System.Reflection.MethodInfo Class

```
[ILAsm]
.class public abstract serializable MethodInfo extends
System.Reflection.MethodBase

[C#]
public abstract class MethodInfo: MethodBase
```

Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 2.0.x.x
- *Attributes:*
    - CLSCompliantAttribute(true)

Summary

Discovers the attributes of a method and provides access to method metadata.

Inherits From: System.Reflection.MethodBase

Library: Reflection

Thread Safety: This type is safe for multithreaded operations.

Permissions

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

Description

Instances of `System.Reflection.MethodInfo` are obtained by calling the `System.Type.GetMethods` or `System.Type.GetMethod` method of a `System.Type` object or of an object that derives from `System.Type`, or by calling the `System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])` method of a `System.Reflection.MethodInfo` that represents a generic method definition.

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

[*Note:* When operating on the given kinds of methods, the following properties return the result as shown:

Property	Non-Generic	Open Generic	Closed Generic
`IsGenericMethodDefinition`	False	True	False
`ContainsGenericParameters`	False	True	False
`IsGenericMethod`	False	True	True

]

MethodInfo() Constructor

```
[ILAsm]
family rtspecialname specialname instance void .ctor()

[C#]
protected MethodInfo()
```

Summary

Constructs a new instance of the `System.Reflection.MethodInfo` class.

MethodInfo.GetBaseDefinition() Method

```
[ILAsm]
.method public hidebysig virtual abstract class
System.Reflection.MethodInfo GetBaseDefinition()

[C#]
public abstract MethodInfo GetBaseDefinition()
```

Summary

Returns a new `System.Reflection.MethodInfo` instance that reflects the first definition of the method reflected by the current instance in the inheritance hierarchy of that method.

Return Value

A new `System.Reflection.MethodInfo` instance that reflects the first definition of the method reflected by the current instance in the inheritance hierarchy of that method.

Behaviors

`System.Reflection.MethodInfo.GetBaseDefinition` proceeds along the inheritance hierarchy of the method reflected by the current instance, returning a `System.Reflection.MethodInfo` instance that reflects the first definition in the hierarchy of that method.

The method declaration to be reflected by the new `System.Reflection.MethodInfo` instance is determined as follows:

- If the method reflected by the current instance overrides a virtual definition in the base class, the virtual definition is reflected.
- If the method reflected by the current instance is specified with the `new` keyword, the current instance is returned.
- If the method reflected by the current instance is not defined in the type of the object on which `System.Reflection.MethodInfo.GetBaseDefinition` is called, the method definition of the furthest ancestor in the class hierarchy is reflected.

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

MethodInfo.GetGenericArguments() Method

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

[C#]
public override Type[] GetGenericArguments()
```

Summary

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

Return Value

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

Description

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

If the current method is a closed constructed method (that is, the `System.Reflection.MethodInfo.ContainsGenericParameters` property returns `false`), the array returned by the `System.Reflection.MethodInfo.GetGenericArguments` method contains the types that have been assigned to the generic type parameters of the generic method definition.

If the current method is a generic method definition, the array contains the type parameters.

If the current method is an open constructed method (that is, the `System.Reflection.MethodInfo.ContainsGenericParameters` property returns `true`) in which specific types have been assigned to some type parameters and type parameters of enclosing generic types have been assigned to other type parameters, the array contains both types and type parameters. Use the `System.Type.IsGenericParameter` property to tell them apart.

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

Example

The following code shows how to get the type arguments of a generic method and display them. (It is part of a larger example for the method `System.Reflection.MethodInfo.MakeGenericMethod`.)

[C#]

```
// If this is a generic method, display its type arguments.
//
if (mi.IsGenericMethod)
{
        Type[] typeArguments = mi.GetGenericArguments();

        Console.WriteLine("\tList type arguments ({0}):",
                typeArguments.Length);

        foreach (Type tParam in typeArguments)
        {
                // IsGenericParameter is true only for generic type
                // parameters.
                //
                if (tParam.IsGenericParameter)
                {
                    Console.WriteLine("\t\t{0}  (unbound - parameter position
{1})",
                        tParam,
                        tParam.GenericParameterPosition);
                }
                else
                {
                    Console.WriteLine("\t\t{0}", tParam);
                }
            }
        }
        else
        {
            Console.WriteLine("\tThis is not a generic method.");
        }
}
```

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

MethodInfo.GetGenericMethodDefinition() Method

