Access Modifiers (C# Programming Guide)

Article • 08/20/2024

All types and type members have an accessibility level. The accessibility level controls whether they can be used from other code in your assembly or other assemblies. An assembly is a *.dll* or *.exe* created by compiling one or more *.cs* files in a single compilation. Use the following access modifiers to specify the accessibility of a type or member when you declare it:

- public: Code in any assembly can access this type or member. The accessibility level of the containing type controls the accessibility level of public members of the type.
- private: Only code declared in the same `class` or `struct` can access this member.
- protected: Only code in the same `class` or in a derived `class` can access this type or member.
- internal: Only code in the same assembly can access this type or member.
- protected internal: Only code in the same assembly *or* in a derived class in another assembly can access this type or member.
- private protected: Only code in the same assembly *and* in the same class or a derived class can access the type or member.
- file: Only code in the same file can access the type or member.

The record modifier on a type causes the compiler to synthesize extra members. The `record` modifier doesn't affect the default accessibility for either a `record class` or a `record struct`.

Summary table

Expand table

Caller's location	`public`	`protected internal`	`protected`	`internal`	`private protected`	`private`	`file`
Within the file	✔	✔	✔	✔	✔	✔	✔
Within the class	✔	✔	✔	✔	✔	✔	✘

Caller's location	`public`	`protected internal`	`protected`	`internal`	`private protected`	`private`	`file`
Derived class (same assembly)	✔	✔	✔	✔	✔	✘	✘
Non-derived class (same assembly)	✔	✔	✘	✔	✘	✘	✘
Derived class (different assembly)	✔	✔	✔	✘	✘	✘	✘
Non-derived class (different assembly)	✔	✘	✘	✘	✘	✘	✘

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

C#

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

Not all access modifiers are valid for all types or members in all contexts. In some cases, the accessibility of the containing type constrains the accessibility of its members.

Multiple declarations of a partial class or partial member must have the same accessibility. If one declaration of the partial class or member doesn't include an access modifier, the other declarations can't declare an access modifier. The compiler generates an error if multiple declarations for the partial class or method declare different accessibilities.

Class and struct accessibility

Classes and structs declared directly within a namespace (aren't nested within other classes or structs) can have `public`, `internal` or `file` access. `internal` is the default if no access modifier is specified.

Struct members, including nested classes and structs, can be declared `public`, `internal`, or `private`. Class members, including nested classes and structs, can be `public`, `protected internal`, `protected`, `internal`, `private protected`, or `private`. Class and struct members, including nested classes and structs, have `private` access by default.

Derived classes can't have greater accessibility than their base types. You can't declare a public class `B` that derives from an internal class `A`. If 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.

Other types

Interfaces declared directly within a namespace can be `public` or `internal` and, just like classes and structs, interfaces default to `internal` access. Interface members are `public` by default because the purpose of an interface is to enable other types to access a class or struct. Interface member declarations might include any access modifier. You use access modifiers on `interface` members to provide a common implementation needed by all implementors of an interface.

A delegate type declared directly in a namespace has `internal` access by default.

For more information about access modifiers, see the Accessibility Levels page.

Member accessibility

Members of a `class` or `struct` (including nested classes and structs) can be declared with any of the six types of access. Struct members can't be declared as `protected`, `protected internal`, or `private protected` because structs don't support inheritance.

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

User-defined operators must always be declared as `public` and `static`. For more information, see Operator overloading.

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#

```
// 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; }
    }
}
```

Finalizers can't have accessibility modifiers. Members of an `enum` type are always `public`, and no access modifiers can be applied.

The `file` access modifier is allowed only on top-level (non-nested) type declarations.

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

- Specify modifier order (style rule IDE0036)
- The C# type system
- Interfaces
- Accessibility Levels
- private
- public
- internal
- protected
- protected internal
- private protected
- sealed
- class
- struct
- interface
- Anonymous types