

1 System.Reflection.Binder Class

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

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 Performs custom overload resolution and argument coercion to bind a member when
14 reflection is used to invoke a member of a `System.Type`.

15 Inherits From: System.Object

16

17 **Library:** Reflection

18

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

21

22 Description

23 Late binding is controlled by a customized binding interface through reflection. The
24 `System.Reflection.Binder` class is designed to provide this functionality.
25 `System.Reflection.Binder` objects are used in overload resolution and argument
26 coercion for dynamic invocation of members at runtime.

27

28 Access to information obtained from reflection is controlled at two levels: untrusted code
29 and code with `System.Security.Permissions.ReflectionPermission`.

30

31 Untrusted code is code with no special level of trust (such as code downloaded from the
32 Internet). Such code is allowed to invoke anything that it would have been able to
33 invoke in an early bound way.

34

35 `System.Security.Permissions.ReflectionPermission` controls access to metadata
36 through reflection. If this permission is granted to code, that code has access to all the
37 types in its application domain, assembly, and module. It can access information about
38 public, family, and private members of all types it has access to. Two primary
39 capabilities are granted:

- 40 • The ability to read the metadata for family and private members of any type.

- 1 • The ability to access peer classes in the module and peer modules in the assembly.

2 [Note: The term "reflection" refers to the ability to obtain information about a
3 `System.Object` during runtime. The primary means through which this information is
4 accessed is via the `System.Type` of the object. Reflection allows the programmatic discovery
5 of a type's metadata. The information included in the metadata includes details about the
6 assembly or module in which the type is defined as well as members of the type. Reflection
7 uses this information to provide the following primary services:

- 8 • Access to type information at runtime.
- 9 • The ability to use this type information to create instances, invoke methods, and
10 access data members of the type.

11 The primary users of these services are script engines, object viewers, compilers, and object
12 persistence formatters.

13

14 Through reflection, methods can be bound and invoked at runtime. If more than one
15 member exists for a given member name, overload resolution determines which
16 implementation of that method is invoked by the system. Coercion can occur when a
17 parameter specified for a method call does not match the type specified for the parameter
18 in the method signature. When possible, the binder converts the parameter (coerces it) to
19 the type specified by the method signature. Coercion might not be possible depending on
20 the types involved.

21

22 To bind to a method, field, or property, typically a list of probable candidates is obtained
23 from the `System.Type` of a `System.Object`. That list is then passed to the appropriate
24 method of a `System.Reflection.Binder` instance. Based on the other parameters passed
25 to that method, typically (although not necessarily) one of the members of the list is
26 chosen, and an object that reflects that member is returned.

27

28 The system supplies a default binder that provides default binding rules. Because binding
29 rules vary among programming languages, it is recommended that each programming
30 language provide a custom implementation of `System.Reflection.Binder`.

31

32]

33

1 Binder() Constructor

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

6 Summary

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

8

1
2 **Binder.BindToField(System.Reflection.Binding**
3 **Flags, System.Reflection.FieldInfo[],**
4 **System.Object,**
5 **System.Globalization.CultureInfo) Method**

```
6 [ILAsm]  
7 .method public hidebysig virtual abstract class  
8 System.Reflection.FieldInfo BindToField(valuetype  
9 System.Reflection.BindingFlags bindingAttr, class  
10 System.Reflection.FieldInfo[] match, object value, class  
11 System.Globalization.CultureInfo culture)  
  
12 [C#]  
13 public abstract FieldInfo BindToField(BindingFlags bindingAttr,  
14 FieldInfo[] match, object value, CultureInfo culture)
```

15 **Summary**

16 Selects a field from the specified set of fields, based on the specified criteria.

17 **Parameters**

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. For requirements, see the Behaviors section.
<i>match</i>	An array of System.Reflection.FieldInfo objects whose elements represent the set of fields that reflection has determined to be a possible match, typically because the fields have the correct member name.
<i>value</i>	An object of a type that is assignment-compatible with the type of the field being searched for. [Note: For example, if value is an instance of a class, the type of that instance can be assigned to the type of the field returned by this method. Fields in match that cannot be assigned to this value are eliminated from the search.]
<i>culture</i>	The only defined value for this parameter is null.