```
[ILAsm]
.method public hidebysig virtual class System.Reflection.MethodInfo
GetGenericMethodDefinition()

[C#]
public override MethodInfo GetGenericMethodDefinition()
```

Summary

Returns a `System.Reflection.MethodInfo` object that represents a generic method definition from which the current method can be constructed.

Return Value

A `System.Reflection.MethodInfo` object representing a generic method definition from which the current method can be constructed.

Description

If you call `System.Reflection.MethodInfo.GetGenericMethodDefinition` on a `System.Reflection.MethodInfo` that already represents a generic method definition, it returns the current `System.Reflection.MethodInfo`.

If a generic method definition includes generic parameters of the declaring type, there will be a generic method definition specific to each constructed type.

A generic method definition is a template from which methods can be constructed. For example, from the generic method definition `T M<T>(T t)` you can construct and invoke the method `int M<int>(int t)`. Given a `System.Reflection.MethodInfo` object representing this constructed method, the `System.Reflection.MethodInfo.GetGenericMethodDefinition` method returns the generic method definition.

If two constructed methods are created from the same generic method definition, the `System.Reflection.MethodInfo.GetGenericMethodDefinition` method returns the same `System.Reflection.MethodInfo` object for both methods.

If you call `System.Reflection.MethodInfo.GetGenericMethodDefinition` on a `System.Reflection.MethodInfo` that already represents a generic method definition, it returns the current `System.Reflection.MethodInfo`.

If a generic method definition includes generic parameters of the declaring type, there will be a generic method definition specific to each constructed type. For example, consider the following C# code:

```
class B<U,V> {}

class C<T> { B<T,S> M<S>() {}}
```

In the constructed type `C<int>`, the generic method `M` returns `B<int, S>`. In the open type `C<T>`, `M` returns `B<T, S>`. In both cases, the `System.Reflection.MethodInfo.IsGenericMethodDefinition` property returns true for the `System.Reflection.MethodInfo` that represents `M`, so`System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])` can be called on both `System.Reflection.MethodInfo` objects. In the case of the constructed type, the result of calling `System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])` is a `System.Reflection.MethodInfo` that can be invoked. In the case of the open type, the `System.Reflection.MethodInfo` returned by `System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])` cannot be invoked.

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

Exceptions

Exception	Condition
System.InvalidOperationException	The current method is not a generic method. That is, `System.Reflection.MethodInfo.IsGenericMethod` returns `false`.

Example

The following code shows a class with a generic method and the code required to obtain a `System.Reflection.MethodInfo` for the method, bind the method to type arguments, and get the original generic type definition back from the bound method. (It is part of a larger example for the method `System.Reflection.MethodInfo.MakeGenericMethod`.)

[C#]

```
// Define a class with a generic method.
public class Example
{
    public static void Generic<T>(T toDisplay)
    {
        Console.WriteLine("\nHere it is: {0}", toDisplay);
    }
}

//...
// Create a Type object representing class Example, and
```

```
1   // get a MethodInfo representing the generic method.
2   //
3   Type ex = Type.GetType("Example");
4   MethodInfo mi = ex.GetMethod("Generic");
5
6   DisplayGenericMethodInfo(mi);
7
8   // Bind the type parameter of the Example method to
9   // type int.
10  //
11  Type[] arguments = {typeof(int)};
12  MethodInfo miBound = mi.MakeGenericMethod(arguments);
13
14  DisplayGenericMethodInfo(miBound);
15
16
17  //...
18  // Get the generic type definition from the closed method,
19  // and show it's the same as the original definition.
20  //
21  MethodInfo miDef = miBound.GetGenericMethodDefinition();
22  Console.WriteLine("\nThe definition is the same: {0}",
23         miDef == mi);
24
```

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

MethodInfo.MakeGenericMethod(System.Type[]) Method

```
[ILAsm]
.method public hidebysig virtual class System.Reflection.MethodInfo
MakeGenericMethod(class System.Type[] typeArguments)

