherwise, `false`.

### 14 Description

15 This property is read-only.

16  
17 If the current instance represents a generic type, this property applies to the definition  
18 of the type. If the current instance represents an unassigned type parameter of a  
19 generic type or method, this property always returns `false`.

20

# 1 Type.IsInterface Property

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

## 7 Summary

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

## 10 Property Value

11 `true` if the current `System.Type` represents an interface; otherwise `false`.

## 12 Description

13 This property is read-only.

14  
15 If the current instance represents an unassigned type parameter of a generic type or  
16 method, this property always returns `false`.

17

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

## 3 Type.IsLayoutSequential Property

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

### 9 Summary

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

### 13 Property Value

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

### 17 Description

18 This property is read-only.

19  
20 [Note: The System.Reflection.TypeAttributes.SequentialLayout attribute is used  
21 to indicate that the system is to preserve field order as emitted, but otherwise the  
22 specific offsets are calculated based on the type of the field; these might be shifted by  
23 explicit offset, padding, or alignment information.

24  
25 ]

26  
27 If the current instance represents a generic type, this property applies to the definition  
28 of the type. If the current instance represents an unassigned type parameter of a  
29 generic type or method, this property always returns false.

30

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

## 3 Type.IsMarshalByRef Property

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

### 9 Summary

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

### 12 Property Value

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

### 14 Description

15 This property is read-only.

16

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

## 3 Type.IsNestedAssembly Property

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

### 9 Summary

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

### 12 Property Value

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

### 15 Description

16 This property is read-only.

17  
18 If the current instance represents an unassigned type parameter of a generic type, this  
19 property returns `false`.

20

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

## 3 Type.IsNestedFamANDAssem Property

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

### 9 Summary

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

### 12 Property Value

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

### 15 Description

16 This property is read-only.

17  
18 A `System.Type` object's family is defined as all objects of the exact same `System.Type`  
19 and of its subclasses.

20  
21 If the current instance represents an unassigned type parameter of a generic type, this  
22 property returns `false`.

23

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

## 3 Type.IsNestedFamily Property

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

### 9 Summary

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

### 12 Property Value

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

### 15 Description

16 This property is read-only.

17  
18 A `System.Type` object's family is defined as all objects of the exact same `System.Type`  
19 and of its subclasses.

20  
21 If the current instance represents an unassigned type parameter of a generic type, this  
22 property returns `false`.

23

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

## 3 Type.IsNestedFamORAssem Property

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

### 9 Summary

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

### 12 Property Value

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

### 15 Description

16 This property is read-only.

17  
18 A `System.Type` object's family is defined as all objects of the exact same `System.Type`  
19 and of its subclasses.

20  
21 If the current instance represents an unassigned type parameter of a generic type, this  
22 property returns `false`.

23

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

## 3 Type.IsNestedPrivate Property

```
4 [ILAsm]  
5 .property bool IsNestedPrivate { public hidebysig specialname instance  
6 bool get_IsNestedPrivate() }
```

```
7 [C#]  
8 public bool IsNestedPrivate { get; }
```

### 9 Summary

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

### 12 Property Value

13 `true` if the `System.Type` is nested and declared private; otherwise, `false`.

### 14 Description

15 This property is read-only.

16  
17 If the current instance represents an unassigned type parameter of a generic type, this  
18 property returns `false`.

19

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

## 3 Type.IsNestedPublic Property

```
4 [ILAsm]  
5 .property bool IsNestedPublic { public hidebysig specialname instance bool  
6 get_IsNestedPublic() }  
7 [C#]  
8 public bool IsNestedPublic { get; }
```

### 9 Summary

10 Gets a `System.Boolean` value indicating whether the current `System.Type` is a public  
11 nested class.

### 12 Property Value

13 `true` if the class is nested and declared public; otherwise, `false`.

### 14 Description

15 This property is read-only.

16  
17 If the current instance represents an unassigned type parameter of a generic type, this  
18 property returns `false`.

19

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

## 3 Type.IsNotPublic Property

```
4 [ILAsm]  
5 .property bool IsNotPublic { public hidebysig specialname instance bool  
6 get_IsNotPublic() }  
  
7 [C#]  
8 public bool IsNotPublic { get; }
```

### 9 Summary

10 Gets a `System.Boolean` value indicating whether the top-level `System.Type` is not  
11 declared public.

### 12 Property Value

13 true if the top-level `System.Type` is not declared public; otherwise, false.