18
19 **Return Value**

1 A `System.Reflection.FieldInfo` instance that reflects the field that matches the
2 specified criteria. It is not required that this instance be contained in *match*. If a suitable
3 field is not found, returns `null`.

4 **Behaviors**

5 For the *bindingAttr* parameter, the caller is required to specify either
6 `System.Reflection.BindingFlags.Public` OR
7 `System.Reflection.BindingFlags.NonPublic`, and either
8 `System.Reflection.BindingFlags.Instance` OR
9 `System.Reflection.BindingFlags.Static`. If at least one value from each pair is not
10 specified, this method is required to return `null`.

11

12

1
 2 **Binder.BindToMethod(System.Reflection.Bind**
 3 **ingFlags, System.Reflection.MethodBase[],**
 4 **System.Object[]&**
 5 **System.Reflection.ParameterModifier[],**
 6 **System.Globalization.CultureInfo,**
 7 **System.String[], System.Object&) Method**

```

8  [ILAsm]
9  .method public hidebysig virtual abstract class
10 System.Reflection.MethodBase BindToMethod(valuetype
11 System.Reflection.BindingFlags bindingAttr, class
12 System.Reflection.MethodBase[] match, object[]& args, class
13 System.Reflection.ParameterModifier[] modifiers, class
14 System.Globalization.CultureInfo culture, string[] names, object& state)

15 [C#]
16 public abstract MethodBase BindToMethod(BindingFlags bindingAttr,
17 MethodBase[] match, ref object[] args, ParameterModifier[] modifiers,
18 CultureInfo culture, string[] names, ref object state)

```

19 **Summary**

20 Selects a method based on the specified criteria.

21 **Parameters**

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of <code>System.Reflection.BindingFlags</code> values that control the binding process. For requirements, see the Behaviors section.
<i>match</i>	An array of <code>System.Reflection.MethodBase</code> objects that represent the set of methods that Reflection has determined to be a possible match, typically because they have the correct member name.
<i>args</i>	An array of objects that represent the parameters passed in the method invocation. The types, values, and order of the elements of this array might be changed by this method to match the signature of the selected method.
<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>names</i>	A <code>System.String</code> array containing the names of methods to be searched.
<i>state</i>	A binder-provided <code>System.Object</code> that keeps track of parameter reordering. The <i>state</i> object is totally defined by the implementer of the <code>System.Reflection.Binder</code> class. This object is <code>null</code> if the binder does not reorder the argument array of the bound method.

1

2 **Return Value**

3 A `System.Reflection.MethodBase` instance that reflects the method that matches to
 4 the specified criteria. It is not required that this instance be contained in *match*. If a
 5 suitable method is not found, returns `null`.

6 **Description**

7 If *state* is not `null`, the system invokes
 8 `System.Reflection.Binder.ReorderArgumentArray` after this method returns. [*Note:*
 9 This allows a caller to map the argument array of a method back to the original form if
 10 the order has been altered by `System.Reflection.Binder.BindToMethod`. This is useful
 11 if `ByRef` arguments are in the argument array, because the caller can retrieve those
 12 arguments in their original order on return from this method. When arguments are
 13 passed by name (i.e., using named arguments), the binder reorders the argument array
 14 and that is what the caller sees. This method insures that the original order of the
 15 arguments is restored.]

16

17

18 **Behaviors**

19 For the *bindingAttr* parameter, the caller is required to specify either
 20 `System.Reflection.BindingFlags.Public` OR
 21 `System.Reflection.BindingFlags.NonPublic`, and either
 22 `System.Reflection.BindingFlags.Instance` OR
 23 `System.Reflection.BindingFlags.Static`. If at least one value from each pair is not
 24 specified, this method is required to return `null`.

