

# C# Compiler Errors

Article • 05/13/2022 • 2 minutes to read

Some C# compiler errors have corresponding topics that explain why the error is generated, and, in some cases, how to fix the error. Use one of the following steps to see whether help is available for a particular error message.

- If you're using Visual Studio, choose the error number (for example, CS0029) in the [Output Window](#), and then choose the F1 key.
- Type the error number in the *Filter by title* box in the table of contents.

If none of these steps leads to information about your error, go to the end of this page, and send feedback that includes the number or text of the error.

For information about how to configure error and warning options in C#, see [C# compiler options](#) or the Visual Studio [Build Page, Project Designer \(C#\)](#).

## ⓘ Note

Your computer might show different names or locations for some of the Visual Studio user interface elements in the following instructions. The Visual Studio edition that you have and the settings that you use determine these elements. For more information, see [Personalizing the IDE](#).

## See also

- [C# Compiler Options](#)
- [Build Page, Project Designer \(C#\)](#)
- [WarningLevel \(C# Compiler Options\)](#)
- [NoWarn \(C# Compiler Options\)](#)

# Compiler Error CS0001

Article • 09/15/2021 • 2 minutes to read

Internal compiler error

Try to determine whether the compiler is failing because of its inability to parse unexpected syntax. If you receive this error repeatedly, please contact Microsoft.

# Compiler Error CS0003

Article • 09/15/2021 • 2 minutes to read

## Out of memory

The compiler was unable to allocate enough virtual memory to complete compilation.  
Close all unnecessary applications and compile again.

You may also want to increase the pagefile size and make sure that you have free disk space.

This error may also appear as the result of having mismatched versions of the Windows SDK and the C# compiler or one or more corrupted files that support the C# compiler; reinstall Visual Studio.

# Compiler Error CS0004

Article •

• 2 minutes to read

## Warning treated as error

You compiled with the `/warnaserror` compiler option, which causes the compiler to generate warnings as errors.

# Compiler Error CS0005

Article • 09/15/2021 • 2 minutes to read

Compiler option 'compiler\_option' must be followed by an argument

Some compiler options require parameters. If you do not pass the arguments required by the compiler option, CS0005 is generated.

For more information, see [C# Compiler Options](#).

# Compiler Error CS0006

Article • 09/15/2021 • 2 minutes to read

Metadata file 'dll\_name' could not be found

The program was compiled and explicitly passed the name of a file that contained metadata; however, the .dll was not found. For more information, see [References \(C# Compiler Options\)](#).

# Compiler Error CS0007

Article • 09/15/2021 • 2 minutes to read

Unexpected common language runtime initialization error — 'description'

This error occurs if the runtime could not be loaded. This could occur if the version of the common language runtime that the compiler attempts to load is not present on the machine, or if the common language runtime installation or configuration is corrupt.

This can happen if the *csc.exe.config* file was changed. This file is configured during setup and should not be changed. If there is a possibility that the *csc.exe.config* file was changed, check the file to make sure that the version of the runtime specified in the file is present on the machine. If the correct version is present, it may be corrupted. Reinstall the common language runtime.

# Compiler Error CS0008

Article • 09/15/2021 • 2 minutes to read

Unexpected error reading metadata from file 'file' — 'description'

A DLL was successfully opened for retrieving metadata but is corrupted, such that data could not be read from it. For more information, see [References \(C# Compiler Options\)](#).

# Compiler Error CS0009

Article • 09/15/2021 • 2 minutes to read

Metadata file 'file' could not be opened — 'description'

The file specified with the [References](#) compiler option does not contain valid metadata.

# Compiler Error CS0010

Article • 09/15/2021 • 2 minutes to read

Unexpected fatal error -- 'error'.

This error is generated when something completely unexpected occurs to stop compilation.

## To correct this error

Recompile another project. If you receive the same error, try reinstalling Visual Studio. If you receive this error, send an error report to Microsoft.

# Compiler Error CS0011

Article • 09/15/2021 • 2 minutes to read

The base class or interface 'class' in assembly 'assembly' referenced by type 'type' could not be resolved

A class that was imported from a file with `/reference`, is derived from a class or implements an interface that is not found. This can occur if a required DLL is not also included in the compilation with `/reference`.

For more information, see [Add Reference Dialog Box](#) and [References \(C# Compiler Options\)](#).

## Examples

C#

```
// CS0011_1.cs
// compile with: /target:library

public class Outer
{
    public class B { }
}
```

The second file creates a DLL that defines a class `C` that is derived from the class `B` that was created in the previous example.

C#

```
// CS0011_2.cs
// compile with: /target:library /reference:CS0011_1.dll
// post-build command: del /f CS0011_1.dll
public class C : Outer.B {}
```

The third file replaces the DLL created by the first step, and omits the definition of the inner class `B`.

C#

```
// CS0011_3.cs
// compile with: /target:library /out:cs0011_1.dll
public class Outer {}
```

Finally, the fourth file references the class `C` defined in the second example, which is derived from class `B`, and which is now missing.

The following sample generates CS0011.

C#

```
// CS0011_4.cs
// compile with: /reference:CS0011_1.dll /reference:CS0011_2.dll
// CS0011 expected

class M
{
    public static void Main()
    {
        C c = new C();
    }
}
```

# Compiler Error CS0012

Article • 09/15/2021 • 2 minutes to read

The type 'type' is defined in an assembly that is not referenced. You must add a reference to assembly 'assembly'.

The definition for a referenced type was not found. This could occur if a required .DLL file is not included in the compilation. For more information, see [Add Reference Dialog Box](#) and [References \(C# Compiler Options\)](#).

The following sequence of compilations will result in CS0012:

```
C#  
  
// cs0012a.cs  
// compile with: /target:library  
public class A {}
```

Then:

```
C#  
  
// cs0012b.cs  
// compile with: /target:library /reference:cs0012a.dll  
public class B  
{  
    public static A f()  
    {  
        return new A();  
    }  
}
```

Then:

```
C#  
  
// cs0012c.cs  
// compile with: /reference:cs0012b.dll  
class C  
{  
    public static void Main()  
    {  
        object o = B.f(); // CS0012  
    }  
}
```

You could resolve this CS0012 by compiling with `/reference:cs0012b.dll;cs0012a.dll`, or in Visual Studio by using the [Add Reference Dialog Box](#) to add a reference to `cs0012a.dll` in addition to `cs0012b.dll`.

# Compiler Error CS0013

Article • 09/15/2021 • 2 minutes to read

Unexpected error writing metadata to file 'ModelStore.dll' -- 'No logical space left to create more user strings.

The .NET runtime did not emit metadata. Check to make sure that the path is correct and that the disk is not full. If the problem persists, you may have to repair or reinstall Visual Studio and/or .NET.

# Compiler Error CS0014

Article • 09/15/2021 • 2 minutes to read

Required file 'file' could not be found

A file is needed by the compiler but is not on the system. Make sure that the path is correct. If the file is a Visual Studio system file, then you may need to repair your installation or else remove and reinstall Visual Studio completely.

# Compiler Error CS0015

Article • 09/15/2021 • 2 minutes to read

The name of type 'type' is too long

The fully qualified name of a user-defined type must have fewer than 1024 characters, including the periods.

The following sample generates CS0015:

C#

```
// CS0015.cs

// Remove a C from one of the namespace names or the class name to resolve
// the error by
// reducing the number of characters in the qualified class name to fewer
// than 1024.

namespace CC
{
    // The following namespace name has 510 characters.
    namespace
CCCC510charsCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    {
        // The following class name has 510 characters. The qualified class
name,
        // CC.CCC510chars..C.CCCC510chars..C (namespace.namespace.class), has
1024
        // characters, which causes compiler error CS0015.
        public class
CCCC510charsCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
    {
        public static void Main()
        {
        }
    }
}
```



# Compiler Error CS0016

Article • 09/15/2021 • 2 minutes to read

Could not write to output file 'file' — 'reason'

The compiler could not write to an output file. Check the path to the file to make sure it exists. If a previously built file is already at the location, make sure it is writeable, and that no process currently has a lock on the file. For example, make sure your executable is not running when you attempt to build.

# Compiler Error CS0017

Article • 09/15/2021 • 2 minutes to read

Program 'output file name' has more than one entry point defined. Compile with /main to specify the type that contains the entry point.

A program can only have one [Main](#) method.

To resolve this error, you can either delete all Main methods in your code, except one, or you can use the [StartupObject](#) compiler option to specify which Main method you want to use.

The following sample generates CS0017:

C#

```
// CS0017.cs
// compile with: /target:exe
public class clk
{
    static public void Main()
    {
    }
}

public class cly
{
    public static void Main()    // CS0017, delete one Main or use /main
    {
    }
}
```

# Compiler Error CS0019

Article • 09/15/2021 • 2 minutes to read

Operator 'operator' cannot be applied to operands of type 'type' and 'type'

A binary operator is applied to data types that do not support it. For example, you cannot use the `||` operator on strings, you cannot use `+`, `-`, `<`, or `>` operators on `bool` variables, and you cannot use the `==` operator with a `struct` type unless the type explicitly overloads that operator.

You can overload an operator to make it support operands of certain types. For more information, see [Operator overloading](#).

## Example 1

In the following example, CS0019 is generated in three places because `bool` in C# is not convertible to `int`. CS0019 is also generated when the subtraction operator `-` is applied to a string. The addition operator `+` can be used with string operands because that operator is overloaded by the `String` class to perform string concatenation.

C#

```
static void Main()
{
    bool result = true;
    if (result > 0) //CS0019
    {
        // Do something.
    }

    int i = 1;
    // You cannot compare an integer and a boolean value.
    if (i == true) //CS0019
    {
        //Do something...
    }

    string s = "Just try to subtract me.";
    float f = 100 - s; // CS0019
}
```

## Example 2

In the following example, conditional logic must be specified outside the `ConditionalAttribute`. You can pass only one predefined symbol to the `ConditionalAttribute`.

The following sample generates CS0019:

```
C#  
  
// CS0019_a.cs  
// compile with: /target:library  
using System.Diagnostics;  
  
public class MyClass  
{  
    [ConditionalAttribute("DEBUG" || "TRACE")] // CS0019  
    public void TestMethod() {}  
  
    // OK  
    [ConditionalAttribute("DEBUG"), ConditionalAttribute("TRACE")]  
    public void TestMethod2() {}  
}
```

## See also

- [C# operators](#)

# Compiler Error CS0020

Article • 09/15/2021 • 2 minutes to read

## Division by constant zero

An expression uses a literal (not variable) value of zero in the denominator of a division operation. Division by zero is not defined, and is therefore invalid.

The following sample generates CS0020:

C#

```
// CS0020.cs
namespace x
{
    public class b
    {
        public static void Main()
        {
            1 / 0;    // CS0020
        }
    }
}
```

## See also

- [Arithmetic operators](#)



# Compiler Error CS0021

Article • 09/15/2021 • 2 minutes to read

Cannot apply indexing with [] to an expression of type 'type'

An attempt was made to access a value through an indexer on a data type that does not support [Indexers](#).

You may get CS0021 when trying to use an indexer in a C++ assembly. In this case, decorate the C++ class with the `DefaultMember` attribute so the C# compiler knows which indexer is the default. The following sample generates CS0021.

## Example

The following C++ example compiles to a .dll file. Note that the `DefaultMember` attribute is commented out in order to generate the error.

C++

```
// CPP0021.cpp
// compile with: /clr /LD
using namespace System::Reflection;
// Uncomment the following line to resolve
//[DefaultMember("myItem")]
public ref class MyClassMC
{
    public:
        property int myItem[int]
        {
            int get(int i){ return 5; }
            void set(int i, int value) {}
        }
};
```

The following C# example calls the .dll file. It attempts to access the class via an indexer, but because no member was declared as the default indexer, the error is generated. You can address the error by uncommenting the line `[DefaultMember("myItem")]` in the .cpp file in the preceding example.

C#

```
// CS0021.cs
// compile with: /reference:CPP0021.dll
public class MyClass
{
```

```
public static void Main()
{
    MyClassMC myMC = new MyClassMC();
    int j = myMC[1]; // CS0021
}
```

# Compiler Error CS0022

Article • 09/15/2021 • 2 minutes to read

Wrong number of indices inside [], expected 'number'

An array-access operation specified the incorrect number of dimensions within the square brackets. For more information, see [Arrays](#).

## Example

The following sample generates CS0022:

C#

```
// CS0022.cs
public class MyClass
{
    public static void Main()
    {
        int[] a = new int[10];
        a[0] = 0;      // single-dimension array
        a[0,1] = 9;   // CS0022, the array does not have two dimensions
    }
}
```

# Compiler Error CS0023

Article • 10/27/2021 • 2 minutes to read

Operator 'operator' cannot be applied to operand of type 'type'

An attempt was made to apply an operator to a variable whose type was not designed to work with the operator. For more information, see [Types](#) and [C# Operators](#).

The following sample generates CS0023:

C#

```
// CS0023.cs
namespace x
{
    public class a
    {
        public static void Main()
        {
            string s = "hello";
            s = -s;      // CS0023, minus operator not allowed on strings
        }
    }
}
```

# Compiler Error CS0025

Article • 09/15/2021 • 2 minutes to read

Standard library file 'file' could not be found

A file that is needed by the compiler was not found. Check that the path is correct and that the file exists.

If the file is a Visual Studio system file, you may need to repair your Visual Studio installation, or reinstall it completely.

# Compiler Error CS0026

Article • 09/15/2021 • 2 minutes to read

Keyword 'this' is not valid in a static property, static method, or static field initializer

The [this](#) keyword refers to an object, which is an instance of a type. Since static methods are independent of any instance of the containing class, the "this" keyword is meaningless and is therefore not allowed. For more information, see [Static Classes and Static Class Members and Objects](#).

## Example

The following example generates CS0026:

```
C#  
  
// CS0026.cs  
public class A  
{  
    public static int i = 0;  
  
    public static void Main()  
    {  
        // CS0026  
        this.i = this.i + 1;  
        // Try the following line instead:  
        // i = i + 1;  
    }  
}
```

# Compiler Error CS0027

Article • 09/15/2021 • 2 minutes to read

Keyword 'this' is not available in the current context

The `this` keyword was found outside of a property, method, or constructor.

To fix this error, either modify the statement to eliminate use of the `this` keyword, and/or move part or all of the statement inside a property, method, or constructor.

The following example generates CS0027:

```
C#  
  
using System;  
using System.Collections.Generic;  
using System.Text;  
  
namespace ConsoleApplication3  
{  
    class MyClass  
    {  
  
        int err1 = this.Fun() + 1; // CS0027  
  
        public int Fun()  
        {  
            return 10;  
        }  
  
        public void Test()  
        {  
            // valid use of this  
            int err = this.Fun() + 1;  
            Console.WriteLine(err);  
        }  
  
        public static void Main()  
        {  
            MyClass c = new MyClass();  
            c.Test();  
        }  
    }  
}
```

# Compiler Error CS0029

Article • 09/15/2021 • 3 minutes to read

Cannot implicitly convert type 'type' to 'type'

The compiler requires an explicit conversion. For example, you may need to cast an r-value to be the same type as an l-value. Or, you must provide conversion routines to support certain operator overloads.

Conversions must occur when assigning a variable of one type to a variable of a different type. When making an assignment between variables of different types, the compiler must convert the type on the right-hand side of the assignment operator to the type on the left-hand side of the assignment operator. Take the following code:

C#

```
int i = 50;
long lng = 100;
i = lng;
```

`i = lng;` makes an assignment, but the data types of the variables on the left and right-hand side of the assignment operator don't match. Before making the assignment the compiler is implicitly converting the variable `lng`, which is of type long, to an int. This is implicit because no code explicitly instructed the compiler to perform this conversion. The problem with this code is that this is considered a narrowing conversion, and the compiler does not allow implicit narrowing conversions because there could be a potential loss of data.

A narrowing conversion exists when converting to a data type that occupies less storage space in memory than the data type we are converting from. For example, converting a long to an int would be considered a narrowing conversion. A long occupies 8 bytes of memory while an int occupies 4 bytes. To see how data loss can occur, consider the following sample:

C#

```
int i = 50;
long lng = 3147483647;
i = lng;
```

The variable `lng` now contains a value that cannot be stored in the variable `i` because it is too large. If we were to convert this value to an int type we would be losing some of

our data and the converted value would not be the same as the value before the conversion.

A widening conversion would be the opposite of a narrowing conversion. With widening conversions, we are converting to a data type that occupies more storage space in memory than the data type we are converting from. Here is an example of a widening conversion:

```
C#
```

```
int i = 50;
long lng = 100;
lng = i;
```

Notice the difference between this code sample and the first. This time the variable `lng` is on the left-hand side of the assignment operator, so it is the target of our assignment. Before the assignment can be made, the compiler must implicitly convert the variable `i`, which is of type `int`, to type `long`. This is a widening conversion since we are converting from a type that occupies 4 bytes of memory (an `int`) to a type that occupies 8 bytes of memory (a `long`). Implicit widening conversions are allowed because there is no potential loss of data. Any value that can be stored in an `int` can also be stored in a `long`.

We know that implicit narrowing conversions are not allowed, so to be able to compile this code we need to explicitly convert the data type. Explicit conversions are done using casting. Casting is the term used in C# to describe converting one data type to another. To get the code to compile we would need to use the following syntax:

```
C#
```

```
int i = 50;
long lng = 100;
i = (int) lng; // Cast to int.
```

The third line of code tells the compiler to explicitly convert the variable `lng`, which is of type `long`, to an `int` before making the assignment. Remember that with a narrowing conversion, there is a potential loss of data. Narrowing conversions should be used with caution and even though the code will compile you may get unexpected results at run-time.

This discussion has only been for value types. When working with value types you work directly with the data stored in the variable. However, .NET also has reference types. When working with reference types you are working with a reference to a variable, not the actual data. Examples of reference types would be classes, interfaces and arrays. You

cannot implicitly or explicitly convert one reference type to another unless the compiler allows the specific conversion or the appropriate conversion operators are implemented.

The following sample generates CS0029:

```
C#  
  
// CS0029.cs  
public class MyInt  
{  
    private int x = 0;  
  
    // Uncomment this conversion routine to resolve CS0029.  
    /*  
    public static implicit operator int(MyInt i)  
    {  
        return i.x;  
    }  
    */  
  
    public static void Main()  
    {  
        var myInt = new MyInt();  
        int i = myInt; // CS0029  
    }  
}
```

## See also

- [User-defined conversion operators](#)

# Compiler Error CS0030

Article • 09/15/2021 • 2 minutes to read

Cannot convert type 'type' to 'type'

There is no predefined conversion between types. You can define a custom conversion between those types. For more information, see [User-defined conversion operators](#).

The following sample generates CS0030:

C#

```
// CS0030.cs
namespace x
{
    public class iii
    {
        /*
        public static implicit operator iii(int aa)
        {
            return null;
        }

        public static implicit operator int(iii aa)
        {
            return 0;
        }
    */

        public static iii operator ++(iii aa)
        {
            return (iii)0;    // CS0030
            // uncomment the conversion routines to resolve CS0030
        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0031

Article • 08/24/2022 • 2 minutes to read

Constant value 'value' cannot be converted to a 'type'.

An attempt was made to assign a value to a variable whose type cannot store the value.  
For more information, see [Types](#).

The following sample generates CS0031 in both checked and unchecked contexts:

C#

```
// CS0031.cs
namespace CS0031
{
    public class Program
    {
        public static void Main()
        {
            int num = (int)2147483648M; //CS0031
            // Try using a larger numeric type instead.
            // long num = (long)2147483648M; //CS0031

            const decimal d = -10M; // Decimal literal
            unchecked
            {
                const byte b = (byte)d; // CS0031
                // For small values try using a signed byte instead.
                // const sbyte b = (sbyte)d;
            }
        }
    }
}
```

## See also

- [unchecked](#)



# Compiler Error CS0034

Article • 09/15/2021 • 2 minutes to read

Operator 'operator' is ambiguous on operands of type 'type1' and 'type2'

An operator was used on two objects and the compiler found more than one conversion. Because conversions have to be unique, you either have to make a cast or to make one of the conversions explicit. For more information, see [User-defined conversion operators](#).

The following sample generates CS0034:

C#

```
// CS0034.cs
public class A
{
    // Allows for the conversion of A object to int.
    public static implicit operator int (A s)
    {
        return 0;
    }

    public static implicit operator string (A i)
    {
        return null;
    }
}

public class B
{
    public static implicit operator int (B s)
    // One way to resolve this CS0034 is to make one conversion explicit.
    // public static explicit operator int (B s)
    {
        return 0;
    }

    public static implicit operator string (B i)
    {
        return null;
    }

    public static implicit operator B (string i)
    {
        return null;
    }

    public static implicit operator B (int i)
    {
```

```

        return null;
    }

}

public class C
{
    public static void Main()
    {
        A a = new A();
        B b = new B();
        b = b + a;    // CS0034
        // Another way to resolve this CS0034 is to make a cast.
        // b = b + (int)a;
    }
}

```

In C# 1.1, a compiler bug made it possible to define a class that has implicit user-defined conversions to both `int` and `bool`, and to use a bitwise `AND` operator (`&`) on objects of that type. In C# 2.0, this bug was fixed to bring the compiler into compliance with the C# specification, and therefore the following code will now cause CS0034:

C#

```

namespace CS0034
{
    class TestClass2
    {
        public void Test()
        {
            TestClass o1 = new TestClass();
            TestClass o2 = new TestClass();
            TestClass o3 = o1 & o2; //CS0034
        }
    }

    class TestClass
    {
        public static implicit operator int(TestClass o)
        {
            return 1;
        }

        public static implicit operator TestClass(int v)
        {
            return new TestClass();
        }

        public static implicit operator bool(TestClass o)
        {
            return true;
        }
    }
}

```

}

# Compiler Error CS0035

Article • 09/15/2021 • 2 minutes to read

Operator 'operator' is ambiguous on an operand of type 'type'

The compiler has more than one available conversion and does not know which to choose before applying the operator.

The following sample generates CS0035:

C#

```
// CS0035.cs
class MyClass
{
    private int i;

    public MyClass(int i)
    {
        this.i = i;
    }

    public static implicit operator double(MyClass x)
    {
        return (double) x.i;
    }

    public static implicit operator decimal(MyClass x)
    {
        return (decimal) x.i;
    }
}

class MyClass2
{
    static void Main()
    {
        MyClass x = new MyClass(7);
        object o = - x;      // CS0035
        // try a cast:
        // object o = - (double)x;
    }
}
```

## See also

- [User-defined conversion operators](#)



# Compiler Error CS0036

Article • 09/15/2021 • 2 minutes to read

An out parameter cannot have the '[In]' attribute

Currently, the **In** attribute is not permitted on an **out** parameter.

The following sample generates CS0036:

C#

```
// CS0036.cs

using System;
using System.Runtime.InteropServices;

public class MyClass
{
    public static void TestOut([In] out char TestChar) // CS0036
        // try the following line instead
        // public static void TestOut(out char TestChar)
    {
        TestChar = 'b';
        Console.WriteLine(TestChar);
    }

    public static void Main()
    {
        char i;           //variable to receive the value
        TestOut(out i); // the arg must be passed as out
        Console.WriteLine(i);
    }
}
```

# Compiler Error CS0037

Article • 09/15/2021 • 2 minutes to read

Cannot convert null to 'type' because it is a non-nullable value type

The compiler cannot assign `null` to a value type; `null` can only be assigned to a [reference type](#) or to a [nullable value type](#). `struct` is a value type.

The following sample generates CS0037:

C#

```
// CS0037.cs
public struct s
{
}

class a
{
    public static void Main()
    {
        int i = null;    // CS0037
        s ss = null;    // CS0037
    }
}
```

# Compiler Error CS0038

Article • 09/15/2021 • 2 minutes to read

Cannot access a nonstatic member of outer type 'type1' via nested type 'type2'

A field in a class is not automatically available to a nested class. To be available to a nested class, the field must be [static](#). Otherwise, you must create an instance of the outer class. For more information, see [Nested Types](#).

The following sample generates CS0038:

```
C#  
  
// CS0038.cs  
class OuterClass  
{  
    public int count;  
    // Try the following line instead.  
    // public static int count;  
  
    class InnerClass  
    {  
        void Func()  
        {  
            // or, create an instance  
            // OuterClass class_inst = new OuterClass();  
            // int count2 = class_inst.count;  
            int count2 = count;    // CS0038  
        }  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0039

Article • 09/15/2021 • 2 minutes to read

Cannot convert type 'type1' to 'type2' via a reference conversion, boxing conversion, unboxing conversion, wrapping conversion, or null type conversion

A conversion with the `as` operator is allowed by inheritance, reference conversions, and boxing conversions.

## Example

The following example generates CS0039:

```
C#  
  
using System;  
  
class A { }  
class B : A { }  
class C : A { }  
  
class Example  
{  
    static void Main()  
    {  
        C c;  
  
        // This compiles, because  
        // there is an explicit reference conversion from type A to type C.  
        A a = new C();  
        c = a as C;  
  
        // This generates CS0039, because  
        // there is no implicit or explicit reference conversion between B  
        // and C types.  
        B b = new B();  
        c = b as C; // CS0039  
    }  
}
```

# Compiler Error CS0040

Article • 09/15/2021 • 2 minutes to read

Unexpected error creating debug information file — 'reason'

This error can occur when using the [DebugType](#) compiler option and indicates that the compiler was unable to write to the .pdb file. Possible resolutions to this error include reinstalling Visual Studio, ensuring that the compiler has write access to a file or directory, or not compiling with /debug.

# Compiler Error CS0041

Article • 09/15/2021 • 2 minutes to read

Unexpected error writing debug information -- '{error}'

# Compiler Error CS0042

Article • 09/15/2021 • 2 minutes to read

Unexpected error creating debug information file 'file' — 'reason'

The compiler could not create the debug information; *reason* contains additional information on the error condition.

# Compiler Error CS0043

Article • 09/15/2021 • 2 minutes to read

PDB file 'file' has an incorrect or out-of-date format. Delete it and rebuild.

Delete the .pdb file for this compilation and recompile.

## ⓘ Note

This compiler error is no longer used in Roslyn.

# Compiler Error CS0050

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: return type 'type' is less accessible than method 'method'

The return type and each of the types referenced in the formal parameter list of a method must be at least as accessible as the method itself. For more information, see [Access Modifiers](#).

## Example

The following sample generates CS0050 because no accessibility modifier is supplied for `MyClass`, and its accessibility therefore defaults to `private`:

C#

```
// CS0050.cs
class MyClass // Accessibility defaults to private.
// Try the following line instead.
// public class MyClass
{
}

public class MyClass2
{
    public static MyClass MyMethod()    // CS0050
    {
        return new MyClass();
    }

    public static void Main() { }
}
```

# Compiler Error CS0051

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: parameter type 'type' is less accessible than method 'method'

The return type and each of the types referenced in the formal parameter list of a method must be at least as accessible as the method itself. Make sure the types used in method signatures are not accidentally private due to the omission of the `public` modifier. For more information, see [Access Modifiers](#).

## Example

The following sample generates CS0051:

```
C#  
  
// CS0051.cs  
public class A  
{  
    // Try making B public since F is public  
    // B is implicitly private here.  
    class B  
    {  
    }  
  
    public static void F(B b) // CS0051  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0052

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: field type 'type' is less accessible than field 'field'

The type of a field cannot be less accessible than the field itself because all public constructs must return a publicly accessible object.

## Example

The following sample generates CS0052:

C#

```
// CS0052.cs
public class MyClass2
{
    // The following line causes an error because the field, M, is declared
    // as public, but the type, MyClass, is private. Therefore the type is
    // less accessible than the field.
    public MyClass M;    // CS0052

    private class MyClass
    {
    }
    // One way to resolve the error is to change the accessibility of the
    // type
    // to public.
    //public class MyClass
    // Another solution is to change the accessibility of the field to
    private.
    // private MyClass M;
}

public class MainClass
{
    public static void Main()
    {
    }
}
```

## See also

- [C# Keywords](#)
- [Access Modifiers](#)
- [Accessibility Levels](#)

- Modifiers

# Compiler Error CS0053

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: property type 'type' is less accessible than property 'property'

A public construct must return a publicly accessible object. For more information, see [Access Modifiers](#).

The following sample generates CS0053:

C#

```
// CS0053.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public MyClass myProperty // CS0053
    {
        get
        {
            return new MyClass();
        }
        set
        {
        }
    }
}

public class MyClass3
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0054

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: indexer return type 'type' is less accessible than indexer 'indexer'

A public construct must return a publicly accessible object. For more information, see [Access Modifiers](#).

The following sample generates CS0054:

```
C#  
  
// CS0054.cs  
class MyClass  
// try the following line instead  
// public class MyClass  
{  
}  
  
public class MyClass3  
{  
    public MyClass this[int i]    // CS0054  
    {  
        get  
        {  
            return new MyClass();  
        }  
    }  
}  
  
public class MyClass2  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0055

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: parameter type 'type' is less accessible than indexer 'indexer'

A public construct must return a publicly accessible object. For more information, see [Access Modifiers](#).

The following sample generates CS0055:

C#

```
// CS0055.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public int this[MyClass myClass]    // CS0055
    {
        get
        {
            return 0;
        }
    }
}

public class MyClass3
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0056

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: return type 'type' is less accessible than operator 'operator'

A public construct must return a publicly accessible object. For more information, see [Access Modifiers](#).

The following sample generates CS0056:

C#

```
// CS0056.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class A
{
    public static implicit operator MyClass(A a)    // CS0056
    {
        return new MyClass();
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0057

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: parameter type 'type' is less accessible than operator 'operator'

A public construct must return a publicly accessible object. For more information, see [Access Modifiers](#).

The following sample generates CS0057:

C#

```
// CS0057.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public static implicit operator MyClass2(MyClass iii)    // CS0057
    {
        return new MyClass2();
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0058

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: return type 'type' is less accessible than delegate 'delegate'

A public construct must return a publicly accessible object. For more information, see [Access Modifiers](#).

The following sample generates CS0058 because no access modifier is applied to MyClass and therefore it is given private accessibility by default:

```
C#  
  
// CS0058.cs  
class MyClass  
// try the following line instead  
// public class MyClass  
{  
}  
  
public delegate MyClass MyClassDel(); // CS0058  
  
public class A  
{  
    public static void Main()  
    {  
    }  
}
```

## See also

- [private](#)



# Compiler Error CS0059

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: parameter type 'type' is less accessible than delegate 'delegate'

The return type and each of the types referenced in the formal parameter list of a method must be at least as accessible as the method itself. For more information, see [Access Modifiers](#).

## Example

The following sample generates CS0059:

```
C#  
  
// CS0059.cs  
class MyClass //defaults to private accessibility  
// try the following line instead  
// public class MyClass  
{  
}  
  
public delegate void MyClassDel( MyClass myClass); // CS0059  
  
public class Program  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0060

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: base class 'class1' is less accessible than class 'class2'

Class accessibility should be consistent between the base class and inherited class.

The following sample generates CS0060:

C#

```
// CS0060.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class MyClass2 : MyClass    // CS0060
{
    public static void Main()
    {
    }
}
```

## See also

- [Access Modifiers](#)



# Compiler Error CS0061

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: base interface 'interface 1' is less accessible than interface 'interface 2'

A [public](#) construct must return a publicly accessible object.

Interface accessibility cannot be narrowed in a derived interface. For more information, see [Interfaces](#) and [Access Modifiers](#).

The following sample generates CS0061.

C#

```
// CS0061.cs
// compile with: /target:library
internal interface A {}
public interface AA : A {} // CS0061

// OK
public interface B {}
internal interface BB : B {}

internal interface C {}
internal interface CC : C {}
```

# Compiler Error CS0065

Article • 09/15/2021 • 2 minutes to read

'event' : event property must have both add and remove accessors

An event that is not a field must have both access methods.

The following sample generates CS0065:

C#

```
// CS0065.cs
using System;
public delegate void EventHandler(object sender, int e);
public class MyClass
{
    public event EventHandler Click // CS0065,
    {
        // to fix, uncomment the add and remove definitions
        /*
        add
        {
            Click += value;
        }
        remove
        {
            Click -= value;
        }
        */
    }

    public static void Main()
    {
    }
}
```

## See also

- [Events](#)



# Compiler Error CS0066

Article • 09/15/2021 • 2 minutes to read

'event': event must be of a delegate type

The event keyword requires a [delegate](#) type. For more information, see [Events](#) and [Delegates](#).

The following sample generates CS0066:

C#

```
// CS0066.cs
using System;

public class EventHandler
{
}

// to fix the error, remove the event declaration and the
// EventHandler class declaration, and uncomment the following line
// public delegate void EventHandler();

public class a
{
    public event EventHandler Click;    // CS0066

    private void TestMethod()
    {
        if (Click != null)
            Click();
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0068

Article • 09/15/2021 • 2 minutes to read

'event': event in interface cannot have initializer

An event in an interface cannot have an initializer. For more information, see [Interfaces](#).

The following sample generates CS0068:

C#

```
// CS0068.cs

delegate void MyDelegate();

interface I
{
    event MyDelegate d = new MyDelegate(M.f);      // CS0068
    // try the following line instead
    // event MyDelegate d2;
}

class M
{
    event MyDelegate d = new MyDelegate(M.f);

    public static void f()
    {
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0069

Article • 09/15/2021 • 2 minutes to read

An event in an interface cannot have add or remove accessors

You cannot define an event's accessor functions in an [interface](#). For more information, see [Events](#) and [Interfaces](#).

The following sample generates CS0069:

C#

```
// CS0069.cs
// compile with: /target:library

public delegate void EventHandler();

public interface a
{
    event EventHandler Click { remove {} }    // CS0069
    event EventHandler Click2;    // OK
}
```

# Compiler Error CS0070

Article • 09/15/2021 • 2 minutes to read

The event 'event' can only appear on the left hand side of += or -= (except when used from within the type 'type')

Outside of the class it is defined in, an [event](#) can only add or subtract references. For more information, see [Events](#).

The following sample generates CS0070:

```
C#  
  
// CS0070.cs  
using System;  
public delegate void EventHandler();  
  
public class A  
{  
    public event EventHandler Click;  
  
    public static void OnClick()  
    {  
        EventHandler eh;  
        A a = new A();  
        eh = a.Click;  
    }  
  
    public static void Main()  
    {  
    }  
}  
  
public class B  
{  
    public int Foo ()  
    {  
        EventHandler eh = new EventHandler(A.OnClick);  
        A a = new A();  
        eh = a.Click;    // CS0070  
        // try the following line instead  
        // a.Click += eh;  
        return 1;  
    }  
}
```

# Compiler Error CS0071

Article • 09/15/2021 • 2 minutes to read

An explicit interface implementation of an event must use event accessor syntax

When explicitly implementing an [event](#) that was declared in an interface, you must manually provide the `add` and `remove` event accessors that are typically provided by the compiler. The accessor code can connect the interface event to another event in your class (shown later in this topic) or to its own delegate type. For more information, see [How to implement interface events](#).

## Example

The following sample generates CS0071.

