Type Class

Reference

Definition

Namespace: System

Assembly: System.Runtime.dll

Represents type declarations: class types, interface types, array types, value types, enumeration types, type parameters, generic type definitions, and open or closed constructed generic types.

C#

```csharp
public abstract class Type : System.Reflection.MemberInfo,
System.Reflection.IReflect
```

Inheritance Object → MemberInfo → Type

Derived System.Reflection.TypeInfo

Implements IReflect

Examples

The following example shows a few representative features of Type. The C# `typeof` operator (`GetType` operator in Visual Basic) is used to get a Type object representing String. From this Type object, the GetMethod method is used to get a MethodInfo representing the String.Substring overload that takes a starting location and a length.

To identify the overload signature, the code example creates a temporary array containing two Type objects representing `int` (`Integer` in Visual Basic).

Note

To be precise, the array contains two references to the instance of **Type** that represents `int` in the current application domain. For any type, there is only one instance of **Type** per application domain.

The code example uses the MethodInfo to invoke the Substring method on the string "Hello, World!", and displays the result.

C#

```csharp
using System;
using System.Reflection;

class Example
{
    static void Main()
    {
        Type t = typeof(String);

        MethodInfo substr = t.GetMethod("Substring",
            new Type[] { typeof(int), typeof(int) });

        Object result =
            substr.Invoke("Hello, World!", new Object[] { 7, 5 });
        Console.WriteLine("{0} returned \"{1}\".", substr, result);
    }
}

/* This code example produces the following output:

System.String Substring(Int32, Int32) returned "World".
 */
```

Remarks

`Type` is the root of the System.Reflection functionality and is the primary way to access metadata. Use the members of Type to get information about a type declaration, about the members of a type (such as the constructors, methods, fields, properties, and events of a class), as well as the module and the assembly in which the class is deployed.

No permissions are required for code to use reflection to get information about types and their members, regardless of their access levels. No permissions are required for code to use reflection to access public members, or other members whose access levels would make them visible during normal compilation. However, in order for your code to use reflection to access members that would normally be inaccessible, such as private or internal methods, or protected fields of a type your class does not inherit, your code must have ReflectionPermission. See Security Considerations for Reflection.

`Type` is an abstract base class that allows multiple implementations. The system will always provide the derived class `RuntimeType`. In reflection, all classes beginning with the word Runtime are created only once per object in the system and support comparison operations.

> ⓘ **Note**
>
> In multithreading scenarios, do not lock **Type** objects in order to synchronize access to `static` data. Other code, over which you have no control, might also lock your class type. This might result in a deadlock. Instead, synchronize access to static data by locking a private `static` object.

> ⓘ **Note**
>
> A derived class can access protected members of the calling code's base classes. Also, access is allowed to assembly members of the calling code's assembly. As a rule, if you are allowed access in early-bound code, then you are also allowed access in late-bound code.

> ⓘ **Note**
>
> Interfaces that extend other interfaces do not inherit the methods defined in the extended interfaces.

In this section:

What types does a Type object represent?

This class is thread safe; multiple threads can concurrently read from an instance of this type. An instance of the Type class can represent any of the following types:

- Classes
- Value types
- Arrays
- Interfaces
- Enumerations

- Delegates
- Constructed generic types and generic type definitions
- Type arguments and type parameters of constructed generic types, generic type definitions, and generic method definitions

Retrieving a Type object

The Type object associated with a particular type can be obtained in the following ways:

- The instance Object.GetType method returns a Type object that represents the type of an instance. Because all managed types derive from Object, the GetType method can be called on an instance of any type.

  The following example calls the Object.GetType method to determine the runtime type of each object in an object array.

  C#

  ```
  object[] values = { "word", true, 120, 136.34, 'a' };
  foreach (var value in values)
     Console.WriteLine("{0} - type {1}", value,
                       value.GetType().Name);

  // The example displays the following output:
  //       word - type String
  //       True - type Boolean
  //       120 - type Int32
  //       136.34 - type Double
  //       a - type Char
  ```

- The static Type.GetType methods return a Type object that represents a type specified by its fully qualified name.