### 14 Description

15 This property is read-only.

16  
17 If the current instance represents an unassigned type parameter of a generic type, this  
18 property returns `false`.

19

# 1 Type.IsPointer Property

```
2 [ILAsm]  
3 .property bool IsPointer { public hidebysig specialname instance bool  
4 get_IsPointer() }  
5 [C#]  
6 public bool IsPointer { get; }
```

## 7 Summary

8 Gets a `System.Boolean` value that indicates whether the current `System.Type`  
9 represents a pointer.

## 10 Property Value

11 This property is read-only.

12  
13 `true` if the current `System.Type` represents a pointer; otherwise `false`.

## 14 Description

15 This property is read-only.

16  
17 If the current instance represents a generic type, or a type parameter of a generic type  
18 or method, this property always returns `false`.

19

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

## 3 Type.IsPrimitive Property

```
4 [ILAsm]  
5 .property bool IsPrimitive { public hidebysig specialname instance bool  
6 get_IsPrimitive() }  
  
7 [C#]  
8 public bool IsPrimitive { get; }
```

### 9 Summary

10 Gets a `System.Boolean` value indicating whether the current `System.Type` is one of the  
11 primitive types.

### 12 Property Value

13 true if the `System.Type` is one of the primitive types; otherwise, false.

### 14 Description

15 This property is read-only.

16  
17 The primitive types are `System.Boolean`, `System.Byte`, `System.SByte`, `System.Int16`,  
18 `System.UInt16`, `System.Int32`, `System.UInt32`, `System.Int64`, `System.UInt64`,  
19 `System.Char`, `System.Double`, and `System.Single`.

20  
21 If the current instance represents a generic type, or a type parameter of a generic type  
22 or method, this property always returns false.

23

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

## 3 Type.IsPublic Property

```
4 [ILAsm]  
5 .property bool IsPublic { public hidebysig specialname instance bool  
6 get_IsPublic() }
```

```
7 [C#]  
8 public bool IsPublic { get; }
```

### 9 Summary

10 Gets a System.Boolean value indicating whether the top-level System.Type is declared  
11 public.

### 12 Property Value

13 true if the top-level System.Type is declared public; otherwise, false.

### 14 Description

15 This property is read-only.

16  
17 If the current instance represents an unassigned type parameter of a generic type, this  
18 property returns true.

19

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

## 3 Type.IsSealed Property

```
4 [ILAsm]  
5 .property bool IsSealed { public hidebysig specialname instance bool  
6 get_IsSealed() }
```

```
7 [C#]  
8 public bool IsSealed { get; }
```

### 9 Summary

10 Gets a System.Boolean value indicating whether the current System.Type is declared  
11 sealed.

### 12 Property Value

13 true if the System.Type is declared sealed; otherwise, false.

### 14 Description

15 This property is read-only.

16  
17 If the current instance represents an unassigned type parameter of a generic type, this  
18 property returns true.

19



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

## 3 Type.IsUnicodeClass Property

```
4 [ILAsm]  
5 .property bool IsUnicodeClass { public hidebysig specialname virtual  
6 instance bool get_IsUnicodeClass() }
```

```
7 [C#]  
8 public virtual bool IsUnicodeClass { get; }
```

### 9 Summary

10 Indicates whether the type attribute  
11 System.Reflection.TypeAttributes.UnicodeClass is selected for the current type.

### 12 Property Value

13 true if the type attribute System.Reflection.TypeAttributes.UnicodeClass is selected  
14 for the current type; otherwise, false.

### 15 Description

16 This property is read-only.

17  
18 If the current System.Type represents a generic type, this property applies to the  
19 definition of the type. If the current System.Type represents a type parameter of a  
20 generic type or method, this property always returns false.

21

# 1 Type.IsValueType Property

```
2 [ILAsm]  
3 .property bool IsValueType { public hidebysig specialname instance bool  
4 get_IsValueType() }  
5 [C#]  
6 public bool IsValueType { get; }
```

## 7 Summary

8 Gets a `System.Boolean` value that indicates whether the current `System.Type`  
9 represents a value type.

## 10 Property Value

11 true if the current `System.Type` represents a value type (structure); otherwise false.