```
C#  
  
// CS0071.cs  
public delegate void MyEvent(object sender);  
  
interface ITest  
{  
    event MyEvent Clicked;  
}  
  
class Test : ITest  
{  
    event MyEvent ITest.Clicked; // CS0071  
  
    // Try the following code instead.  
    /*  
     *  
     private MyEvent clicked;  
  
     event MyEvent ITest.Clicked  
    {  
        add  
        {  
            clicked += value;  
        }  
        remove  
        {  
            clicked -= value;  
        }  
    }  
*/  
    public static void Main() { }  
}
```

# Compiler Error CS0072

Article • 09/15/2021 • 2 minutes to read

'event' : cannot override; 'method' is not an event

An [event](#) can only override another event. For more information, see [Events](#).

## Example

The following sample generates CS0072:

```
C#  
  
// CS0072.cs  
delegate void MyDelegate();  
  
class Test1  
{  
    public virtual event MyDelegate MyEvent;  
    public virtual void VMeth()  
    {  
    }  
  
    public void FireAway()  
    {  
        if (MyEvent != null)  
            MyEvent();  
    }  
}  
  
class Test2 : Test1  
{  
    public override event MyDelegate VMeth // CS0072  
    // Uncomment the following lines to resolve.  
    // public override event MyDelegate MyEvent  
    {  
        add  
        {  
            VMeth += value;  
            // MyEvent += value;  
        }  
        remove  
        {  
            VMeth -= value;  
            // MyEvent -= value;  
        }  
    }  
  
    public static void Main()  
    {
```

```
    }  
}
```

# Compiler Error CS0073

Article • 09/15/2021 • 2 minutes to read

An add or remove accessor must have a body

An **add** or **remove** keyword in an [event](#) definition must have a body. For more information, see [Events](#).

The following sample generates CS0073:

C#

```
// CS0073.cs
delegate void del();

class Test
{
    public event del MyEvent
    {
        add; // CS0073
        // try the following lines instead
        // add
        // {
        //     MyEvent += value;
        // }
        remove
        {
            MyEvent -= value;
        }
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0074

Article • 09/15/2021 • 2 minutes to read

'event': abstract event cannot have initializer

If an [event](#) is marked as **abstract**, it cannot be initialized. For more information, see [Events](#).

The following sample generates CS0074:

C#

```
// CS0074.cs
delegate void D();

abstract class Test
{
    public abstract event D e = null;      // CS0074
    // try the following line instead
    // public abstract event D e;

    public static void Main()
    {
    }
}
```

# Compiler Error CS0075

Article • 09/15/2021 • 2 minutes to read

To cast a negative value, you must enclose the value in parentheses

If you are casting using a keyword that identifies a predefined type, then you do not need parentheses. Otherwise, you must put the parentheses because `(x)-y` will not be considered a cast expression. From the C# Specification, Section 7.6.6:

*From the disambiguation rule it follows that, if x and y are identifiers, `(x)y`, `(x)(y)`, and `(x)(-y)` are cast-expressions, but `(x)-y` is not, even if x identifies a type. However, if x is a keyword that identifies a predefined type (such as int), then all four forms are cast-expressions (because such a keyword could not possibly be an expression by itself).*

The following code generates CS0075:

```
C#  
  
// CS0075  
namespace MyNamespace  
{  
    enum MyEnum { }  
    public class MyClass  
    {  
        public static void Main()  
        {  
            // To fix the error, place the negative  
            // values below in parentheses  
            int i = (System.Int32) - 4; //CS0075  
            MyEnum e = (MyEnum) - 1; //CS0075  
            System.Console.WriteLine(i); //to avoid warning  
            System.Console.WriteLine(e); //to avoid warning  
        }  
    }  
}
```

## See also

- [Casting and Type Conversions](#)



# Compiler Error CS0076

Article • 09/15/2021 • 2 minutes to read

The enumerator name 'value\_\_' is reserved and cannot be used

An item in an [enumeration](#) cannot have an identifier called **value\_\_**.

# Compiler Error CS0077

Article • 09/15/2021 • 2 minutes to read

The `as` operator must be used with a reference type or nullable type ('`int`' is a non-nullable value type).

The `as` operator was passed a [value type](#). Because `as` can return [null](#), it can only be passed a [reference type](#) or a [nullable value type](#).

The following sample generates CS0077:

C#

```
// CS0077.cs
using System;

class C
{
}

struct S
{
}

class M
{
    public static void Main()
    {
        object o1, o2;
        C c;
        S s;

        o1 = new C();
        o2 = new S();

        s = o2 as S;
        // CS0077, S is not a reference type.
        // Try the following line instead.
        // c = o1 as C;
    }
}
```

# Compiler Error CS0079

Article • 09/15/2021 • 2 minutes to read

The event 'event' can only appear on the left hand side of += or -=

An [event](#) was called incorrectly. For more information, see [Events](#) and [Delegates](#).

The following sample generates CS0079:

C#

```
// CS0079.cs
using System;

public delegate void MyEventHandler();

public class Class1
{
    private MyEventHandler _e;

    public event MyEventHandler Pow
    {
        add
        {
            _e += value;
            Console.WriteLine("in add accessor");
        }
        remove
        {
            _e -= value;
            Console.WriteLine("in remove accessor");
        }
    }

    public void Handler()
    {
    }

    public void Fire()
    {
        if (_e != null)
        {
            Pow();    // CS0079
            // try the following line instead
            // _e();
        }
    }

    public static void Main()
    {
        Class1 p = new Class1();
```

```
p.Pow += new MyEventHandler(p.Handler);
p._e();
p.Pow += new MyEventHandler(p.Handler);
p._e();
p._e -= new MyEventHandler(p.Handler);
if (p._e != null)
{
    p._e();
}
p.Pow -= new MyEventHandler(p.Handler);
if (p._e != null)
{
    p._e();
}
}
```

# Compiler Error CS0080

Article • 09/15/2021 • 2 minutes to read

Constraints are not allowed on non-generic declarations

The syntax found may only be used in a generic declaration to apply constraints to the type parameter. For more information, see [Generics](#).

The following sample generates CS0080 because MyClass is not a generic class and Foo is not a generic method.

C#

```
namespace MyNamespace
{
    public class MyClass where MyClass : System.IDisposable // CS0080
    //the following line shows an example of correct syntax
    //public class MyClass<T> where T : System.IDisposable
    {
        public void Foo() where Foo : new() // CS0080
        //the following line shows an example of correct syntax
        //public void Foo<U>() where U : struct
        {
        }
    }

    public class Program
    {
        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0081

Article • 09/15/2021 • 2 minutes to read

Type parameter declaration must be an identifier not a type

When you declare a generic method or type, specify the type parameter as an identifier, for example "T" or "inputType". When client code calls the method, it supplies the type, which replaces each occurrence of the identifier in the method or class body. For more information, see [Generic Type Parameters](#).

C#

```
// CS0081.cs
class MyClass
{
    public void F<int>() {}    // CS0081
    public void F<T>(T input) {} // OK

    public static void Main()
    {
        MyClass a = new MyClass();
        a.F<int>(2);
        a.F<double>(.05);
    }
}
```

## See also

- [Generics](#)



# Compiler Error CS0082

Article • 09/15/2021 • 2 minutes to read

Type 'type' already reserves a member called 'name' with the same parameter types

Properties at compile time are translated to methods with `get_` and/or `set_` in front of the identifier. If you define your own method that conflicts with the method name, an error is generated.

## Example

The following example generates CS0082:

```
C#  
  
//cs0082.cs  
class MyClass  
{  
  
    //property  
    public int MyProp  
    {  
        get //CS0082  
        {  
            return 1;  
        }  
    }  
  
    //conflicting Get  
    public int get_MyProp()  
    {  
        return 2;  
    }  
  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Properties](#)



# Compiler Error CS0100

Article • 09/15/2021 • 2 minutes to read

The parameter name 'parameter name' is a duplicate

A method declaration used the same parameter name more than once. Parameter names must be unique in a method declaration. For more information, see [Methods](#).

The following sample generates CS0100:

C#

```
// CS0100.cs
namespace x
{
    public class a
    {
        public static void f(int i, char i)    // CS0100
        // try the following line instead
        // public static void f(int i, char j)
        {

        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0101

Article • 09/15/2021 • 2 minutes to read

The namespace 'namespace' already contains a definition for 'type'

A [namespace](#) has duplicate identifiers. Rename or delete one of the duplicate identifiers.

For more information, see [Namespaces](#)

The following sample generates CS0101:

```
C#  
  
// CS0101.cs  
namespace MyNamespace  
{  
    public class MyClass  
    {  
        static public void Main()  
        {  
        }  
    }  
  
    public class MyClass // CS0101  
    {  
    }  
}
```

A CS0101 is also generated when your class name clashes with your namespace name. This can happen when expanding with helper classes for the base class where you attempt to keep the namespace route the same. In the below example, the UTF8 class should clearly be a subsidiary of the String class, but attempting to force it into the same name space by declaring said namespace as Utilities.String will cause a CS0101 error:

```
C#  
  
//CS0101-Utilities.String.cs  
namespace Utilities  
{  
    public class String  
    {  
        public string MyString;  
    }  
}  
  
//CS0101-Utilities.String.UTF8.cs  
namespace Utilities.String // CS0101  
{  
    public class UTF8
```

```
{  
    public string MySecondString;  
}  
}
```

# Compiler Error CS0102

Article • 09/15/2021 • 2 minutes to read

The type 'type name' already contains a definition for 'identifier'

A class contains multiple declarations for identifiers with the same name at the same scope. To fix the error, rename the duplicate identifiers.

## Example

The following sample generates CS0102.

C#

```
// CS0102.cs
// compile with: /target:library
namespace MyApp
{
    public class MyClass
    {
        string s = "Hello";
        string s = "Goodbye"; // CS0102

        public void GetString()
        {
            string s = "Hello again"; // method scope, no error
        }
    }
}
```

# Compiler Error CS0103

Article • 10/26/2022 • 2 minutes to read

The name 'identifier' does not exist in the current context

An attempt was made to use a name that does not exist in the class, [namespace](#), or scope. Check the spelling of the name and check your using directives and assembly references to make sure that the name that you are trying to use is available.

This error frequently occurs if you declare a variable in a loop or a `try` or `if` block and then attempt to access it from an enclosing code block or a separate code block, as shown in the following example:

## ⓘ Note

This error may also be presented when missing the `>` symbol in the operator `=>` in an expression lambda. For more information, see [expression lambdas](#).

C#

```
using System;

class MyClass1
{
    public static void Main()
    {
        try
        {
            // The following declaration is only available inside the try
block.
            var conn = new MyClass1();
        }
        catch (Exception e)
        {
            // The following expression causes error CS0103, because
variable
            // conn only exists in the try block.
            if (conn != null)
                Console.WriteLine("{0}", e);
        }
    }
}
```

The following example resolves the error:

C#

```
using System;

class MyClass2
{
    public static void Main()
    {
        // To resolve the error in the example, the first step is to
        // move the declaration of conn out of the try block. The following
        // declaration is available throughout the Main method.
        MyClass2 conn = null;
        try
        {
            // Inside the try block, use the conn variable that you declared
            // previously.
            conn = new MyClass2();
        }
        catch (Exception e)
        {
            // The following expression no longer causes an error, because
            // the declaration of conn is in scope.
            if (conn != null)
                Console.WriteLine("{0}", e);
        }
    }
}
```

# Compiler Error CS0104

Article • 10/06/2022 • 2 minutes to read

'reference' is an ambiguous reference between 'identifier' and 'identifier'

Your program contains `using` directives for two namespaces and your code references a name that appears in both namespaces.

The following sample generates CS0104:

C#

```
// CS0104.cs
using x;
using y;

namespace x
{
    public class Test
    {
    }
}

namespace y
{
    public class Test
    {
    }
}

public class a
{
    public static void Main()
    {
        Test test = new Test();    // CS0104, is Test in x or y namespace?
        // try the following line instead
        // y.Test test = new y.Test();
    }
}
```

# Compiler Error CS0106

Article • 09/15/2021 • 2 minutes to read

The modifier 'modifier' is not valid for this item

A class or interface member was marked with an invalid access modifier. The following examples describe some of these invalid modifiers:

- The `static` modifier is not permitted on a [local function](#). The static local function feature is supported starting with C# 8.0. A compiler that doesn't support C# 8.0 produces CS0106 when you try to use this feature. However, a compiler that supports C# 8.0 but the set language version is prior to C# 8.0 will produce a diagnostic suggesting that you use C# 8.0 or later.
- The `public` keyword is not allowed on an explicit interface declaration. In this case, remove the `public` keyword from the explicit interface declaration.
- The `abstract` keyword is not allowed on an explicit interface declaration because an explicit interface implementation can never be overridden.
- Access modifiers are not allowed on a [local function](#). Local functions are always `private`.

In prior releases of Visual Studio, the `static` modifier was not permitted on a class, but `static` classes are allowed starting with Visual Studio 2005.

For more information, see [Interfaces](#).

## Example

The following sample generates CS0106:

```
C#  
  
// CS0106.cs  
namespace MyNamespace  
{  
    interface I  
    {  
        void M();  
    }  
  
    public class MyClass : I  
    {  
        public void I.M() {} // CS0106
```

```
    public static void Main() {}  
}
```

# Compiler Error CS0107

Article • 09/15/2021 • 2 minutes to read

More than one protection modifier

Only one access modifier ([public](#), [private](#), [protected](#), or [internal](#)) is allowed on a class member, with the exception of [protected internal](#) and [private protected](#).

The following sample generates CS0107:

C#

```
// CS0107.cs
public class C
{
    private internal void F()    // CS0107, delete private or internal
    {
    }
}
```

# Compiler Error CS0110

Article • 09/15/2021 • 2 minutes to read

The evaluation of the constant value for 'const declaration' involves a circular definition

The declaration of a `const` variable (a) cannot reference another const variable (b) that also references (a).

The following sample generates CS0110:

C#

```
// CS0110.cs
namespace MyNamespace
{
    public class A
    {
        public static void Main()
        {
        }
    }

    public class B : A
    {
        public const int i = c + 1;    // CS0110, c already references i
        public const int c = i + 1;
        // the following line would be OK
        // public const int c = 10;
    }
}
```

## See also

- [Constants](#)



# Compiler Error CS0111

Article • 09/15/2021 • 2 minutes to read

Type 'class' already defines a member called 'member' with the same parameter types

CS0111 occurs if a class contains two member declarations with the same name and parameter types. For more information, see [Methods](#).

## Example

The following sample generates CS0111.

C#

```
// CS0111.cs
class A
{
    void Test() { }
    public static void Test(){}} // CS0111

    public static void Main() {}
}
```

# Compiler Error CS0112

Article • 09/15/2021 • 2 minutes to read

A static member 'function' cannot be marked as override, virtual or abstract

Any method declaration that uses the `override`, `virtual`, or `abstract` keyword cannot also use the `static` keyword.

For more information, see [Methods](#).

The following sample generates CS0112:

C#

```
// CS0112.cs
namespace MyNamespace
{
    abstract public class MyClass
    {
        public abstract void Foo();
    }
    public class MyClass2 : MyClass
    {
        override public static void Foo()    // CS0112, remove static keyword
        {
        }
        public static int Main()
        {
            return 0;
        }
    }
}
```

# Compiler Error CS0113

Article • 09/15/2021 • 2 minutes to read

A member 'function' marked as override cannot be marked as new or virtual

It is mutually exclusive to mark a method with the [new](#) and [override](#) keywords.

The following sample generates CS0113:

C#

```
// CS0113.cs
namespace MyNamespace
{
    abstract public class MyClass
    {
        public abstract void Foo();
    }

    public class MyClass2 : MyClass
    {
        override new public void Foo()    // CS0113, remove new keyword
        {
        }

        public static int Main()
        {
            return 0;
        }
    }
}
```

# Compiler Error CS0115

Article • 09/15/2021 • 2 minutes to read

'function' : no suitable method found to override

A method was marked as an override, but the compiler found no method to override. For more information, see [override](#), and [Knowing When to Use Override and New Keywords](#).

## Example

The following sample generates CS0115. You can resolve CS0115 in one of two ways:

- Remove the `override` keyword from the method in `MyClass2`.
- Use `MyClass1` as a base class for `MyClass2`.

C#

```
// CS0115.cs
namespace MyNamespace
{
    abstract public class MyClass1
    {
        public abstract int f();
    }

    abstract public class MyClass2
    {
        public override int f() // CS0115
        {
            return 0;
        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0116

Article • 10/27/2021 • 2 minutes to read

A namespace cannot directly contain members such as fields or methods.

A namespace can contain other namespaces, structs, and classes. For more information, see the [namespace keyword](#) article.

## Example

The following sample will cause Visual Studio to flag parts of the code as being in violation of CS0116. Attempting to build this code will result in build failure:

```
C#  
  
// CS0116.cs  
namespace x  
{  
    // A namespace can be placed within another namespace.  
    using System;  
  
    // These variables trigger the CS0116 error as they are declared outside  
    // of a struct or class.  
    public int latitude;  
    public int longitude;  
    Coordinate coord;  
  
    // Auto-properties also fall under the definition of this rule.  
    public string LocationName { get; set; }  
  
    // This method as well: if it isn't in a class or a struct, it's  
    // violating CS0116.  
    public void DisplayLatitude()  
    {  
        Console.WriteLine($"Lat: {latitude}");  
    }  
  
    public struct Coordinate  
    {  
    }  
  
    public class CoordinatePrinter  
    {  
        public void DisplayLongitude()  
        {  
            Console.WriteLine($"Longitude: {longitude}");  
        }  
  
        public void DisplayLocation()
```

```
        {
            Console.WriteLine($"Location: {LocationName}");
        }
    }
}
```

Note that in C#, methods and variables must be declared and defined within a struct or class. For more information on program structure in C#, see the [General Structure of a C# Program](#) article. To fix this error, rewrite your code such that all methods and fields are contained within either a struct or a class:

C#

```
namespace x
{
    // A namespace can be placed within another namespace.
    using System;

    // These variables are now placed within a struct, so CS0116 is no
    longer violated.
    public struct Coordinate
    {
        public int Latitude;
        public int Longitude;
    }

    // The methods and fields are now placed within a class, and the
    compiler is satisfied.
    public class CoordinatePrinter
    {
        Coordinate coord;
        public string LocationName { get; set; }

        public void DisplayLatitude()
        {
            Console.WriteLine($"Lat: {coord.Latitude}");
        }

        public void DisplayLongitude()
        {
            Console.WriteLine($"Longitude: {coord.Longitude}");
        }

        public void DisplayLocation()
        {
            Console.WriteLine($"Location: {LocationName}");
        }
    }
}
```

## See also

- [General Structure of a C# Program](#)
- [The C# type system](#)
- [Namespaces](#)

# Compiler Error CS0117

Article • 09/15/2021 • 2 minutes to read

'type' does not contain a definition for 'identifier'

- This error occurs in some situations when a reference is made to a member that does not exist for the data type.

## Example

The following sample generates CS0117:

C#

```
public class Base { }

public class Derived : Base
{
    public void TestInt()
    {
        int i = base.someMember; // CS0117
    }
}
```

# Compiler Error CS0118

Article • 09/15/2021 • 2 minutes to read

'construct1\_name' is a 'construct1' but is used like a 'construct2'

The compiler detected a situation in which a construct was used in some erroneous way or a disallowed operation was tried on a construct. Some common examples include the following:

- A try to instantiate a namespace (instead of a class)
- A try to call a field (instead of a method)
- A try to use a type as a variable
- A try to use an extern alias as a type.

To resolve this error, make sure that the operation you are performing is valid for the type you are performing the operation on.

## Example

The following sample generates CS0118.

C#

```
// CS0118.cs
// compile with: /target:library
namespace MyNamespace
{
    class MyClass
    {
        // MyNamespace not a class
        MyNamespace ix = new MyNamespace ();    // CS0118
    }
}
```

# Compiler Error CS0119

Article • 09/15/2021 • 2 minutes to read

'construct1\_name' is a 'construct1', which is not valid in the given context.

The compiler detected an unexpected construct such as the following:

- A class constructor is not a valid test expression in a conditional statement.
- A class name was used instead of an instance name to refer to an array element.
- A method identifier is used as if it were a struct or class

## Example

The following sample generates CS0119: 'C.B()' is a method, which is not valid in the given context. You can fix this error by changing the name of the method `C.B`, or using the fully qualified name for the class `B` like `N2.B`.

```
C#  
  
namespace N2  
{  
    public static class B  
    {  
        public static void X() {}  
    }  
}  
  
namespace N1  
{  
    public class C  
    {  
        void B() {}  
        void M() => B.X(); // CS0119  
    }  
}
```

# Compiler Error CS0120

Article • 02/24/2023 • 2 minutes to read

An object reference is required for the nonstatic field, method, or property 'member'

In order to use a non-static field, method, or property, you must first create an object instance. For more information about static methods, see [Static Classes and Static Class Members](#). For more information about creating instances of classes, see [Instance Constructors](#).

## Example 1

The following sample generates CS0120:

```
C#  
  
// CS0120_1.cs  
public class MyClass  
{  
    // Non-static field.  
    public int i;  
    // Non-static method.  
    public void f() {}  
    // Non-static property.  
    int Prop  
    {  
        get  
        {  
            return 1;  
        }  
    }  
  
    public static void Main()  
    {  
        i = 10;    // CS0120  
        f();      // CS0120  
        int p = Prop;  // CS0120  
    }  
}
```

To correct this error, first create an instance of the class:

```
C#  
  
// CS0120_1.cs  
public class MyClass  
{
```

```

// Non-static field.
public int i;
// Non-static method.
public void f() { }
// Non-static property.
int Prop
{
    get
    {
        return 1;
    }
}

public static void Main()
{
    var mc = new MyClass();
    mc.i = 10;
    mc.f();
    int p = mc.Prop;
}

```

## Example 2

CS0120 will also be generated if there is a call to a non-static method from a static method, as follows:

```

C#

// CS0120_2.cs
// CS0120 expected
using System;

public class MyClass
{
    public static void Main()
    {
        TestCall();    // CS0120
    }

    public void TestCall()
    {
    }
}

```

To correct this error, first create an instance of the class:

```
C#
```

```
// CS0120_2.cs
using System;

public class MyClass
{
    public static void Main()
    {
        var anInstanceofMyClass = new MyClass();
        anInstanceofMyClass.TestCall();
    }

    public void TestCall()
    {
    }
}
```

## Example 3

Similarly, a static method cannot call an instance method unless you explicitly give it an instance of the class, as follows:

```
C#

// CS0120_3.cs
using System;

public class MyClass
{
    public static void Main()
    {
        DoIt("Hello There");    // CS0120
    }

    private void DoIt(string sText)
    {
        Console.WriteLine(sText);
    }
}
```

To correct this error, you could also add the keyword static to the method definition:

```
C#

// CS0120_3.cs
using System;

public class MyClass
{
    public static void Main()
```

```
{  
    DoIt("Hello There");    // CS0120  
}  
  
private static void DoIt(string sText)  
{  
    Console.WriteLine(sText);  
}  
}
```

## See also

- [The C# type system](#)

# Compiler error CS0121

Article • 09/15/2021 • 2 minutes to read

The call is ambiguous between the following methods or properties: 'method1' and 'method2'

Due to implicit conversion, the compiler was not able to call one form of an overloaded method. You can resolve this error in one of the following ways:

- Specify the method parameters in such a way that implicit conversion does not take place.
- Remove all overloads for the method.
- Cast to proper type before calling the method.
- Use named arguments.

## Example 1

The following examples generate compiler error CS0121:

C#

```
public class Program
{
    static void f(int i, double d)
    {
    }

    static void f(double d, int i)
    {
    }

    public static void Main()
    {
        f(1, 1);      // CS0121

        // Try the following code instead:
        // f(1, 1.0);
        // or
        // f(1.0, 1);
        // or
        // f(1, (double)1);    // Cast and specify which method to call.
        // or
        // f(i: 1, 1);
        // or
        // f(d: 1, 1);

        // f(i: 1, d: 1); // This still gives CS0121
```

```
    }  
}
```

## Example 2

C#

```
class Program2  
{  
    static int ol_invoked = 0;  
  
    delegate int D1(int x);  
    delegate T D1<T>(T x);  
    delegate T D1<T, U>(U u);  
  
    static void F(D1 d1) { ol_invoked = 1; }  
    static void F<T>(D1<T> d1t) { ol_invoked = 2; }  
    static void F<T, U>(D1<T, U> d1t) { ol_invoked = 3; }  
  
    static int Test001()  
    {  
        F(delegate(int x) { return 1; }); // CS0121  
        if (ol_invoked == 1)  
            return 0;  
        else  
            return 1;  
    }  
  
    static int Main()  
    {  
        return Test001();  
    }  
}
```

# Compiler Error CS0122

Article • 09/15/2021 • 2 minutes to read

'member' is inaccessible due to its protection level

The [access modifier](#) for a class member prevents accessing the member. For more information, see [Access Modifiers](#).

One cause of this (not shown in the sample below) can be omitting the `/out` compiler flag on the target of a friend assembly. For more information, see [Friend Assemblies](#) and [OutputAssembly \(C# Compiler Options\)](#).

## Example

The following sample generates CS0122:

```
C#  
  
// CS0122.cs  
public class MyClass  
{  
    // Make public to resolve CS0122.  
    void MyMethod()  
    {  
    }  
}  
  
public class MyClass2  
{  
    public static int Main()  
    {  
        var a = new MyClass();  
        // MyMethod is private.  
        a.MyMethod();    // CS0122  
        return 0;  
    }  
}
```

# Compiler Error CS0123

Article • 09/15/2021 • 2 minutes to read

No overload for 'method' matches delegate 'delegate'

An attempt to create a delegate failed because the correct signature was not used.

Instances of a delegate must be declared with the same signature as the delegate declaration.

You can resolve this error by adjusting either the method or delegate signature. For more information, see [Delegates](#).

The following sample generates CS0123.

C#

```
// CS0123.cs
delegate void D();
delegate void D2(int i);

public class C
{
    public static void f(int i) {}

    public static void Main()
    {
        D d = new D(f);    // CS0123
        D2 d2 = new D2(f); // OK
    }
}
```

# Compiler Error CS0126

Article • 09/15/2021 • 2 minutes to read

An object of a type convertible to 'type' is required

A return statement is present, but the statement does not return a value of the expected type. For more information, see [Properties](#).

The following sample generates CS0126:

C#

```
// CS0126.cs
public class a
{
    public int i
    {
        set
        {
        }
        get
        {
            return; // CS0126, specify a value to return
        }
    }
}
```

# Compiler Error CS0127

Article • 09/15/2021 • 2 minutes to read

Since 'function' returns void, a return keyword must not be followed by an object expression

A method with a [void](#) return type cannot return a value. For more information, see [Methods](#).

The following sample generates CS0127:

```
C#  
  
// CS0127.cs  
namespace MyNamespace  
{  
    public class MyClass  
    {  
        public int hiddenMember2  
        {  
            get  
            {  
                return 0;  
            }  
            set // CS0127, set has an implicit void return type  
            {  
                return 0; // remove return statement to resolve this CS0127  
            }  
        }  
  
        public static void Main()  
        {  
        }  
    }  
}
```

# Compiler Error CS0128

Article • 09/15/2021 • 2 minutes to read

A local variable named 'variable' is already defined in this scope

The compiler detected declarations of two local variables with the same name. For more information, see [Methods](#).

The following sample generates CS0128:

C#

```
// CS0128.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            char i;
            int i;    // CS0128
        }
    }
}
```

# Compiler Error CS0131

Article • 03/31/2022 • 2 minutes to read

The left-hand side of an assignment must be a variable, property or indexer

In an assignment statement, the value of the right-hand side is assigned to the left-hand side. The left-hand side must be a variable, property, or indexer.

To fix this error, make sure that all operators are on the right-hand side and that the left-hand side is a variable, property, or indexer. For more information, see [Operators and expressions](#).

## Example 1

The following sample generates CS0131.

```
C#  
  
// CS0131.cs  
public class MyClass  
{  
    public int i = 0;  
    public void MyMethod()  
    {  
        i++ = 1;    // CS0131  
        // try the following line instead  
        // i = 1;  
    }  
    public static void Main() { }  
}
```

## Example 2

This error can also occur if you attempt to perform arithmetic operations on the left hand side of an assignment operator, as in the following example.

```
C#  
  
// CS0131b.cs  
public class C  
{  
    public static int Main()  
    {  
        int a = 1, b = 2, c = 3;  
        if (a + b = c) // CS0131
```

```
// try this instead
// if (a + b == c)
    return 0;
return 1;
}
```

# Compiler Error CS0132

Article • 09/15/2021 • 2 minutes to read

'constructor' : a static constructor must be parameterless

A [static](#) constructor cannot be declared with one or more parameters. For more information, see [Constructors](#).

The following sample generates CS0132:

C#

```
// CS0132.cs
namespace MyNamespace
{
    public class MyClass
    {
        public MyClass(int i)
        {
        }
    }

    public class MyClass2 : MyClass
    {
        static MyClass2(int i) // CS0132
        {
        }
    }
}
```

# Compiler Error CS0133

Article • 09/15/2021 • 2 minutes to read

The expression being assigned to 'variable' must be constant

A [const](#) variable cannot take as its value an expression that is not constant. For more information, see [Constants](#).

The following sample generates CS0133:

C#

```
// CS0133.cs
public class MyClass
{
    public const int i = c;    // CS0133, c is not constant
    public static int c = i;
    // try the following line instead
    // public const int i = 6;

    public static void Main()
    {
    }
}
```

# Compiler Error CS0134

Article • 09/15/2021 • 2 minutes to read

'variable' is of type 'type'. A const field of a reference type other than string can only be initialized with null.

A constant-expression is an expression that can be fully evaluated at compile-time. Because the only way to create a non-null value of a reference-type is to apply the new operator, and because the new operator is not permitted in a constant-expression, the only possible value for constants of reference-types other than string is null.

If you encounter this error by trying to create a `const` string array, the solution is to make the array `readonly`, and initialize it in the constructor.

## Example

The following example generates CS0134:

```
C#  
  
// CS0134.cs  
// compile with: /target:library  
class MyTest {}  
  
class MyClass  
{  
    const MyTest test = new MyTest();    // CS0134  
  
    //OK  
    const MyTest test2 = null;  
    const System.String test3 = "test";  
}
```

# Compiler Error CS0135

Article • 01/25/2022 • 2 minutes to read

'declaration1' conflicts with the declaration 'declaration2'

The compiler does not allow hiding names, which commonly leads to logic errors in your code.

## Example

The following sample generates CS0135:

C#

```
// CS0135.cs
public class MyClass2
{
    public static int i = 0;

    public static void Main()
    {
        {
            int i = 4;
            i++;
        }
        i = 0;    // CS0135
    }
}
```

From the [C# Language Specification](#):

It is an error for a local variable declaration space and a nested local variable declaration space to contain elements with the same name. Thus, within a nested declaration space it is not possible to declare a local variable or constant with the same name as a local variable or constant in an enclosing declaration space. It is possible for two declaration spaces to contain elements with the same name as long as neither declaration space contains the other.

# Compiler Error CS0136

Article • 01/25/2022 • 2 minutes to read

A local variable named 'var' cannot be declared in this scope because it would give a different meaning to 'var', which is already used in a 'parent or current/child' scope to denote something else

A variable declaration hides another declaration that would otherwise be in scope.

Rename the variable that is declared on the line that generated CS0136.

## Example

The following sample generates CS0136:

```
C#  
  
// CS0136.cs  
namespace MyNamespace  
{  
    public class MyClass  
    {  
        public static void Main()  
        {  
            int i = 0;  
            {  
                char i = 'a';    // CS0136, hides int i  
            }  
            i++;  
        }  
    }  
}
```

From the [C# Language Specification](#):

It is an error for a local variable declaration space and a nested local variable declaration space to contain elements with the same name. Thus, within a nested declaration space it is not possible to declare a local variable or constant with the same name as a local variable or constant in an enclosing declaration space. It is possible for two declaration spaces to contain elements with the same name as long as neither declaration space contains the other.

# Compiler Error CS0138

Article • 10/06/2022 • 2 minutes to read

A using namespace directive can only be applied to namespaces; 'type' is a type not a namespace

A [using](#) directive can only take the name of a namespace as a parameter. For more information, see [Namespaces](#).

The following sample generates CS0138:

```
C#  
  
// CS0138.cs  
using System.Object; // CS0138
```

# Compiler Error CS0139

Article • 12/09/2021 • 2 minutes to read

No enclosing loop out of which to break or continue

A break or continue statement was encountered outside of a loop.

For more information, see the [break](#) and [continue](#) statements.

The following sample generates CS0139 twice:

C#

```
// CS0139.cs
namespace x
{
    public class a
    {
        public static void Main()
        {
            continue; // CS0139
            break;    // CS0139
        }
    }
}
```

# Compiler Error CS0140

Article • 12/09/2021 • 2 minutes to read

The label 'label' is a duplicate

A label with the same name appeared twice. For more information, see [goto](#).

The following sample generates CS0140:

C#

```
// CS0140.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            label1: int i = 0;
            label1: int j = 0;    // CS0140, comment this line to resolve
            goto label1;
        }
    }
}
```

# Compiler Error CS0143

Article • 09/15/2021 • 2 minutes to read

The type 'class' has no constructors defined

There is no appropriate constructor available. This is the case for built-in numeric value types, which are initialized simply by assigning a value to them.

The following sample generates CS0143:

C#

```
// CS0143.cs
class MyClass
{
    static public void Main ()
    {
        double d = new double(4.5);    // CS0143
        // Try this line instead:
        // double d = 4.5;
    }
}
```

# Compiler Error CS0144

Article • 03/18/2022 • 2 minutes to read

Cannot create an instance of the abstract class or interface 'interface'

You cannot create an instance of an [abstract](#) class or an [interface](#). For more information, see [Interfaces](#).

The following sample generates CS0144:

C#

```
// CS0144.cs
interface MyInterface
{
}
public class MyClass
{
    public static void Main()
    {
        MyInterface myInterface = new MyInterface (); // CS0144
    }
}
```

## How to fix violations

You can solve this problem by implementing one of the two following solutions:

1. Change the type declaration so that it's not abstract: Either remove the `abstract` keyword from the class declaration, or change the type from an interface to a class.
2. Create a type that's derived from the abstract class or that implements the interface.



# Compiler Error CS0145

Article • 09/15/2021 • 2 minutes to read

A const field requires a value to be provided

You must initialize a [const](#) variable. For more information, see [Constants](#).