- The Module.GetTypes, Module.GetType, and Module.FindTypes methods return `Type` objects that represent the types defined in a module. The first method can be used to obtain an array of Type objects for all the public and private types defined in a module. (You can obtain an instance of `Module` through the Assembly.GetModule or Assembly.GetModules method, or through the Type.Module property.)

- The System.Reflection.Assembly object contains a number of methods to retrieve the classes defined in an assembly, including Assembly.GetType, Assembly.GetTypes, and Assembly.GetExportedTypes.

- The FindInterfaces method returns a filtered list of interface types supported by a type.
- The GetElementType method returns a `Type` object that represents the element.
- The GetInterfaces and GetInterface methods return Type objects representing the interface types supported by a type.
- The GetTypeArray method returns an array of Type objects representing the types specified by an arbitrary set of objects. The objects are specified with an array of type Object.
- The GetTypeFromProgID and GetTypeFromCLSID methods are provided for COM interoperability. They return a Type object that represents the type specified by a `ProgID` or `CLSID`.
- The GetTypeFromHandle method is provided for interoperability. It returns a `Type` object that represents the type specified by a class handle.
- The C# `typeof` operator, the C++ `typeid` operator, and the Visual Basic `GetType` operator obtain the `Type` object for a type.
- The MakeGenericType method returns a Type object representing a constructed generic type, which is an open constructed type if its ContainsGenericParameters property returns `true`, and a closed constructed type otherwise. A generic type can be instantiated only if it is closed.
- The MakeArrayType, MakePointerType, and MakeByRefType methods return Type objects that represent, respectively, an array of a specified type, a pointer to a specified type, and the type of a reference parameter (`ref` in C#, 'byref' in F#, `ByRef` in Visual Basic).

Comparing type objects for equality

A Type object that represents a type is unique; that is, two Type object references refer to the same object if and only if they represent the same type. This allows for comparison of Type objects using reference equality. The following example compares the Type objects that represent a number of integer values to determine whether they are of the same type.

C#

```csharp
long number1 = 1635429;
int number2 = 16203;
```

```
double number3 = 1639.41;
long number4 = 193685412;

// Get the type of number1.
Type t = number1.GetType();

// Compare types of all objects with number1.
Console.WriteLine("Type of number1 and number2 are equal: {0}",
                  Object.ReferenceEquals(t, number2.GetType()));
Console.WriteLine("Type of number1 and number3 are equal: {0}",
                  Object.ReferenceEquals(t, number3.GetType()));
Console.WriteLine("Type of number1 and number4 are equal: {0}",
                  Object.ReferenceEquals(t, number4.GetType()));

// The example displays the following output:
//       Type of number1 and number2 are equal: False
//       Type of number1 and number3 are equal: False
//       Type of number1 and number4 are equal: True
```

Notes to Implementers

When you inherit from `Type`, you must override the following members:

- Assembly
- AssemblyQualifiedName
- BaseType
- FullName
- GetAttributeFlagsImpl()
- GetConstructorImpl(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetConstructors(BindingFlags)
- GetElementType()
- GetEvent(String, BindingFlags)
- GetEvents(BindingFlags)
- GetField(String, BindingFlags)
- GetFields(BindingFlags)
- GetInterface(String, Boolean)

- GetInterfaces()
- GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])
- GetMethods(BindingFlags)
- GetNestedType(String, BindingFlags)
- GetNestedTypes(BindingFlags)
- GetProperties(BindingFlags)
- GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], ParameterModifier[])
- GUID
- HasElementTypeImpl()
- InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[])
- IsArrayImpl()
- IsByRefImpl()
- IsCOMObjectImpl()
- IsPointerImpl()
- IsPrimitiveImpl()
- Module
- Namespace
- TypeHandle
- UnderlyingSystemType
- GetCustomAttributes(Boolean)
- GetCustomAttributes(Type, Boolean)
- IsDefined(Type, Boolean)
- Name

Constructors

Type()	Initializes a new instance of the Type class.

Fields

Delimiter	Separates names in the namespace of the Type. This field is read-only.
EmptyTypes	Represents an empty array of type Type. This field is read-only.
FilterAttribute	Represents the member filter used on attributes. This field is read-only.
FilterName	Represents the case-sensitive member filter used on names. This field is read-only.
FilterNameIgnoreCase	Represents the case-insensitive member filter used on names. This field is read-only.
Missing	Represents a missing value in the Type information. This field is read-only.