[C#]
public override MethodInfo MakeGenericMethod(params System.Type[]
typeArguments)
```

Summary

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

Parameters

Parameter	Description
typeArguments	An array of types to be substituted for the type parameters of the current generic method.

Return Value

A `System.Reflection.MethodInfo` object that represents the constructed method formed by substituting the elements of *typeArguments* for the type parameters of the current generic method definition.

Description

The `System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])` method allows you to write code that assigns specific types to the type parameters of a generic method definition, thus creating a `System.Reflection.MethodInfo` object that represents a particular constructed method. If the `System.Reflection.MethodInfo.ContainsGenericParameters` property of this `System.Reflection.MethodInfo` object returns `true`, you can use it to invoke the method or to create a delegate to invoke the method.

Methods constructed with the `System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])` method can be open; that is, some of their type arguments can be type parameters of enclosing generic types. You might use such open constructed methods when you generate dynamic assemblies. For example, consider the following C# code:

```
class C
{
  T N<T,U>(T t, U u) {...}
  public V M<V>(V v)
  {
    return N<V,int>(v, 42);
  }
}
```

The method body of MM contains a call to method `N`, specifying the type parameter of `M` and the type `System.Int32`. The `System.Reflection.MethodInfo.IsGenericMethodDefinition` property returns `false` for method `N<V,int>`. The `System.Reflection.MethodInfo.ContainsGenericParameters` property returns `true`, so method `N<V,int>` cannot be invoked.

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

Exceptions

Exception	Condition
System.ArgumentException	The number of elements in *typeArguments* is not the same as the number of type parameters of the current generic method definition. -or- An element of *typeArguments* does not satisfy the constraints specified for the corresponding type parameter of the current generic method definition.
System.ArgumentNullException	*typeArguments* is `null`. -or- Any element of *typeArguments* is `null`.
System.InvalidOperationException	The current `System.Reflection.MethodInfo` does not represent the definition of a generic method. (That is, `System.Reflection.MethodInfo.IsGenericMethodDefinition` returns `false`).

Example

The following code demonstrates the properties and methods of System.Reflection.MethodInfo that support the examination of generic methods. The example does the following:

1. Defines a class that has a generic method.
2. Creates a `System.Reflection.MethodInfo` that represents the generic method.
3. Displays properties of the generic method definition.
4. Binds the `System.Reflection.MethodInfo` to a type, and invokes it.
5. Displays properties of the bound generic method.
6. Retrieves the generic method definition from the bound method.

[C#]

```
using System;
using System.Reflection;

// Define a class with a generic method.
public class Example
{
    public static void Generic<T>(T toDisplay)
    {
        Console.WriteLine("\nHere it is: {0}", toDisplay);
    }
}

public class Test
{
    public static void Main()
    {
        Console.WriteLine("\n--- Examine a generic method.");

        // Create a Type object representing class Example, and
        // get a MethodInfo representing the generic method.
        //
        Type ex = Type.GetType("Example");
        MethodInfo mi = ex.GetMethod("Generic");

        DisplayGenericMethodInfo(mi);

        // Bind the type parameter of the Example method to
        // type int.
        //
        Type[] arguments = {typeof(int)};
        MethodInfo miBound = mi.MakeGenericMethod(arguments);

        DisplayGenericMethodInfo(miBound);

        // Invoke the method.
        object[] args = {42};
        miBound.Invoke(null, args);

        // Invoke the method normally.
```

```
            Example.Generic<int>(42);

            // Get the generic type definition from the closed method,
            // and show it's the same as the original definition.
            //
            MethodInfo miDef = miBound.GetGenericMethodDefinition();
            Console.WriteLine("\nThe definition is the same: {0}",
                miDef == mi);
        }

        private static void DisplayGenericMethodInfo(MethodInfo mi)
        {
            Console.WriteLine("\n{0}", mi);

            Console.WriteLine("\tIs this a generic method definition? {0}",
                mi.IsGenericMethodDefinition);

            Console.WriteLine("\tDoes it have generic arguments? {0}",
                mi.IsGenericMethod);

            Console.WriteLine("\tDoes it have unbound generic parameters? {0}",
                mi.ContainsGenericParameters);

            // If this is a generic method, display its type arguments.
            //
            if (mi.IsGenericMethod)
            {
                Type[] typeArguments = mi.GetGenericArguments();

                Console.WriteLine("\tList type arguments ({0}):",
                    typeArguments.Length);

                foreach (Type tParam in typeArguments)
                {
                    // IsGenericParameter is true only for generic type
                    // parameters.
                    //
                    if (tParam.IsGenericParameter)
                    {
                        Console.WriteLine("\t\t{0}  (unbound - parameter position
{1})",
                            tParam,
                            tParam.GenericParameterPosition);
                    }
                    else
                    {
                        Console.WriteLine("\t\t{0}", tParam);
                    }
                }
            }
            else
            {
                Console.WriteLine("\tThis is not a generic method.");
            }
        }
    }

```

```
/* This example produces the following output:

--- Examine a generic method.