The following sample generates CS0145:

C#

```
// CS0145.cs
class MyClass
{
    const int i;    // CS0145
    // try the following line instead
    // const int i = 0;

    public static void Main()
    {
    }
}
```

# Compiler Error CS0146

Article • 09/15/2021 • 2 minutes to read

Circular base class dependency involving 'class1' and 'class2'

The inheritance list for a class includes a direct or indirect reference to itself. A class cannot inherit from itself. For more information, see [Inheritance](#).

The following sample generates CS0146:

C#

```
// CS0146.cs
namespace MyNamespace
{
    public interface InterfaceA
    {

        public class MyClass : InterfaceA, MyClass2
        {
            public void Main()
            {
            }
        }

        public class MyClass2 : MyClass // CS0146
        {
        }
    }
}
```

# Compiler Error CS0148

Article • 09/15/2021 • 2 minutes to read

The delegate 'delegate' does not have a valid constructor

You imported and used a managed program (one that uses the .NET runtime) that was created with another compiler. That compiler allowed an ill-formed **delegate** constructor. For more information, see [Delegates](#).

# Compiler Error CS0149

Article • 09/15/2021 • 2 minutes to read

Method name expected

When creating a [delegate](#), specify a method. For more information, see [Delegates](#).

The following sample generates CS0149:

C#

```
// CS0149.cs
using System;

delegate string MyDelegate(int i);

class MyClass
{
    // class member-field of the declared delegate type
    static MyDelegate dt;

    public static void Main()
    {
        dt = new MyDelegate(17.45);    // CS0149
        // try the following line instead
        // dt = new MyDelegate(Func2);
        F(dt);
    }

    public static string Func2(int j)
    {
        Console.WriteLine(j);
        return j.ToString();
    }

    public static void F(MyDelegate myFunc)
    {
        myFunc(8);
    }
}
```

# Compiler Error CS0150

Article • 07/30/2022 • 2 minutes to read

A constant value is expected

A variable was found where a constant was expected. For more information, see [switch](#).

The following sample generates CS0150:

C#

```
// CS0150.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            int i = 0;
            int j = 0;

            switch(i)
            {
                case j: // CS0150, j is a variable int, not a constant int
                // try the following line instead
                // case 0:
            }
        }
    }
}
```

This error is also produced when an array size is specified with a variable value and initialized with an array initializer. To remove the error, initialize the array in a separate statement or statements.

C#

```
// CS0150.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            int size = 2;
            double[] nums = new double[size] { 46.9, 89.4 }; //CS0150
            // Try the following lines instead
            // double[] nums = new double[size];
            // nums[0] = 46.9;
        }
    }
}
```

```
// nums[1] = 89.4;  
}  
}  
}
```

# Compiler Error CS0151

Article • 10/27/2021 • 2 minutes to read

A value of an integral type expected

A variable was used in a situation where an integral data type was required. For more information, see [Types](#).

## Example of ambiguous conversion

This error can occur when there is no conversion or if the available implicit conversions result in an ambiguous situation. The following example generates CS0151:

```
C#  
  
public class MyClass  
{  
    public static implicit operator int (MyClass aa)  
    {  
        return 0;  
    }  
  
    public static implicit operator long (MyClass aa)  
    {  
        return 0;  
    }  
  
    public static void Main()  
    {  
        MyClass a = new MyClass();  
  
        // Compiler cannot choose between int and long.  
        switch (a) // CS0151  
        // try the following line instead  
        // switch ((int)a)  
        {  
            case 1:  
                break;  
        }  
    }  
}
```

# Compiler Error CS0152

Article • 09/15/2021 • 2 minutes to read

The label 'label' already occurs in this switch statement

A label was repeated in a [switch statement](#).

The following sample generates CS0152:

C#

```
// CS0152.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            int i = 0;

            switch (i)
            {
                case 1:
                    i++;
                    return;

                case 1: // CS0152, two case 1 statements
                    i++;
                    return;
            }
        }
    }
}
```

# Compiler Error CS0153

Article • 09/15/2021 • 2 minutes to read

A goto case is only valid inside a switch statement

An attempt was made to use the [switch statement](#) syntax outside of a `switch` statement.

The following sample generates CS0153:

C#

```
// CS0153.cs
public class a
{
    public static void Main()
    {
        goto case 5;    // CS0153
    }
}
```

# Compiler Error CS0154

Article • 09/15/2021 • 2 minutes to read

The property or indexer 'property' cannot be used in this context because it lacks the get accessor

An attempt to use a [property](#) failed because no get accessor method was defined in the property. For more information, see [Fields](#).

## Example

The following sample generates CS0154:

```
C#  
  
// CS0154.cs  
public class MyClass2  
{  
    public int i  
    {  
        set  
        {  
        }  
        // uncomment the get method to resolve this error  
        /*  
        get  
        {  
            return 0;  
        }  
        */  
    }  
}  
  
public class MyClass  
{  
    public static void Main()  
    {  
        MyClass2 myClass2 = new MyClass2();  
        int j = myClass2.i;    // CS0154, no get method  
    }  
}
```

# Compiler Error CS0155

Article • 10/27/2021 • 2 minutes to read

The type caught or thrown must be derived from `System.Exception`

An attempt was made to pass a data type that does not derive from `System.Exception` into a `catch` block. Only data types that derive from `System.Exception` can be passed into a `catch` block. For more information, see [Exceptions and Exception Handling](#).

The following sample generates CS0155:

C#

```
// CS0155.cs
using System;

namespace MyNamespace
{
    public class MyClass2
        // try the following line instead
        // public class MyClass2 : Exception
    {
    }

    public class MyClass
    {
        public static void Main()
        {
            try
            {
            }
            catch (MyClass2)    // CS0155, resolves if you derive MyClass2
from Exception
            {
            }
        }
    }
}
```

# Compiler Error CS0156

Article • 10/27/2021 • 2 minutes to read

A throw statement with no arguments is not allowed in a finally clause that is nested inside the nearest enclosing catch clause

A `throw` statement with no parameters can only appear in a `catch` clause that takes no parameters.

For more information, see [Exceptions and Exception Handling](#).

The following sample generates CS0156:

```
C#  
  
// CS0156.cs  
using System;  
  
namespace MyNamespace  
{  
    public class MyClass2 : Exception  
    {}  
  
    public class MyClass  
    {  
        public static void Main()  
        {  
            try  
            {  
                throw; // CS0156  
            }  
  
            catch(MyClass2)  
            {  
                throw; // this throw is valid  
            }  
        }  
    }  
}
```

# Compiler Error CS0157

Article • 10/27/2021 • 2 minutes to read

Control cannot leave the body of a finally clause

All of the statements in a [finally](#) clause must execute. For more information, see [Exceptions and Exception Handling](#).

The following sample generates CS0157:

C#

```
// CS0157.cs
using System;
namespace MyNamespace
{
    public class MyClass2 : Exception
    {
    }

    public class MyClass
    {
        public static void Main()
        {
            try
            {

                finally
                {
                    return; // CS0157, cannot leave finally clause
                }
            }
        }
    }
}
```

# Compiler Error CS0158

Article • 12/09/2021 • 2 minutes to read

The label 'label' shadows another label by the same name in a contained scope

A label in an inner scope hides a label with the same name in an outer scope. For more information, see [goto](#).

The following sample generates CS0158:

C#

```
// CS0158.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            goto lab1;
            lab1:
            {
                lab1:
                goto lab1;    // CS0158
            }
        }
    }
}
```

# Compiler Error CS0159

Article • 12/09/2021 • 2 minutes to read

No such label 'label' within the scope of the goto statement

The label referenced by the `goto` statement could not be found within the scope of the `goto` statement.

The following sample generates CS0159:

C#

```
// CS0159.cs
public class Class1
{
    public static void Main()
    {
        int i = 0;

        switch (i)
        {
            case 1:
                goto case 3;    // CS0159, case 3 label does not exist
            case 2:
                break;
        }
        goto NOWHERE;    // CS0159, NOWHERE label does not exist
    }
}
```

# Compiler Error CS0160

Article • 10/27/2021 • 2 minutes to read

A previous catch clause already catches all exceptions of this or of a super type ('type')

A series of `catch` statements needs to be in decreasing order of derivation. For example, the most derived objects must appear first.

For more information, see [Exceptions and Exception Handling](#).

The following sample generates CS0160:

C#

```
// CS0160.cs
public class MyClass2 : System.Exception {}
public class MyClass
{
    public static void Main()
    {
        try {}

        catch(System.Exception) {}    // Second-most derived; should be second
    catch
        catch(MyClass2) {}    // CS0160  Most derived; should be first catch
    }
}
```

# Compiler Error CS0161

Article • 09/15/2021 • 2 minutes to read

'method': not all code paths return a value

A method that returns a value must have a `return` statement in all code paths. For more information, see [Methods](#).

## Example

The following sample generates CS0161:

C#

```
// CS0161.cs
public class Test
{
    public static int Main() // CS0161
    {
        int i = 5;
        if (i < 10)
        {
            return i;
        }
        else
        {
            // Uncomment the following line to resolve.
            // return 1;
        }
    }
}
```

# Compiler Error CS0163

Article • 12/09/2021 • 2 minutes to read

Control cannot fall through from one case label ('label') to another

When a [switch statement](#) contains more than one switch section, you must explicitly terminate each section, including the last one, by using one of the following keywords:

- [return](#)
- [goto](#)
- [break](#)
- [throw](#)

If you want to implement "fall through" behavior from one section to the next, use [goto](#) [case #](#).

The following sample generates CS0163.

C#

```
// CS0163.cs
public class MyClass
{
    public static void Main()
    {
        int i = 0;

        switch (i)    // CS0163
        {
            // Compiler error CS0163 is reported on the following line.
            case 1:
                i++;
            // To resolve the error, uncomment one of the following example
            statements.
            // return;
            // break;
            // goto case 3;

            case 2:
                i++;
                return;

            case 3:
                i = 0;
                return;

            // Compiler error CS0163 is reported on the following line.
            default:
                Console.WriteLine("Default");
        }
    }
}
```

```
// To resolve the error, uncomment the following line:  
//break;  
}  
}
```

# Compiler Error CS0165

Article • 09/15/2021 • 2 minutes to read

Use of unassigned local variable 'name'

The C# compiler doesn't allow the use of uninitialized variables. If the compiler detects the use of a variable that might not have been initialized, it generates compiler error CS0165. For more information, see [Fields](#). This error is generated when the compiler encounters a construct that might result in the use of an unassigned variable, even if your particular code does not. This avoids the necessity of overly complex rules for definite assignment.

For more information, see [Why does a recursive lambda cause a definite assignment error?](#).

## Example 1

The following sample generates CS0165:

```
C#  
  
// CS0165.cs  
using System;  
  
class MyClass  
{  
    public int i;  
}  
  
class MyClass2  
{  
    public static void Main(string[] args)  
    {  
        // i and j are not initialized.  
        int i, j;  
  
        // You can provide a value for args[0] in the 'Command line  
        // arguments'  
        // text box on the Debug tab of the project Properties window.  
        if (args[0] == "test")  
        {  
            i = 0;  
        }  
        // If the following else clause is absent, i might not be  
        // initialized.  
        //else  
        //{
    }
}
```

```

//      i = 1;
//}

// Because i might not have been initialized, the following
// line causes CS0165.
j = i;

// To resolve the error, uncomment the else clause of the previous
// if statement, or initialize i when you declare it.

// The following example causes CS0165 because myInstance is
// declared but not instantiated.
MyClass myInstance;
// The following line causes the error.
myInstance.i = 0;

// To resolve the error, replace the previous declaration with
// the following line.
//MyClass myInstance = new MyClass();
}

}

```

## Example 2

Compiler error CS0165 can occur in recursive delegate definitions. You can avoid the error by defining the delegate in two statements so that the variable is not used before it is initialized. The following example demonstrates the error and the resolution.

C#

```

class Program
{
    delegate void Del();
    static void Main(string[] args)
    {
        // The following line causes CS0165 because variable d is used
        // as an argument before it has been initialized.
        Del d = delegate() { System.Console.WriteLine(d); };

        //// To resolve the error, initialize d in a separate statement.
        //Del d = null;
        //// After d is initialized, you can use it as an argument.
        //d = delegate() { System.Console.WriteLine(d); };
        //d();

    }
}

```

[Load\(String\)](#)

# Compiler Error CS0167

Article • 09/15/2021 • 2 minutes to read

The delegate 'delegate' is missing the `Invoke` method

You imported and used a managed program (one that uses the .NET runtime) that was created with another compiler. That compiler allowed an ill-formed [delegate](#). Therefore, the `Invoke` method was not available. For more information, see [Delegates](#).

# Compiler Error CS0170

Article • 09/15/2021 • 2 minutes to read

Use of possibly unassigned field 'field'

A field in a structure was used without first being initialized. To solve this problem, first determine which field was uninitialized and then initialize it before you try to access it. For more information about initializing structs, see [Structure types](#).

The following sample generates CS0170:

```
C#  
  
// CS0170.cs  
public struct error  
{  
    public int i;  
}  
  
public class MyClass  
{  
    public static void Main()  
    {  
        error e;  
        // uncomment the next line to resolve this error  
        // e.i = 0;  
        System.Console.WriteLine( e.i );    // CS0170 because  
                                         // e.i was never assigned  
    }  
}
```

# Compiler Error CS0172

Article • 09/15/2021 • 2 minutes to read

Type of conditional expression cannot be determined because 'type1' and 'type2' implicitly convert to one another

In a conditional statement, you must be able to implicitly convert the types on either side of the `:` token. Also, there cannot be mutual implicit conversions; you only need one conversion.

The following sample generates CS0172:

```
C#  
  
// CS0172.cs  
public class Square  
{  
    public class Circle  
    {  
        public static implicit operator Circle(Square aa)  
        {  
            return null;  
        }  
  
        public static implicit operator Square(Circle aa)  
        // using explicit resolves this error  
        // public static explicit operator Square(Circle aa)  
        {  
            return null;  
        }  
    }  
  
    public static void Main()  
    {  
        Circle aa = new Circle();  
        Square ii = new Square();  
        object o = (1 == 1) ? aa : ii;    // CS0172  
        // the following cast would resolve this error  
        // (1 == 1) ? aa : (Circle)ii;  
    }  
}
```

## See also

- [User-defined conversion operators](#)



# Compiler Error CS0173

Article • 09/15/2021 • 2 minutes to read

Type of conditional expression cannot be determined because there is no implicit conversion between 'class1' and 'class2'

Conversions between classes are useful when you want objects of different classes to work with the same code. However, two classes that work together cannot have mutual and redundant conversions, or no implicit conversions. The types of `class1` and `class2` are determined independently, and the more general type is selected as the type of the conditional expression. For more information about how types are determined, see [Conditional operator cannot cast implicitly](#).

To resolve CS0173, verify that there is one and only one implicit conversion between `class1` and `class2`, regardless of which direction the conversion is in and regardless of which class the conversion is in. For more information, see [User-defined conversion operators](#) and [Built-in numeric conversions](#).

## Example

The following examples generate compiler error CS0173:

C#

```
public class C {}

public class A
{
    // The following code defines an implicit conversion operator from
    // type C to type A.
    //public static implicit operator A(C c)
    //{
    //    A a = new A();
    //    a = c;
    //    return a;
    //}
}

public class MyClass
{
    public static void F(bool b)
    {
        A a = new A();
        C c = new C();

        // The following line causes CS0173 because there is no implicit
```

```
// conversion from a to c or from c to a.  
object o = b ? a : c;  
  
// To resolve the error, you can cast a and c.  
// object o = b ? (object)a : (object)c;  
  
// Alternatively, you can add a conversion operator from class C to  
// class A, or from class A to class C, but not both.  
}  
  
public static void Main()  
{  
    F(true);  
}  
}
```

C#

```
class M  
{  
    static int Main ()  
    {  
        int X = 1;  
        // The following line causes CS0173.  
        object o = (X == 0) ? null : null;  
        return -1;  
    }  
}
```

# Compiler Error CS0174

Article • 09/15/2021 • 2 minutes to read

A base class is required for a 'base' reference

This error occurs only when the .NET runtime compiles the source code for the `System.Object` class, which is the only class that has no base class.

# Compiler Error CS0175

Article • 09/15/2021 • 2 minutes to read

Use of keyword 'base' is not valid in this context

The [base](#) keyword must be used to specify a particular member of the base class. For more information, see [Constructors](#).

The following sample generates CS0175:

C#

```
// CS0175.cs
using System;
class BaseClass
{
    public int TestInt = 0;
}

class MyClass : BaseClass
{
    public static void Main()
    {
        MyClass aClass = new MyClass();
        aClass.BaseTest();
    }

    public void BaseTest()
    {
        Console.WriteLine(base); // CS0175
        // Try the following line instead:
        // Console.WriteLine(base.TestInt);
        base = 9; // CS0175

        // Try the following line instead:
        // base.TestInt = 9;
    }
}
```

# Compiler Error CS0176

Article • 09/15/2021 • 2 minutes to read

Static member 'member' cannot be accessed with an instance reference; qualify it with a type name instead

Only a class name can be used to qualify a `static` variable; an instance name cannot be a qualifier. For more information, see [Static Classes and Static Class Members](#).

The following sample generates CS0176:

```
C#  
  
// CS0176.cs  
public class MyClass2  
{  
    public static int num;  
}  
  
public class Test  
{  
    public static void Main()  
    {  
        MyClass2 mc2 = new MyClass2();  
        int i = mc2.num;    // CS0176  
        // try the following line instead  
        // int i = MyClass2.num;  
    }  
}
```

# Compiler Error CS0177

Article • 09/21/2022 • 2 minutes to read

The out parameter 'parameter' must be assigned to before control leaves the current method

A parameter marked with the `out` keyword was not assigned a value in the method body. For more information, see [Passing Parameters](#)

The following sample generates CS0177:

C#

```
// CS0177.cs
public class MyClass
{
    public static void Foo(out int i)    // CS0177
    {
        // uncomment the following line to resolve this error
        //    i = 0;
    }

    public static void Main()
    {
        int x = -1;
        Foo(out x);
    }
}
```

# Compiler Error CS0178

Article • 04/12/2023

Invalid rank specifier: expected ',' or ']'

An array initialization was ill-formed. For example, when specifying the array dimensions, you can specify the following:

- A number in brackets
- Empty brackets
- A comma enclosed in brackets

For more information, see [Arrays](#) and the C# specification ([C# Language Specification](#)) section on array initializers.

## Example

The following sample generates CS0178.

```
C#  
  
// CS0178.cs  
class MyClass  
{  
    public static void Main()  
    {  
        int a = new int[5][,][][5];    // CS0178  
        int[,] b = new int[3,2];      // OK  
  
        int[][] c = new int[10][];  
        c[0] = new int[5][5];        // CS0178  
        c[0] = new int[2];          // OK  
        c[1] = new int[2]{1,2};      // OK  
    }  
}
```

# Compiler Error CS0179

Article • 09/15/2021 • 2 minutes to read

'member' cannot be extern and declare a body

When a class member is marked `extern`, it means that the member's definition is located in another file. Therefore, a class member marked as `extern` cannot be defined in the class. Either remove the `extern` keyword or delete the definition. For more information, see [Methods](#).

The following sample generates CS0179:

```
C#  
  
// CS0179.cs  
public class MyClass  
{  
    public extern int ExternMethod(int aa)    // CS0179  
    {  
        return 0;  
    }  
    // try the following line instead  
    // public extern int ExternMethod(int aa);  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0180

Article • 09/15/2021 • 2 minutes to read

'member' cannot be both extern and abstract

The [abstract](#) and [extern](#) keywords are mutually exclusive. The `extern` keyword means that the member is defined outside the file, and [abstract](#) means that the implementation is provided in a derived class. For more information, see [Methods](#).

The following sample generates CS0180:

C#

```
// CS0180.cs
namespace MyNamespace
{
    public class MyClass
    {
        public extern abstract int Foo(int a); // CS0180

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0181

Article • 09/16/2022 • 2 minutes to read

Attribute constructor parameter has type, which is not a valid attribute parameter type

## Example

The following sample generates CS0181:

C#

```
// CS0181.cs (12,6)

using System;
using System.Runtime.InteropServices;
[AttributeUsage(AttributeTargets.Method, AllowMultiple = true)]
unsafe class Attr : Attribute
{
    public Attr(delegate*<void> d) {}
}
unsafe class C
{
    [UnmanagedCallersOnly]
    [Attr(&M1)]
    static void M1()
    {
    }
}
```

The CLR currently does not allow generic attribute parameter types, so C# does not allow generic attribute parameter types.

# Compiler Error CS0182

Article • 03/15/2023 • 2 minutes to read

An attribute argument must be a constant expression, `typeof` expression or array creation expression of an attribute parameter type

Certain restrictions apply to what kinds of arguments may be used with attributes. Note that in addition to the restrictions specified in the error message, the following types are NOT allowed as attribute arguments:

- `sbyte`
- `ushort`
- `uint`
- `ulong`
- `decimal`

For more information, see [Attributes](#).

## Example

The following sample generates CS0182:

C#

```
// CS0182.cs
public class MyClass
{
    static string s = "Test";

    [System.Diagnostics.ConditionalAttribute(s)] // CS0182
    // try the following line instead
    // [System.Diagnostics.ConditionalAttribute("Test")]
    void NonConstantArgumentToConditional()
    {

    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0185

Article • 12/17/2021 • 2 minutes to read

'type' is not a reference type as required by the lock statement

The [lock](#) statement can only be used with [reference types](#).

## Example

The following sample generates CS0185:

```
C#  
  
// CS0185.cs  
public class MainClass  
{  
    public static void Main ()  
    {  
        lock (1)    // CS0185  
        // try the following lines instead  
        // MainClass x = new MainClass();  
        // lock(x)  
        {  
        }  
    }  
}
```

# Compiler Error CS0186

Article • 09/15/2021 • 2 minutes to read

Use of null is not valid in this context

The following sample generates CS0186:

C#

```
// CS0186.cs
using System;
using System.Collections;

class MyClass
{
    static void Main()
    {
        // Each of the following lines generates CS0186:
        foreach (int i in null) {}    // CS0186
        foreach (int i in (IEnumerable) null) { };    // CS0186
    }
}
```

# Compiler Error CS0191

Article • 09/15/2021 • 2 minutes to read

Property or indexer 'name' cannot be assigned to -- it is read only

A [readonly](#) field can only take an assignment in a constructor or at declaration. For more information, see [Constructors](#).

CS0191 also results if the `readonly` field is [static](#) and the constructor is not marked

`static`.

## Example

The following sample generates CS0191.

C#

```
// CS0191.cs
class MyClass
{
    public readonly int TestInt = 6; // OK to assign to readonly field in
declaration

    MyClass()
    {
        TestInt = 11; // OK to assign to readonly field in constructor
    }

    public void TestReadOnly()
    {
        TestInt = 19; // CS0191
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0192

Article • 09/15/2021 • 2 minutes to read

Fields of static readonly field 'name' cannot be passed ref or out (except in a static constructor)

A field (variable) marked with the [readonly](#) keyword cannot be passed either to a [ref](#) or [out](#) parameter except inside a constructor. For more information, see [Fields](#).

CS0192 also results if the `readonly` field is [static](#) and the constructor is not marked `static`.

## Example

The following sample generates CS0192.

```
C#  
  
// CS0192.cs  
class MyClass  
{  
    public readonly int TestInt = 6;  
    static void TestMethod(ref int testInt)  
    {  
        testInt = 0;  
    }  
  
    MyClass()  
    {  
        TestMethod(ref TestInt); // OK  
    }  
  
    public void PassReadOnlyRef()  
    {  
        TestMethod(ref TestInt); // CS0192  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0193

Article • 09/15/2021 • 2 minutes to read

The \* or -> operator must be applied to a pointer

The \* or -> operator was used with a nonpointer type. For more information, see [Pointer types](#).

The following sample generates CS0193:

C#

```
// CS0193.cs
using System;

public struct Age
{
    public int AgeYears;
    public int AgeMonths;
    public int AgeDays;
}

public class MyClass
{
    public static void SetAge(ref Age anAge, int years, int months, int days)
    {
        anAge->Months = 3;    // CS0193, anAge is not a pointer
        // try the following line instead
        // anAge.AgeMonths = 3;
    }

    public static void Main()
    {
        Age MyAge = new Age();
        Console.WriteLine(MyAge.AgeMonths);
        SetAge(ref MyAge, 22, 4, 15);
        Console.WriteLine(MyAge.AgeMonths);
    }
}
```

# Compiler Error CS0196

Article • 09/15/2021 • 2 minutes to read

A pointer must be indexed by only one value

A pointer cannot have multiple indices. For more information, see [Pointer types](#)

The following sample generates CS0196:

C#

```
// CS0196.cs
public class MyClass
{
    public static void Main ()
    {
        int *i = null;
        int j = 0;
        j = i[1,2];    // CS0196
        // try the following line instead
        // j = i[1];
    }
}
```

# Compiler Error CS0198

Article • 09/15/2021 • 2 minutes to read

Fields of static readonly field 'name' cannot be assigned to (except in a static constructor or a variable initializer)

A [readonly](#) variable must have the same [static](#) usage as the constructor in which you want to initialize it. For more information, see [Static Constructors](#).

The following sample generates CS0198:

C#

```
// CS0198.cs
class MyClass
{
    public static readonly int TestInt = 6;

    MyClass()
    {
        TestInt = 11;      // CS0198, constructor is not static and readonly
field is
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0199

Article • 09/21/2022 • 2 minutes to read

Fields of static readonly field 'name' cannot be passed ref or out (except in a static constructor)

A [readonly](#) variable must have the same [static](#) usage as the constructor in which you want to pass it as a [ref](#) or [out](#) parameter. For more information, see [Method Parameters](#).

## Example

The following sample generates CS0199:

C#

```
// CS0199.cs
class MyClass
{
    public static readonly int TestInt = 6;

    static void TestMethod(ref int testInt)
    {
        testInt = 0;
    }

    MyClass()
    {
        TestMethod(ref TestInt);    // CS0199, TestInt is static
    }

    public static void Main()
    {
    }
}
```

# Compiler error CS0200

Article • 09/15/2021 • 2 minutes to read

Property or indexer 'property' cannot be assigned to -- it is read only

An attempt was made to assign a value to a [property](#), but the property does not have a set accessor or the assignment was outside of the constructor. Resolve the error by adding a set accessor. For more information, see [How to declare and use read-write properties](#).

## Examples

The following sample generates CS0200:

```
C#  
  
// CS0200.cs  
public class Example  
{  
    private int _mi;  
    int I  
    {  
        get  
        {  
            return _mi;  
        }  
        // uncomment the set accessor and declaration for _mi  
        /*  
        set  
        {  
            _mi = value;  
        }  
        */  
    }  
  
    public static void Main()  
    {  
        Example example = new Example();  
        example.I = 9;    // CS0200  
    }  
}
```

The following sample uses [auto-implemented properties](#) and [object initializers](#) and still generates CS0200:

```
C#
```

```
// CS0200.cs
public class Example
{
    int I
    {
        get;
        // uncomment the set accessor and declaration
        //set;
    }

    public static void Main()
    {
        var example = new Example
        {
            I = 9    // CS0200
        };
    }
}
```

To assign to a property or indexer 'property' that's read-only, add a set accessor or assign the value in the object's constructor.

C#

```
public class Example
{
    int I { get; }

    public Example()
    {
        I = -7;
    }
}
```

# Compiler Error CS0201

Article • 03/31/2022 • 2 minutes to read

Only assignment, call, increment, decrement, and new object expressions can be used as a statement

The compiler generates an error when it encounters an invalid statement. An invalid statement is any line or series of lines ending in a semicolon that does not represent an assignment (`=`), method call `()`, `new`, `--` or `++` operation. For more information, see [Statements](#) and [Operators and expressions](#).

## Example 1

The following sample generates CS0201 because `2 * 3` is an expression, not a statement. To make the code compile, try assigning the value of the expression to a variable.

C#

```
// CS0201.cs
public class MainClass
{
    public static void Main()
    {
        2 * 3;    // CS0201
        // Try the following line instead.
        // int i = 2 * 3;
    }
}
```

## Example 2

The following sample generates CS0201 because `checked` by itself is not a statement, even though it is parameterized by an increment operation.

C#

```
// CS0201_b.cs
// compile with: /target:library
public class myList<T>
{
    public void Add(T x)
    {
        int i = 0;
        if ( (object)x == null)
```

```
{  
    checked(i++);      // CS0201  
  
    // OK  
    checked {  
        i++;  
    }  
}  
}
```

## See also

- [C# Compiler Errors](#)

# Compiler Error CS0202

Article • 09/15/2021 • 2 minutes to read

foreach requires that the return type 'type' of 'type.GetEnumerator()' must have a suitable public MoveNext method and public Current property

A [GetEnumerator](#) function, used to enable the use of foreach statements, cannot return a pointer or array; it must return an instance of a class that is able to act as an enumerator. The proper requirements to serve as an enumerator include a public Current property and a public MoveNext method.

## ⓘ Note

In C# 2.0, the compiler will automatically generate Current and MoveNext for you. For more information, see the code example in [Generic Interfaces](#).

The following sample generates CS0202:

```
C#  
  
// CS0202.cs  
  
public class C1  
{  
    public int Current  
    {  
        get  
        {  
            return 0;  
        }  
    }  
  
    public bool MoveNext ()  
    {  
        return false;  
    }  
  
    public static implicit operator C1 (int c1)  
    {  
        return 0;  
    }  
}  
  
public class C2  
{  
    public int Current  
    {  
        get  
    }
```

```
get
{
    return 0;
}

public bool MoveNext ()
{
    return false;
}

public C1[] GetEnumerator ()
// try the following line instead
// public C1 GetEnumerator ()
{
    return null;
}
}

public class MainClass
{
    public static void Main ()
    {
        C2 c2 = new C2();

        foreach (C1 x in c2)    // CS0202
        {
            System.Console.WriteLine(x.Current);
        }
    }
}
```

# Compiler Error CS0204

Article • 09/15/2021 • 2 minutes to read

Only 65534 locals are allowed

The limit of 65534 local variables has been exceeded in a method.

# Compiler Error CS0205

Article • 09/15/2021 • 2 minutes to read

Cannot call an abstract base member: 'method'

You cannot call an [abstract](#) method because it does not have a method body. For more information, see [Abstract and Sealed Classes and Class Members](#).

The following sample generates CS0205:

C#

```
// CS0205.cs
abstract public class MyClass
{
    abstract public void M();
}

public class MyClass2 : MyClass
{
    public override void M()
    {
        base.M();    // CS0205, delete this line
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0206

Article • 09/21/2022 • 2 minutes to read

A property or indexer may not be passed as an out or ref parameter

A [property](#) is not available to be passed as a [ref](#) or [out](#) parameter. For more information, see [Method Parameters](#).

## Example

The following sample generates CS0206:

```
C#  
  
// CS0206.cs  
public class MyClass  
{  
    public static int P  
    {  
        get  
        {  
            return 0;  
        }  
        set  
        {  
        }  
    }  
  
    public static void MyMethod(ref int i)  
    // public static void MyMethod(int i)  
    {  
    }  
  
    public static void Main()  
    {  
        MyMethod(ref P);    // CS0206  
        // try the following line instead  
        // MyMethod(P);    // CS0206  
    }  
}
```

# Compiler Error CS0208

Article • 09/15/2021 • 2 minutes to read

Cannot take the address of, get the size of, or declare a pointer to a managed type ('type')

Even when used with the [unsafe](#) keyword, taking the address of a managed object, getting the size of a managed object, or declaring a pointer to a managed type is not allowed. A managed type is:

- any reference type
- any struct that contains a reference type as a field or property

For more information, see [Unmanaged types](#).

## Example

The following sample generates CS0208:

```
C#  
  
// CS0208.cs  
// compile with: /unsafe  
  
class myClass  
{  
    public int a = 98;  
}  
  
struct myProblemStruct  
{  
    string s;  
    float f;  
}  
  
struct myGoodStruct  
{  
    int i;  
    float f;  
}  
  
public class MyClass  
{  
    unsafe public static void Main()  
    {  
        // myClass is a class, a managed type.  
        MyClass s = new MyClass();
```

```
myClass* s2 = &s;      // CS0208

// The struct contains a string, a managed type.
int i = sizeof(myProblemStruct); //CS0208

// The struct contains only value types.
i = sizeof(myGoodStruct); //OK

}
```

# Compiler Error CS0209

Article • 09/10/2022 • 2 minutes to read

The type of local declared in a fixed statement must be a pointer type

The variable that you declare in a [fixed statement](#) must be a pointer. For more information, see [Unsafe Code and Pointers](#).

The following sample generates CS0209:

C#

```
// CS0209.cs
// compile with: /unsafe

class Point
{
    public int x, y;
}

public class MyClass
{
    unsafe public static void Main()
    {
        Point pt = new Point();

        fixed (int i)      // CS0209
        {
        }
        // try the following lines instead
        /*
        fixed (int* p = &pt.x)
        {
        }
        fixed (int* q = &pt.y)
        {
        }
        */
    }
}
```

# Compiler Error CS0210

Article • 03/14/2023 • 2 minutes to read

You must provide an initializer in a fixed or using statement declaration

You must declare and initialize the variable in a [fixed statement](#). For more information, see [Unsafe Code and Pointers](#).

The following sample generates CS0210:

```
C#  
  
// CS0210a.cs  
// compile with: /unsafe  
  
class Point  
{  
    public int x, y;  
}  
  
public class MyClass  
{  
    unsafe public static void Main()  
    {  
        Point pt = new Point();  
  
        fixed (int i)      // CS0210  
        {  
        }  
        // try the following lines instead  
        /*  
        fixed (int* p = &pt.x)  
        {  
        }  
        fixed (int* q = &pt.y)  
        {  
        }  
        */  
    }  
}
```

The following sample also generates CS0210 because the [using statement](#) has no initializer.

```
C#  
  
// CS0210b.cs  
  
using System.IO;
```

```
class Test
{
    static void Main()
    {
        using (StreamWriter w) // CS0210
        // Try this line instead:
        // using (StreamWriter w = new StreamWriter("TestFile.txt"))
        {
            w.WriteLine("Hello there");
        }
    }
}
```

# Compiler Error CS0211

Article • 09/15/2021 • 2 minutes to read

Cannot take the address of the given expression

You can take the address of fields, local variables, and indirection of pointers, but you cannot take, for example, the address of the sum of two local variables. For more information, see [Unsafe Code and Pointers](#).

The following sample generates CS0211:

```
C#  
  
// CS0211.cs  
// compile with: /unsafe  
  
public class MyClass  
{  
    unsafe public void M()  
    {  
        int a = 0, b = 0;  
        int *i = &(a + b);    // CS0211, the addition of two local variables  
        // try the following line instead  
        // int *i = &a;  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0212

Article • 09/15/2021 • 2 minutes to read

You can only take the address of an unfixed expression inside of a fixed statement initializer

For more information, see [Unsafe Code and Pointers](#).