Properties

Assembly	Gets the Assembly in which the type is declared. For generic types, gets the Assembly in which the generic type is defined.
AssemblyQualifiedName	Gets the assembly-qualified name of the type, which includes the name of the assembly from which this Type object was loaded.
Attributes	Gets the attributes associated with the Type.
BaseType	Gets the type from which the current Type directly inherits.
ContainsGenericParameters	Gets a value indicating whether the current Type object has type parameters that have not been replaced by specific types.
CustomAttributes	Gets a collection that contains this member's custom attributes. (Inherited from MemberInfo)
DeclaringMethod	Gets a MethodBase that represents the declaring method, if the current Type represents a type parameter of a generic method.
DeclaringType	Gets the type that declares the current nested type or generic

	type parameter.
DefaultBinder	Gets a reference to the default binder, which implements internal rules for selecting the appropriate members to be called by InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[]).
FullName	Gets the fully qualified name of the type, including its namespace but not its assembly.
GenericParameterAttributes	Gets a combination of GenericParameterAttributes flags that describe the covariance and special constraints of the current generic type parameter.
GenericParameterPosition	Gets the position of the type parameter in the type parameter list of the generic type or method that declared the parameter, when the Type object represents a type parameter of a generic type or a generic method.
GenericTypeArguments	Gets an array of the generic type arguments for this type.
GUID	Gets the GUID associated with the Type.
HasElementType	Gets a value indicating whether the current Type encompasses or refers to another type; that is, whether the current Type is an array, a pointer, or is passed by reference.
IsAbstract	Gets a value indicating whether the Type is abstract and must be overridden.
IsAnsiClass	Gets a value indicating whether the string format attribute `AnsiClass` is selected for the Type.
IsArray	Gets a value that indicates whether the type is an array.
IsAutoClass	Gets a value indicating whether the string format attribute `AutoClass` is selected for the Type.
IsAutoLayout	Gets a value indicating whether the fields of the current type are laid out automatically by the common language runtime.
IsByRef	Gets a value indicating whether the Type is passed by reference.
IsByRefLike	Gets a value that indicates whether the type is a byref-like structure.
IsClass	Gets a value indicating whether the Type is a class or a delegate; that is, not a value type or interface.
IsCollectible	Gets a value that indicates whether this MemberInfo object is part of an assembly held in a collectible AssemblyLoadContext. (Inherited from MemberInfo)

IsCOMObject	Gets a value indicating whether the Type is a COM object.
IsConstructedGenericType	Gets a value that indicates whether this object represents a constructed generic type. You can create instances of a constructed generic type.
IsContextful	Gets a value indicating whether the Type can be hosted in a context.
IsEnum	Gets a value indicating whether the current Type represents an enumeration.
IsExplicitLayout	Gets a value indicating whether the fields of the current type are laid out at explicitly specified offsets.
IsGenericMethodParameter	Gets a value that indicates whether the current Type represents a type parameter in the definition of a generic method.
IsGenericParameter	Gets a value indicating whether the current Type represents a type parameter in the definition of a generic type or method.
IsGenericType	Gets a value indicating whether the current type is a generic type.
IsGenericTypeDefinition	Gets a value indicating whether the current Type represents a generic type definition, from which other generic types can be constructed.
IsGenericTypeParameter	Gets a value that indicates whether the current Type represents a type parameter in the definition of a generic type.
IsImport	Gets a value indicating whether the Type has a ComImportAttribute attribute applied, indicating that it was imported from a COM type library.
IsInterface	Gets a value indicating whether the Type is an interface; that is, not a class or a value type.
IsLayoutSequential	Gets a value indicating whether the fields of the current type are laid out sequentially, in the order that they were defined or emitted to the metadata.
IsMarshalByRef	Gets a value indicating whether the Type is marshaled by reference.
IsNested	Gets a value indicating whether the current Type object represents a type whose definition is nested inside the definition of another type.
IsNestedAssembly	Gets a value indicating whether the Type is nested and visible only within its own assembly.