25

26 The `System.Reflection.Binder.BindToMethod` method is permitted to change the
 27 order of the argument array of a method call only if the binder returns, via the *state*
 28 parameter, a non-null opaque object that records the original order of the argument
 29 array. If, on return from `System.Reflection.Binder.BindToMethod`, *state* is not `null`,
 30 the system calls `System.Reflection.Binder.ReorderArgumentArray`.

31

1 Binder.ChangeType(System.Object, 2 System.Type, 3 System.Globalization.CultureInfo) Method

```
4 [ILAsm]  
5 .method public hidebysig virtual abstract object ChangeType(object value,  
6 class System.Type type, class System.Globalization.CultureInfo culture)  
  
7 [C#]  
8 public abstract object ChangeType(object value, Type type, CultureInfo  
9 culture)
```

10 Summary

11 Converts the type of the specified object to the specified type.

12 Parameters

Parameter	Description
<i>value</i>	The object to be converted to a new <code>System.Type</code> .
<i>type</i>	The <code>System.Type</code> to which <i>value</i> is converted.
<i>culture</i>	The only defined value for this parameter is <code>null</code> .

13

14 Return Value

15 A new object of the type specified by *type*. The contents of this object are equal to those
16 of *value*.

17 Behaviors

18 As described above.

19

20 How and When to Override

21 Implement this method to change the type of a member of a parameter array. Typically,
22 it is recommended that implementations of this method perform only widening
23 conversions.

24

1 **Usage**

2 This method is used to change the type of a element in a parameter array to match the
3 type required by the signature of a bound method.

4

5

Binder.ReorderArgumentArray(System.Object []&, System.Object) Method

```
[ILAsm]
.method public hidebysig virtual abstract void
ReorderArgumentArray(object[]& args, object state)

[C#]
public abstract void ReorderArgumentArray(ref object[] args, object state)
```

Summary

Restores the specified set of parameters to their original order after a call to `System.Reflection.Binder.BindToMethod`.

Parameters

Parameter	Description
<i>args</i>	An array of objects whose elements represent the parameters passed to the bound method in their original order.
<i>state</i>	A binder-provided opaque object that keeps track of parameter reordering. This object is the same object that was passed as the <i>state</i> parameter in the invocation of <code>System.Reflection.Binder.BindToMethod</code> that caused <code>System.Reflection.Binder.ReorderArgumentArray</code> to be called.

Description

[Note: When a method call is bound to a method through reflection using `System.Reflection.Binder.BindToMethod`, the order, value, and type of the parameters in the original method call can be changed to match the signature of the bound method. The binder creates *state* as an opaque object that records the original order of the argument array. If, on return from `System.Reflection.Binder.BindToMethod`, *state* is not null, the system calls `System.Reflection.Binder.ReorderArgumentArray`. This allows a caller to map the argument array of a method back to the original form if the order had been altered by `System.Reflection.Binder.BindToMethod`. This is useful if `ByRef` arguments are in the argument array, because the caller can retrieve those arguments in their original order on return from this method. When arguments are passed by name (i.e., using named arguments), the binder reorders the argument array and that is what the caller sees. This method insures that the original order of the arguments is restored.]

Behaviors

1 *state* is required to be a non-null `System.Object` that tracks the original ordering of
2 *args* if *args* is reordered by a call to `System.Reflection.Binder.BindToMethod`. This
3 method is required to restore the elements of *args* to their original order, value, and
4 `System.Type`

5

6 **How and When to Override**

7 Implement this method to insure that the parameters contained in *args* are returned to
8 their original order, `System.Type` and value, after being used by a bound method.

9

10 **Usage**

11 Use this method to insure that the parameters contained in *args* are returned to their
12 original order, `System.Type` and value, after being used by a bound method.