The following sample shows how to take the address of an unfixed expression. The following sample generates CS0212.

```
C#  
  
// CS0212a.cs  
// compile with: /unsafe /target:library  
  
public class A {  
    public int iField = 5;  
  
    unsafe public void M() {  
        A a = new A();  
        int* ptr = &a.iField;    // CS0212  
    }  
  
    // OK  
    unsafe public void M2() {  
        A a = new A();  
        fixed (int* ptr = &a.iField) {}  
    }  
}
```

The following sample also generates CS0212 and shows how to resolve the error:

```
C#  
  
// CS0212b.cs  
// compile with: /unsafe /target:library  
using System;  
  
public class MyClass  
{  
    unsafe public void M()  
    {  
        // Null-terminated ASCII characters in an sbyte array  
        sbyte[] sbArr1 = new sbyte[] { 0x41, 0x42, 0x43, 0x00 };  
        sbyte* pAsciiUpper = &sbArr1[0];    // CS0212  
        // To resolve this error, delete the previous line and  
        // uncomment the following code:  
        // fixed (sbyte* pAsciiUpper = sbArr1)
```

```
// {  
//     String szAsciiUpper = new String(pAsciiUpper);  
// }  
}  
}
```

# Compiler Error CS0213

Article • 09/10/2022 • 2 minutes to read

You cannot use the fixed statement to take the address of an already fixed expression

A local variable in an [unsafe](#) method or a parameter is already fixed (on the stack), so you cannot take the address of either of these two variables in a [fixed](#) expression. For more information, see [Unsafe Code and Pointers](#).

## Example

The following sample generates CS0213.

C#

```
// CS0213.cs
// compile with: /unsafe
public class MyClass
{
    unsafe public static void Main()
    {
        int i = 45;
        fixed (int *j = &i) { } // CS0213
        // try the following line instead
        // int* j = &i;

        int[] a = new int[] {1,2,3};
        fixed (int *b = a)
        {
            fixed (int *c = b) { } // CS0213
            // try the following line instead
            // int *c = b;
        }
    }
}
```

# Compiler Error CS0214

Article • 09/15/2021 • 2 minutes to read

Pointers and fixed size buffers may only be used in an unsafe context

Pointers can only be used with the [unsafe](#) keyword. For more information, see [Unsafe Code and Pointers](#).

The following sample generates CS0214:

C#

```
// CS0214.cs
// compile with: /target:library /unsafe
public struct S
{
    public int a;
}

public class MyClass
{
    public static void Test()
    {
        S s = new S();
        S * s2 = &s;      // CS0214
        s2->a = 3;       // CS0214
        s.a = 0;
    }

    // OK
    unsafe public static void Test2()
    {
        S s = new S();
        S * s2 = &s;
        s2->a = 3;
        s.a = 0;
    }
}
```

# Compiler Error CS0215

Article • 09/15/2021 • 2 minutes to read

The return type of operator True or False must be bool

User-defined `true` and `false` operators must have a return type of `bool`.

The following sample generates CS0215:

C#

```
// CS0215.cs
class MyClass
{
    public static int operator true (MyClass MyInt)    // CS0215
    // try the following line instead
    // public static bool operator true (MyClass MyInt)
    {
        return true;
    }

    public static int operator false (MyClass MyInt)   // CS0215
    // try the following line instead
    // public static bool operator false (MyClass MyInt)
    {
        return true;
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0216

Article • 09/15/2021 • 2 minutes to read

The operator 'operator' requires a matching operator 'missing\_operator' to also be defined

A user-defined `==` operator requires a user-defined `!=` operator, and vice versa.

The same applies also to a user-defined `true` operator and a user-defined `false` operator.

The following sample generates CS0216:

C#

```
// CS0216.cs
class MyClass
{
    public static bool operator == (MyClass MyIntLeft, MyClass MyIntRight)
// CS0216
    {
        return MyIntLeft == MyIntRight;
    }

    // to resolve, uncomment the following operator definition
/*
    public static bool operator != (MyClass MyIntLeft, MyClass MyIntRight)
    {
        return MyIntLeft != MyIntRight;
    }
*/
}

public override bool Equals (object obj)
{
    return base.Equals (obj);
}

public override int GetHashCode()
{
    return base.GetHashCode();
}

public static void Main()
{
}
}
```

# Compiler Error CS0217

Article • 04/08/2023 • 2 minutes to read

In order to be applicable as a short circuit operator a user-defined logical operator ('operator') must have the same return type as the type of its 2 parameters.

If you define an operator for a user-defined type, and then try to use the operator as a short-circuit operator, the user-defined operator must have parameters and return values of the same type. For more information about short-circuit operators, see [&& operator](#) and [|| operator](#). For more information about user-defined short-circuit, or conditional, operators, see the [User-defined conditional logical operators](#) section of the [C# language specification](#).

The following sample generates CS0217:

C#

```
// CS0217.cs
using System;

public class MyClass
{
    public static bool operator true (MyClass f)
    {
        return false;
    }

    public static bool operator false (MyClass f)
    {
        return false;
    }

    public static implicit operator int(MyClass x)
    {
        return 0;
    }

    public static int operator & (MyClass f1, MyClass f2)    // CS0217
    // try the following line instead
    // public static MyClass operator & (MyClass f1, MyClass f2)
    {
        return new MyClass();
    }

    public static void Main()
    {
        MyClass f = new MyClass();
        int i = f && f;
```

```
    }  
}
```

## See also

- [Operator overloading](#)
- [true and false operators](#)

# Compiler Error CS0218

Article • 09/15/2021 • 2 minutes to read

The type ('type') must contain declarations of operator true and operator false

If a user-defined type overloads the [& operator](#) or [| operator](#), it must also define [true](#) and [false](#) operators, in order to make short-circuiting [&& operator](#) or [|| operator](#) defined.

The following sample generates CS0218:

C#

```
// CS0218.cs
using System;
public class MyClass
{
    // uncomment these operator declarations to resolve this CS0218
    /*
    public static bool operator true (MyClass f)
    {
        return false;
    }

    public static bool operator false (MyClass f)
    {
        return false;
    }
    */

    public static implicit operator int(MyClass x)
    {
        return 0;
    }

    public static MyClass operator & (MyClass f1, MyClass f2)
    {
        return new MyClass();
    }

    public static void Main()
    {
        MyClass f = new MyClass();
        int i = f && f;    // CS0218, requires operators true and false
    }
}
```

## See also

- Operator overloading

# Compiler Error CS0220

Article • 08/24/2022 • 2 minutes to read

The operation overflows at compile time in checked mode

An operation was detected by [checked](#), which is the default for constant expressions, and a data loss resulted. Either correct the inputs to the assignment or use [unchecked](#) to resolve this error. For more information, see the [checked and unchecked statements](#) article.

The following sample generates CS0220:

C#

```
// CS0220.cs
using System;

class TestClass
{
    const int x = 1000000;
    const int y = 1000000;

    public int MethodCh()
    {
        int z = (x * y);    // CS0220
        return z;
    }

    public int MethodUnCh()
    {
        unchecked
        {
            int z = (x * y);
            return z;
        }
    }

    public static void Main()
    {
        TestClass myObject = new TestClass();
        Console.WriteLine("Checked : {0}", myObject.MethodCh());
        Console.WriteLine("Unchecked: {0}", myObject.MethodUnCh());
    }
}
```

# Compiler Error CS0221

Article • 08/24/2022 • 2 minutes to read

Constant value 'value' cannot be converted to a 'type' (use 'unchecked' syntax to override)

An assignment operation that would result in a data loss was detected by [checked](#), which is on by default for constant expressions. Either correct the assignment or use [unchecked](#) to resolve this error. For more information, see the [checked and unchecked statements](#) article.

The following sample generates CS0221:

```
C#  
  
// CS0221.cs  
public class MyClass  
{  
    public static void Main()  
    {  
        // unchecked  
        // {  
        int a = (int)0xFFFFFFFF;    // CS0221  
        a++;  
        // }  
    }  
}
```

# Compiler Error CS0225

Article • 09/15/2021 • 2 minutes to read

The `params` parameter must be a single dimensional array

When using the `params` keyword, you must specify a single-dimensional array of the data type. For more information, see [Methods](#).

## Example

The following sample generates CS0225:

C#

```
// CS0225.cs
public class MyClass
{
    public static void TestParams(params int a)    // CS0225
    // try the following line instead
    // public static void TestParams(params int[] a)
    {
    }

    public static void Main()
    {
        TestParams(1);
    }
}
```

# Compiler Error CS0226

Article • 09/15/2021 • 2 minutes to read

An `_arglist` expression may only appear inside of a call or new expression.

The unsupported keyword `_arglist` can only appear in a method call or a new expression.

## Example

The following code generates CS0226:

C#

```
// cs0226.cs
using System;

public class C
{
    public static int Main ()
    {
        _arglist(1, "This is a string"); // CS0226
        return 0;
    }
}
```

# Compiler Error CS0227

Article • 09/15/2021 • 2 minutes to read

Unsafe code may only appear if compiling with /unsafe

If source code contains the `unsafe` keyword, then the [AllowUnsafeBlocks](#) compiler option must also be used. For more information, see [Unsafe Code and Pointers](#).

To set the unsafe option in Visual Studio 2012, click on **Project** in the main menu, select the **Build** pane, and check the box that says "allow unsafe code."

The following sample, when compiled without `/unsafe`, generates CS0227:

```
C#  
  
// CS0227.cs  
public class MyClass  
{  
    unsafe public static void Main()    // CS0227  
    {  
    }  
}
```

## See also

- [C# Compiler Errors](#)



# Compiler Error CS0228

Article • 09/15/2021 • 2 minutes to read

'type' does not contain a definition for 'member', or it is not accessible

You may have replaced the system version of **System.Hashtable**, **System.String**, or **System.Array**, and the replacement does not contain `member`.

Otherwise, you may need to repair or reinstall Visual Studio.

# Compiler Error CS0229

Article • 09/15/2021 • 2 minutes to read

Ambiguity between 'member1' and 'member2'

Members of different interfaces have the same name. If you want to keep the same names, you must qualify the names. For more information, see [Interfaces](#).

## ⓘ Note

In some cases, this ambiguity can be resolved by providing an explicit prefix to the identifier via a **using** alias.

## Example

The following example generates CS0229:

```
C#  
  
// CS0229.cs  
  
interface IList  
{  
    int Count  
    {  
        get;  
        set;  
    }  
  
    void Counter();  
}  
  
interface ICounter  
{  
    double Count  
    {  
        get;  
        set;  
    }  
}  
  
interface IListCounter : IList, ICounter {}  
  
class MyClass  
{  
    void Test(IListCounter x)  
    {
```

```
x.Count = 1; // CS0229
// Try one of the following lines instead:
// ((IList)x).Count = 1;
// or
// ((ICounter)x).Count = 1;
}

public static void Main() {}
}
```

# Compiler Error CS0230

Article • 09/15/2021 • 2 minutes to read

Type and identifier are both required in a foreach statement

A `foreach` statement was poorly formed.

The following sample generates CS0230:

C#

```
// CS0230.cs
using System;

class MyClass
{
    public static void Main()
    {
        int[] myarray = new int[3] {1,2,3};

        foreach (int in myarray) // CS0230
        // try the following line instead
        // foreach (int x in myarray)
        {
            Console.WriteLine(x);
        }
    }
}
```

# Compiler Error CS0231

Article • 09/08/2022 • 2 minutes to read

A `params` parameter must be the last parameter in a formal parameter list.

The `params` parameter supports a variable number of arguments and must be after all other parameters. For more information, see [Methods](#).

The following sample generates CS0231:

C#

```
// CS0231.cs
class Test
{
    public void TestMethod(params int[] p, int i) {} // CS0231
    // To resolve the error, use the following line instead:
    // public void TestMethod(int i, params int[] p) {}

    static void Main()
    {
    }
}
```

# Compiler Error CS0233

Article • 09/03/2022 • 2 minutes to read

'identifier' does not have a predefined size, therefore sizeof can only be used in an unsafe context

Without `unsafe` context, the `sizeof` operator can only be used for types whose size is a compile-time constant. If you are getting this error, use an unsafe context.

The following example generates CS0233:

```
C#  
  
// CS0233.cs  
using System;  
using System.Runtime.InteropServices;  
  
[StructLayout(LayoutKind.Sequential)]  
public struct S  
{  
    public int a;  
}  
  
public class MyClass  
{  
    public static void Main()  
    {  
        S myS = new S();  
        Console.WriteLine(sizeof(S));    // CS0233  
        // Try the following instead:  
        // unsafe  
        // {  
        //     Console.WriteLine(sizeof(S));  
        // }  
    }  
}
```

# Compiler Error CS0234

Article • 09/15/2021 • 2 minutes to read

The type or namespace name 'name' does not exist in the namespace 'namespace' (are you missing an assembly reference?)

A type was expected. Possible causes for this error include the following:

- An assembly that contains the definition of a type was not referenced in the compilation; use [References \(Import Metadata\)](#) to specify the assembly
- You passed a variable name to the `typeof` operator.
- You tried to reference an assembly that is not part of your .NET target framework moniker (TFM). For more information, see [Troubleshooting .NET Targeting Errors](#).

If you see this error after moving code from one development machine to another, make sure that the project on the new machine has the correct references, and that the versions of the assemblies are the same as on the old machine. You can also use the Object Browser to inspect an assembly and verify whether it contains the types that you expect it to contain.

The following sample generates CS0234:

```
C#  
  
// CS0234.cs  
public class C  
{  
    public static void Main()  
    {  
        System.DateTime x = new System.DateTim(); // CS0234  
        // try the following line instead  
        // System.DateTime x = new System.DateTime();  
    }  
}
```

# Compiler Error CS0236

Article • 09/15/2021 • 2 minutes to read

A field initializer cannot reference the non-static field, method, or property 'name'.

Instance fields cannot be used to initialize other instance fields outside a method.

## To correct this error

If you are trying to initialize a variable outside a method, consider performing the initialization inside the class constructor. For more information, see [Methods](#).

## Example

The following sample generates CS0236, and shows how to fix it:

C#

```
public class MyClass
{
    public int i = 5;

    // To fix the error, remove "= i", and uncomment the line in
    // constructor.
    public int j = i; // CS0236

    public MyClass()
    {
        // Uncomment the following.
        //j = i;
    }
}
```

# Compiler Error CS0238

Article • 09/15/2021 • 2 minutes to read

'member' cannot be sealed because it is not an override

[sealed](#) was used on a member that was not also marked with [override](#). For more information, see [Inheritance](#).

The following sample generates CS0238:

C#

```
// CS0238.cs
abstract class MyClass
{
    public abstract void f();
}

class MyClass2 : MyClass
{
    public static void Main()
    {

        public sealed void f() // CS0238
        // Try the following definition instead:
        // public override sealed void f()
        {
        }
}
```

# Compiler Error CS0239

Article • 08/18/2022 • 2 minutes to read

'member' : cannot override inherited member 'inherited member' because it is sealed

A member cannot [override](#) a [sealed](#) inherited member. For more information, see [Inheritance](#).

The following sample generates CS0239:

C#

```
// CS0239.cs
abstract class MyClass
{
    public abstract void f();
}

class MyClass2 : MyClass
{
    public static void Main()
    {
    }

    public override sealed void f()
    {
    }
}

class MyClass3 : MyClass2
{
    public override void f()    // CS0239
    // Try the following definition instead:
    // public new void f()
    {
    }
}
```

# Compiler Error CS0241

Article • 09/21/2022 • 2 minutes to read

Default parameter specifiers are not permitted

[Method parameters](#) cannot have default values. Use method overloads if you want to achieve the same effect.

## Example

The following sample generates CS0241. In addition, the sample shows how to simulate, with overloading, a method with default arguments.

C#

```
// CS0241.cs
public class A
{
    public void Test(int i = 9) {}    // CS0241
}

public class B
{
    public void Test() { Test(9); }
    public void Test(int i) {}
}

public class C
{
    public static void Main()
    {
        B x = new B();
        x.Test();
    }
}
```

# Compiler Error CS0242

Article • 09/15/2021 • 2 minutes to read

The operation in question is undefined on void pointers

Incrementing a void pointer is not allowed. For more information, see [Unsafe Code and Pointers](#).

The following sample generates CS0242:

C#

```
// CS0242.cs
// compile with: /unsafe
class TestClass
{
    public unsafe void Test()
    {
        void * p = null;
        p++;    // CS0242, incrementing a void pointer not allowed
    }

    public static void Main()
    {
    }
}
```

# Compiler error CS0243

Article • 09/15/2021 • 2 minutes to read

The Conditional attribute is not valid on 'method' because it is an override method

The [ConditionalAttribute](#) attribute is not allowed on a method that is marked with the [override](#) keyword. For more information, see [Knowing When to Use Override and New Keywords](#).

The compiler never binds to override methods. It only binds to the base method, and the common language runtime calls the override, as appropriate.

The following code generates CS0243:

C#

```
// CS0243.cs
// compile with: /target:library
public class MyClass
{
    public virtual void M() {}

public class MyClass2 : MyClass
{
    [System.Diagnostics.ConditionalAttribute("MySymbol")] // CS0243
    // remove Conditional attribute or remove override keyword
    public override void M() {}
}
```

# Compiler Error CS0244

Article • 09/15/2021 • 2 minutes to read

Neither 'is' nor 'as' is valid on pointer types

The [is](#) and [as](#) operators are not valid for use on pointer types. For more information, see [Unsafe Code and Pointers](#).

The following sample generates CS0244:

C#

```
// CS0244.cs
// compile with: /unsafe

class UnsafeTest
{
    unsafe static void SquarePtrParam (int* p)
    {
        bool b = p is object;    // CS0244 p is pointer
    }

    unsafe public static void Main()
    {
        int i = 5;
        SquarePtrParam (&i);
    }
}
```

# Compiler Error CS0245

Article • 09/17/2021 • 2 minutes to read

Destructors and object.Finalize cannot be called directly. Consider calling IDisposable.Dispose if available.

For more information, see [Programming Essentials for Garbage Collection](#) and [Finalizers](#).

The following sample generates CS0245:

C#

```
// CS0245.cs
using System;
using System.Collections;

class MyClass // : IDisposable
{
    /*
    public void Dispose()
    {
        // cleanup code goes here
    }
    */

    void m()
    {
        this.Finalize();    // CS0245
        // this.Dispose();
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0246

Article • 11/01/2022 • 4 minutes to read

The type or namespace name 'type/namespace' could not be found (are you missing a using directive or an assembly reference?)

A type or namespace that is used in the program was not found. You might have forgotten to reference ([References](#)) the assembly that contains the type, or you might not have added the required [using directive](#). Or, there might be an issue with the assembly you are trying to reference.

The following situations cause compiler error CS0246.

- Did you misspell the name of the type or namespace? Without the correct name, the compiler cannot find the definition for the type or namespace. This often occurs because the casing used in the name of the type is not correct. For example, `Dataset ds;` generates CS0246 because the s in `Dataset` must be capitalized.
- If the error is for a namespace name, did you add a reference ([References](#)) to the assembly that contains the namespace? For example, your code might contain the directive `using Accessibility`. However, if your project does not reference the assembly `Accessibility.dll`, error CS0246 is reported. For more information, see [Managing references in a project](#)
- If the error is for a type name, did you include the proper [using directive](#), or, alternatively, fully qualify the name of the type? Consider the following declaration: `DataSet ds`. To use the `DataSet` type, you need two things. First, you need a reference to the assembly that contains the definition for the `DataSet` type. Second, you need a `using` directive for the namespace where `DataSet` is located. For example, because `DataSet` is located in the `System.Data` namespace, you need the following directive at the beginning of your code: `using System.Data`.

The `using` directive is not required. However, if you omit the directive, you must fully qualify the `DataSet` type when referring to it. Full qualification means that you specify both the namespace and the type each time you refer to the type in your code. If you omit the `using` directive in the previous example, you must write `System.Data.DataSet ds` to declare `ds` instead of `DataSet ds`.

- Did you use a variable or some other language element where a type was expected? For example, in an `is` statement, if you use a `Type` object instead of an actual type, you get error CS0246.

- Did you reference the assembly that was built against a higher framework version than the target framework of the program? Or did you reference the project that is targeting a higher framework version than the target framework of the program? For example, you work on the project that is targeting .NET Framework 4.6.1 and use the type from the project that is targeting .NET Framework 4.7.1. Then you get error CS0246.
- Are all referenced projects included in the selected build configuration and platform? Use the Visual Studio Configuration Manager to make sure all referenced projects are marked to be built with the selected configuration and platform.
- Did you use a *using alias directive* without fully qualifying the type name? A `using` alias directive does not use the `using` directives in the source code file to resolve types. The following example generates CS0246 because the type `List<int>` is not fully qualified. The `using` directive for `System.Collections.Generic` does not prevent the error.

C#

```
using System.Collections.Generic;

// The following declaration generates CS0246.
using myAliasName = List<int>;

// To avoid the error, fully qualify List.
using myAliasName2 = System.Collections.Generic.List<int>;
```

If you get this error in code that was previously working, first look for missing or unresolved references in Solution Explorer. Do you need to reinstall a [NuGet](#) package? For information about how the build system searches for references, see [Resolving file references in team build](#). If all references seem to be correct, look in your source control history to see what has changed in your .csproj file and/or your local source file.

If you haven't successfully accessed the reference yet, use the Object Browser to inspect the assembly that is supposed to contain this namespace and verify that the namespace is present. If you verify with Object Browser that the assembly contains the namespace, try removing the `using` directive for the namespace and see what else breaks. The root problem may be with some other type in another assembly.

The following example generates CS0246 because a necessary `using` directive is missing.

```
C#  
  
// CS0246.cs  
//using System.Diagnostics;  
  
public class MyClass  
{  
    // The following line causes CS0246. To fix the error, uncomment  
    // the using directive for the namespace for this attribute,  
    // System.Diagnostics.  
    [Conditional("A")]  
    public void Test()  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

The following example causes CS0246 because an object of type `Type` was used where an actual type was expected.

```
C#  
  
// CS0246b.cs  
using System;  
  
class ExampleClass  
{  
    public bool supports(object o, Type t)  
    {  
        // The following line causes CS0246. You must use an  
        // actual type, such as ExampleClass, String, or Type.  
        if (o is t)  
        {  
            return true;  
        }  
        return false;  
    }  
}  
  
class Program  
{  
    public static void Main()  
    {  
        ExampleClass myC = new ExampleClass();  
        myC.supports(myC, myC.GetType());  
    }  
}
```



# Compiler Error CS0247

Article • 09/15/2021 • 2 minutes to read

Cannot use a negative size with stackalloc

A negative number was passed to a `stackalloc` statement.

The following sample generates CS0247:

C#

```
// CS0247.cs
// compile with: /unsafe
public class MyClass
{
    unsafe public static void Main()
    {
        int *p = stackalloc int[-30]; // CS0247
    }
}
```

# Compiler Error CS0248

Article • 09/15/2021 • 2 minutes to read

Cannot create an array with a negative size

An array size was specified with a negative number. For more information, see [Arrays](#).

## Example

The following sample generates CS0248:

C#

```
// CS0248.cs
class MyClass
{
    public static void Main()
    {
        int[] myArray = new int[-3] {1,2,3};    // CS0248, pass a nonnegative
number
    }
}
```

# Compiler Error CS0249

Article • 09/15/2021 • 2 minutes to read

Do not override `object.Finalize`. Instead, provide a destructor.

Use finalizer syntax to specify instructions to execute when your object is destroyed.

The following sample generates CS0249:

C#

```
// CS0249.cs
class MyClass
{
    protected override void Finalize()    // CS0249
    // try the following line instead
    // ~MyClass()
    {

    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0250

Article • 09/17/2021 • 2 minutes to read

Do not directly call your base class `Finalize` method. It is called automatically from your `destructor`.

A program cannot attempt to force cleanup of base class resources.

See [Finalizers](#) for more information.

The following sample generates CS0250

C#

```
// CS0250.cs
class B
{
}

class C : B
{
    ~C()
    {
        base.Finalize();    // CS0250
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0254

Article • 09/10/2022 • 2 minutes to read

The right hand side of a fixed statement assignment may not be a cast expression

The right side of a [fixed](#) expression may not use a cast. For more information, see [Unsafe Code and Pointers](#).

The following sample generates CS0254:

C#

```
// CS0254.cs
// compile with: /unsafe
class Point
{
    public uint x, y;
}

class FixedTest
{
    unsafe static void SquarePtrParam (int* p)
    {
        *p *= *p;
    }

    unsafe public static void Main()
    {
        Point pt = new Point();
        pt.x = 5;
        pt.y = 6;

        fixed (int* p = (int*)&pt.x)    // CS0254
        // try the following line instead
        // fixed (uint* p = &pt.x)
        {
            SquarePtrParam ((int*)p);
        }
    }
}
```

# Compiler Error CS0255

Article • 10/27/2021 • 2 minutes to read

stackalloc may not be used in a catch or finally block

You cannot use the [stackalloc operator](#) in a [catch](#) or [finally](#) block. For more information, see [Exceptions and Exception Handling](#).

The following sample generates CS0255:

C#

```
// CS0255.cs
// compile with: /unsafe
using System;

public class TestTryFinally
{
    public static unsafe void Test()
    {
        int i = 123;
        string s = "Some string";
        object o = s;

        try
        {
            // Conversion is not valid; o contains a string not an int
            i = (int) o;
        }
        finally
        {
            Console.WriteLine("i = {0}", i);
            int* fib = stackalloc int[100];    // CS0255
        }
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0260

Article • 09/15/2021 • 2 minutes to read

Missing partial modifier on declaration of type 'type'; another partial declaration of this type exists

This error indicates that you have declared multiple classes that have the same name. In addition, at least one but not all of the declarations contains the `partial` modifier. If you want to define a class in several parts, you must declare each part by using the keyword `partial`.

This error also occurs if you declare a class and accidentally give it the same name as a partial class that's declared elsewhere in the same namespace.

The following sample generates CS0260:

C#

```
// CS0260.cs
// You must mark both parts of the definition of class C
// by using the partial keyword.

// The following line causes CS0260. To resolve the error, add
// the 'partial' keyword to the declaration.
class C
{
}

partial class C
{
}
```

## See also

- [Partial Classes and Methods](#)



# Compiler Error CS0261

Article • 09/15/2021 • 2 minutes to read

Partial declarations of 'type' must be all classes, all structs, or all interfaces

This error occurs if a partial type is declared as a different type of construct in various places. For more information, see [Partial Classes and Methods](#).

The following sample generates CS0261:

C#

```
// CS0261.cs
partial class A // CS0261 - A declared as a class here, but as a struct
below
{
}

partial struct A
{
}
```

# Compiler Error CS0262

Article • 09/15/2021 • 2 minutes to read

Partial declarations of 'type' have conflicting accessibility modifiers

This error occurs if a partial type has inconsistent modifiers such as public, private, protected, internal, or abstract. These modifiers must be consistent in all partial declarations for that type. For more information, see [Partial Classes and Methods](#).

## Example

The following sample generates CS0262:

C#

```
// CS0262.cs
class A
{
    public partial class C // CS0262
    {
    }
    private partial class C
    {
    }
}
```

# Compiler Error CS0263

Article • 09/15/2021 • 2 minutes to read

Partial declarations of 'type' must not specify different base classes

When defining a type in partial declarations, specify the same base types in each of the partial declarations. For more information, see [Partial Classes and Methods](#).

The following sample generates CS0263:

C#

```
// CS0263.cs
// compile with: /target:library
class B1
{
}

class B2
{
}
partial class C : B1 // CS0263 - is the base class B1 or B2?
{
}

partial class C : B2
{
}
```

# Compiler Error CS0264

Article • 09/15/2021 • 2 minutes to read

Partial declarations of 'type' must have the same type parameter names in the same order

This error occurs if you are defining a generic type in partial declarations and the type parameters are not consistent in name or order throughout all of the partial declarations. To get rid of this error, check the type parameters for each partial declaration and make sure the same name and order of parameters is used. For more information, see [Partial Classes and Methods](#) and [Generic Type Parameters](#).

## Example

The following example generates CS0264.

C#

```
// CS0264.cs

partial class MyClass<T> // CS0264
{
}

partial class MyClass <MyType>
{
```

# Compiler Error CS0265

Article • 09/15/2021 • 2 minutes to read

Partial declarations of 'type' have inconsistent constraints for type parameter 'type parameter'

This error happens when you define a generic class as a partial class, so that its partial definitions occur in more than one place, and the constraints on the generic type are inconsistent or different in two or more places. If you specify the constraints in more than one place, they must all be identical. The easiest solution is to specify the constraints in one place, and omit them everywhere else. For more information, see [Partial Classes and Methods](#) and [Constraints on Type Parameters](#).

The following code generates error CS0265.

## Example

In this code, the partial class definitions are all in a single file, but they could also be spread across multiple files.

C#

```
// CS0265.cs
public class GenericsErrors
{
    interface IFace1 { }
    interface IFace2 { }
    partial class PartialBadBounds<T> where T : IFace1 { } // CS0265
    partial class PartialBadBounds<T> where T : IFace2 { }
}
```

# Compiler Error CS0266

Article • 09/15/2021 • 2 minutes to read

Cannot implicitly convert type 'type1' to 'type2'. An explicit conversion exists (are you missing a cast?)

This error occurs when your code tries to convert between two types that cannot be implicitly converted, but where an explicit conversion is available. For more information, see [Casting and Type Conversions](#).

The following code shows examples that generate CS0266:

```
C#  
  
// CS0266.cs  
class MyClass  
{  
    public static void Main()  
    {  
        // You cannot implicitly convert a double to an integer.  
        double d = 3.2;  
  
        // The following line causes compiler error CS0266.  
        int i1 = d;  
  
        // However, you can resolve the error by using an explicit  
        // conversion.  
        int i2 = (int)d;  
  
        // You cannot implicitly convert an object to a class type.  
        object obj = new MyClass();  
  
        // The following assignment statement causes error CS0266.  
        MyClass myClass = obj;  
  
        // You can resolve the error by using an explicit conversion.  
        MyClass c = (MyClass)obj;  
  
        // You cannot implicitly convert a base class object to a derived  
        // class type.  
        MyClass mc = new MyClass();  
        DerivedClass dc = new DerivedClass();  
  
        // The following line causes compiler error CS0266.  
        dc = mc;  
  
        // You can resolve the error by using an explicit conversion.  
        dc = (DerivedClass)mc;  
    }  
}
```

```
class DerivedClass : MyClass
{
}
```

# Compiler Error CS0267

Article • 09/15/2021 • 2 minutes to read

The partial modifier can only appear immediately before 'class', 'record', 'struct', 'interface', or a method return type.

The placement of the **partial** modifier was incorrect in the declaration of the class, record, struct, interface, or method. To fix the error, reorder the modifiers. For more information, see [Partial Classes and Methods](#).

The following sample generates CS0267:

C#

```
public partial class MyClass
{
}

partial public class MyClass // CS0267
// Try this line instead:
// public partial class MyClass
{}
```

# Compiler Error CS0268

Article • 09/15/2021 • 2 minutes to read

Imported type 'type' is invalid. It contains a circular base class dependency.

This error occurs if a type imported from another language has a circular base class dependency. Such a type cannot be used in a C# program. To resolve this error, check types imported from other languages in any referenced assemblies or modules.

# Compiler Error CS0269

Article • 09/21/2022 • 2 minutes to read

Use of unassigned out parameter 'parameter'

The compiler could not verify that the out parameter was assigned a value before it was used; its value may be undefined when assigned. Be sure to assign a value to `out` parameters in the called method before accessing the value. If you need to use the value of the variable passed in, use a `ref` parameter instead. For more information, see [Method Parameters](#).

## Example 1

The following sample generates CS0269:

```
C#  
  
// CS0269.cs  
class C  
{  
    public static void F(out int i)  
    // One way to resolve the error is to use a ref parameter instead  
    // of an out parameter.  
    //public static void F(ref int i)  
    {  
        // The following line causes a compiler error because no value  
        // has been assigned to i.  
        int k = i;  // CS0269  
        i = 1;  
        // The error does not occur if the order of the two previous  
        // lines is reversed.  
    }  
  
    public static void Main()  
    {  
        int myInt = 1;  
        F(out myInt);  
        // If the declaration of method F is changed to require a ref  
        // parameter, ref must be specified in the call as well.  
        //F(ref myInt);  
    }  
}
```

## Example 2

This could also occur if the initialization of a variable occurs in a try block, which the compiler is unable to verify will execute successfully:

C#

```
// CS0269b.cs
class C
{
    public static void F(out int i)
    {
        try
        {
            // Assignment occurs, but compiler can't verify it
            i = 1;
        }
        catch
        {
        }

        int k = i; // CS0269
        i = 1;
    }

    public static void Main()
    {
        int myInt;
        F(out myInt);
    }
}
```

# Compiler Error CS0270

Article • 09/15/2021 • 2 minutes to read

Array size cannot be specified in a variable declaration (try initializing with a 'new' expression)

This error occurs when a size is specified as part of an array declaration. To resolve, use the [new operator](#) expression.

The following example generates CS0270:

```
C#  
  
// CS0270.cs  
// compile with: /t:module  
  
public class Test  
{  
    int[10] a;    // CS0270  
    // To resolve, use the following line instead:  
    // int[] a = new int[10];  
}
```

# Compiler Error CS0271

Article • 09/15/2021 • 2 minutes to read

The property or indexer 'property/indexer' cannot be used in this context because the get accessor is inaccessible

This error occurs when you try to access an inaccessible `get` accessor. To resolve this error, increase the accessibility of the accessor, or change the calling location. For more information, see [Accessor Accessibility](#) and [Properties](#).

The following example generates CS0271:

```
C#  
  
// CS0271.cs  
public class MyClass  
{  
    public int Property  
    {  
        private get { return 0; }  
        set { }  
    }  
  
    public int Property2  
    {  
        get { return 0; }  
        set { }  
    }  
}  
  
public class Test  
{  
    public static void Main(string[] args)  
    {  
        MyClass c = new MyClass();  
        int a = c.Property;    // CS0271  
        int b = c.Property2;   // OK  
    }  
}
```

# Compiler Error CS0272

Article • 09/21/2022 • 2 minutes to read

The property or indexer 'property/indexer' cannot be used in this context because the set accessor is inaccessible.

This error occurs when the `set` accessor is not accessible to the program code.

## To correct this error

Increase the accessibility of the accessor, or change the calling location. For more information, see [Restricting Accessor Accessibility](#).