IsNestedFamANDAssem	Gets a value indicating whether the Type is nested and visible only to classes that belong to both its own family and its own assembly.
IsNestedFamily	Gets a value indicating whether the Type is nested and visible only within its own family.
IsNestedFamORAssem	Gets a value indicating whether the Type is nested and visible only to classes that belong to either its own family or to its own assembly.
IsNestedPrivate	Gets a value indicating whether the Type is nested and declared private.
IsNestedPublic	Gets a value indicating whether a class is nested and declared public.
IsNotPublic	Gets a value indicating whether the Type is not declared public.
IsPointer	Gets a value indicating whether the Type is a pointer.
IsPrimitive	Gets a value indicating whether the Type is one of the primitive types.
IsPublic	Gets a value indicating whether the Type is declared public.
IsSealed	Gets a value indicating whether the Type is declared sealed.
IsSecurityCritical	Gets a value that indicates whether the current type is security-critical or security-safe-critical at the current trust level, and therefore can perform critical operations.
IsSecuritySafeCritical	Gets a value that indicates whether the current type is security-safe-critical at the current trust level; that is, whether it can perform critical operations and can be accessed by transparent code.
IsSecurityTransparent	Gets a value that indicates whether the current type is transparent at the current trust level, and therefore cannot perform critical operations.
IsSerializable	Gets a value indicating whether the Type is binary serializable.
IsSignatureType	Gets a value that indicates whether the type is a signature type.
IsSpecialName	Gets a value indicating whether the type has a name that requires special handling.
IsSZArray	Gets a value that indicates whether the type is an array type that can represent only a single-dimensional array with a zero lower bound.

IsTypeDefinition	Gets a value that indicates whether the type is a type definition.
IsUnicodeClass	Gets a value indicating whether the string format attribute `UnicodeClass` is selected for the Type.
IsValueType	Gets a value indicating whether the Type is a value type.
IsVariableBoundArray	Gets a value that indicates whether the type is an array type that can represent a multi-dimensional array or an array with an arbitrary lower bound.
IsVisible	Gets a value indicating whether the Type can be accessed by code outside the assembly.
MemberType	Gets a MemberTypes value indicating that this member is a type or a nested type.
MetadataToken	Gets a value that identifies a metadata element. (Inherited from MemberInfo)
Module	Gets the module (the DLL) in which the current Type is defined.
Name	Gets the name of the current member. (Inherited from MemberInfo)
Namespace	Gets the namespace of the Type.
ReflectedType	Gets the class object that was used to obtain this member.
StructLayoutAttribute	Gets a StructLayoutAttribute that describes the layout of the current type.
TypeHandle	Gets the handle for the current Type.
TypeInitializer	Gets the initializer for the type.
UnderlyingSystemType	Indicates the type provided by the common language runtime that represents this type.

Methods

Equals(Object)	Determines if the underlying system type of the current Type object is the same as the underlying system type of the specified Object.
Equals(Type)	Determines if the underlying system type of the current Type is the same as the underlying system type of the specified Type.

FindInterfaces(TypeFilter, Object)	Returns an array of Type objects representing a filtered list of interfaces implemented or inherited by the current Type.
FindMembers(MemberTypes, BindingFlags, MemberFilter, Object)	Returns a filtered array of MemberInfo objects of the specified member type.
GetArrayRank()	Gets the number of dimensions in an array.
GetAttributeFlagsImpl()	When overridden in a derived class, implements the Attributes property and gets a bitwise combination of enumeration values that indicate the attributes associated with the Type.
GetConstructor(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetConstructor(BindingFlags, Binder, Type[], ParameterModifier[])	Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetConstructor(BindingFlags, Type[])	Searches for a constructor whose parameters match the specified argument types, using the specified binding constraints.
GetConstructor(Type[])	Searches for a public instance constructor whose parameters match the types in the specified array.
GetConstructorImpl(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	When overridden in a derived class, searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetConstructors()	Returns all the public constructors defined for the current Type.
GetConstructors(BindingFlags)	When overridden in a derived class, searches for the constructors defined for the current Type, using the specified `BindingFlags`.
GetCustomAttributes(Boolean)	When overridden in a derived class, returns an array of all custom attributes applied to this member. (Inherited from MemberInfo)
GetCustomAttributes(Type, Boolean)	When overridden in a derived class, returns an array of custom attributes applied to this member and identified by Type. (Inherited from MemberInfo)