Void Generic[T](T)
        Is this a generic method definition? True
        Does it have generic arguments? True
        Does it have unbound generic parameters? True
        List type arguments (1):
                T  (unbound - parameter position 0)

Void Generic[Int32](Int32)
        Is this a generic method definition? False
        Does it have generic arguments? True
        Does it have unbound generic parameters? False
        List type arguments (1):
                System.Int32

Here it is: 42

Here it is: 42

The definition is the same: True

 */
```

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

MethodInfo.ContainsGenericParameters Property

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

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

Summary

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

Property Value

`true` if the `System.Reflection.MethodInfo` contains unassigned generic type parameters; otherwise `false`.

Description

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

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

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

Behaviors

This property is read-only.

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

MethodInfo.IsGenericMethod Property

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

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

Summary

Returns a value that indicates whether the current method is a generic method.

Property Value

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

Description

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

The following table summarizes the invariant conditions for terms specific to generic methods. For other terms used in generic reflection, such as generic type parameter and generic type, see the `System.Type.IsGenericType` property.

Term	Invariant
generic method definition	The `System.Reflection.MethodInfo.IsGenericMethodDefinition` property is `true`. Defines a generic method. A constructed method is created by calling the `System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])` method on a `System.Reflection.MethodInfo` object that represents a generic method definition, and specifying an array of type arguments. `System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])` can be called only on generic method definitions. Any generic method definition is a generic method, but the converse is not true.
generic	The `System.Reflection.MethodInfo.IsGenericMethod` property is `true`.

method	Can be a generic method definition, an open constructed method, or a closed constructed method.
open constructed method	The `System.Reflection.MethodInfo.ContainsGenericParameters` property is `true`. It is not possible to invoke an open constructed method.
closed constructed method	The `System.Reflection.MethodInfo.ContainsGenericParameters` property is `false`. When examined recursively, the method has no unassigned generic parameters. The containing type has no generic type parameters, and none of the type arguments have generic type parameters.

1
2

3 **Behaviors**

4 This property is read-only.

5

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

MethodInfo.IsGenericMethodDefinition Property

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

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

Summary

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

Property Value

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

Description

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

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

Use the `System.Reflection.MethodInfo.IsGenericMethodDefinition` property to determine whether type arguments have been assigned to the type parameters of a generic method. If type arguments have been assigned, the `System.Reflection.MethodInfo.IsGenericMethodDefinition` property returns `false` even if some of the type arguments are `System.Type` objects that represent type parameters of enclosing types. For example, consider the following C# code:

```
class C
{
  T N<T,U>(T t, U u) {...}
  public V M<V>(V v)
  {
    return N<V,int>(v, 42);
```

```
  }
}
```

The method body of `M` contains a call to method `N`, specifying the type parameter of `M` and the type `System.Int32`. The `System.Reflection.MethodInfo.IsGenericMethodDefinition` property returns `false` for method `N<V,int>`.

[*Note:* Although the open constructed method `N<V,int>` is not encountered when reflecting over class `C`, it must be generated using `System.Reflection.MethodInfo.MakeGenericMethod(System.Type[])`.

]

If a generic method definition includes generic parameters of the declaring type, there will be a generic method definition specific to each constructed type. For example, consider the following C# code:

```
class B<U,V> {}
class C<T> { B<T,S> M<S>() {}}
```

In the constructed type `C<int>`, the generic method `M` returns `B<int, S>`. In the open type `C<T>`, `M` returns `B<T, S>`. In both cases, the `System.Reflection.MethodInfo.IsGenericMethodDefinition` property returns `true` for the `System.Reflection.MethodInfo` that represents `M`.

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

[*Note:* See the `System.Reflection.MethodInfo` type for an example of the use of this property.

]

Behaviors

This property is read-only.

MethodInfo.ReturnType Property

```
[ILAsm]
.property class System.Type ReturnType { public hidebysig virtual abstract
specialname class System.Type get_ReturnType() }

[C#]
public abstract Type ReturnType { get; }
```

Summary

Gets the type of the return value of the method reflected by the current instance.

Property Value

The `System.Type` of the return value of the method reflected by the current instance. This property is equal to the `System.Type` object representing `System.Void` if the return value of the method is `void`.

Behaviors

This property is read-only.