## Example

The following example generates CS0272:

C#

```
// CS0272.cs
public class MyClass
{
    public int Property
    {
        get { return 0; }
        private set { }
    }
}

public class Test
{
    static void Main()
    {
        MyClass c = new MyClass();
        c.Property = 10;          // CS0272
        // To resolve, remove the previous line
        // or use an appropriate modifier on the set accessor.
    }
}
```

# Compiler error CS0273

Article • 09/22/2022 • 2 minutes to read

The accessibility modifier of the 'property\_accessor' accessor must be more restrictive than the property or indexer 'property'

The accessibility modifier of the set/get accessor must be more restrictive than the property or indexer 'property/indexer'

This error occurs when the accessibility of the accessor you declared isn't less restrictive than the accessibility of the property or indexer.

## To correct this error

Use the appropriate access modifier on either the property or the accessor. For more information, see [Restricting Accessor Accessibility](#) and [Accessors](#).

## Example

This sample contains an internal property with an internal set method. The following sample generates CS0273.

```
C#  
  
// CS0273.cs  
// compile with: /target:library  
public class MyClass  
{  
    internal int Property  
    {  
        get { return 0; }  
        internal set {} // CS0273  
        // try the following line instead  
        // private set {}  
    }  
}
```

# Compiler Error CS0274

Article • 09/15/2021 • 2 minutes to read

Cannot specify accessibility modifiers for both accessors of the property or indexer 'property/indexer'

This error occurs when you declare a property or indexer with access modifiers on both its accessors. To resolve this error, use an access modifier on only one of the two accessors. For more information, see [Accessor Accessibility](#).

The following example generates CS0274:

C#

```
// CS0274.cs
public class MyClass
{
    public int Property // CS0274
    {
        public get { return 0; }
        protected set { }
    }
}
```

# Compiler Error CS0275

Article • 09/15/2021 • 2 minutes to read

'accessor': accessibility modifiers may not be used on accessors in an interface

This error occurs when you use an access modifier on any one of the accessors of a property or indexer in an interface. To resolve, remove the access modifier.

## Example

The following example generates CS0275:

C#

```
// CS0275.cs
public interface MyInterface
{
    int Property
    {
        get;
        internal set;    // CS0275
    }
}
```

# Compiler Error CS0276

Article • 09/15/2021 • 2 minutes to read

'property/indexer': accessibility modifiers on accessors may only be used if the property or indexer has both a get and a set accessor

This error occurs when you declare a property or indexer with one accessor only, and use an access modifier on the accessor. To resolve, remove the access modifier or add another accessor.

## Example

The following example generates CS0276:

```
C#  
  
// CS0276.cs  
public class MyClass  
{  
    public int Property  
    {  
        protected set { } // CS0276  
    }  
    public int Property2  
    {  
        internal get { } // CS0276  
    }  
}
```

# Compiler Error CS0277

Article • 09/15/2021 • 2 minutes to read

'class' does not implement interface member 'accessor'. 'class accessor' is not public

This error occurs when you try to implement a property of an interface, but the implementation of the property accessor in the class is not public. Methods that implement interface members need to have public accessibility. To resolve, remove the access modifier on the property accessor.

## Example

The following example generates CS0277:

```
C#  
  
// CS0277.cs  
public interface MyInterface  
{  
    int Property  
    {  
        get;  
        set;  
    }  
}  
  
public class MyClass : MyInterface // CS0277  
{  
    public int Property  
    {  
        get { return 0; }  
        // Try this instead:  
        //set { }  
        protected set { }  
    }  
}
```

# Compiler Error CS0281

Article • 09/15/2021 • 2 minutes to read

Friend access was granted to 'AssemblyName1', but the output assembly is named 'AssemblyName2'. Try adding a reference to 'AssemblyName1' or changing the output assembly name to match.

Friend access is a new common language runtime (CLR) feature that enables an assembly to see another assembly's non-public types. This error occurs when the assembly granting friend access specifies the wrong name for the grantee assembly. For more information, see [Friend Assemblies](#).

## Example 1

The following sequence of code samples will generate CS0281.

The files used to create the strong named assemblies are generated as follows:

- sn -d CS0281.snk
- sn -k CS0281.snk
- sn -i CS0281.snk CS0281.snk
- sn -pc CS0281.snk key.publickey
- sn -tp key.publickey

C#

```
// CS0281.cs
// compile with: /target:library /keyfile:CS0281.snk
public class A {}
```

## Example 2

C#

```
// CS0281_b.cs
// compile with: /target:library /keyfile:CS0281.snk /reference:CS0281.dll
[assembly:System.Runtime.CompilerServices.InternalsVisibleTo("CS0281,
PublicKey=00240000480000940000006020000024000525341310004000010001004b
2d4d56af7c50be2fcbbf97cb880b9e73ad84467a587191fef63aadcc118a96cecf9d508cd679c
907b6e20f71684300bdc2c0a851019af0c96b29bf8f1339753276041aefd67db46139e6348b3
```

```
a12f29537b4dc6c2c19829df2c9ed6803f3c63c3b84cfa2728849386aea575c543a5f70fa857  
93d2946f15f7fe1ccb0c5e8fe0")]
class B : A {}
```

## Example 3

The following sample generates CS0281.

Notice that this sample creates an output file with the same name as the output file in the first sample. To resolve, do not change the assembly attributes of the component and add class C.

C#

```
// CS0281_c.cs
// compile with: /target:library /out:CS0281.dll /keyfile:CS0281.snk
//reference:CS0281_b.dll
// CS0281 expected
[assembly:System.Reflection.AssemblyVersion("3")]
[assembly:System.Reflection.AssemblyCulture("en-us")]
class C : B {}
public class A {}
```

# Compiler Error CS0283

Article • 09/15/2021 • 2 minutes to read

The type 'type' cannot be declared const

The type specified in a `constant` declaration must be `byte`, `sbyte`, `ushort`, `short`, `uint`, `int`, `ulong`, `long`, `char`, `float`, `double`, `decimal`, `bool`, `string`, an `enum` type, or a reference type that is assigned a value of `null`. Each constant expression must yield a value of the target type or of a type that is implicitly convertible to the target type.

## Example

The following example generates CS0283.

C#

```
// CS0283.cs
struct MyTest
{
}
class MyClass
{
    // To resolve the error but retain the "const-ness",
    // change const to readonly.
    const MyTest test = new MyTest();    // CS0283

    public static int Main() {
        return 1;
    }
}
```

# Compiler Error CS0304

Article • 09/29/2021 • 2 minutes to read

Cannot create an instance of the variable type 'type' because it does not have the new() constraint

When you implement a generic class, and you want to use the `new` keyword to create a new instance of any type that is supplied for a type parameter `T`, you must apply the [new\(\) constraint](#) to `T` in the class declaration, as shown in the following example.

C#

```
class C<T> where T : new()
```

The `new()` constraint enforces type safety by guaranteeing that any concrete type that is supplied for `T` has a parameterless constructor. CS0304 occurs if you attempt to use the `new` operator in the body of the class to create an instance of type parameter `T` when `T` does not specify the `new()` constraint. On the client side, if code attempts to instantiate the generic class with a type that has no parameterless constructor, that code will generate [Compiler Error CS0310](#).

The following example generates CS0304.

C#

```
// CS0304.cs
// Compile with: /target:library.
class C<T>
{
    // The following line generates CS0304.
    T t = new T();
}
```

The `new` operator also is not allowed in methods of the class.

C#

```
// Compile with: /target:library.
class C<T>
{
    public void ExampleMethod()
    {
        // The following line generates CS0304.
        T t = new T();
```

```
    }  
}
```

To avoid the error, declare the class by using the `new()` constraint, as shown in the following example.

C#

```
// Compile with: /target:library.  
class C<T> where T : new()  
{  
    T t = new T();  
  
    public void ExampleMethod()  
    {  
        T t = new T();  
    }  
}
```

## See also

- [C# Compiler Errors](#)

# Compiler Error CS0305

Article • 09/15/2021 • 2 minutes to read

Using the generic type 'generic type' requires 'number' type arguments

This error occurs when the expected number of type arguments was not found. To resolve C0305, use the required number of type arguments.

## Example

The following sample generates CS0305.

C#

```
// CS0305.cs
public class MyList<T> {}
public class MyClass<T> {}

class MyClass
{
    public static void Main()
    {
        MyList<MyClass, MyClass> list1 = new MyList<MyClass>(); // CS0305
        MyList<MyClass> list2 = new MyList<MyClass>(); // OK
    }
}
```

# Compiler Error CS0306

Article • 09/15/2021 • 2 minutes to read

The type 'type' may not be used as a type argument

The type used as a type parameter is not allowed. This could be because the type is a pointer type.

The following example generates CS0306:

C#

```
// CS0306.cs
class C<T>
{
}

class M
{
    // CS0306 - int* not allowed as a type parameter
    C<int*> f;
}
```

# Compiler Error CS0307

Article • 09/15/2021 • 2 minutes to read

The 'construct' 'identifier' is not a generic method. If you intended an expression list, use parentheses around the < expression.

The construct named was not a type or a method, the only constructs that can take generic arguments. Remove the type arguments in angle brackets. If a generic is needed, declare your generic construct as a generic type or method.

The following sample generates CS0307:

```
C#  
  
// CS0307.cs  
class C  
{  
    public int P { get { return 1; } }  
    public static void Main()  
    {  
        C c = new C();  
        int p = c.P<int>(); // CS0307 - C.P is a property  
        // Try this instead  
        // int p = c.P;  
    }  
}
```

# Compiler Error CS0308

Article • 09/15/2021 • 2 minutes to read

The non-generic type-or-method 'identifier' cannot be used with type arguments.

The method or type is not generic, but it was used with type arguments. To avoid this error, remove the angled brackets and type arguments, or redeclare the method or type as a generic method or type.

The following example generates CS0308:

```
C#  
  
// CS0308a.cs  
class MyClass  
{  
    public void F() {}  
    public static void Main()  
    {  
        F<int>(); // CS0308 - F is not generic.  
        // Try this instead:  
        // F();  
    }  
}
```

The following example also generates CS0308. To resolve the error, use the directive "using System.Collections.Generic."

```
C#  
  
// CS0308b.cs  
// compile with: /t:library  
using System.Collections;  
// To resolve, uncomment the following line:  
// using System.Collections.Generic;  
public class MyStack<T>  
{  
    // Store the elements of the stack:  
    private T[] items = new T[100];  
    private int stack_counter = 0;  
  
    // Define the iterator block:  
    public I IEnumerator<T> GetEnumerator() // CS0308  
    {  
        for (int i = stack_counter - 1 ; i >= 0; i--)  
            yield return items[i];  
    }  
}
```

# Compiler Error CS0310

Article • 09/29/2021 • 2 minutes to read

The type 'typename' must be a non-abstract type with a public parameterless constructor in order to use it as parameter 'parameter' in the generic type or method 'generic'

The generic type or method defines the [new\(\) constraint](#) in its `where` clause, so any type must have a public parameterless constructor in order to be used as a type argument for that generic type or method. To avoid this error, make sure that the type has the correct constructor, or modify the constraint clause of the generic type or method.

## Example

The following sample generates CS0310:

C#

```
// CS0310.cs
using System;

class G<T> where T : new()
{
    T t;

    public G()
    {
        t = new T();
        Console.WriteLine(t);
    }
}

class B
{
    private B() { }
    // Try this instead:
    // public B() { }
}

class CMain
{
    public static void Main()
    {
        G<B> g = new G<B>();    // CS0310
        Console.WriteLine(g.ToString());
    }
}
```

# Compiler Error CS0311

Article • 09/15/2021 • 2 minutes to read

The type 'type1' cannot be used as type parameter 'T' in the generic type or method '<name>'. There is no implicit reference conversion from 'type1' to 'type2'.

When a constraint is applied to a generic type parameter, an implicit identity or reference conversion must exist from the concrete argument to the type of the constraint.

## To correct this error

1. Change the argument you are using to create the class.
2. If you own the class, you can remove the constraint or else do something to enable an implicit reference or identity conversion. For example, you can make the second type inherit from the first.

## Example

C#

```
// cs0311.cs
class B {}
class C {}
class Test<T> where T : C
{ }

class Program
{
    static void Main()
    {
        Test<B> test = new Test<B>(); //CS0311
    }
}
```

If this error occurs when trying to use a value-type argument, notice that an implicit numeric conversion, for example from `short` to `int`, does not satisfy a generic type parameter.

## See also

- Constraints on Type Parameters

# Compiler Error CS0312

Article • 09/15/2021 • 2 minutes to read

The type 'type1' cannot be used as type parameter 'name' in the generic type or method 'name'. The nullable type 'type1' does not satisfy the constraint of 'type2'.

A nullable value type is distinct from its non-nullable counterpart; no implicit reference conversion or identify conversion exists between them. A nullable boxing conversion does not satisfy a generic type constraint. In the example that follows, the first type parameter is a `Nullable<int>` and the second type parameter is a `System.Int32`.

## To correct this error

1. Remove the constraint.
2. In the following example, make the second type argument either `int?` or `object`.

## Example

The following code generates CS0312:

```
C#  
  
// cs0312.cs  
class Program  
{  
    static void MTyVar<T, U>() where T : U { }  
  
    static int Main()  
    {  
        MTyVar<int?, int>(); // CS0312  
        return 1;  
    }  
}
```

Although a nullable value type is distinct from a non-nullable type, various kinds of conversions are allowed between nullable and non-nullable values.

## See also

- [Nullable value types](#)



# Compiler Error CS0313

Article • 09/15/2021 • 2 minutes to read

The type 'type1' cannot be used as type parameter 'parameter name' in the generic type or method 'type2'. The nullable type 'type1' does not satisfy the constraint of 'type2'.

Nullable types cannot satisfy any interface constraints.

A nullable value type is not equivalent to its non-nullable counterpart. In the example that follows, `ImplStruct` satisfies the `BaseInterface` constraint but `ImplStruct?` does not because `Nullable<ImplStruct>` does not implement `BaseInterface`.

## To correct this error

1. Using the code that follows as an example, one solution is to specify an ordinary `ImplStruct` as the first type argument in the call to `TestMethod`. Then modify `TestMethod` to create a nullable version of `ImplStruct` in its return statement:

C#

```
return new Nullable<T>(t);
```

## Example

The following code generates CS0313:

```
C#  
  
// cs0313.cs  
public interface BaseInterface { }  
public struct ImplStruct : BaseInterface { }  
  
public class TestClass  
{  
    public T? TestMethod<T, U>(T t) where T : struct, U  
    {  
        return t;  
    }  
}  
  
public class NullableTest  
{  
    public static void Run()  
    {
```

```
    TestClass tc = new TestClass();
    tc.TestMethod<ImplStruct?, BaseInterface>(new ImplStruct?()); // CS0313
}
public static void Main()
{
}
}
```

## See also

- [Nullable value types](#)

# Compiler Error CS0314

Article • 09/15/2021 • 2 minutes to read

The type 'type1' cannot be used as type parameter 'name' in the generic type or method 'name'. There is no boxing conversion or type parameter conversion from 'type1' to 'type2'.

When a generic type uses a type parameter that is constrained, the new class must also satisfy those same constraints.

## To correct this error

1. In the example that follows, add `where T : ClassConstraint` to class `B`.

## Example

The following code generates CS0314:

```
C#  
  
// cs0314.cs  
// Compile with: /target:library  
public class ClassConstraint { }  
  
public class A<T> where T : ClassConstraint  
{ }  
  
public class B<T> : A<T> //CS0314  
{ }  
  
// Try using this instead.  
public class C<T> : A<T> where T : ClassConstraint  
{ }
```

## See also

- [Constraints on Type Parameters](#)



# Compiler Error CS0315

Article • 09/15/2021 • 2 minutes to read

The type 'valueType' cannot be used as type parameter 'T' in the generic type or method 'TypeorMethod<T>'. There is no boxing conversion from 'valueType' to 'referenceType'.

This error occurs when you constrain a generic type to a particular class, and try to construct an instance of that class by using a value type that cannot be implicitly boxed to it.

## To correct this error

1. One solution is to redefine the struct as a class.

## Example

The following example generates CS0315:

C#

```
// cs0315.cs
public class ClassConstraint { }
public struct ViolateClassConstraint { }

public class Gen<T> where T : ClassConstraint
{
}
public class Test
{
    public static int Main()
    {
        Gen<ViolateClassConstraint> g = new Gen<ViolateClassConstraint>();
//CS0315
        return 1;
    }
}
```

## See also

- [Constraints on Type Parameters](#)
- [Boxing and Unboxing](#)



# Compiler Error CS0316

Article • 09/15/2021 • 2 minutes to read

The parameter name 'name' conflicts with an automatically-generated parameter name.

Reserved words cannot be used as parameter names. In the example that follows, `value` is a reserved word in the context of a default property or indexer accessor.

## To correct this error

1. Change the name of the parameter.

## Example

The following code generates CS0316:

```
C#  
  
// cs0316.cs  
// Compile with: /target:library  
public class Test  
{  
    public int this[int value] // CS0316  
    {  
        get { return 1; }  
        set { }  
    }  
}
```

## See also

- [Indexers](#)
- [C# Keywords](#)



# Compiler Error CS0400

Article • 09/15/2021 • 2 minutes to read

The type or namespace name 'identifier' could not be found in the global namespace  
(are you missing an assembly reference?)

The identifier scoped with the global scope operator ( :: ) was not found in the global namespace. You may be missing an assembly reference that contains the identifier, or the identifier may be declared in a class or namespace other than the global namespace. This error could also occur if a globally scoped identifier is not declared or is misspelled.

To avoid this error, locate the declaration of the identifier, verify the correct spelling, and if the declaration is in a separate assembly, make sure that you have the appropriate assembly reference. If the identifier is declared inside another type or namespace, use the fully-qualified name after the ::. The following sample generates CS0400:

```
C#  
  
// CS0400.cs  
class C  
{  
    public static void Main()  
    {  
        // CS0400 - D could not be found  
        // in the global namespace.  
        global::D d = new global::D();  
    }  
}
```

# Compiler Error CS0401

Article • 09/15/2021 • 2 minutes to read

The new() constraint must be the last constraint specified

When using multiple constraints, list all other constraints before the new() constraint.

## Example

The following sample generates CS0401.

C#

```
// CS0401.cs
// compile with: /target:library
using System;
class C<T> where T : new(), IDisposable {} // CS0401

class D<T> where T : IDisposable
{
    static void F<U>() where U : new(), IDisposable{} // CS0401
}
```

# Compiler Error CS0403

Article • 09/15/2021 • 2 minutes to read

Cannot convert null to type parameter 'name' because it could be a non-nullable value type. Consider using default('T') instead.

You cannot assign null to the unknown type named because it might be a value type, which does not allow null assignment. If your generic class is not intended to accept value types, use the class type constraint. If it can accept value types, such as the built-in types, you may be able to replace the assignment to null with the expression `default(T)`, as shown in the following example.

## Example

The following sample generates CS0403.

```
C#  
  
// CS0403.cs  
// compile with: /target:library  
class C<T>  
{  
    public void f()  
    {  
        T t = null; // CS0403  
        T t2 = default(T); // OK  
    }  
}  
  
class D<T> where T : class  
{  
    public void f()  
    {  
        T t = null; // OK  
    }  
}
```

# Compiler Error CS0404

Article • 09/15/2021 • 2 minutes to read

'<' unexpected : attributes cannot be generic

Generic type parameters are not allowed in attributes. Remove the type parameter and angled brackets.

The following sample generates CS0404:

C#

```
// CS0404.cs
[MyAttrib<int>] // CS0404
class C
{
    public static void Main()
    {

    }
}
```

# Compiler Error CS0405

Article • 09/15/2021 • 2 minutes to read

Duplicate constraint 'constraint' for type parameter 'type parameter'

Two of the constraints on the generic declaration are identical. To get rid of the error, remove the duplicate.

The following sample generates CS0405:

C#

```
// CS0405.cs
interface I
{
}

class C<T> where T : I, I // CS0405.cs
{}
```

# Compiler Error CS0406

Article • 09/15/2021 • 2 minutes to read

The class type constraint 'constraint' must come before any other constraints

When a generic type or method has a class type constraint, that constraint must be listed first. To avoid this error, move the class type constraint to the beginning of the constraint list.

## Example

The following sample generates CS0406.

C#

```
// CS0406.cs
// compile with: /target:library
interface I {}
class C {}
class D<T> where T : I, C {}    // CS0406
class D2<T> where T : C, I {}   // OK
```

# Compiler Error CS0407

Article • 09/15/2021 • 2 minutes to read

'return-type method' has the wrong return type

The method was not compatible with the delegate type. The argument types matched, but the return type was not the correct return type for that delegate. To avoid this error, use a different method, change the method's return type, or change the delegate's return type.

## Example

The following sample generates CS0407:

```
C#  
  
// CS0407.cs  
public delegate int MyDelegate();  
  
class C  
{  
    MyDelegate d;  
  
    public C()  
    {  
        d = new MyDelegate(F); // OK: F returns int  
        d = new MyDelegate(G); // CS0407 - G doesn't return int  
    }  
  
    public int F()  
    {  
        return 1;  
    }  
  
    public void G()  
    {  
    }  
  
    public static void Main()  
    {  
        C c1 = new C();  
    }  
}
```

# Compiler Error CS0409

Article • 09/15/2021 • 2 minutes to read

A constraint clause has already been specified for type parameter 'type parameter'. All of the constraints for a type parameter must be specified in a single where clause.

Multiple constraint clauses (where clauses) were found for a single type parameter. Remove the extraneous where clause, or correct the where clauses so that a unique type parameter in each clause.

C#

```
// CS0409.cs
interface I
{
}

class C<T1, T2> where T1 : I where T1 : I // CS0409 - T1 used twice
{}
```

# Compiler Error CS0410

Article • 09/15/2021 • 2 minutes to read

No overload for 'method' has the correct parameter and return types

This error occurs if you try to instantiate a delegate with a function that has the wrong parameter types. The parameter types of the delegate must match the function that you are assigning to the delegate.

## Example

The following example generates CS0410:

```
C#  
  
// CS0410.cs  
// compile with: /langversion:ISO-1  
  
class Test  
{  
    delegate void D(double d );  
    static void F(int i) { }  
  
    static void Main()  
    {  
        D d = new D(F); // CS0410  
    }  
}
```

### ⓘ Note

This compiler error is no longer used in Roslyn. The previous example generates CS0123 when compiled with Roslyn.

# Compiler Error CS0411

Article • 09/15/2021 • 2 minutes to read

The type arguments for method 'method' cannot be inferred from the usage. Try specifying the type arguments explicitly.

This error occurs if you call a generic method without explicitly providing the type arguments and the compiler cannot infer which type arguments are intended. To avoid this error, add the intended type arguments in angle brackets.

## Example 1

The following sample generates CS0411:

```
C#  
  
// CS0411.cs  
class C  
{  
    void G<T>()  
    {  
    }  
  
    public static void Main()  
    {  
        G(); // CS0411  
        // Try this instead:  
        // G<int>();  
    }  
}
```

## Example 2

Other possible error cases include when the parameter is `null`, which has no type information:

```
C#  
  
// CS0411b.cs  
class C  
{  
    public void F<T>(T t) where T : C  
    {  
    }
```

```
public static void Main()
{
    C c = new C();
    c.F(null); // CS0411
}
}
```

# Compiler Error CS0412

Article • 09/15/2021 • 2 minutes to read

'generic': a parameter or local variable cannot have the same name as a method type parameter

There is a name conflict between the type parameter of a generic method and a local variable in the method or one of the method's parameters. To avoid this error, rename any conflicting parameters or local variables.

## Example

The following sample generates CS0412:

```
C#  
  
// CS0412.cs  
using System;  
  
class C  
{  
    // Parameter name is the same as method type parameter name  
    public void G<T>(int T)    // CS0412  
    {  
    }  
    public void F<T>()  
    {  
        // Method local variable name is the same as method type  
        // parameter name  
        double T = 0.0;    // CS0412  
        Console.WriteLine(T);  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0413

Article • 09/15/2021 • 2 minutes to read

The type parameter 'type parameter' cannot be used with the 'as' operator because it does not have a class type constraint nor a 'class' constraint

This error occurs if a generic type uses the `as` operator, but that generic type does not have a class type constraint. The `as` operator is only allowed with reference and nullable value types, so the type parameter must be constrained to guarantee that it is not a value type. To avoid this error, use a class type constraint or a reference type constraint.

This is because the `as` operator could return `null`, which is not a possible value for a value type, and the type parameter must be treated as a value type unless it is a class type constraint or a reference type constraint.

## Example

The following sample generates CS0413.

```
C#  
  
// CS0413.cs  
// compile with: /target:library  
class A {}  
class B : A {}  
  
class CMain  
{  
    A a = null;  
    public void G<T>()  
    {  
        a = new A();  
        System.Console.WriteLine (a as T); // CS0413  
    }  
  
    // OK  
    public void H<T>() where T : A  
    {  
        a = new A();  
        System.Console.WriteLine (a as T);  
    }  
}
```

# Compiler Error CS0415

Article • 09/15/2021 • 2 minutes to read

The 'IndexerName' attribute is valid only on an indexer that is not an explicit interface member declaration

This error occurs if you use an IndexerName attribute on an indexer that was an explicit implementation of an interface. This error may be avoided by removing the interface name from the declaration of the indexer, if possible. For more information, see the [IndexerNameAttribute Class](#).

The following sample generates CS0415:

C#

```
// CS0415.cs
using System;
using System.Runtime.CompilerServices;

public interface IA
{
    int this[int index]
    {
        get;
        set;
    }
}

public class A : IA
{
    [IndexerName("Item")] // CS0415
    int IA.this[int index]
    // Try this line instead:
    // public int this[int index]
    {
        get { return 0; }
        set { }
    }

    static void Main()
    {
    }
}
```

# Compiler Error CS0416

Article • 09/15/2021 • 2 minutes to read

'type parameter': an attribute argument cannot use type parameters

A type parameter was used as an attribute argument, which is not allowed. Use a non-generic type.

The following sample generates CS0416:

C#

```
// CS0416.cs
public class MyAttribute : System.Attribute
{
    public MyAttribute(System.Type t)
    {
    }
}

class G<T>
{
    [MyAttribute(typeof(G<T>))] // CS0416
    public void F()
    {
    }
}
```

# Compiler Error CS0417

Article • 09/15/2021 • 2 minutes to read

'identifier': cannot provide arguments when creating an instance of a variable type

This error occurs if a call to the `new` operator on a type parameter has arguments. The only constructor that can be called by using the `new` operator on an unknown parameter type is a constructor that has no arguments. If you need to call another constructor, consider using a class type constraint or interface constraint.

## Example

The following example generates CS0417:

C#

```
// CS0417
class ExampleClass<T> where T : new()
{
    // The following line causes CS0417.
    T instance1 = new T(1);

    // The following line doesn't cause the error.
    T instance2 = new T();
}
```

## See also

- [Constraints on Type Parameters](#)



# Compiler Error CS0418

Article • 09/15/2021 • 2 minutes to read

'class name': an abstract class cannot be sealed or static

An abstract class cannot be used to create objects unless it is derived from, so it makes no sense to be sealed. An abstract class cannot meaningfully be static either; abstract classes are designed to support an object hierarchy that will use the abstract class as a base.

## Example

The following sample generates CS0418:

```
C#  
  
// CS0418.cs  
public abstract sealed class C // CS0418  
{  
}  
  
sealed static class S // CS0418  
{  
}  
  
public class MyClass  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0423

Article • 09/15/2021 • 2 minutes to read

Since 'class' has the ComImport attribute, 'method' must be extern or abstract

Specifying the ComImport attribute implies that the implementation for the class is to be imported from a COM module. Additional methods may not be defined.

The following sample generates CS0423:

```
C#  
  
// CS0423.cs  
  
using System.Runtime.InteropServices;  
  
[  
    ComImport,  
    Guid("7ab770c7-0e23-4d7a-8aa2-19bfad479829")  
]  
class ImageProperties  
{  
    public static void Main() // CS0423  
    {  
        ImageProperties i = new ImageProperties();  
    }  
}
```

# Compiler Error CS0424

Article • 09/15/2021 • 2 minutes to read

'class': a class with the ComImport attribute cannot specify a base class

Specifying the [ComImportAttribute](#) attribute implies that the implementation for the class is to be imported from a COM module. Additional methods or fields inherited from the base class are not allowed to be added to the implementation defined in the COM module.

The following sample generates CS0424:

C#

```
// CS0424.cs
// compile with: /target:library
using System.Runtime.InteropServices;
public class A {}

[ ComImport, Guid("7ab770c7-0e23-4d7a-8aa2-19bfad479829") ]
class B : A {} // CS0424 error
```

# Compiler Error CS0425

Article • 09/15/2021 • 2 minutes to read

The constraints for type parameter 'type parameter' of method 'method' must match the constraints for type parameter 'type parameter' of interface method 'method'.

Consider using an explicit interface implementation instead.

This error occurs if a virtual generic method is overridden in a derived class and the constraints on the method in the derived class do not match the constraints on the method in the base class. To avoid this error, make sure the `where` clause is identical in both declarations, or implement the interface explicitly.

## Example 1

The following example generates CS0425:

```
C#  
  
// CS0425.cs  
  
class C1  
{  
}  
  
class C2  
{  
}  
  
interface IBase  
{  
    void F<ItemType>(ItemType item) where ItemType : C1;  
}  
  
class Derived : IBase  
{  
    public void F<ItemType>(ItemType item) where ItemType : C2 // CS0425  
    {  
    }  
}  
  
class CMain  
{  
    public static void Main()  
    {  
    }  
}
```

## Example 2

The constraints do not have to be a literal match, as long as the set of constraints has the same meaning. For example, the following is okay:

C#

```
// CS0425b.cs

interface J<Z>
{
}

interface I<S>
{
    void F<T>(S s, T t) where T: J<S>, J<int>;
}

class C : I<int>
{
    public void F<X>(int s, X x) where X : J<int>
    {
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0426

Article • 10/27/2021 • 2 minutes to read

The type name 'identifier' does not exist in the type 'type'

This error occurs when you reference a type nested within another type, but no such nested type exists. This can occur if you mistype the name of the nested type. Check the spelling of the names used, and verify that the enclosing type has the expected member.

The following sample generates CS0426 because class C has no nested type A:

```
C#  
  
// CS0426.cs  
  
class C  
{  
    // No nested types are declared.  
}  
  
class D  
{  
    public static void Main()  
    {  
        C c = new C();  
        // Attempt to reference a nested type A:  
        C.A a; // CS0426 because no such type C.A  
    }  
}
```

## See also

- [The C# type system](#)



# Compiler Error CS0428

Article • 09/15/2021 • 2 minutes to read

Cannot convert method group 'Identifier' to non-delegate type 'type'. Did you intend to invoke the method?

This error occurs when converting a method group to a non-delegate type, or attempting to invoke a method without using parentheses.

## Example

The following sample generates CS0428:

C#

```
// CS0428.cs
namespace ConsoleApplication1
{
    class Program
    {
        delegate int Del1();
        delegate object Del2();

        static void Main(string[] args)
        {
            ExampleClass ec = new ExampleClass();

            // The following assignment statement causes compiler error
CS0428.
            // It attempts to assign the address of Method1 to an integer
variable.
            // You can only assign the address to an appropriate delegate
type.
            int i = ec.Method1;

            // Del1 is a delegate type that is appropriate for a method like
            // Method1 that returns an int. The following assignment
statement
            // does not cause an error.
            Del1 d1 = ec.Method1;

            // You can invoke Method1 and assign the int that is returned to
            // integer variable i.
            i = ec.Method1();

            // The following assignment statement causes compiler error
CS0428.
            // It attempts to assign the address of Method2 to an instance
of
```

```
// ExampleClass. You can only assign the address to a delegate
type.
    ec = ExampleClass.Method2;

    // Del2 is a delegate type that is appropriate for a method like
    // Method2 that returns an instance of a class. The following
assignment
    // statement does not cause an error.
    Del2 d2 = ExampleClass.Method2;

    // Similarly, you can invoke Method2 and assign the result
returned to ec.
    ec = ExampleClass.Method2();
}
}

public class ExampleClass
{
    public int Method1() { return 1; }
    public static ExampleClass Method2() { return null; }
}
```

# Compiler Error CS0430

Article • 09/15/2021 • 2 minutes to read

The extern alias 'alias' was not specified in a /reference option

This error occurs when extern Alias is encountered but Alias was not specified as a reference on the command line. To resolve CS0430, compile with **/reference**.

## Example 1

```
C#  
  
// CS0430_a.cs  
// compile with: /target:library  
public class MyClass {}
```

## Example 2

Compiling with **/reference:MyType=cs0430\_a.dll** to refer to the DLL created in the previous sample resolves this error. The following sample generates CS0430.

```
C#  
  
// CS0430_b.cs  
extern alias MyType; // CS0430  
public class Test  
{  
    public static void Main() {}  
}
```

# Compiler Error CS0431

Article • 09/15/2021 • 2 minutes to read

Cannot use alias 'identifier' with '::' since the alias references a type. Use '.' instead.

You used "::" with an alias that references a type. To resolve this error, use the "." operator.

The following sample generates CS0431:

C#

```
// CS0431.cs
using A = Outer;

public class Outer
{
    public class Inner
    {
        public static void Meth() {}
    }
}

public class MyClass
{
    public static void Main()
    {
        A::Inner.Meth();    // CS0431
        A.Inner.Meth();    // OK
    }
}
```

# Compiler Error CS0432

Article • 09/15/2021 • 2 minutes to read

Alias 'identifier' not found

This error occurs when you use ":" to the right of an identifier that is not an alias. To resolve the error, use "." instead.

The following example generates CS0432:

C#

```
// CS0432.cs
namespace A {
    public class B {
        public static void Meth() { }
    }
}

public class Test
{
    public static void Main()
    {
        A::B.Meth();    // CS0432
        // To resolve, use the following line instead:
        // A.B.Meth();
    }
}
```

# Compiler Error CS0433

Article • 05/10/2022 • 2 minutes to read

The type TypeName1 exists in both TypeName2 and TypeName3

Two different assemblies referenced in your application contain the same namespace and type, which produces ambiguity.

To resolve this error, use the alias feature of the ([References](#)) compiler option or do not reference one of your assemblies.

This error can also occur if:

- The `@ Page` directive has a `CodeFile` attribute when it should be a `CodeBehind` attribute.
- Code is placed in an `App_Code` folder that shouldn't reside there.

## Examples

This code creates the DLL with the first copy of the ambiguous type.

```
C#  
  
// CS0433_1.cs  
// compile with: /target:library  
namespace TypeBindConflicts  
{  
    public class AggPubImpAggPubImp {}  
}
```

This code creates the DLL with the second copy of the ambiguous type.

```
C#  
  
// CS0433_2.cs  
// compile with: /target:library  
namespace TypeBindConflicts  
{  
    public class AggPubImpAggPubImp {}  
}
```

The following example generates CS0433.