GetCustomAttributesData()	Returns a list of CustomAttributeData objects representing data about the attributes that have been applied to the target member. (Inherited from MemberInfo)
GetDefaultMembers()	Searches for the members defined for the current Type whose DefaultMemberAttribute is set.
GetElementType()	When overridden in a derived class, returns the Type of the object encompassed or referred to by the current array, pointer or reference type.
GetEnumName(Object)	Returns the name of the constant that has the specified value, for the current enumeration type.
GetEnumNames()	Returns the names of the members of the current enumeration type.
GetEnumUnderlyingType()	Returns the underlying type of the current enumeration type.
GetEnumValues()	Returns an array of the values of the constants in the current enumeration type.
GetEnumValuesAsUnderlyingType()	Retrieves an array of the values of the underlying type constants of this enumeration type.
GetEvent(String)	Returns the EventInfo object representing the specified public event.
GetEvent(String, BindingFlags)	When overridden in a derived class, returns the EventInfo object representing the specified event, using the specified binding constraints.
GetEvents()	Returns all the public events that are declared or inherited by the current Type.
GetEvents(BindingFlags)	When overridden in a derived class, searches for events that are declared or inherited by the current Type, using the specified binding constraints.
GetField(String)	Searches for the public field with the specified name.
GetField(String, BindingFlags)	Searches for the specified field, using the specified binding constraints.
GetFields()	Returns all the public fields of the current Type.
GetFields(BindingFlags)	When overridden in a derived class, searches for the fields defined for the current Type, using the specified binding constraints.

GetGenericArguments()	Returns an array of Type objects that represent the type arguments of a closed generic type or the type parameters of a generic type definition.
GetGenericParameterConstraints()	Returns an array of Type objects that represent the constraints on the current generic type parameter.
GetGenericTypeDefinition()	Returns a Type object that represents a generic type definition from which the current generic type can be constructed.
GetHashCode()	Returns the hash code for this instance.
GetInterface(String)	Searches for the interface with the specified name.
GetInterface(String, Boolean)	When overridden in a derived class, searches for the specified interface, specifying whether to do a case-insensitive search for the interface name.
GetInterfaceMap(Type)	Returns an interface mapping for the specified interface type.
GetInterfaces()	When overridden in a derived class, gets all the interfaces implemented or inherited by the current Type.
GetMember(String)	Searches for the public members with the specified name.
GetMember(String, BindingFlags)	Searches for the specified members, using the specified binding constraints.
GetMember(String, MemberTypes, BindingFlags)	Searches for the specified members of the specified member type, using the specified binding constraints.
GetMembers()	Returns all the public members of the current Type.
GetMembers(BindingFlags)	When overridden in a derived class, searches for the members defined for the current Type, using the specified binding constraints.
GetMemberWithSameMetadata DefinitionAs(MemberInfo)	Searches for the MemberInfo on the current Type that matches the specified MemberInfo.
GetMethod(String)	Searches for the public method with the specified name.
GetMethod(String, BindingFlags)	Searches for the specified method, using the specified binding constraints.
GetMethod(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.

GetMethod(String, BindingFlags, Binder, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetMethod(String, BindingFlags, Type[])	Searches for the specified method whose parameters match the specified argument types, using the specified binding constraints.
GetMethod(String, Int32, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethod(String, Int32, BindingFlags, Binder, Type[], ParameterModifier[])	Searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints.
GetMethod(String, Int32, Type[])	Searches for the specified public method whose parameters match the specified generic parameter count and argument types.
GetMethod(String, Int32, Type[], ParameterModifier[])	Searches for the specified public method whose parameters match the specified generic parameter count, argument types and modifiers.
GetMethod(String, Type[])	Searches for the specified public method whose parameters match the specified argument types.
GetMethod(String, Type[], ParameterModifier[])	Searches for the specified public method whose parameters match the specified argument types and modifiers.
GetMethodImpl(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	When overridden in a derived class, searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethodImpl(String, Int32, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[])	When overridden in a derived class, searches for the specified method whose parameters match the specified generic parameter count, argument types and modifiers, using the specified binding constraints and the specified calling convention.
GetMethods()	Returns all the public methods of the current Type.
GetMethods(BindingFlags)	When overridden in a derived class, searches for the methods defined for the current Type, using the specified binding constraints.