13

14

1
2 **Binder.SelectMethod(System.Reflection.BindingFlags, System.Reflection.MethodBase[],**
3 **System.Type[],**
4 **System.Reflection.ParameterModifier[])**
5 **Method**
6

```
7 [ILAsm]  
8 .method public hidebysig virtual abstract class  
9 System.Reflection.MethodBase SelectMethod(valuetype  
10 System.Reflection.BindingFlags bindingAttr, class  
11 System.Reflection.MethodBase[] match, class System.Type[] types, class  
12 System.Reflection.ParameterModifier[] modifiers)  
  
13 [C#]  
14 public abstract MethodBase SelectMethod(BindingFlags bindingAttr,  
15 MethodBase[] match, Type[] types, ParameterModifier[] modifiers)
```

16 **Summary**

17 Selects a method from the specified set of methods, based on the argument type.

18 **Parameters**

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of <code>System.Reflection.BindingFlags</code> values that control the binding process. For requirements, see the Behaviors section.
<i>match</i>	An array of <code>System.Reflection.MethodBase</code> objects that represent the set of methods that Reflection has determined to be a possible match, typically because they have the correct member name.
<i>types</i>	An array of <code>System.Type</code> objects that represent the values used to locate a matching method.
<i>modifiers</i>	The only defined value for this parameter is <code>null</code> .

19
20 **Return Value**

21 A `System.Reflection.MethodBase` instance that reflects the method that is matched to
22 the specified criteria. It is not required that this instance be contained in *match*. If a
23 suitable method is not found, returns `null`.

1 **Behaviors**

2 For the *bindingAttr* parameter, the caller is required to specify either
3 `System.Reflection.BindingFlags.Public` OR
4 `System.Reflection.BindingFlags.NonPublic`, and either
5 `System.Reflection.BindingFlags.Instance` OR
6 `System.Reflection.BindingFlags.Static`. If at least one value from each pair is not
7 specified, this method is required to return `null`.

8

9

1
2 **Binder.SelectProperty(System.Reflection.Bind**
3 **ingFlags, System.Reflection.PropertyInfo[],**
4 **System.Type, System.Type[],**
5 **System.Reflection.ParameterModifier[])**
6 **Method**

```
7 [ILAsm]  
8 .method public hidebysig virtual abstract class  
9 System.Reflection.PropertyInfo SelectProperty(valuetype  
10 System.Reflection.BindingFlags bindingAttr, class  
11 System.Reflection.PropertyInfo[] match, class System.Type returnType,  
12 class System.Type[] indexes, class System.Reflection.ParameterModifier[]  
13 modifiers)  
14 [C#]  
15 public abstract PropertyInfo SelectProperty(BindingFlags bindingAttr,  
16 PropertyInfo[] match, Type returnType, Type[] indexes, ParameterModifier[]  
17 modifiers)
```

18 **Summary**

19 Selects a property from the specified set of properties, based on the specified criteria.

20 **Parameters**

Parameter	Description
<i>bindingAttr</i>	A bitwise combination of System.Reflection.BindingFlags values that control the binding process. For requirements, see the Behaviors section.
<i>match</i>	An array of System.Reflection.PropertyInfo objects that represent the set of properties that Reflection has determined to be a possible match, typically because they have the correct member name.
<i>returnType</i>	The System.Type of the property being searched for.
<i>indexes</i>	An array of System.Type objects that represent the index types of the property being searched for. [Note: Use this parameter for index properties such as the indexer for a class.]
<i>modifiers</i>	The only defined value for this parameter is null.

21
22 **Return Value**

1 A `System.Reflection.PropertyInfo` instance that reflects the property that matches
2 the specified criteria. It is not required that this instance be contained in *match*. If a
3 suitable property is not found, returns `null`.

4 **Behaviors**

5 For the *bindingAttr* parameter, the caller is required to specify either
6 `System.Reflection.BindingFlags.Public` OR
7 `System.Reflection.BindingFlags.NonPublic`, and either
8 `System.Reflection.BindingFlags.Instance` OR
9 `System.Reflection.BindingFlags.Static`. If at least one value from each pair is not
10 specified, this method is required to return `null`.

11

12