```
C#
```

```
// CS0433_3.cs
// compile with: /reference:cs0433_1.dll /reference:cs0433_2.dll
using TypeBindConflicts;
public class Test
{
    public static void Main()
    {
        AggPubImpAggPubImp n6 = new AggPubImpAggPubImp(); // CS0433
    }
}
```

The following example shows how you can use the alias feature of the `/reference` compiler option to resolve this CS0433 error.

C#

```
// CS0433_4.cs
// compile with: /reference:cs0433_1.dll
/referencet:TypeBindConflicts=cs0433_2.dll
using TypeBindConflicts;
public class Test
{
    public static void Main()
    {
        AggPubImpAggPubImp n6 = new AggPubImpAggPubImp();
    }
}
```

# Compiler Error CS0434

Article • 09/15/2021 • 2 minutes to read

The namespace NamespaceName1 in NamespaceName2 conflicts with the type TypeName1 in NamespaceName3

This error occurs when an imported type and an imported nested namespace have the same fully qualified name. When that name is referenced, the compiler is unable to distinguish between the two. If you can change the imported source code, you can resolve the error by changing the name of either the type or the namespace so that both are unique within the assembly.

The following code generates error CS0434.

## Example 1

This code creates the first copy of the type with the identical fully qualified name.

```
C#  
  
// CS0434_1.cs  
// compile with: /t:library  
namespace TypeBindConflicts  
{  
    namespace NsImpAggPubImp  
    {  
        public class X { }  
    }  
}
```

## Example 2

This code creates the second copy of the type with the identical fully qualified name.

```
C#  
  
// CS0434_2.cs  
// compile with: /t:library  
namespace TypeBindConflicts {  
    // Conflicts with another import (import2.cs).  
    public class NsImpAggPubImp { }  
    // Try this instead:  
    // public class UniqueClassName { }  
}
```

## Example 3

This code references the type with the identical fully qualified name.

C#

```
// CS0434.cs
// compile with: /r:cs0434_1.dll /r:cs0434_2.dll
using TypeBindConflicts;
public class Test
{
    public TypeBindConflicts.NsImpAggPubImp.X n2 = null; // CS0434
}
```

# Compiler Error CS0438

Article • 09/15/2021 • 2 minutes to read

The type 'type' in 'module\_1' conflicts with the namespace 'namespace' in 'module\_2'.

This error occurs when a type in a source file conflicts with a namespace in another source file. This typically happens when one or both come from an added module. To resolve, rename the type or the namespace that caused the conflict.

The following example generates CS0438:

Compile this file first:

```
C#  
  
// CS0438_1.cs  
// compile with: /target:module  
public class Util  
{  
    public class A { }  
}
```

Then compile this file:

```
C#  
  
// CS0438_2.cs  
// compile with: /target:module  
namespace Util  
{  
    public class A { }  
}
```

And then compile this file:

```
C#  
  
// CS0438_3.cs  
// compile with: /addmodule:CS0438_1.netmodule /addmodule:CS0438_2.netmodule  
using System;  
public class Test  
{  
    public static void Main() {  
        Console.WriteLine(typeof(Util.A)); // CS0438  
    }  
}
```

# Compiler Error CS0439

Article • 09/15/2021 • 2 minutes to read

An extern alias declaration must precede all other elements defined in the namespace

This error occurs when an `extern` declaration comes after something else, such as a `using` declaration, in the same namespace. The `extern` declarations must come before all other namespace elements.

## Example

The following example generates CS0439:

C#

```
// CS0439.cs
using System;

extern alias MyType;    // CS0439
// To resolve the error, make the extern alias the first line in the file.

public class Test
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0441

Article • 09/15/2021 • 2 minutes to read

'class': a class cannot be both static and sealed

This error occurs when you declare a class that is both static and sealed. Static classes are inherently sealed, so the sealed modifier is not necessary. To resolve, use one modifier only.

The following example generates CS0441:

C#

```
// CS0441.cs
sealed static class MyClass { } // CS0441
```

# Compiler Error CS0442

Article • 09/15/2021 • 2 minutes to read

'Property': abstract properties cannot have private accessors

This error occurs when you use the access modifier "private" to modify an abstract accessor. To resolve, use a different access modifier, or make the property non-abstract.

## Example

The following sample generates CS0442:

```
C#  
  
// CS0442.cs  
public abstract class MyClass  
{  
    public abstract int AbstractProperty  
    {  
        get;  
        private set; // CS0442  
        // Try this instead:  
        // set;  
    }  
}
```

# Compiler Error CS0443

Article • 09/15/2021 • 2 minutes to read

Syntax error, value expected

This error occurs when you reference an array without specifying a value for the array index.

## Example

The following code generates CS0443.

C#

```
// CS0443.cs
using System;
class MyClass
{
    public static void Main()
    {
        int[,] x = new int[1,5];
        if (x[] == 5) {} // CS0443
        // if (x[0, 0] == 5) {}
    }
}
```

# Compiler Error CS0445

Article • 09/15/2021 • 2 minutes to read

Cannot modify the result of an unboxing conversion

The result of an unboxing conversion is a temporary variable. The compiler prevents you from modifying such variables because any modification would go away when the temporary variable goes away. To fix this, declare a new value-type variable to store the intermediate expression, and assign the result of the unboxing conversion to that variable.

The following code generates CS0455.

C#

```
// CS0445.cs
class UnboxingTest
{
    public static void Main()
    {
        Point p;
        p.x = 1;
        p.y = 2;
        object obj = p;
        // The following line generates CS0445, because the result
        // of unboxing obj is a temporary variable.
        ((Point)obj).x = 2;

        // The following lines resolve the error.

        // Store the result of the unboxing conversion in p2.
        Point p2;
        p2 = (Point)obj;
        // Then you can modify the unboxed value.
        p2.x = 2;
    }
}

struct Point
{
    public int x, y;
}
```

# Compiler Error CS0446

Article • 09/15/2021 • 2 minutes to read

Foreach cannot operate on a 'Method or Delegate'. Did you intend to invoke the 'Method or Delegate'?

This error is caused by specifying a method without parentheses or an anonymous method without parentheses in the part of the `foreach` statement where you would normally put a collection class. Note that it is valid, though unusual, to put a method call in that location, if the method returns a collection class.

## Example

The following code will generate CS0446.

C#

```
// CS0446.cs
using System;
class Tester
{
    static void Main()
    {
        int[] intArray = new int[5];
        foreach (int i in M) { } // CS0446
    }
    static void M() { }
}
```

# Compiler Error CS0447

Article • 09/15/2021 • 2 minutes to read

Attributes cannot be used on type arguments, only on type parameters

This error occurs when you apply an attribute to a type argument that occurs in an invocation statement. It is acceptable to apply an attribute to a type parameter in a class or method declaration statement such as the following:

```
C#
```

```
class C<[some attribute] T> {...}
```

The following line of code will generate this error. It is assumed that the class `C`, defined in the previous line of code, has a static method called `MyStaticMethod`.

```
C#
```

```
C<[some attribute] T>.MyStaticMethod();
```

## Example

The following code generates error CS0447:

```
C#
```

```
// CS0447.cs
using System;

namespace Test41
{
    public interface I<A>
    {
        void Meth<B>();
    }
    public class B : I<int>
    {
        void I<[Test] int>.Meth<X>() { } // CS0447
    }
}
```

 Note

This compiler error is no longer used in Roslyn.

# Compiler Error CS0448

Article • 09/15/2021 • 2 minutes to read

The return type for `++` or `--` operator must be the containing type or derived from the containing type

When you override the `++` or `--` operators, they must return the same type as the containing type, or return a type that is derived from the containing type.

## Example 1

The following sample generates CS0448.

C#

```
// CS0448.cs
class C5
{
    public static int operator ++(C5 c) { return null; }    // CS0448
    public static C5 operator --(C5 c) { return null; }    // OK
    public static void Main() {}
}
```

## Example 2

The following sample generates CS0448.

C#

```
// CS0448_b.cs
public struct S
{
    public static S? operator ++(S s) { return new S(); }    // CS0448
    public static S? operator --(S s) { return new S(); }    // CS0448
}

public struct T
{
    // OK
    public static T operator --(T t) { return new T(); }
    public static T operator ++(T t) { return new T(); }

    public static T? operator --(T? t) { return new T(); }
    public static T? operator ++(T? t) { return new T(); }
```

```
    public static void Main() {}  
}
```

# Compiler Error CS0449

Article • 09/15/2021 • 2 minutes to read

The 'class' or 'struct' constraint must come before any other constraints

The constraints on the type parameter of a generic type or method must occur in a specific order: `class` or `struct` must be first, if present, then any interface constraints, and finally any constructor constraints. This error is caused by the `class` or `struct` constraint not appearing first. To resolve this error, reorder the constraint clauses.

## Example

The following sample generates CS0449.

C#

```
// CS0449.cs
// compile with: /target:library
interface I {}
public class C4
{
    public void F1<T>() where T : class, struct, I {} // CS0449
    public void F2<T>() where T : I, struct {} // CS0449
    public void F3<T>() where T : I, class {} // CS0449

    // OK
    public void F4<T>() where T : class {}
    public void F5<T>() where T : struct {}
    public void F6<T>() where T : I {}
}
```

# Compiler Error CS0450

Article • 09/15/2021 • 2 minutes to read

'Type Parameter Name': cannot specify both a constraint class and the 'class' or 'struct' constraint

If a type parameter is constrained by the struct type constraint, it is logically contradictory for it to also be constrained by a specific class type, since struct and class are mutually exclusive categories. If a type parameter is constrained by a specific class type constraint, then it is by definition constrained by the class type constraint, and so specifying the class type constraint is redundant.

## Example

C#

```
// CS0450.cs
// compile with: /t:library
public class GenericsErrors
{
    public class B { }
    public class G3<T> where T : struct, B { } // CS0450
    // To resolve, use the following line instead:
    // public class G3<T> where T : B { }
}
```

# Compiler Error CS0451

Article • 09/15/2021 • 2 minutes to read

The 'new()' constraint cannot be used with the 'struct' constraint

When specifying constraints on the type of a generic, the `new()` constraint may only be used with class type constraints, interface type constraints, reference type constraints, and type parameter constraints, but not with value type constraints.

## Example

The following sample generates CS0451.

C#

```
// CS0451.cs
using System;
public class C4
{
    public void F4<T>() where T : struct, new() {} // CS0451
}

// OK
public class C5
{
    public void F5<T>() where T : struct {}
}

public class C6
{
    public void F6<T>() where T : new() {}
}
```

# Compiler Error CS0452

Article • 09/15/2021 • 2 minutes to read

The type 'type name' must be a reference type in order to use it as parameter 'parameter name' in the generic type or method 'identifier of generic'

This error occurs when you pass a value type such as a `struct` or `int` as a parameter to a generic type or method that has a reference type constraint.

## Example

The following code generates error CS0452.

C#

```
// CS0452.cs
using System;
public class BaseClass<S> where S : class { }
public class Derived1 : BaseClass<int> { } // CS0452
public class Derived2<S> : BaseClass<S> where S : struct { } // CS0452
```

## See also

- [Constraints on Type Parameters](#)



# Compiler Error CS0453

Article • 09/15/2021 • 2 minutes to read

The type 'Type Name' must be a non-nullable value type in order to use it as parameter 'Parameter Name' in the generic type or method 'Generic Identifier'

This error occurs when you use a non-value type argument in instantiating a generic type or method that has the **value** constraint on it. It can also occur when you use a nullable value type argument. See the last two lines of code in the following example.

## Example

The following code generates this error.

C#

```
// CS0453.cs
using System;
public class HV<S> where S : struct { }
public class H1 : HV<string> { } // CS0453
public class H2 : HV<H1> { } // CS0453
public class H3<S> : HV<S> where S : class { } // CS0453
public class H4 : HV<int?> { } // CS0453
public class H5 : HV<Nullable<Nullable<int>>> { } // CS0453
```

# Compiler Error CS0454

Article • 09/15/2021 • 2 minutes to read

Circular constraint dependency involving 'Type Parameter 1' and 'Type Parameter 2'

This error arises because at some point one type parameter refers to another, and the second refers back to the first. To fix this error, break the circular dependency by removing one of the constraints. Note that the circular constraint dependency can be indirect.

## Example 1

The following code generates error CS0454.

C#

```
// CS0554
using System;
public class GenericsErrors
{
    public class G4<T> where T : T { } // CS0454
}
```

## Example 2

The following example demonstrates a circular dependency between two type constraints.

C#

```
public class Gen<T,U> where T : U where U : T // CS0454
{ }
```

# Compiler Error CS0455

Article • 09/15/2021 • 2 minutes to read

Type parameter 'Type Parameter Name' inherits conflicting constraints 'Constraint Name 1' and 'Constraint Name 2'

Two common ways to get this error are to set up constraints so that the type parameter derives from two non-related classes, or so that it derives from a class type or reference type constraint and a `struct` type or value type constraint. To resolve this error, remove the conflict from your inheritance hierarchy.

## Example

The following code generates error CS0455.

C#

```
// CS0455.cs
using System;

public class GenericsErrors {
    public class B { }
    public class B2 { }
    public class G6<T> where T : B { public class N<U> where U : B2, T { } }
// CS0455
}
```

# Compiler Error CS0456

Article • 09/15/2021 • 2 minutes to read

Type parameter 'Type Parameter Name 1' has the 'struct' constraint so 'Type Parameter Name 1' cannot be used as a constraint for 'Type Parameter Name 2'

Value type constraints are implicitly sealed so those constraints cannot be used as constraints on a second type parameter. This is because value types cannot be overridden. To resolve this error, put a value type constraint directly on the second type parameter, instead of doing so indirectly by means of the first type parameter.

## Example

The following sample generates CS0456.

C#

```
// CS0456.cs
// compile with: /target:library
public class GenericsErrors
{
    public class G5<T> where T : struct
    {
        public class N<U> where U : T {} // CS0456
        public class N2<U> where U : struct {} // OK
    }
}
```

# Compiler Error CS0457

Article • 09/15/2021 • 2 minutes to read

Ambiguous user defined conversions 'Conversion method name 1' and 'Conversion method name 2' when converting from 'type name 1' to 'type name 2'

Two conversion methods are applicable, and the compiler is unable to decide which one to use.

One scenario that can cause this error is as follows:

- You want to convert class A to class B where A and B are unrelated.
- A is derived from A0, and there is a method that converts from A0 to B.
- B has a subclass B1 and there is a method that converts from A to B1.

The compiler will weight the two conversion methods equally, because the first conversion provides the best destination type, and the second conversion provides the best source type. Since the compiler will be unable to choose, this error will be generated. To resolve, write a new explicit method converting A to B.

Another scenario that causes this error is if there are two methods that convert A to B. To fix, specify which conversion to use through an explicit cast.

## Example

The following sample generates CS0457.

```
C#  
  
// CS0457.cs  
using System;  
public class A { }  
  
public class G0 { }  
public class G1<R> : G0 { }  
  
public class H0 {  
    public static implicit operator G0(H0 h) {  
        return new G0();  
    }  
}  
public class H1<R> : H0 {  
    public static implicit operator G1<R>(H1<R> h) {  
        return new G1<R>();  
}
```

```
    }

}

public class Test
{
    public static void F0(G0 g) {  }
    public static void Main()
    {
        H1<A> h1a = new H1<A>();
        F0(h1a); // CS0457
    }
}
```

# Compiler Error CS0459

Article • 09/15/2021 • 2 minutes to read

Cannot take the address of a read-only local variable

There are three common scenarios in the C# language that generate read-only local variables: `foreach`, `using`, and `fixed`. In each of these cases, you are not allowed to write to the read-only local variable, or to take its address. This error is generated when the compiler realizes you are trying to take the address of a read-only local variable.

## Example

The following example generates CS0459 when an attempt is made to take the address of a read-only local variable in a `foreach` loop and in a `fixed` statement block:

```
C#  
  
// CS0459.cs  
// compile with: /unsafe  
  
class Program  
{  
    public unsafe void M1()  
    {  
        int[] ints = new int[] { 1, 2, 3 };  
        foreach (int i in ints)  
        {  
            int *j = &i; // CS0459  
        }  
  
        fixed (int *i = &_i)  
        {  
            int **j = &i; // CS0459  
        }  
    }  
  
    private int _i = 0;  
}
```

### ⓘ Note

Roslyn compiler was updated and this compiler error was removed starting with Visual Studio 2017 version 15.5, so the previous code would compile successfully with this version and later.

# Compiler Error CS0460

Article • 09/15/2021 • 2 minutes to read

Constraints for override and explicit interface implementation methods are inherited from the base method, so they cannot be specified directly

When a generic method that is part of a derived class overrides a method in the base class, you may not specify constraints on the overridden method. The override method in the derived class inherits its constraints from the method in the base class.

## Example

The following sample generates CS0460.

```
C#  
  
// CS0460.cs  
// compile with: /target:library  
class BaseClass  
{  
    BaseClass() { }  
}  
  
interface I  
{  
    void F1<T>() where T : BaseClass;  
    void F2<T>() where T : struct;  
    void F3<T>() where T : BaseClass;  
}  
  
class ExpImpl : I  
{  
    void I.F1<T>() where T : BaseClass {} // CS0460  
    void I.F2<T>() where T : class {} // CS0460  
}
```

# Compiler Error CS0462

Article • 09/15/2021 • 2 minutes to read

The inherited members 'member1' and 'member2' have the same signature in type 'type', so they cannot be overridden

This error arises with the introduction of generics. Normally, you cannot have two versions of a method in a class with the same signature. But with generics, you can specify a generic method that might duplicate another method if it is instantiated with a particular type.

## Example

When `C<int>` is instantiated, two versions of the method `F` are created with the same signature, so the override in class `D` cannot decide which one to apply the override to.

The following sample generates CS0462.

```
C#  
  
// CS0462.cs  
// compile with: /target:library  
class C<T>  
{  
    public virtual void F(T t) {}  
    public virtual void F(int t) {}  
}  
  
class D : C<int>  
{  
    public override void F(int t) {}    // CS0462  
}
```

# Compiler Error CS0463

Article • 09/08/2022 • 2 minutes to read

Evaluation of the decimal constant expression failed with error: 'error'

This error occurs when a constant decimal expression overflows at compile time.

Typically you get overflow errors at run time. In this case, you defined the constant expression in such a way that the compiler could evaluate the result and know that an overflow would happen.

## Example

The following code generates error CS0463.

C#

```
// CS0463.cs
using System;
class MyClass
{
    public static void Main()
    {
        const decimal myDec = 79000000000000000000000000000000.0m +
79000000000000000000000000000000.0m; // CS0463
        Console.WriteLine(myDec.ToString());
    }
}
```

# Compiler Error CS0466

Article • 09/15/2021 • 2 minutes to read

'method1' should not have a params parameter since 'method2' does not

You cannot use `params` parameter on a class member if the implemented interface doesn't use it.

## Example

The following sample generates CS0466.

C#

```
// CS0466.cs
interface I
{
    void F1(params int[] a);
    void F2(int[] a);
}

class C : I
{
    void I.F1(params int[] a) {}
    void I.F2(params int[] a) {} // CS0466
    void I.F2(int[] a) {} // OK

    public static void Main()
    {
        I i = (I) new C();

        i.F1(new int[] {1, 2} );
        i.F2(new int[] {1, 2} );
    }
}
```

# Compiler Error CS0468

Article • 09/15/2021 • 2 minutes to read

Ambiguity between type 'type1' and type 'type2'

This error is generated when there are two types in the assembly being compiled that have the same fully qualified name. This could occur if they are both in added modules or one is in an added module and one in source.

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS0470

Article • 09/15/2021 • 2 minutes to read

Method 'method' cannot implement interface accessor 'accessor' for type 'type'. Use an explicit interface implementation.

This error is generated when an accessor is trying to implement an interface. Explicit interface implementation must be used.

## Example

The following sample generates CS0470.

C#

```
// CS0470.cs
// compile with: /target:library

interface I
{
    int P { get; }
}

class MyClass : I
{
    public int get_P() { return 0; }    // CS0470
    public int P2 { get { return 0; } }   // OK
}
```

# Compiler Error CS0471

Article • 09/15/2021 • 2 minutes to read

The method 'name' is not a generic method. If you intended an expression list, use parentheses around the < expression.

This error is generated when your code contains an expression list without parentheses.

## Example

The following example generates CS0471:

```
C#  
  
// CS0471.cs  
// compile with: /t:library  
class Test  
{  
    public void F(bool x, bool y) {}  
    public void F1()  
    {  
        int a = 1, b = 2, c = 3;  
        F(a<b, c>(3));      // CS0471  
        // To resolve, try the following instead:  
        // F((a<b), c>(3));  
    }  
}
```

### ⓘ Note

This compiler error is no longer used in Roslyn. The previous example should compile successfully.

# Compiler Error CS0473

Article • 09/15/2021 • 2 minutes to read

Explicit interface implementation 'method name' matches more than one interface member. Which interface member is actually chosen is implementation-dependent. Consider using a non-explicit implementation instead.

In some cases a generic method might acquire the same signature as a non-generic method. The problem is that there is no way in the common language infrastructure (CLI) metadata system to unambiguously state which method binds to which slot. It is up to the CLI to make that determination.

## To correct this error

To correct the error, eliminate the explicit implementation and implement both of the interface methods in the implicit implementation `public int TestMethod(int)`.

## Example

The following code generates CS0473:

```
C#  
  
public interface ITest<T>  
{  
    int TestMethod(int i);  
    int TestMethod(T i);  
}  
  
public class ImplementingClass : ITest<int>  
{  
    int ITest<int>.TestMethod(int i) // CS0473  
    {  
        return i + 1;  
    }  
  
    public int TestMethod(int i)  
    {  
        return i - 1;  
    }  
}  
  
class T  
{  
    static int Main()  
    {
```

```
ImplementingClass a = new ImplementingClass();
if (a.TestMethod(0) != -1)
    return -1;

ITest<int> i_a = a;
System.Console.WriteLine(i_a.TestMethod(0).ToString());
if (i_a.TestMethod(0) != 1)
    return -1;

return 0;
}
```

## See also

- [Odious ambiguous overloads, part two](#)

# Compiler Error CS0500

Article • 09/15/2021 • 2 minutes to read

'class member' cannot declare a body because it is marked abstract

An [abstract](#) method cannot contain its implementation.

The following sample generates CS0500:

C#

```
// CS0500.cs
namespace x
{
    abstract public class clk
    {
        abstract public void f(){} // CS0500
        // try the following line instead
        // abstract public void f();
    }

    public class cly
    {
        public static int Main()
        {
            return 0;
        }
    }
}
```

# Compiler Error CS0501

Article • 10/14/2022 • 2 minutes to read

'member function' must declare a body because it is not marked abstract, extern, or partial

Nonabstract methods must have implementations.

## Explanation

In C#, methods/functions that are a part of a class must have a "body", or implementation. The compiler needs to know what should happen when these methods are called, so that it knows what to execute. A method with no body is not acceptable to the compiler because it wants to avoid confusion about the intent of the code.

There are exceptions to this rule:

- When the method is marked `abstract` as an [Abstract Method](#)
- When the method is marked `extern` as an [External Method](#)
- When the method is marked `partial` as a [Partial Method](#)

## Example

The following sample generates CS0501:

```
C#  
  
public class MyClass  
{  
    public void MethodWithNoBody(); // CS0501 declared but not defined  
}
```

This could be fixed by declaring a body (by adding brackets):

```
C#  
  
public class MyClass  
{  
    public void MethodWithNoBody() { }; // No error; compiler now  
    interprets as an empty method  
}
```

Or, using an appropriate keyword, such as defining an `abstract` method:

C#

```
abstract class MyClass // class is abstract; classes that inherit from it
will have to define MyAbstractMethod
{
    public abstract void MyAbstractMethod(); // Compiler now knows that
this method must be defined by inheriting classes.
}
```

# Compiler Error CS0502

Article • 09/15/2021 • 2 minutes to read

'member' cannot be both abstract and sealed

A class member cannot be both [abstract](#) and [sealed](#).

The following sample generates CS0502:

C#

```
// CS0502.cs
public class B
{
    abstract public void F();
}

public class C : B
{
    abstract sealed override public void F()    // CS0502
    {
    }
}

public class CMain
{
    public static void Main()
    { }
}
```

# Compiler Error CS0503

Article • 09/15/2021 • 2 minutes to read

The abstract method 'method' cannot be marked virtual

It is redundant to mark a member method as both `abstract` and `virtual` because `abstract` implies `virtual`.

The following sample generates CS0503:

C#

```
// CS0503.cs
namespace x
{
    abstract public class clk
    {
        abstract virtual public void f();    // CS0503
    }
}
```

# Compiler Error CS0504

Article • 09/15/2021 • 2 minutes to read

The constant 'variable' cannot be marked static

If a variable is **const**, it is also **static**. If you want a **const** and **static** variable, just declare that variable as **const**; if all you want is a **static** variable, just mark it **static**.

The following sample generates CS0504:

C#

```
// CS0504.cs
namespace x
{
    abstract public class clk
    {
        static const int i = 0;    // CS0504, cannot be both static and const
        abstract public void f();
    }
}
```

# Compiler Error CS0505

Article • 09/15/2021 • 2 minutes to read

'member1': cannot override because 'member2' is not a function

A class declaration attempted to override a non-method in a base class. Overrides must match the member type. If a method with the same name as a method in a base class is desired, use [new](#) (and not [override](#)) on the method declaration in the base class.

The following sample generates CS0505:

```
C#  
  
// CS0505.cs  
// compile with: /target:library  
public class clk  
{  
    public int i;  
}  
  
public class cly : clk  
{  
    public override int i() { return 0; }    // CS0505  
}
```

# Compiler Error CS0506

Article • 09/15/2021 • 2 minutes to read

'function1' : cannot override inherited member 'function2' because it is not marked "virtual", "abstract", or "override"

A method was overridden that was not explicitly marked as **virtual**, **abstract**, or **override**.

The following sample generates CS0506:

C#

```
// CS0506.cs
namespace MyNameSpace
{
    abstract public class ClassX
    {
        public int i = 0;

        public int f()
        {
            return 0;
        }
        // Try the following definition for f() instead:
        // abstract public int f();
    }

    public class ClassY : ClassX
    {
        public override int f()    // CS0506
        {
            return 0;
        }

        public static int Main()
        {
            return 0;
        }
    }
}
```

# Compiler Error CS0507

Article • 09/15/2021 • 2 minutes to read

'function1' : cannot change access modifiers when overriding 'access' inherited member  
'function2'

An attempt was made to change the access specification in a method override.

## Example 1

The following sample generates CS0507.

```
C#  
  
// CS0507.cs  
abstract public class clx  
{  
    virtual protected void f() {}  
}  
  
public class cly : clx  
{  
    public override void f() {} // CS0507  
    public static void Main() {}  
}
```

## Example 2

CS0507 can also occur if a class attempts to override a method marked as `protected internal` defined in referenced metadata. In this situation, the overriding method should be marked as `protected`.

```
C#  
  
// CS0507_b.cs  
// compile with: /target:library  
abstract public class clx  
{  
    virtual protected internal void f() {}  
}
```

## Example 3

The following sample generates CS0507.

C#

```
// CS0507_c.cs
// compile with: /reference:cs0507_b.dll
public class cly : clx
{
    protected internal override void f() {}    // CS0507
    // try the following line instead
    // protected override void f() {}    // OK

    public static void Main() {}
}
```

# Compiler Error CS0508

Article • 09/15/2021 • 2 minutes to read

'Type 1': return type must be 'Type 2' to match overridden member 'Member Name'

An attempt was made to change the return type in a method override. To resolve this error, make sure both methods declare the same return type.

## Example

The following sample generates CS0508.

C#

```
// CS0508.cs
// compile with: /target:library
abstract public class Clx
{
    public int i = 0;
    // Return type is int.
    abstract public int F();
}

public class Cly : Clx
{
    public override double F()
    {
        return 0.0;    // CS0508
    }
}
```

# Compiler Error CS0509

Article • 09/15/2021 • 2 minutes to read

'class1' : cannot derive from sealed type 'class2'

A [sealed](#) class cannot act as a [base](#) class. Structs are sealed by default.

The following sample generates CS0509:

C#

```
// CS0509.cs
// compile with: /target:library
sealed public class clx {}
public class cly : clx {} // CS0509
```

# Compiler Error CS0513

Article • 09/15/2021 • 2 minutes to read

'function' is abstract but it is contained in nonabstract class 'class'

A method cannot be an [abstract](#) member of a nonabstract class.

The following sample generates CS0513:

C#

```
// CS0513.cs
namespace x
{
    public class clk
    {
        abstract public void f();    // CS0513, abstract member of nonabstract
    class
    }
}
```

# Compiler Error CS0514

Article • 09/15/2021 • 2 minutes to read

'constructor' : static constructor cannot have an explicit 'this' or 'base' constructor call

Calling `this` in the static constructor is not allowed because the static constructor is called automatically before creating any instance of the class. Also, static constructors are not inherited, and cannot be called directly.

For more information, see [this](#) and [base](#).

## Example

The following example generates CS0514:

```
C#  
  
// CS0514.cs  
class A  
{  
    static A() : base(0) // CS0514  
    {  
    }  
  
    public A(object o)  
    {  
    }  
}  
  
class B  
{  
    static B() : this(null) // CS0514  
    {  
    }  
  
    public B(object o)  
    {  
    }  
}
```

# Compiler Error CS0515

Article • 09/15/2021 • 2 minutes to read

'function' : access modifiers are not allowed on static constructors

A static constructor cannot have an [access modifier](#).

## Example

The following sample generates CS0515:

C#

```
// CS0515.cs
public class Clx
{
    public static void Main()
    {
    }
}

public class Clz
{
    public static Clz()    // CS0515, remove public keyword
    {
    }
}
```

# Compiler Error CS0516

Article • 09/15/2021 • 2 minutes to read

Constructor 'constructor' can not call itself

A program cannot recursively call constructors.

The following sample generates CS0516:

C#

```
// CS0516.cs
namespace x
{
    public class clx
    {
        public clx() : this() // CS0516, delete "this()"
        {

        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0517

Article • 09/15/2021 • 2 minutes to read

'class' has no base class and cannot call a base constructor

CS0517 can occur only when the .NET runtime compiles the source code for the object class, which is the only class that has no base class.

# Compiler Error CS0518

Article • 09/15/2021 • 2 minutes to read

Predefined type 'type' is not defined or imported

The main cause for this problem is that the project is not importing mscorel.dll, which defines the entire System namespace. This can be caused by one of the following:

- The **NoStandardLib** option from the command line compiler has been specified.  
The **NoStandardLib** option prevents the import of mscorel.dll. Use this option if you want to define or create a user-specific System namespace.
- An incorrect mscorel.dll is referenced.
- A corrupt Visual Studio .NET or .NET Framework common language runtime installation exists.
- Residual components from an earlier installation that are incompatible with the latest installation remain.

To resolve this problem, take one of the following actions:

- Do not specify the /nostdlib option from the command line compiler.
- Make sure that the project refers to the correct mscorel.dll.
- Reinstall the .NET Framework common language runtime (if the previous solutions do not solve the problem).



# Compiler Error CS0520

Article • 09/15/2021 • 2 minutes to read

Predefined type 'name' is declared incorrectly

One or more files needed by the compiler were not found. Reinstall the .NET runtime.

# Compiler Error CS0522

Article • 09/15/2021 • 2 minutes to read

'constructor' : structs cannot call base class constructors

A [struct](#) cannot call a base class constructor; remove the call to the base class constructor.

The following sample generates CS0522:

C#

```
// CS0522.cs
public class clk
{
    public clk(int i)
    {
    }

    public static void Main()
    {
    }
}

public struct cly
{
    public cly(int i):base(0) // CS0522
    // try the following line instead
    // public cly(int i)
    {
    }
}
```

# Compiler Error CS0523

Article • 09/15/2021 • 2 minutes to read

Struct member 'struct2 field' of type 'struct1' causes a cycle in the struct layout

The definitions of two structs include recursive references. Change the `struct` definitions such that each does not define itself on the other. This limitation applies only to structs, since structs are value types. If using recursive references, declare your types as classes.

The following sample generates CS0523:

```
C#  
  
// CS0523.cs  
// compile with: /target:library  
struct RecursiveLayoutStruct1  
{  
    public RecursiveLayoutStruct2 field;  
}  
  
struct RecursiveLayoutStruct2  
{  
    public RecursiveLayoutStruct1 field;    // CS0523  
}
```

# Compiler Error CS0524

Article • 09/15/2021 • 2 minutes to read

'type' : interfaces cannot declare types

An [interface](#) cannot contain a user-defined type; it should contain only methods and properties.

## Example

The following sample generates CS0524:

C#

```
// CS0524.cs
public interface Clx
{
    public class Cly    // CS0524, delete user-defined type
    {
    }
}
```

# Compiler Error CS0525

Article • 09/15/2021 • 2 minutes to read

Interfaces cannot contain fields

An [interface](#) can contain methods and properties but not fields.

The following sample generates CS0525:

C#

```
// CS0525.cs
namespace x
{
    public interface clx
    {
        public int i;    // CS0525
    }
}
```

# Compiler Error CS0526

Article • 09/15/2021 • 2 minutes to read

Interfaces cannot contain constructors

Constructors cannot be defined for [interfaces](#). A method is considered a constructor if it has the same name as the class and no return type.

The following sample generates CS0526:

C#

```
// CS0526.cs
namespace x
{
    public interface clx
    {
        public clx() // CS0526
    }
}

public class cly
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0527

Article • 09/15/2021 • 2 minutes to read

Type 'type' in interface list is not an interface

It is possible for a [struct](#) or [interface](#) to inherit from another interface but not from any other type.

The following sample generates CS0527:

C#

```
// CS0527.cs
// compile with: /target:library
public struct clk : int {} // CS0527 int not an interface
```

# Compiler Error CS0528

Article • 09/15/2021 • 2 minutes to read

'interface' is already listed in interface list

An interface-inheritance list includes a duplicate. An [interface](#) can only be specified once in the inheritance list.

The following sample generates CS0528:

C#

```
// CS0528.cs
namespace x
{
    public interface a
    {

        public class b : a, a    // CS0528
        {
            public void Main()
            {
            }
        }
    }
}
```

# Compiler Error CS0529

Article • 09/15/2021 • 2 minutes to read

Inherited interface 'interface1' causes a cycle in the interface hierarchy of 'interface2'

The inheritance list for an [interface](#) includes a direct or indirect reference to itself. An interface cannot inherit from itself.

The following sample generates CS0529:

C#

```
// CS0529.cs
namespace x
{
    public interface a
    {
    }

    public interface b : a, c
    {
    }

    public interface c : b    // CS0529, b inherits from c
    {
    }
}
```

# Compiler Error CS0531

Article • 09/15/2021 • 2 minutes to read

'member' : interface members cannot have a definition

Methods that are declared in an [interface](#) must be implemented in a class that inherits from it and not in the interface itself.