GetNestedType(String)	Searches for the public nested type with the specified name.
GetNestedType(String, BindingFlags)	When overridden in a derived class, searches for the specified nested type, using the specified binding constraints.
GetNestedTypes()	Returns the public types nested in the current Type.
GetNestedTypes(BindingFlags)	When overridden in a derived class, searches for the types nested in the current Type, using the specified binding constraints.
GetProperties()	Returns all the public properties of the current Type.
GetProperties(BindingFlags)	When overridden in a derived class, searches for the properties of the current Type, using the specified binding constraints.
GetProperty(String)	Searches for the public property with the specified name.
GetProperty(String, BindingFlags)	Searches for the specified property, using the specified binding constraints.
GetProperty(String, BindingFlags, Binder, Type, Type[], Parameter Modifier[])	Searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetProperty(String, Type)	Searches for the public property with the specified name and return type.
GetProperty(String, Type, Type[])	Searches for the specified public property whose parameters match the specified argument types.
GetProperty(String, Type, Type[], ParameterModifier[])	Searches for the specified public property whose parameters match the specified argument types and modifiers.
GetProperty(String, Type[])	Searches for the specified public property whose parameters match the specified argument types.
GetPropertyImpl(String, BindingFlags, Binder, Type, Type[], Parameter Modifier[])	When overridden in a derived class, searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.
GetType()	Gets the current Type.
GetType(String)	Gets the Type with the specified name, performing a case-sensitive search.

GetType(String, Boolean)	Gets the Type with the specified name, performing a case-sensitive search and specifying whether to throw an exception if the type is not found.
GetType(String, Boolean, Boolean)	Gets the Type with the specified name, specifying whether to throw an exception if the type is not found and whether to perform a case-sensitive search.
GetType(String, Func<Assembly Name,Assembly>, Func<Assembly,String,Boolean,Type>)	Gets the type with the specified name, optionally providing custom methods to resolve the assembly and the type.
GetType(String, Func<Assembly Name,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean)	Gets the type with the specified name, specifying whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.
GetType(String, Func<Assembly Name,Assembly>, Func<Assembly,String,Boolean,Type>, Boolean, Boolean)	Gets the type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.
GetTypeArray(Object[])	Gets the types of the objects in the specified array.
GetTypeCode(Type)	Gets the underlying type code of the specified Type.
GetTypeCodeImpl()	Returns the underlying type code of this Type instance.
GetTypeFromCLSID(Guid)	Gets the type associated with the specified class identifier (CLSID).
GetTypeFromCLSID(Guid, Boolean)	Gets the type associated with the specified class identifier (CLSID), specifying whether to throw an exception if an error occurs while loading the type.
GetTypeFromCLSID(Guid, String)	Gets the type associated with the specified class identifier (CLSID) from the specified server.
GetTypeFromCLSID(Guid, String, Boolean)	Gets the type associated with the specified class identifier (CLSID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.
GetTypeFromHandle(RuntimeType Handle)	Gets the type referenced by the specified type handle.
GetTypeFromProgID(String)	Gets the type associated with the specified program identifier (ProgID), returning null if an error is encountered while loading the Type.

