Access Modifiers (C# Programming Guide)

07/20/2015 • 4 minutes to read • Contributors all

In this article

All types and type members have an accessibility level, which controls whether they can be used from other code in your assembly or other assemblies. You can use the following access modifiers to specify the accessibility of a type or member when you declare it:

public
The type or member can be accessed by any other code in the same assembly or another assembly that references it.

private
The type or member can be accessed only by code in the same class or struct.

protected
The type or member can be accessed only by code in the same class, or in a class that is derived from that class.
internal
The type or member can be accessed by any code in the same assembly, but not from another assembly.

protected internal The type or member can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly.

private protected The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.

The following examples demonstrate how to specify access modifiers on a type and member:

C# Copy

```csharp
public class Bicycle
{
    public void Pedal() { }
}
```

Not all access modifiers can be used by all types or members in all contexts, and in some cases the accessibility of a type member is constrained by the accessibility of its containing type. The following sections provide more details about accessibility.

Class and Struct Accessibility

Classes and structs that are declared directly within a namespace (in other words, that are not nested within other classes or structs) can be either public or internal. Internal is the default if no access modifier is specified.

Struct members, including nested classes and structs, can be declared as public, internal, or private. Class members, including nested classes and structs, can be public, protected internal, protected, internal, private protected or private. The access level for class members and struct members, including nested classes and structs, is private by default. Private nested types are not accessible from outside the containing type.

Derived classes cannot have greater accessibility than their base types. In other words, you cannot have a public class `B` that derives from an internal class `A`. If this were allowed, it would have the effect of making `A` public, because all protected or internal members of `A` are accessible from the derived class.

You can enable specific other assemblies to access your internal types by using the InternalsVisibleToAttribute. For more information, see Friend Assemblies.

Class and Struct Member Accessibility

Class members (including nested classes and structs) can be declared with any of the six types of access. Struct members cannot be declared as protected because structs do not support inheritance.

Normally, the accessibility of a member is not greater than the accessibility of the type that contains it. However, a public member of an internal class might be accessible from outside the assembly if the member implements interface methods or overrides virtual methods that are defined in a public base class.

The type of any member that is a field, property, or event must be at least as accessible as the member itself. Similarly, the return type and the parameter types of any member that is a method, indexer, or delegate must be at least as accessible as the member itself. For example, you cannot have a public method `M` that returns a class `C` unless `C` is also public. Likewise, you cannot have a protected property of type `A` if `A` is declared as private.

User-defined operators must always be declared as public. For more information, see operator (C# Reference).

Finalizers cannot have accessibility modifiers.

To set the access level for a class or struct member, add the appropriate keyword to the member declaration, as shown in the following example.

C# Copy

```csharp
// public class:
public class Tricycle
{
    // protected method:
    protected void Pedal() { }
    // private field:
    private int wheels = 3;

    // protected internal property:
    protected internal int Wheels
    {
        get { return wheels; }
    }
}
```

Note

The protected internal accessibility level means protected OR internal, not protected AND internal. In other words, a protected internal member can be accessed from any class in the same assembly, including derived classes. To limit accessibility to only derived classes in the same assembly, declare the class itself internal, and declare its members as protected. Also, starting with C# 7.2, you can use the private protected access modifier to achieve the same result without need to make the containing class internal.

Other Types

Interfaces declared directly within a namespace can be declared as public or internal and, just like classes and structs, interfaces default to internal access. Interface members are always public because the purpose of an interface is to enable other types to access a class or struct. No access modifiers can be applied to interface members.

Enumeration members are always public, and no access modifiers can be applied.

Delegates behave like classes and structs. By default, they have internal access when declared directly within a namespace, and private access when nested.

C# Language Specification

For more information, see the C# Language Specification. The language specification is the definitive source for C# syntax and usage.

See also

- C# Programming Guide
- Classes and Structs
- Interfaces
- private
- public
- internal
- protected
- protected internal
- private protected
- class
- struct
- interface