The following sample generates CS0531:

C#

```
// CS0531.cs
namespace x
{
    public interface clk
    {
        int xclk(); // CS0531, cannot define xclk
        // Try the following declaration instead:
        // int xclk();
        {
            return 0;
        }
    }

    public class cly
    {
        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0533

Article • 09/15/2021 • 2 minutes to read

'derived-class member' hides inherited abstract member 'base-class member'

A base [class](#) method is hidden. Check the syntax of your declaration to see if it is correct.

For more information, see [base](#).

The following sample generates CS0533:

C#

```
// CS0533.cs
namespace x
{
    abstract public class a
    {
        abstract public void f();
    }

    abstract public class b : a
    {
        new abstract public void f(); // CS0533
        // try the following lines instead
        // override public void f()
        // {
        // }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0534

Article • 09/15/2021 • 2 minutes to read

'function1' does not implement inherited abstract member 'function2'

A class is required to implement all the [abstract](#) members in the base class, unless the class is also abstract.

The following sample generates CS0534:

C#

```
// CS0534.cs
namespace x
{
    abstract public class clx
    {
        public abstract void f();
    }

    public class cly : clx // CS0534, no override for clx::f
    {
        // uncomment the following sample override to resolve CS0534
        // override public void f()
        // {
        // }

        public static int Main()
        {
            return 0;
        }
    }
}
```

# Compiler Error CS0535

Article • 09/15/2021 • 2 minutes to read

'class' does not implement interface member 'member'

A [class](#) derived from an [interface](#), but the class did not implement one or more of the interface's members. A class must implement all members of interfaces from which it derives or else be declared [abstract](#).

## Example 1

The following sample generates CS0535.

```
C#  
  
// CS0535.cs  
public interface A  
{  
    void F();  
}  
  
public class B : A {} // CS0535 A::F is not implemented  
  
// OK  
public class C : A {  
    public void F() {}  
    public static void Main() {}  
}
```

## Example 2

The following sample generates CS0535.

```
C#  
  
// CS0535_b.cs  
using System;  
class C : IDisposable {} // CS0535  
  
// OK  
class D : IDisposable {  
    void IDisposable.Dispose() {}  
    public void Dispose() {}  
  
    static void Main() {  
        using (D d = new D()) {}
```

```
    }  
}
```

# Compiler Error CS0537

Article • 09/15/2021 • 2 minutes to read

The class `System.Object` cannot have a base class or implement an interface

CS0537 occurs when rebuilding the [System](#) class libraries, and where [Object](#) derives from another class. You are very unlikely to encounter this error. If you do, do not derive [Object](#) from a class or interface: it is the root of the entire .NET class hierarchy, and as such, does not derive from anything else.

# Compiler Error CS0538

Article • 09/15/2021 • 2 minutes to read

'name' in explicit interface declaration is not an interface

An attempt was made to explicitly declare an [interface](#), but an interface was not specified.

The following sample generates CS0538:

C#

```
// CS0538.cs
interface MyIFace
{
    void F();
}

public class MyClass
{
    public void G()
    {
    }
}

class C: MyIFace
{
    void MyIFace.F()
    {

    }

    void MyClass.G() // CS0538, MyClass not an interface
    {
    }
}
```

# Compiler Error CS0539

Article • 09/15/2021 • 2 minutes to read

'member' in explicit interface declaration is not a member of interface

An attempt was made to explicitly declare an [interface](#) member that does not exist. You should either delete the declaration or change it so that it refers to a valid interface member.

The following sample generates CS0539:

```
C#  
  
// CS0539.cs  
namespace x  
{  
    interface I  
    {  
        void m();  
    }  
  
    public class clx : I  
    {  
        void I.x() // CS0539  
        {  
        }  
  
        public static int Main()  
        {  
            return 0;  
        }  
    }  
}
```

# Compiler Error CS0540

Article • 09/15/2021 • 2 minutes to read

'interface member' : containing type does not implement interface 'interface'

You attempted to implement an interface member in a [class](#) that does not derive from the [interface](#). You should either delete the implementation of the interface member or add the interface to the base-class list of the class.

## Example 1

The following sample generates CS0540.

```
C#  
  
// CS0540.cs  
interface I  
{  
    void m();  
}  
  
public class Clx  
{  
    void I.m() {}    // CS0540  
}  
  
// OK  
public class Cly : I  
{  
    void I.m() {}  
    public static void Main() {}  
}
```

## Example 2

The following sample generates CS0540.

```
C#  
  
// CS0540_b.cs  
using System;  
class C {  
    void IDisposable.Dispose() {}    // CS0540  
}  
  
class D : IDisposable {
```

```
void IDisposable.Dispose() {}
public void Dispose() {}

static void Main()
    using (D d = new D()) {}
}

}
```

# Compiler Error CS0541

Article • 09/15/2021 • 2 minutes to read

'declaration' : explicit interface declaration can only be declared in a class or struct

An explicit [interface](#) declaration was found outside a [class](#) or [struct](#).

The following sample generates CS0541:

C#

```
// CS0541.cs
namespace x
{
    interface IFace
    {
        void F();
    }

    interface IFace2 : IFace
    {
        void IFace.F(); // CS0541
    }
}
```

# Compiler Error CS0542

Article • 06/18/2022 • 2 minutes to read

'user-defined type' : member names cannot be the same as their enclosing type

The members of a class or struct cannot have the same name as the class or struct, unless the member is a constructor.

The following sample generates CS0542:

```
C#  
  
// CS0542.cs  
class C  
{  
    public int C;  
}
```

This error might be caused if you inadvertently put a return type on a constructor, which in effect makes it into an ordinary method. The following example generates CS0542 because `F` is a method, not a constructor, because it has a return type:

```
C#  
  
// CS0542.cs  
class F  
{  
    // Remove void from F() to resolve the problem.  
    void F()    // CS0542, same name as the class  
    {  
    }  
}  
  
class MyClass  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0543

Article • 09/15/2021 • 2 minutes to read

'enumeration' : the enumerator value is too large to fit in its type

A value that was assigned to an element in an [enumeration](#) is outside the range of the data type.

The following sample generates CS0543:

C#

```
// CS0543.cs
namespace x
{
    enum I : byte
    {a = 255, b, c}    // CS0543
    public class clk
    {
        public static int Main()
        {
            return 0;
        }
    }
}
```

# Compiler Error CS0544

Article • 09/15/2021 • 2 minutes to read

'property override': cannot override because 'non-property' is not a property

An attempt was made to override a nonproperty data type as a [property](#), which is not allowed.

The following sample generates CS0544:

C#

```
// CS0544.cs
// compile with: /target:library
public class a
{
    public int i;
}

public class b : a
{
    public override int i {    // CS0544
        // try the following line instead
        // public new int i {
        get
        {
            return 0;
        }
    }
}
```

# Compiler Error CS0545

Article • 09/15/2021 • 2 minutes to read

'function' : cannot override because 'property' does not have an overridable get accessor

A try was made to define an override for a property accessor when the base class has no such definition to override. You can resolve this error by:

- Adding a `set` accessor in the base class.
- Removing the `set` accessor from the derived class.
- Hiding the base class property by adding the `new` keyword to a property in a derived class.
- Making the base class property `virtual`.

For more information, see [Using Properties](#).

## Example

The following sample generates CS0545.

```
C#  
  
// CS0545.cs  
// compile with: /target:library  
// CS0545  
public class a  
{  
    public virtual int i  
    {  
        set {}  
  
        // Uncomment the following line to resolve.  
        // get { return 0; }  
    }  
}  
  
public class b : a  
{  
    public override int i  
    {  
        get { return 0; }  
        set {}    // OK
```

```
    }  
}
```

# Compiler Error CS0546

Article • 09/15/2021 • 2 minutes to read

'accessor' : cannot override because 'property' does not have an overridable set accessor

An attempt to override one of the accessor methods for a property failed because the accessor cannot be overridden. This error can occur if:

- the base class property is not declared as [virtual](#).
- the base class property does not declare the [get](#) or [set](#) accessor you are trying to override.

If you do not want to override the base class property, you can use the [new](#) keyword before the property in derived class.

For more information, see [Using Properties](#).

## Example

The following sample generates CS0546 because the base class does not declare a set accessor for the property.

C#

```
// CS0546.cs
// compile with: /target:library
public class a
{
    public virtual int i
    {
        get
        {
            return 0;
        }
    }

    public virtual int i2
    {
        get
        {
            return 0;
        }

        set {}
    }
}
```

```
public class b : a
{
    public override int i
    {
        set {} // CS0546 error no set
    }

    public override int i2
    {
        set {} // OK
    }
}
```

## See also

- [Properties](#)

# Compiler Error CS0547

Article • 09/15/2021 • 2 minutes to read

'property' : property or indexer cannot have void type

[void](#) is invalid as a return value for a property.

For more information, see [Properties](#).

The following sample generates CS0547:

C#

```
// CS0547.cs
public class a
{
    public void i    // CS0547
    // Try the following declaration instead:
    // public int i
    {
        get
        {
            return 0;
        }
    }
}

public class b : a
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0548

Article • 09/15/2021 • 2 minutes to read

'property' : property or indexer must have at least one accessor

A property must have at least one accessor (get or set) method.

For more information, see and [Using Properties](#).

## Example

The following sample generates CS0548.

C#

```
// CS0548.cs
// compile with: /target:library
public class b
{
    public int MyProp {}      // CS0548

    public int MyProp2     // OK
    {
        get
        {
            return 0;
        }
        set {}
    }
}
```

# Compiler Error CS0549

Article • 09/15/2021 • 2 minutes to read

'function' is a new virtual member in sealed class 'class'

A [sealedclass](#) cannot be used as a base class. Therefore, it is useless to have a virtual method in sealed class.

The following sample generates CS0549:

C#

```
// CS0549.cs
// compile with: /target:library
sealed public class MyClass
{
    virtual public void TestMethod() {}    // CS0549
    public void TestMethod2() {}    // OK
}
```

# Compiler Error CS0550

Article • 09/15/2021 • 2 minutes to read

'accessor' adds an accessor not found in interface member 'property'

The implementation of a property in a derived class contains an accessor that was not specified in the base interface.

For more information, see [Using Properties](#).

## Example

The following sample generates CS0550.

C#

```
// CS0550.cs
namespace x
{
    interface ii
    {
        int i
        {
            get;
            // add the following accessor to resolve this CS0550
            // set;
        }
    }

    public class a : ii
    {
        int ii.i
        {
            get
            {
                return 0;
            }
            set {}    // CS0550  no set in interface
        }

        public static void Main() {}
    }
}
```

# Compiler Error CS0551

Article • 09/15/2021 • 2 minutes to read

Explicit interface implementation 'implementation' is missing accessor 'accessor'

A class that explicitly implements an interface's property must implement all the accessors that the interface defines.

For more information, see [Using Properties](#).

## Example

The following sample generates CS0551.

C#

```
// CS0551.cs
// compile with: /target:library
interface ii
{
    int i
    {
        get;
        set;
    }
}

public class a : ii
{
    int ii.i { set {} } // CS0551

    // OK
    int ii.i
    {
        set {}
        get { return 0; }
    }
}
```

# Compiler Error CS0552

Article • 09/15/2021 • 2 minutes to read

'conversion routine' : user defined conversion to/from interface

You cannot create a user-defined conversion to or from an interface. If you need the conversion routine, resolve this error by making the interface a class or derive a class from the interface.

The following sample generates CS0552:

```
C#  
  
// CS0552.cs  
public interface ii  
{  
}  
  
public class a  
{  
    // delete the routine to resolve CS0552  
    public static implicit operator ii(a aa) // CS0552  
    {  
        return new ii();  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0553

Article • 09/15/2021 • 2 minutes to read

'conversion routine' : user defined conversion to/from base class

User-defined conversions to values of a base class are not allowed; you do not need such an operator.

The following sample generates CS0553:

C#

```
// CS0553.cs
namespace x
{
    public class ii
    {
    }

    public class a : ii
    {
        // delete the conversion routine to resolve CS0553
        public static implicit operator ii(a aa) // CS0553
        {
            return new ii();
        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0554

Article • 09/15/2021 • 2 minutes to read

'conversion routine' : user defined conversion to/from derived class

User-defined conversions to values of a derived class are not allowed; you do not need such an operator.

See chapter 6 in the C# language specification for more information on user-defined conversions.

The following sample generates CS0554:

C#

```
// CS0554.cs
namespace x
{
    public class ii
    {
        // delete the conversion routine to resolve CS0554
        public static implicit operator ii(a aa) // CS0554
        {
            return new ii();
        }
    }

    public class a : ii
    {
        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0555

Article • 09/15/2021 • 2 minutes to read

User-defined operator cannot take an object of the enclosing type and convert to an object of the enclosing type

User-defined conversions to values of the enclosing class are not allowed; you do not need such an operator.

The following sample generates CS0555:

C#

```
// CS0555.cs
public class MyClass
{
    // delete the following operator to resolve this CS0555
    public static implicit operator MyClass(MyClass aa)    // CS0555
    {
        return new MyClass();
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0556

Article • 09/15/2021 • 2 minutes to read

User-defined conversion must convert to or from the enclosing type

A user-defined conversion routine must convert to or from the class that contains the routine.

The following sample generates CS0556:

C#

```
// CS0556.cs
namespace x
{
    public class ii
    {
        public class iii
        {
            public static implicit operator int(byte aa) // CS0556
                // try the following line instead
                // public static implicit operator int(iii aa)
            {
                return 0;
            }
        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0557

Article • 09/15/2021 • 2 minutes to read

Duplicate user-defined conversion in type 'class'

Duplicate conversion routines are not allowed in a class.

The following example generates CS0557:

C#

```
// CS0557.cs
namespace x
{
    public class ii
    {
        public class iii
        {
            public static implicit operator int(iii aa)
            {
                return 0;
            }

            // CS0557, delete duplicate
            public static explicit operator int(iii aa)
            {
                return 0;
            }
        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0558

Article • 09/15/2021 • 2 minutes to read

User-defined operator 'operator' must be declared static and public

Both the **static** and **public** access [modifiers](#) must be specified on user-defined operators.

The following sample generates CS0558:

C#

```
// CS0558.cs
namespace x
{
    public class ii
    {
        public class iii
        {
            static implicit operator int(iii aa) // CS0558, add public
            {
                return 0;
            }
        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0559

Article • 09/15/2021 • 2 minutes to read

The parameter type for `++` or `--` operator must be the containing type

The method declaration for an operator overload must follow certain guidelines. For the `++` and `--` operators, it is required that the parameter be of the same type as the type in which the operator is being overloaded.

## Example 1

The following sample generates CS0559:

```
C#  
  
// CS0559.cs  
// compile with: /target:library  
public class iii  
{  
    public static implicit operator int(iii x)  
    {  
        return 0;  
    }  
  
    public static implicit operator iii(int x)  
    {  
        return null;  
    }  
  
    public static int operator ++(int aa)    // CS0559  
    // try the following line instead  
    // public static iii operator ++(iii aa)  
    {  
        return (iii)0;  
    }  
}
```

## Example 2

The following sample generates CS0559.

```
C#  
  
// CS0559_b.cs  
// compile with: /target:library  
public struct S
```

```
{  
    public static S operator ++(S? s) { return new S(); } // CS0559  
    public static S operator --(S? s) { return new S(); } // CS0559  
}  
  
public struct T  
{  
// OK  
    public static T operator --(T t) { return new T(); }  
    public static T operator ++(T t) { return new T(); }  
  
    public static T? operator --(T? t) { return new T(); }  
    public static T? operator +(T? t) { return new T(); }  
}
```

# Compiler Error CS0562

Article • 09/15/2021 • 2 minutes to read

The parameter of a unary operator must be the containing type

The method declaration for an operator overload must follow certain guidelines. For more information, see [Operator overloading](#).

The following sample generates CS0562:

C#

```
// CS0562.cs
public class iii
{
    public static implicit operator int(iii x)
    {
        return 0;
    }

    public static implicit operator iii(int x)
    {
        return null;
    }

    public static iii operator +(int aa)    // CS0562
    // try the following line instead
    // public static iii operator +(iii aa)
    {
        return (iii)0;
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0563

Article • 09/15/2021 • 2 minutes to read

One of the parameters of a binary operator must be the containing type

The method declaration for an [operator overload](#) must follow certain guidelines.

## Example

The following sample generates CS0563:

C#

```
// CS0563.cs
public class iii
{
    public static implicit operator int(iii x)
    {
        return 0;
    }
    public static implicit operator iii(int x)
    {
        return null;
    }
    public static int operator +(int aa, int bb) // CS0563
    // Use the following line instead:
    // public static int operator +(int aa, iii bb)
    {
        return 0;
    }
    public static void Main()
    {
    }
}
```

# Compiler Error CS0564

Article • 09/15/2021 • 2 minutes to read

The first operand of an overloaded shift operator must have the same type as the containing type, and the type of the second operand must be int

You attempted to overload a shift operator (<< or >>) with incorrectly typed operands. The first operand must be the type and the second operand must be of the type `int`.

The following sample generates CS0564:

C#

```
// CS0564.cs
using System;
class C
{
    public static int operator << (C c1, C c2) // CS0564
// To correct, change second operand to int, like so:
// public static int operator << (C c1, int c2)
    {
        return 0;
    }
    static void Main()
    {
    }
}
```

# Compiler Error CS0567

Article • 09/15/2021 • 2 minutes to read

Interfaces cannot contain operators

Operators are not permitted in [interface](#) definitions.

The following sample generates CS0567:

C#

```
// CS0567.cs
interface IA
{
    int operator +(int aa, int bb);    // CS0567
}

class Sample
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0568

Article • 05/21/2022 • 2 minutes to read

Structs cannot contain explicit parameterless constructors

Each `struct` already has a parameterless constructor that initializes the object to zero. Therefore, the constructors that you can create for a struct must take one or more parameters.

The following sample generates CS0568:

C#

```
// CS0568.cs
public struct ClassY
{
    public int field1;
    public ClassY(){} // CS0568, cannot have no param constructor
    // Try following instead:
    // public ClassY(int i)
    // {
    //     field1 = i;
    // }
}

public class ClassX
{
    public static void Main()
    {
    }
}
```

## ⓘ Note

Beginning with C# 10, a structure type can contain an explicit parameterless constructor. For more information, see the [Struct initialization and default values](#) section of the [Structure types](#) article.

# Compiler Error CS0569

Article • 09/15/2021 • 2 minutes to read

'method2' : cannot override 'method1' because it is not supported by the language

This error occurs when you derive from a base class that was written in another language and when the compiler does not support the method that you are attempting to override.

# Compiler Error CS0570

Article • 09/15/2021 • 2 minutes to read

Property, indexer, or event 'name' is not supported by the language; try directly calling accessor method 'name!'

This error occurs when using imported metadata that was generated by another compiler. Your code attempted to use a class member that the compiler cannot process.

## Example 1

The following C++ program uses an attribute, RequiredAttributeAttribute, which may not be consumed by other languages.

C++

```
// CPP0570.cpp
// compile with: /clr /LD

using namespace System;
using namespace System::Runtime::CompilerServices;

namespace CS0570_Server {
    [RequiredAttributeAttribute(Int32::typeid)]
    public ref struct Scenario1 {
        int intVar;
    };

    public ref struct CS0570Class {
        Scenario1 ^ sc1_field;

        property virtual Scenario1 ^ sc1_prop {
            Scenario1 ^ get() { return sc1_field; }
        }

        Scenario1 ^ sc1_method() { return sc1_field; }
    };
}
```

## Example 2

The following sample generates CS0570.

C#

```
// CS0570.cs
// compile with: /reference:CPP0570.dll
using System;
using CS0570_Server;

public class C {
    public static int Main() {
        CS0570Class r = new CS0570Class();
        r.sc1_field = null;    // CS0570
        object o = r.sc1_prop; // CS0570
        r.sc1_method();       // CS0570
    }
}
```

# Compiler Error CS0571

Article • 09/15/2021 • 2 minutes to read

'function' : cannot explicitly call operator or accessor

Certain operators have internal names. For example, `op_Increment` is the internal name of the `++` operator. You should not use or explicitly call such method names.

The following sample generates CS0571:

C#

```
// CS0571.cs
public class MyClass
{
    public static MyClass operator ++ (MyClass c)
    {
        return null;
    }

    public static int prop
    {
        get
        {
            return 1;
        }
        set
        {
        }
    }

    public static void Main()
    {
        op_Increment(null);    // CS0571
        // use the increment operator as follows
        // MyClass x = new MyClass();
        // x++;

        set_prop(1);          // CS0571
        // try the following line instead
        // prop = 1;
    }
}
```

# Compiler Error CS0572

Article • 09/15/2021 • 2 minutes to read

'type' : cannot reference a type through an expression; try 'path\_to\_type' instead

An attempt was made to access a member of a class through an identifier, which is not permitted in this situation.

The following sample generates CS0572:

C#

```
// CS0572.cs
using System;
class C
{
    public class Inner
    {
        public static int v = 9;
    }
}

class D : C
{
    public static void Main()
    {
        C cValue = new C();
        Console.WriteLine(cValue.Inner.v);    // CS0572
        // try the following line instead
        // Console.WriteLine(C.Inner.v);
    }
}
```

# Compiler Error CS0573

Article • 05/21/2022 • 2 minutes to read

'field declaration' : cannot have instance field initializers in structs

You can't initialize an instance field of a [struct](#). Fields of value types will be initialized to their default values, and reference-type fields will be initialized to `null`.

## ⓘ Note

Beginning with C# 10, you can initialize a struct's instance field or property at its declaration. For more information, see the [Struct initialization and default values](#) section of the [Structure types](#) article.

## Example

The following sample generates CS0573:

```
C#  
  
// CS0573.cs  
namespace x  
{  
    public class clx  
    {  
        public static void Main()  
        {  
        }  
    }  
  
    public struct cly  
    {  
        clx a = new clx(); // CS0573  
        // clx a; // OK  
        int i = 7; // CS0573  
        // int i; // OK  
    }  
}
```

# Compiler Error CS0574

Article • 09/16/2021 • 2 minutes to read

Name of destructor must match name of class

The name of a finalizer must be the class name preceded by a tilde (~).

The following sample generates CS0574:

C#

```
// CS0574.cs
namespace x
{
    public class iii
    {
        ~iiii()    // CS0574
        {

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0575

Article • 09/15/2021 • 2 minutes to read

Only class types can contain destructors

A [struct](#) cannot contain a finalizer.

The following sample generates CS0575:

C#

```
// CS0575.cs
namespace x
{
    public struct iii
    {
        ~iii()    // CS0575
        {

        }

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0576

Article • 09/15/2021 • 2 minutes to read

Namespace 'namespace' contains a definition conflicting with alias 'identifier'

An attempt was made to use the same namespace twice.

## Example

The following sample generates CS0576:

```
C#  
  
// CS0576.cs  
using SysIO = System.IO;  
public class SysIO  
{  
    public void MyMethod() {}  
}  
  
public class Test  
{  
    public static void Main()  
    {  
        SysIO.Stream s; // CS0576  
    }  
}
```

# Compiler Error CS0577

Article • 09/15/2021 • 2 minutes to read

The `Conditional` attribute is not valid on 'function' because it is a constructor, destructor, operator, or explicit interface implementation

`Conditional` cannot be applied to the specified methods.

For example, you cannot use some attributes on an explicit interface definition. The following sample generates CS0577:

C#

```
// CS0577.cs
// compile with: /target:library
interface I
{
    void m();
}

public class MyClass : I
{
    [System.Diagnostics.Conditional("a")] // CS0577
    void I.m() {}
}
```

# Compiler Error CS0578

Article • 09/15/2021 • 2 minutes to read

The Conditional attribute is not valid on 'function' because its return type is not void

[ConditionalAttribute](#) cannot be applied to a method that has a return type other than `void`. The reason for this is that any other return type for a method may be needed by another part of your program.

## Example

The following sample generates CS0578. To resolve this error, you must either delete [ConditionalAttribute](#), or you must change the return value of the method to `void`.

C#

```
// CS0578.cs
// compile with: /target:library
public class MyClass
{
    [System.Diagnostics.ConditionalAttribute("a")] // CS0578
    public int TestMethod()
    {
        return 0;
    }
}
```

# Compiler Error CS0579

Article • 09/15/2021 • 2 minutes to read

## Duplicate 'attribute' attribute

It is not possible to specify the same attribute more than once unless the attribute specifies `AllowMultiple=true` in its [AttributeUsage](#).

## Example

The following example generates CS0579.

C#

```
// CS0579.cs
using System;
public class MyAttribute : Attribute
{
}

[AttributeUsage(AttributeTargets.All,AllowMultiple=true)]
public class MyAttribute2 : Attribute
{
}

public class z
{
    [MyAttribute, MyAttribute]      // CS0579
    public void zz()
    {
    }

    [MyAttribute2, MyAttribute2]    // OK
    public void zzz()
    {
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0582

Article • 09/15/2021 • 2 minutes to read

The Conditional not valid on interface members

**ConditionalAttribute** is not valid on an interface member.

The following sample generates CS0582:

C#

```
// CS0582.cs
// compile with: /target:library
using System.Diagnostics;
interface MyIFace
{
    [ConditionalAttribute("DEBUG")]
    void zz();
}
```

# Compiler Error CS0583

Article • 09/15/2021 • 2 minutes to read

Internal Compiler Error. An internal error has occurred in the compiler. To work around this problem, try simplifying or changing the program near the locations listed below. Locations at the top of the list are closer to the point at which the internal error occurred.

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS0584

Article • 09/15/2021 • 2 minutes to read

Internal Compiler Error: stage 'stage' symbol 'symbol'

Try to determine whether the compiler is failing because of its inability to parse unexpected syntax. If that is not the case, see [Visual Studio feedback options](#).

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS0585

Article • 09/15/2021 • 2 minutes to read

Internal Compiler Error: stage 'stage'

Try to determine if the compiler is failing due to its inability to parse unexpected syntax. If that is not the case, see [Visual Studio feedback options](#).

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS0586

Article • 09/15/2021 • 2 minutes to read

Internal Compiler Error: stage 'stage'

Try to determine if the compiler is failing due to its inability to parse unexpected syntax. If that is not the case, see [Visual Studio feedback options](#).

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS0587

Article • 09/15/2021 • 2 minutes to read

Internal Compiler Error: stage 'stage'

Try to determine if the compiler is failing due to its inability to parse unexpected syntax.  
If this is not the case, see [Visual Studio feedback options](#).

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS0588

Article • 09/15/2021 • 2 minutes to read

Internal Compiler Error: stage 'LEX'

Try to determine whether the compiler is failing because of its inability to parse unexpected syntax. If this is not the case, see [Visual Studio feedback options](#).

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS0589

Article • 09/15/2021 • 2 minutes to read

Internal Compiler Error: stage 'PARSE'

Try to determine if the compiler is failing due to its inability to parse unexpected syntax. If that is not the case, see [Visual Studio feedback options](#).

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS0590

Article • 09/15/2021 • 2 minutes to read

User-defined operators cannot return void

The purpose of a user-defined operator is to return an object.

The following sample generates CS0590:

C#

```
// CS0590.cs
namespace x
{
    public class a
    {
        public static void operator+(a A1, a A2)    // CS0590
        {
        }

        // try the following user-defined operator
        /*
        public static a operator+(a A1, a A2)
        {
            return A2;
        }
        */
    }

    public static int Main()
    {
        return 1;
    }
}
```

# Compiler Error CS0591

Article • 09/15/2021 • 2 minutes to read

Invalid value for argument to 'attribute' attribute

An attribute was passed either an invalid argument or two mutually exclusive arguments.

## Example

The following sample generates CS0591:

C#

```
// CS0591.cs
using System;

[AttributeUsage(0)] // CS0591
class I: Attribute
{
}

public class a
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0592

Article • 03/15/2023 • 2 minutes to read

Attribute 'attribute' is not valid on this declaration type. It is valid on 'type' declarations only.

When you define an attribute, you define what constructs it can be applied to by specifying an `AttributeTargets` value. In the following example, the `MyAttribute` attribute can be applied to interfaces only, because the `AttributeUsage` attribute specifies `AttributeTargets.Interface`. The error is generated because the attribute is applied to a class (class `A`).

## Example

The following sample generates CS0592:

C#

```
// CS0592.cs
using System;

[AttributeUsage(AttributeTargets.Interface)]
public class MyAttribute : Attribute
{
}

[MyAttribute]
// Generates CS0592 because MyAttribute is not valid for a class.
public class A
{
    public static void Main()
    {
    }
}
```

## See also

- [Attributes](#)

# Compiler Error CS0594

Article • 09/15/2021 • 2 minutes to read

Floating-point constant is outside the range of type 'type'

A value was assigned to a floating-point variable that is too large for the variables of this data type. See [Integral Types Table](#) for information about the range of values allowed in data types.

The following sample generates CS0594:

C#

```
// CS0594.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            float f = 6.77777777777E400;    // CS0594, value too large
        }
    }
}
```

# Compiler Error CS0596

Article • 09/15/2021 • 2 minutes to read

The Guid attribute must be specified with the ComImport attribute

The **Guid** attribute must be present when using the **ComImport** attribute.

The following sample generates CS0596:

C#

```
// CS0596.cs
using System.Runtime.InteropServices;

namespace x
{
    [ComImport] // CS0596
    // try the following line to resolve this CS0596
    // [ComImport, Guid("00000000-0000-0000-0000-000000000001")]
    public class a
    {

    }

    public class b
    {
        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0599

Article • 09/15/2021 • 2 minutes to read

Invalid value for named attribute argument 'argument'

An invalid argument was passed to an attribute.

# Compiler Error CS0601

Article • 09/15/2021 • 2 minutes to read

The `DllImport` attribute must be specified on a method marked 'static' and 'extern'

The `DllImport` attribute was used on a method that did not have the correct access keywords.

The following sample generates CS0601:

C#

```
// CS0601.cs
using System.Runtime.InteropServices;
using System.Text;

public class C
{
    [DllImport("KERNEL32.DLL")]
    extern int GetCurDirectory(int bufSize, StringBuilder buf); // CS0601
    // Try the following line instead:
    // static extern int GetCurDirectory(int bufSize, StringBuilder buf);
}

public class MainClass
{
    public static void Main ()
    {
    }
}
```

# Compiler Error CS0609

Article • 09/15/2021 • 2 minutes to read

Cannot set the `IndexerName` attribute on an indexer marked override

The `name` attribute (`IndexerNameAttribute`) cannot be applied to an indexed property that is an override. For more information, see [Indexers](#).

The following sample generates CS0609:

C#

```
// CS0609.cs
using System;
using System.Runtime.CompilerServices;

public class idx
{
    public virtual int this[int iPropIndex]
    {
        get
        {
            return 0;
        }
        set
        {
        }
    }
}

public class MonthDays : idx
{
    [IndexerName("MonthInfoIndexer")] // CS0609, delete to resolve this
CS0609
    public override int this[int iPropIndex]
    {
        get
        {
            return 0;
        }
        set
        {
        }
    }
}

public class test
{
    public static void Main(string[] args)
    {
```

```
    }  
}
```

### ⓘ Note

This compiler error is no longer used in Roslyn. The previous code should compile successfully.

# Compiler Error CS0610

Article • 09/15/2021 • 2 minutes to read

Field or property cannot be of type 'type'

There are some types that cannot be used as fields or properties. These types include **System.ArgIterator** and **System.TypedReference**.

The following sample generates CS0610 as a result of using **System.TypedReference** as a field:

C#

```
// CS0610.cs
public class MainClass
{
    System.TypedReference i;    // CS0610
    public static void Main ()
    {
    }

    public static void Test(System.TypedReference i)    // OK
    {
    }
}
```

# Compiler Error CS0611

Article • 09/15/2021 • 2 minutes to read

Array elements cannot be of type 'type'

There are some types that cannot be used as the type of an array. These types include **System.TypedReference** and **System.ArgIterator**.

The following sample generates CS0611 as a result of using **System.TypedReference** as an array element:

C#

```
// CS0611.cs
public class a
{
    public static void Main()
    {
        System.TypedReference[] ao = new System.TypedReference [10];    // CS0611
        // try the following line instead
        // int[] ao = new int[10];
    }
}
```

# Compiler Error CS0616

Article • 09/15/2021 • 2 minutes to read

'class' is not an attribute class

An attempt was made to use a non-attribute class in an attribute block. All the attribute types need to be inherited from [System.Attribute](#).

## Example 1

The following sample generates CS0616.

```
C#  
  
// CS0616.cs  
// compile with: /target:library  
[CMyClass(i = 5)] // CS0616  
public class CMyClass {}
```

## Example 2

The following sample shows how you might define an attribute:

```
C#  
  
// CreateAttrib.cs  
// compile with: /target:library  
using System;  
  
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Interface)]  
public class MyAttr : Attribute  
{  
    public int Name = 0;  
    public int Count = 0;  
  
    public MyAttr (int iCount, int sName)  
    {  
        Count = iCount;  
        Name = sName;  
    }  
}  
  
[MyAttr(5, 50)]  
class Class1 {}
```

```
[MyAttr(6, 60)]  
interface Interface1 {}
```

# Compiler Error CS0617

Article • 09/15/2021 • 2 minutes to read

'reference' is not a valid named attribute argument because it is not a valid attribute parameter type

An attempt was made to access a [private](#) member of an attribute class.

## Example

The following sample generates CS0617.

C#

```
// CS0617.cs
using System;

[AttributeUsage(AttributeTargets.Struct | 
                AttributeTargets.Class | 
                AttributeTargets.Interface)]
public class MyClass : Attribute
{
    public int Name;

    public MyClass (int sName)
    {
        Name = sName;
        Bad = -1;
        Bad2 = -1;
    }

    public readonly int Bad;
    public int Bad2;
}

[MyClass(5, Bad=0)] class Class1 {} // CS0617
[MyClass(5, Bad2=0)] class Class2 {}
```

# Compiler Error CS0619

Article • 09/15/2021 • 2 minutes to read

'member' is obsolete: 'text'

A class member was marked with the [Obsolete](#) attribute, such that an error will be issued when the class member is referenced.

## Example

The following sample generates CS0619:

C#

```
using System;

public class C
{
    [Obsolete("Use NewMethod instead", true)] // generates an error on use
    public static void OldMethod()
    {
    }

    // this is the method you should be using
    public static void NewMethod()
    {
    }
}

class MyClass
{
    public static void Main()
    {
        C.OldMethod();    // CS0619
    }
}
```

# Compiler Error CS0620

Article • 09/15/2021 • 2 minutes to read

Indexers cannot have void type

The return type of an [indexer](#) cannot be `void`. An indexer must return a value.

The following sample generates CS0620:

C#

```
// CS0620.cs
class MyClass
{
    public static void Main()
    {
        MyClass test = new MyClass();
        System.Console.WriteLine(test[2]);
    }

    void this [int intI]    // CS0620, return type cannot be void
    {
        get
        {
            // will need to return some value
        }
    }
}
```

## See also

- [void](#)



# Compiler Error CS0621

Article • 09/15/2021 • 2 minutes to read

'member' : virtual or abstract members cannot be private

Private **virtual** or **abstract** members are not allowed.