GetTypeFromProgID(String, Boolean)	Gets the type associated with the specified program identifier (ProgID), specifying whether to throw an exception if an error occurs while loading the type.
GetTypeFromProgID(String, String)	Gets the type associated with the specified program identifier (progID) from the specified server, returning null if an error is encountered while loading the type.
GetTypeFromProgID(String, String, Boolean)	Gets the type associated with the specified program identifier (progID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.
GetTypeHandle(Object)	Gets the handle for the Type of a specified object.
HasElementTypeImpl()	When overridden in a derived class, implements the HasElementType property and determines whether the current Type encompasses or refers to another type; that is, whether the current Type is an array, a pointer, or is passed by reference.
HasSameMetadataDefinitionAs(MemberInfo)	(Inherited from MemberInfo)
InvokeMember(String, BindingFlags, Binder, Object, Object[])	Invokes the specified member, using the specified binding constraints and matching the specified argument list.
InvokeMember(String, BindingFlags, Binder, Object, Object[], CultureInfo)	Invokes the specified member, using the specified binding constraints and matching the specified argument list and culture.
InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[])	When overridden in a derived class, invokes the specified member, using the specified binding constraints and matching the specified argument list, modifiers and culture.
IsArrayImpl()	When overridden in a derived class, implements the IsArray property and determines whether the Type is an array.
IsAssignableFrom(Type)	Determines whether an instance of a specified type `c` can be assigned to a variable of the current type.
IsAssignableTo(Type)	Determines whether the current type can be assigned to a variable of the specified `targetType`.
IsByRefImpl()	When overridden in a derived class, implements the IsByRef property and determines whether the Type is passed by reference.

IsCOMObjectImpl()	When overridden in a derived class, implements the IsCOMObject property and determines whether the Type is a COM object.
IsContextfulImpl()	Implements the IsContextful property and determines whether the Type can be hosted in a context.
IsDefined(Type, Boolean)	When overridden in a derived class, indicates whether one or more attributes of the specified type or of its derived types is applied to this member. (Inherited from MemberInfo)
IsEnumDefined(Object)	Returns a value that indicates whether the specified value exists in the current enumeration type.
IsEquivalentTo(Type)	Determines whether two COM types have the same identity and are eligible for type equivalence.
IsInstanceOfType(Object)	Determines whether the specified object is an instance of the current Type.
IsMarshalByRefImpl()	Implements the IsMarshalByRef property and determines whether the Type is marshaled by reference.
IsPointerImpl()	When overridden in a derived class, implements the IsPointer property and determines whether the Type is a pointer.
IsPrimitiveImpl()	When overridden in a derived class, implements the IsPrimitive property and determines whether the Type is one of the primitive types.
IsSubclassOf(Type)	Determines whether the current Type derives from the specified Type.
IsValueTypeImpl()	Implements the IsValueType property and determines whether the Type is a value type; that is, not a class or an interface.
MakeArrayType()	Returns a Type object representing a one-dimensional array of the current type, with a lower bound of zero.
MakeArrayType(Int32)	Returns a Type object representing an array of the current type, with the specified number of dimensions.
MakeByRefType()	Returns a Type object that represents the current type when passed as a `ref` parameter (`ByRef` parameter in Visual Basic).
MakeGenericMethodParameter(Int32)	Returns a signature type object that can be passed into the `Type[]` array parameter of a GetMethod method to

	represent a generic parameter reference.
MakeGenericSignatureType(Type, Type[])	Creates a generic signature type, which allows third party reimplementations of Reflection to fully support the use of signature types in querying type members.
MakeGenericType(Type[])	Substitutes the elements of an array of types for the type parameters of the current generic type definition and returns a Type object representing the resulting constructed type.
MakePointerType()	Returns a Type object that represents a pointer to the current type.
MemberwiseClone()	Creates a shallow copy of the current Object. (Inherited from Object)
ReflectionOnlyGetType(String, Boolean, Boolean)	**Obsolete.** Gets the Type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found. The type is loaded for reflection only, not for execution.
ToString()	Returns a `String` representing the name of the current `Type`.

Operators

Equality(Type, Type)	Indicates whether two Type objects are equal.
Inequality(Type, Type)	Indicates whether two Type objects are not equal.