## 12 Description

13 This property is read-only.

14  
15 This property returns true for enumerations, but not for the `System.Enum` type itself,  
16 which is a class. [*Note:* For an example that demonstrates this behavior, see  
17 `System.Type.IsEnum`.]  
18  
19

20

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

## 3 Type.Module Property

```
4 [ILAsm]  
5 .property class System.Reflection.Module Module { public hidebysig virtual  
6 abstract specialname class System.Reflection.Module get_Module() }  
7 [C#]  
8 public abstract Module Module { get; }
```

### 9 Summary

10 Gets the module in which the current System.Type is defined.

### 11 Property Value

12 A System.Reflection.Module that reflects the module in which the current  
13 System.Type is defined.

### 14 Description

15 If the current instance represents a generic type, this property returns the module in  
16 which the type was defined.

17  
18 Similarly, if the current instance represents a generic parameter T, this property returns  
19 the assembly that contains the generic type that defines T.

### 20 Behaviors

21 This property is read-only.

22

23

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

## 3 Type.Namespace Property

```
4 [ILAsm]  
5 .property string Namespace { public hidebysig virtual abstract specialname  
6 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 A System.String containing the namespace of the current System.Type.

### 13 Description

14 If the current instance represents a generic type, this property returns the namespace  
15 that contains the generic type definition. Similarly, if the current instance represents a  
16 generic parameter T, this property returns the namespace that contains the generic type  
17 that defines T.

18  
19 [*Note:* A namespace is a logical design-time naming convenience, used mainly to define  
20 scope in an application and organize classes and other types in a hierarchical structure.  
21 From the viewpoint of the system, there are no namespaces.]  
22  
23

### 24 Behaviors

25 This property is read-only.  
26  
27

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

### 3 Type.ReflectedType Property

```
4 [ILAsm]  
5 .property class System.Type ReflectedType { public hidebysig virtual  
6 specialname class System.Type get_ReflectedType() }  
7 [C#]  
8 public override Type ReflectedType { get; }
```

#### 9 Summary

10 Gets the type that was used to obtain the current instance.

#### 11 Property Value

12 The Type object through which the current instance was obtained.

#### 13 Description

14 This property is read-only.

15

16 If the current instance represents a generic type, or a type parameter of a generic type  
17 or method, this property returns the current instance.

18

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 { public hidebysig  
6 virtual abstract specialname valuetype System.RuntimeTypeHandle  
7 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 The System.RuntimeTypeHandle for the current System.Type.

### 14 Description

15 This property is read-only.

16  
17 The System.RuntimeTypeHandle encapsulates a pointer to an internal data structure  
18 that represents the type. This handle is unique during the process lifetime. The handle is  
19 valid only in the application domain in which it was obtained.

20

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

## 3 Type.TypeInitializer Property

```
4 [ILAsm]  
5 .property class System.Reflection.ConstructorInfo TypeInitializer { public  
6 hidebysig specialname instance class System.Reflection.ConstructorInfo  
7 get_TypeInitializer() }  
8 [C#]  
9 public ConstructorInfo TypeInitializer { get; }
```

### 10 Summary

11 Gets the initializer for the type represented by the current instance.

### 12 Property Value

13 A System.Reflection.ConstructorInfo containing the name of the static constructor  
14 for the type represented by the current instance

### 15 Description

16 This property is read-only.

17  
18 [*Note:* Type initializers are available through System.Type.GetMember,  
19 System.Type.GetMembers, and System.Type.GetConstructors.]  
20  
21  
22

23 If the current instance represents an unassigned type parameter of a generic type or  
24 method, this property returns null.

25

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

## 3 **Type.UnderlyingSystemType** Property

```
4 [ILAsm]  
5 .property class System.Type UnderlyingSystemType { public hidebysig  
6 virtual abstract specialname class System.Type get_UnderlyingSystemType()  
7 }  
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 The underlying system type for the `System.Type`.

### 14 **Description**

15 This property is read-only.

### 16 **Behaviors**

17 As described above.

18

19