Extension Methods

GetCustomAttribute(MemberInfo, Type)	Retrieves a custom attribute of a specified type that is applied to a specified member.
GetCustomAttribute(MemberInfo, Type, Boolean)	Retrieves a custom attribute of a specified type that is applied to a specified member, and optionally inspects the ancestors of that member.
GetCustomAttribute<T>(MemberInfo)	Retrieves a custom attribute of a specified type that is applied to a specified member.
GetCustomAttribute<T>(MemberInfo, Boolean)	Retrieves a custom attribute of a specified type that is applied to a specified member, and optionally inspects the ancestors of

	that member.
GetCustomAttributes(MemberInfo)	Retrieves a collection of custom attributes that are applied to a specified member.
GetCustomAttributes(MemberInfo, Boolean)	Retrieves a collection of custom attributes that are applied to a specified member, and optionally inspects the ancestors of that member.
GetCustomAttributes(MemberInfo, Type)	Retrieves a collection of custom attributes of a specified type that are applied to a specified member.
GetCustomAttributes(MemberInfo, Type, Boolean)	Retrieves a collection of custom attributes of a specified type that are applied to a specified member, and optionally inspects the ancestors of that member.
GetCustomAttributes<T>(MemberInfo)	Retrieves a collection of custom attributes of a specified type that are applied to a specified member.
GetCustomAttributes<T>(MemberInfo, Boolean)	Retrieves a collection of custom attributes of a specified type that are applied to a specified member, and optionally inspects the ancestors of that member.
IsDefined(MemberInfo, Type)	Indicates whether custom attributes of a specified type are applied to a specified member.
IsDefined(MemberInfo, Type, Boolean)	Indicates whether custom attributes of a specified type are applied to a specified member, and, optionally, applied to its ancestors.
GetTypeInfo(Type)	Returns the TypeInfo representation of the specified type.
GetMetadataToken(MemberInfo)	Gets a metadata token for the given member, if available.
HasMetadataToken(MemberInfo)	Returns a value that indicates whether a metadata token is available for the specified member.
GetRuntimeEvent(Type, String)	Retrieves an object that represents the specified event.
GetRuntimeEvents(Type)	Retrieves a collection that represents all the events defined on a specified type.
GetRuntimeField(Type, String)	Retrieves an object that represents a specified field.
GetRuntimeFields(Type)	Retrieves a collection that represents all the fields defined on a specified type.
GetRuntimeMethod(Type, String, Type[])	Retrieves an object that represents a specified method.
GetRuntimeMethods(Type)	Retrieves a collection that represents all methods defined on a

	specified type.
GetRuntimeProperties(Type)	Retrieves a collection that represents all the properties defined on a specified type.
GetRuntimeProperty(Type, String)	Retrieves an object that represents a specified property.
GetConstructor(Type, Type[])	
GetConstructors(Type)	
GetConstructors(Type, BindingFlags)	
GetDefaultMembers(Type)	
GetEvent(Type, String)	
GetEvent(Type, String, BindingFlags)	
GetEvents(Type)	
GetEvents(Type, BindingFlags)	
GetField(Type, String)	
GetField(Type, String, BindingFlags)	
GetFields(Type)	
GetFields(Type, BindingFlags)	
GetGenericArguments(Type)	
GetInterfaces(Type)	
GetMember(Type, String)	
GetMember(Type, String, BindingFlags)	
GetMembers(Type)	
GetMembers(Type, BindingFlags)	
GetMethod(Type, String)	
GetMethod(Type, String, BindingFlags)	
GetMethod(Type, String, Type[])	
GetMethods(Type)	
GetMethods(Type, BindingFlags)	

GetNestedType(Type, String, BindingFlags)
GetNestedTypes(Type, BindingFlags)
GetProperties(Type)
GetProperties(Type, BindingFlags)
GetProperty(Type, String)
GetProperty(Type, String, BindingFlags)
GetProperty(Type, String, Type)
GetProperty(Type, String, Type, Type[])
IsAssignableFrom(Type, Type)
IsInstanceOfType(Type, Object)

Applies to

Product	Versions
.NET	Core 1.0, Core 1.1, Core 2.0, Core 2.1, Core 2.2, Core 3.0, Core 3.1, 5, 6, 7, 8
.NET Framework	1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1, 4.5.2, 4.6, 4.6.1, 4.6.2, 4.7, 4.7.1, 4.7.2, 4.8, 4.8.1
.NET Standard	1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 2.0, 2.1
UWP	10.0
Xamarin.iOS	10.8
Xamarin.Mac	3.0

Thread Safety

This type is thread safe.

See also

- Object
- System.Reflection
- ReflectionPermission
- Viewing Type Information