The following sample generates CS0621:

C#

```
// CS0621.cs
abstract class MyClass
{
    private virtual void DoNothing1()    // CS0621
    {
    }

    private abstract void DoNothing2();   // CS0621

    public static void Main()
    {
    }
}
```

# Compiler Error CS0622

Article • 09/15/2021 • 2 minutes to read

Can only use array initializer expressions to assign to array types. Try using a new expression instead.

Syntax that is appropriate to initialize an array was used in the declaration of a non-array.

## Example

The following sample generates CS0622:

```
C#  
  
// CS0622.cs  
using System;  
  
public class Test  
{  
    public static void Main ()  
    {  
        Test t = { new Test() }; // CS0622  
        // Try the following instead:  
        // Test[] t = { new Test() };  
    }  
}
```

# Compiler Error CS0623

Article • 09/15/2021 • 2 minutes to read

Array initializers can only be used in a variable or field initializer. Try using a new expression instead.

An attempt was made to initialize an array by using an array initializer in a context where it is not allowed.

## Example

The following example produces CS0623 because the compiler interprets the {4} as embedded array initializer inside the outer array initializer:

```
C#  
  
//cs0623.cs  
using System;  
  
class X  
{  
    public int[] x = { 2, 3, {4}}; //CS0623  
}
```

## See also

- [Arrays](#)



# Compiler Error CS0625

Article • 09/15/2021 • 2 minutes to read

'field': instance field types marked with StructLayout(LayoutKind.Explicit) must have a FieldOffset attribute.

When a struct is marked with an explicit **StructLayout** attribute, all fields in the struct must have the **FieldOffset** attribute. For more information, see [StructLayoutAttribute Class](#).

The following sample generates CS0625:

C#

```
// CS0625.cs
// compile with: /target:library
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Explicit)]
struct A
{
    public int i;    // CS0625 not static; an instance field
}

// OK
[StructLayout(LayoutKind.Explicit)]
struct B
{
    [FieldOffset(5)]
    public int i;
}
```

# Compiler error CS0629

Article • 09/15/2021 • 2 minutes to read

Conditional member 'member' cannot implement interface member 'base class member' in type 'Type Name'

The [Conditional attribute](#) cannot be used on the implementation of an interface.

The following sample generates CS0629:

C#

```
// CS0629.cs
interface MyInterface
{
    void MyMethod();
}

public class MyClass : MyInterface
{
    [System.Diagnostics.Conditional("debug")]
    public void MyMethod()      // CS0629, remove the Conditional attribute
    {
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0631

Article • 09/15/2021 • 2 minutes to read

ref and out are not valid in this context

The declaration for an [indexer](#) cannot include the use of [ref](#) or [out](#) parameters.

## Example

The following sample generates CS0631:

```
C#  
  
// CS0631.cs  
public class MyClass  
{  
    public int this[ref int i]    // CS0631  
    // try the following line instead  
    // public int this[int i]  
    {  
        get  
        {  
            return 0;  
        }  
    }  
}  
  
public class MyClass2  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0633

Article • 09/15/2021 • 2 minutes to read

The argument to the 'attribute' attribute must be a valid identifier

Any argument that you pass to the [ConditionalAttribute](#) or [IndexerNameAttribute](#) attributes must be a valid identifier. This means that it may not contain characters such as "+" that are illegal when they occur in identifiers.

## Example 1

This example illustrates CS0633 using the [ConditionalAttribute](#). The following sample generates CS0633.

```
C#  
  
// CS0633a.cs  
#define DEBUG  
using System.Diagnostics;  
public class Test  
{  
    [Conditional("DEB+UG")] // CS0633  
    // try the following line instead  
    // [Conditional("DEBUG")]  
    public static void Main() { }  
}
```

## Example 2

This example illustrates CS0633 using the [IndexerNameAttribute](#).

```
C#  
  
// CS0633b.cs  
// compile with: /target:module  
#define DEBUG  
using System.Runtime.CompilerServices;  
public class Test  
{  
    [IndexerName("Invalid+Identifier")] // CS0633  
    // try the following line instead  
    // [IndexerName("DEBUG")]  
    public int this[int i]  
    {  
        get { return i; }  
    }
```

```
    }  
}
```

# Compiler Error CS0635

Article • 09/15/2021 • 2 minutes to read

'attribute' : System.Interop.UnmanagedType.CustomMarshaller requires named arguments ComType and Marshal

The **ComType** and **Marshal** arguments must be specified when the marshal format is **System.Interop.UnmanagedType.CustomMarshaller**.

# Compiler Error CS0636

Article • 09/15/2021 • 2 minutes to read

The **FieldOffset** attribute can only be placed on members of types marked with the **StructLayout(LayoutKind.Explicit)**

You must use the **StructLayout(LayoutKind.Explicit)** attribute on the struct declaration, if it contains any members marked with the **FieldOffset** attribute. For more information, see [FieldOffset](#).

The following sample generates CS0636:

C#

```
// CS0636.cs
using System;
using System.Runtime.InteropServices;

// To resolve the error, uncomment the following line:
// [StructLayout(LayoutKind.Explicit)]
struct Worksheet
{
    [FieldOffset(4)]public int i;    // CS0636
}

public class MainClass
{
    public static void Main ()
    {
    }
}
```

# Compiler Error CS0637

Article • 09/15/2021 • 2 minutes to read

The `FieldOffset` attribute is not allowed on static or `const` fields.

The `FieldOffset` attribute cannot be used on fields marked with `static` or `const`.

The following sample generates CS0637:

C#

```
// CS0637.cs
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Explicit)]
public class MainClass
{
    [FieldOffset(3)] // CS0637
    public static int i;
    public static void Main ()
    {
    }
}
```

# Compiler Error CS0641

Article • 09/15/2021 • 2 minutes to read

'attribute' : attribute is only valid on classes derived from System.Attribute

An attribute was used that can only be used on a class that derives from **System.Attribute**.

The following sample generates CS0641:

C#

```
// CS0641.cs
using System;

[AttributeUsage(AttributeTargets.All)]
public class NonAttrClass // CS0641
// try the following line instead
// public class NonAttrClass : Attribute
{
}

class MyClass
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0643

Article • 03/15/2023 • 2 minutes to read

'arg' duplicate named attribute argument

A parameter, `arg`, on a user-defined attribute was specified twice. For more information, see [Attributes](#).

## Example

The following sample generates CS0643:

C#

```
// CS0643.cs
using System;
using System.Runtime.InteropServices;

[AttributeUsage(AttributeTargets.Class)]
public class MyAttribute : Attribute
{
    public MyAttribute()
    {
    }

    public int x;
}

[MyAttribute(x = 5, x = 6)] // CS0643, error setting x twice
// try the following line instead
// [MyAttribute(x = 5)]
class MyClass
{

}

public class MainClass
{
    public static void Main ()
    {
    }
}
```

# Compiler Error CS0644

Article • 09/15/2021 • 2 minutes to read

'class1' cannot derive from special class 'class2'

Classes cannot explicitly inherit from any of the following base classes:

- **System.Enum**
- **System.ValueType**
- **System.Delegate**
- **System.Array**

These are used as implicit base classes by the compiler. For example, **System.ValueType** is the implicit base class of structs.

The following sample generates CS0644:

C#

```
// CS0644.cs
class MyClass : System.ValueType // CS0644
{}
```

# Compiler Error CS0645

Article • 09/15/2021 • 2 minutes to read

## Identifier too long

A class name or other identifier can be no longer than 512 characters.

# Compiler Error CS0646

Article • 09/15/2021 • 2 minutes to read

Cannot specify the DefaultMember attribute on a type containing an indexer

If a class or other type specifies `System.Reflection.DefaultMemberAttribute`, it cannot contain an indexer. For more information, see [Properties](#).

The following sample generates CS0646:

```
C#  
  
// CS0646.cs  
// compile with: /target:library  
[System.Reflection.DefaultMemberAttribute("x")] // CS0646  
class MyClass  
{  
    public int this[int index] // an indexer  
    {  
        get  
        {  
            return 0;  
        }  
    }  
  
    public int x = 0;  
}  
  
// OK  
[System.Reflection.DefaultMemberAttribute("x")]  
class MyClass2  
{  
    public int prop  
    {  
        get  
        {  
            return 0;  
        }  
    }  
  
    public int x = 0;  
}  
  
class MyClass3  
{  
    public int this[int index] // an indexer  
    {  
        get  
        {  
            return 0;  
        }  
    }  
}
```

```
}

public int x = 0;
}
```

# Compiler Error CS0647

Article • 09/15/2021 • 2 minutes to read

Error emitting 'attribute' attribute -- 'reason'

The following sample generates CS0647:

C#

```
// CS0647.cs
using System.Runtime.InteropServices;

[Guid("z")]    // CS0647, incorrect uuid format.
// try the following line instead
// [Guid("00000000-0000-0000-0000-000000000001")]
public class MyClass
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0648

Article • 09/15/2021 • 2 minutes to read

'type' is a type not supported by the language

Metadata that was generated from another language, possibly C++, contained a type that was not marked as managed. Information about this type is included in the metadata, but the type is not available to programs written in C#.

# Compiler Error CS0650

Article • 09/10/2022 • 2 minutes to read

Bad array declarator: To declare a managed array the rank specifier precedes the variable's identifier. To declare a fixed size buffer field, use the `fixed` keyword before the field type.

An array was declared incorrectly. In C#, unlike in C and C++, the square brackets follow the type, not the variable name. Also, realize that the syntax for a fixed size buffer differs from that of an array.

## Example

The following example code generates CS0650.

```
C#  
  
// CS0650.cs  
public class MyClass  
{  
    public static void Main()  
    {  
        // Generates CS0650. Incorrect array declaration syntax:  
        int myarray[2];  
  
        // Correct declaration.  
        int[] myarray2;  
  
        // Declaration and initialization in one statement  
        int[] myArray3= new int[2] {1,2};  
  
        // Access an array element.  
        myarray3[0] = 0;  
    }  
}
```

The following example shows how to use the `fixed` keyword when you create a fixed size buffer:

```
C#  
  
// This code must appear in an unsafe block.  
public struct MyArray  
{  
    public fixed char pathName[128];  
}
```

---

## See also

- [Fixed Size Buffers](#)
- [Arrays](#)

# Compiler Error CS0653

Article • 09/15/2021 • 2 minutes to read

Cannot apply attribute class 'class' because it is abstract

An [abstract](#) custom attribute class cannot be used as an attribute.

The following sample generates CS0653:

C#

```
// CS0653.cs
using System;

public abstract class MyAttribute : Attribute
{
}

public class My2Attribute : MyAttribute
{
}

[My] // CS0653
// try the following line instead
// [My2]
class MyClass
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0655

Article • 03/15/2023 • 2 minutes to read

'parameter' is not a valid named attribute argument because it is not a valid attribute parameter type

See [Attributes](#) for a discussion of valid parameter types for an attribute.

## Example

The following sample generates CS0655:

```
C#  
  
// CS0655.cs  
using System;  
  
class MyAttribute : Attribute  
{  
    // decimal is not valid attribute parameter type  
    public decimal d = 0;  
    public int e = 0;  
}  
  
[My(d = 0)] // CS0655  
// Try the following line instead:  
// [My(e = 0)]  
class C  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0656

Article • 09/15/2021 • 2 minutes to read

Missing compiler required member 'object.member'

One of the following problems exists:

- Your installation of the common language runtime is corrupt.
- You have a reference to an assembly that defines a type that is also found in the common language runtime. However, your assembly's type is not defined the way the C# compiler expects.

Check your references to ensure that you are using the correct version of the common language runtime.

# Compiler Error CS0662

Article • 09/15/2021 • 2 minutes to read

'method' cannot specify only Out attribute on a ref parameter. Use both In and Out attributes, or neither.

An interface method has a parameter that uses `ref` with just the `Out` attribute. A `ref` parameter that uses the `Out` attribute must also use the `In` attribute.

The following sample generates CS0662:

C#

```
// CS0662.cs
using System.Runtime.InteropServices;

interface I
{
    void method([Out] ref int i);    // CS0662
    // try one of the following lines instead
    // void method(ref int i);
    // void method([Out, In]ref int i);
}

class test
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0663

Article • 09/15/2021 • 2 minutes to read

Cannot define overloaded methods that differ only on ref and out.

Methods that differ only on their use of [in](#), [ref](#) and [out](#) on a parameter are not allowed.

The following sample generates CS0663:

C#

```
// CS0663.cs
class TestClass
{
    public static void Main()
    {

        public void Test(ref int i)
        {

        }

        public void Test(out int i)    // CS0663
        {
        }
    }
}
```

# Compiler Error CS0664

Article • 09/15/2021 • 2 minutes to read

Literal of type double cannot be implicitly converted to type 'type'; use an 'suffix' suffix to create a literal of this type

An assignment could not be completed; use a suffix to correct the instruction. The documentation for each type identifies the corresponding suffix for the type. For more information on conversions, see [Casting and Type Conversions](#).

The following sample generates CS0664:

```
C#  
  
// CS0664.cs  
class Example  
{  
    static void Main()  
    {  
        decimal d1 = 1.0;    // CS0664, because 1.0 is interpreted  
                            // as a double.  
  
        // Try the following line instead.  
        decimal d2 = 1.0M;   // The M tells the compiler that 1.0 is a  
                            // decimal.  
        Console.WriteLine(d2);  
    }  
}
```

## See also

- [Type Conversion Tables](#)



# Compiler Error CS0666

Article • 09/15/2021 • 2 minutes to read

'member' : new protected member declared in struct

A [struct](#) cannot be [abstract](#) and is always implicitly [sealed](#). Because structs do not support inheritance, the concept of a [protected](#) member in a struct makes no sense. For more information, see [Inheritance](#).

## Example

The following sample generates CS0666:

C#

```
// CS0666.cs
class M
{
    static void Main()
    {
    }
}

struct S
{
    protected int x;    // CS0666
}
```

# Compiler Error CS0667

Article • 09/15/2021 • 2 minutes to read

The feature 'invalid feature' is deprecated. Please use 'valid feature' instead.

The feature you are attempting to use is now deprecated. Update your code to use the valid feature instead.

# Compiler Error CS0668

Article • 09/15/2021 • 2 minutes to read

Two indexers have different names; the **IndexerName** attribute must be used with the same name on every indexer within a type

The values passed to the **IndexerName** attribute must be the same for all indexers in a type. For more information on the **IndexerName** attribute, see [IndexerNameAttribute Class](#).

The following sample generates CS0668:

C#

```
// CS0668.cs
using System;
using System.Runtime.CompilerServices;

class IndexerClass
{
    [IndexerName("IName1")]
    public int this [int index]    // indexer declaration
    {
        get
        {
            return index;
        }
        set
        {
        }
    }

    [IndexerName("IName2")]
    public int this [string s]    // CS0668, change IName2 to IName1
    {
        get
        {
            return int.Parse(s);
        }
        set
        {
        }
    }

    void Main()
    {
    }
}
```

# Compiler Error CS0669

Article • 09/15/2021 • 2 minutes to read

A class with the `ComImport` attribute cannot have a user-defined constructor.

The COM interop layer in the common language runtime supplies the constructor for `ComImport` classes. Consequently, a COM object can be used as a managed object in the runtime.

The following sample generates CS0669:

```
C#  
  
// CS0669.cs  
using System.Runtime.InteropServices;  
[ComImport, Guid("00000000-0000-0000-0000-000000000001")]  
class TestClass  
{  
    TestClass() // CS0669, delete constructor to resolve  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0670

Article • 09/15/2021 • 2 minutes to read

Field cannot have void type

A field was declared to be of type [void](#).

The following sample generates CS0670:

C#

```
// CS0670.cs
class C
{
    void f;    // CS0670
    // try the following line instead
    // public int f;

    public static void Main()
    {
        C myc = new C();
        myc.f = 0;
    }
}
```

# Compiler Error CS0673

Article • 09/15/2021 • 2 minutes to read

System.Void cannot be used from C# -- use typeof(void) to get the void type object.

**System.Void** cannot be used in C#.

The following sample generates CS0673:

C#

```
// CS0673.cs
class MyClass
{
    public static void Main()
    {
        System.Type t = typeof(System.Void);    // CS0673
        // try the following line instead
        // System.Type t = typeof(void);
    }
}
```

# Compiler Error CS0674

Article • 09/15/2021 • 2 minutes to read

Do not use 'System.ParamArrayAttribute'. Use the 'params' keyword instead.

The C# compiler does not allow for the use of [System.ParamArrayAttribute](#); use [params](#) instead.

The following sample generates CS0674:

C#

```
// CS0674.cs
using System;
public class MyClass
{
    public static void UseParams([ParamArray] int[] list) // CS0674
        // try the following line instead
        // public static void UseParams(params int[] list)
    {
        for ( int i = 0 ; i < list.Length ; i++ )
            Console.WriteLine(list[i]);
        Console.WriteLine();
    }

    public static void Main()
    {
        UseParams(1, 2, 3);
    }
}
```

# Compiler Error CS0677

Article • 09/15/2021 • 2 minutes to read

'variable': a volatile field cannot be of the type 'type'

Fields declared with the `volatile` keyword must be one of the following types:

- Any reference type
- Any pointer type (in an `unsafe` context)
- The types `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `char`, `float`, `bool`
- Enum types based on any of the above types

The following sample generates CS0677:

C#

```
// CS0677.cs
class TestClass
{
    private volatile long i;    // CS0677

    public static void Main()
    {
    }
}
```

# Compiler Error CS0678

Article • 09/15/2021 • 2 minutes to read

'variable': a field can not be both volatile and readonly

Use of the [volatile](#) and [readonly](#) keywords is mutually exclusive.

The following sample generates CS0678:

C#

```
// CS0678.cs
using System;

class TestClass
{
    private readonly volatile int i;    // CS0678
    // try the following line instead
    // private volatile int i;

    public static void Main()
    {
    }
}
```

# Compiler Error CS0681

Article • 09/15/2021 • 2 minutes to read

The modifier 'abstract' is not valid on fields. Try using a property instead

You cannot make a field abstract. You can, however, have an abstract property that accesses the field.

## Example 1

The following sample generates CS0681:

```
C#  
  
// CS0681.cs  
// compile with: /target:library  
abstract class C  
{  
    abstract int num; // CS0681  
}
```

## Example 2

Try the following code instead:

```
C#  
  
// CS0681b.cs  
// compile with: /target:library  
abstract class C  
{  
    public abstract int num  
    {  
        get;  
        set;  
    }  
}
```

# Compiler Error CS0682

Article • 09/15/2021 • 2 minutes to read

'type1' cannot implement 'type2' because it is not supported by the language

This error occurs when you try to implement an interface written in another language and the compiler does not support the interface.

# Compiler Error CS0683

Article • 09/15/2021 • 2 minutes to read

'explicitmethod' explicit method implementation cannot implement 'method' because it is an accessor

The following sample generates CS0683:

C#

```
// CS0683.cs
interface IExample
{
    int Test { get; }

class CExample : IExample
{
    int IExample.get_Test() { return 0; } // CS0683
    int IExample.Test { get{ return 0; } } // correct
}
```

# Compiler Error CS0685

Article • 09/15/2021 • 2 minutes to read

Conditional member 'member' cannot have an out parameter

When using the [ConditionalAttribute](#) attribute on a method, that method may not have an out parameter. This is because the value of the variable used for the out parameter would not be defined in the case that the method call is compiled to nothing. To avoid this error, remove the out parameter from the conditional method declaration, or don't use the Conditional Attribute.

## Example

The following sample generates CS0685:

C#

```
// CS0685.cs
using System.Diagnostics;

class C
{
    [Conditional("DEBUG")]
    void trace(out int i) // CS0685
    {
        i = 1;
    }
}
```

# Compiler Error CS0686

Article • 09/15/2021 • 2 minutes to read

Accessor 'accessor' cannot implement interface member 'member' for type 'type'. Use an explicit interface implementation.

Suggested: This error can occur when implementing an interface that contains method names which conflict with the auto-generated methods associated with a property or event. The get/set methods for properties are generated as get\_property and set\_property, and the add/remove methods for events are generated as add\_event and remove\_event. If an interface contains either of these methods, a conflict occurs. To avoid this error, implement the methods using an explicit interface implementation. To do this, specify the function as:

C#

```
Interface.get_property() { /* */ }
Interface.set_property() { /* */ }
```

## Example 1

The following sample generates CS0686:

C#

```
// CS0686.cs
interface I
{
    int get_P();
}

class C : I
{
    public int P
    {
        get { return 1; } // CS0686
    }
}
// But the following is valid:
class D : I
{
    int I.get_P() { return 1; }
    public static void Main() {}
}
```

## Example 2

This error can also occur when declaring events. The event construct automatically generates the `add_event` and `remove_event` methods, which could conflict with the methods of the same name in an interface, as in the following sample:

```
C#  
  
// CS0686b.cs  
using System;  
  
interface I  
{  
    void add_OnMyEvent(EventHandler e);  
}  
  
class C : I  
{  
    public event EventHandler OnMyEvent  
    {  
        add { } // CS0686  
        remove { }  
    }  
}  
  
// Correct (using explicit interface implementation):  
class D : I  
{  
    void I.add_OnMyEvent(EventHandler e) {}  
    public static void Main() {}  
}
```

# Compiler Error CS0687

Article • 09/15/2021 • 2 minutes to read

The namespace alias qualifier '::' always resolves to a type or namespace so is illegal here. Consider using '.' instead.

This error occurs if you used something which the parser interpreted as a type in an unexpected place. A type or namespace name is valid only in a member access expression, using the member access (.) operator. This could occur if you used the global scope operator (::) in another context.

## Example

The following sample generates CS0687:

C#

```
// CS0687.cs

using M = Test;
using System;

public class Test
{
    public static int x = 77;

    public static void Main()
    {
        Console.WriteLine(M::x); // CS0687
        // To resolve use the following line instead:
        // Console.WriteLine(M.x);
    }
}
```

# Compiler Error CS0689

Article • 09/15/2021 • 2 minutes to read

Cannot derive from 'identifier' because it is a type parameter

Base classes or interfaces for generic classes cannot be specified by a type parameter. Derive from a specific class or interface, or a specific generic class instead, or include the unknown type as a member.

The following sample generates CS0689:

C#

```
// CS0689.cs
class A<T> : T    // CS0689
{                   }
```

# Compiler Error CS0690

Article • 09/15/2021 • 2 minutes to read

Input file 'file' contains invalid metadata.

You are able to open the metadata file, but due to some localization problem, the metadata was corrupted. This is similar to error [CS0009](#), except that you are able to open the metadata file.

A PE (Process Executable) is an executable file.

# Compiler Error CS0692

Article • 09/15/2021 • 2 minutes to read

Duplicate type parameter 'identifier'

The same name may not be used more than once in a type parameter list. Rename or remove the duplicate type parameter(s).

## Example

The following sample generates CS0692:

C#

```
// CS0692.cs
// compile with: /target:library
class C <T, A, T>    // CS0692
{
}

class D <T, T>    // CS0692
{}
```

# Compiler Error CS0694

Article • 09/15/2021 • 2 minutes to read

Type parameter 'identifier' has the same name as the containing type, or method

You must use a different name for the type parameter since the type parameter's name cannot be identical to the type or method name that contains the type parameter.

## Example 1

The following sample generates CS0694.

```
C#  
  
// CS0694.cs  
// compile with: /target:library  
class C<C> {} // CS0694
```

## Example 2

In addition to the above case involving a generic class, this error may occur with a method:

```
C#  
  
// CS0694_2.cs  
// compile with: /target:library  
class A  
{  
    public void F<F>(F arg); // CS0694  
}
```

# Compiler Error CS0695

Article • 09/15/2021 • 2 minutes to read

'generic type' cannot implement both 'generic interface' and 'generic interface' because they may unify for some type parameter substitutions

This error occurs when a generic class implements more than one parameterization of the same generic interface, and there exists a type parameter substitution which would make the two interfaces identical. To avoid this error, implement only one of the interfaces, or change the type parameters to avoid the conflict.

The following sample generates CS0695:

```
C#  
  
// CS0695.cs  
// compile with: /target:library  
  
interface I<T>  
{  
}  
  
class G<T1, T2> : I<T1>, I<T2> // CS0695  
{  
}
```

# Compiler Error CS0698

Article • 09/15/2021 • 2 minutes to read

A generic type cannot derive from 'class' because it is an attribute class

Any class that derives from an attribute class is an attribute. Attributes are not allowed to be generic types.

The following sample generates CS0698:

C#

```
// CS0698.cs
class C<T> : System.Attribute // CS0698
{}
```

# Compiler Error CS0699

Article • 09/15/2021 • 2 minutes to read

'generic' does not define type parameter 'identifier'

A type parameter was used in a generic definition that was not found in the declaration list of the type parameters for that generic. This can happen if the name used for the type parameter was inconsistent.

The following sample generates CS0699:

C#

```
// CS0699.cs
class C<T> where U : I    // CS0699 - U is not a valid type parameter
{}
```

# Compiler Error CS0701

Article • 09/15/2021 • 2 minutes to read

'identifier' is not a valid constraint. A type used as a constraint must be an interface, a non-sealed class or a type parameter.

This error occurs if a sealed type is used as a constraint. To resolve this error, use only non-sealed types as constraints.

## Example

The following sample generates CS0701.

C#

```
// CS0701.cs
// compile with: /target:library
class C<T> where T : System.String {}    // CS0701
class D<T> where T : System.Attribute {}   // OK
```

# Compiler error CS0702

Article • 09/15/2021 • 2 minutes to read

Constraint cannot be special class 'identifier'

The following types may not be used as constraints: [Object](#), [Array](#), or [ValueType](#).

## Example

The following sample generates CS0702:

C#

```
// CS0702.cs
class C<T> where T : System.Array // CS0702
{
}
```

## See also

- [Constraints on Type Parameters](#)



# Compiler Error CS0703

Article • 09/15/2021 • 2 minutes to read

Inconsistent accessibility: constraint type 'identifier' is less accessible than 'identifier'

A constraint may not force the generic parameter to be less accessible than the generic class itself. In the following example, while the generic class C<T> is declared public, the constraint attempts to force T to implement an internal interface. Even if this were allowed, only clients with internal access would be able to create the parameter for the class, so in effect, the class could be used only by clients with internal access.

To get rid of this error, make sure the access level of the generic class is not less restrictive than any classes or interfaces that appear in the bounds.

The following sample generates CS0703:

C#

```
// CS0703.cs
internal interface I {}
public class C<T> where T : I // CS0703 - I is internal; C<T> is public
{
}
```

# Compiler Error CS0704

Article • 09/15/2021 • 2 minutes to read

Cannot do member lookup in 'type' because it is a type parameter

An inner type cannot be specified through a type parameter. Try using the desired type explicitly.

## Example

The following sample generates CS0704:

```
C#  
  
// CS0704.cs  
class B  
{  
    public class I { }  
}  
  
class C<T> where T : B  
{  
    T.I f; // CS0704 - member lookup on type parameter  
    // Try this instead:  
    // B.I f;  
}  
  
class CMain  
{  
    public static void Main() {}  
}
```

# Compiler Error CS0706

Article • 09/15/2021 • 2 minutes to read

Invalid constraint type. A type used as a constraint must be an interface, a non-sealed class or a type parameter.

This error occurs when an invalid construct is used in a constraint clause. To avoid this error, use an interface or non-sealed class instead of the construct that caused the error.

## Example

The following sample generates CS0706.

```
C#  
  
// CS0706.cs  
// compile with: /target:library  
class A {}  
class C<T> where T : int[] {} // CS0706  
class D<T> where T : A {} // OK
```

# Compiler Error CS0708

Article • 09/15/2021 • 2 minutes to read

'field': cannot declare instance members in a static class

This error occurs if you declare a non-static member in a class that is declared static. It is not possible to create instances of static classes, so instance variables would not be meaningful. The **static** keyword should be applied to all members of static classes.

The following sample generates CS0708:

```
C#  
  
// CS0708.cs  
// compile with: /target:library  
public static class C  
{  
    int i; // CS0708  
    static int j; // OK  
}
```

# Compiler Error CS0709

Article • 09/15/2021 • 2 minutes to read

'derived class': cannot derive from static class 'base class'

A static class cannot be instantiated or derived from. That is, a static class cannot be a base class of any other class.

## Example

The following sample generates CS0709.

C#

```
// CS0709.cs
// compile with: /target:library
public static class Base {}
public class Derived : Base {} // CS0709
```

# Compiler Error CS0710

Article • 09/15/2021 • 2 minutes to read

Static classes cannot have instance constructors

A static class cannot be instantiated, hence it has no need for constructors. To avoid this error, remove any constructors from static classes, or if you really want to construct instances, make the class non-static.

The following sample generates CS0710:

```
C#  
  
// CS0710.cs  
public static class C  
{  
    public C() // CS0710  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0711

Article • 09/16/2021 • 2 minutes to read

Static classes cannot contain destructors

A static class cannot be instantiated, hence it has no need for constructors or finalizer.

To avoid this error, remove finalizer from static classes, or, if you really want to construct and destroy instances, make the class non-static.

The following sample generates CS0711:

C#

```
// CS0711.cs
public static class C
{
    ~C() // CS0711
    {

        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS0712

Article • 09/15/2021 • 2 minutes to read

Cannot create an instance of the static class 'static class'

It is not possible to create instances of static classes. Static classes are designed to contain static fields and methods, but may not be instantiated.

## Example

The following sample generates CS0712:

C#

```
// CS0712.cs
public static class SC
{
}

public class CMain
{
    public static void Main()
    {
        SC sc = new SC(); // CS0712
    }
}
```

# Compiler Error CS0713

Article • 09/15/2021 • 2 minutes to read

Static class 'static type' cannot derive from type 'type'. Static classes must derive from object.

If this were allowed, the static class would inherit methods and non-static members from the base class, so it would not be static. Therefore, it is not allowed.

The following sample generates CS0713:

```
C#  
  
// CS0713.cs  
public class Base  
{  
}  
  
public static class Derived : Base // CS0713  
{  
}  
  
public class CMain  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0714

Article • 09/15/2021 • 2 minutes to read

'static type' : static classes cannot implement interfaces

Interfaces may define non-static methods on objects and hence may not be implemented by static classes. To resolve this error, make sure your class does not attempt to implement any interfaces.

## Example

The following sample generates CS0714:

C#

```
// CS0714.cs
interface I
{
}

public static class C : I // CS0714
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0715

Article • 09/15/2021 • 2 minutes to read

'static class' : static classes cannot contain user defined operators

User defined operators operate on instances of classes. Static classes cannot be instantiated, so it is not possible to create instances for operators to act upon. Hence, user defined operators are not allowed for static classes.

The following sample generates CS0715:

```
C#  
  
// CS0715.cs  
public static class C  
{  
    public static C operator+(C c) // CS0715  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS0716

Article • 09/15/2021 • 2 minutes to read

Cannot convert to static type 'type'

This error occurs if your code uses a cast to convert to a static type. Since it is not possible for an object to be an instance of a static type, casting to a static type can never be a meaningful cast.

## Example

The following sample generates CS0716:

C#

```
// CS0716.cs

public static class SC
{
    static void F() { }

public class Test
{
    public static void Main()
    {
        object o = new object();
        System.Console.WriteLine((SC)o); // CS0716
    }
}
```

# Compiler Error CS0717

Article • 09/15/2021 • 2 minutes to read

'static class': static classes cannot be used as constraints

Static classes cannot be extended as they only contain static members and not instance members. Because they cannot be extended, this makes them useless as type parameters and constraints, as no type can exist that is a specialization of a static class.

## Example

The following sample generates CS0717:

C#

```
// CS0717.cs

public static class SC
{
    public static void F()
    {
    }
}

public class G<T> where T : SC // CS0717
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS0718

Article • 09/15/2021 • 2 minutes to read

'type': static types cannot be used as type arguments

Because a static type cannot be instantiated, it cannot be used as a generic argument. To resolve this error, remove the static type from the generic argument.

## Example

The following sample generates CS0718:

C#

```
// CS0718.cs
public static class SC
{
    public static void F()
    {
    }
}

public class G<T>
{
}

public class CMain
{
    public static void Main()
    {
        G<SC> gsc = new G<SC>(); // CS0718
    }
}
```

# Compiler Error CS0719

Article • 09/15/2021 • 2 minutes to read

'type': array elements cannot be of static type

An array of static type does not make sense since array elements are instances, but it is not possible to create instances of static types.

The following sample generates CS0719:

C#

```
// CS0719.cs
public static class SC
{
    public static void F()
    {
    }
}

public class CMain
{
    public static void Main()
    {
        SC[] sca = new SC[10]; // CS0719
    }
}
```

# Compiler Error CS0720

Article • 09/15/2021 • 2 minutes to read

'static class': cannot declare indexers in a static class

Indexers are not meaningful in static classes, since they can only be used with instances, and it is not possible to create instances of a static type.

## Example

The following sample generates CS0720:

```
C#  
  
// CS0720.cs  
  
public static class Test  
{  
    public int this[int index] // CS0720  
    {  
        get { return 1; }  
        set {}  
    }  
  
    static void Main() {}  
}
```

# Compiler Error CS0721

Article • 09/15/2021 • 2 minutes to read

'type': static types cannot be used as parameters

A static type is not meaningful as a parameter. Since no instances of static types may be created, no instance could ever be passed as a parameter.

The following sample generates CS0721:

C#

```
// CS0721.cs
public static class SC
{
}

public class CMain
{
    public void F(SC sc) // CS0721
    {
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0722

Article • 09/15/2021 • 2 minutes to read

'type': static types cannot be used as return types

A static type as a return type is not meaningful since instances of static types cannot be created.

The following sample generates CS0722:

C#

```
// CS0722.cs
public static class SC
{
}

public class CMain
{
    public SC F() // CS0722
    {
        return null;
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS0723

Article • 09/15/2021 • 2 minutes to read

Cannot declare variable of static type 'type'

Instances of static types cannot be created.

The following sample generates CS0723:

C#

```
// CS0723.cs
public static class SC
{
}

public class CMain
{
    public static void Main()
    {
        SC sc = null; // CS0723
    }
}
```

# Compiler Error CS0724

Article • 09/15/2021 • 2 minutes to read

A throw statement with no arguments is not allowed in a finally clause that is nested inside the nearest enclosing catch clause

## Example

The following example generates CS0724 because of the `throw` statement inside the `finally` clause block:

C#

```
// CS0724.cs
using System;

class Program
{
    static void Test()
    {
        try
        {
            throw new Exception();
        }
        catch
        {
            try
            {
            }
            finally
            {
                throw; // CS0724
            }
        }
    }

    static void Main()
    {
    }
}
```

# Compiler Error CS0726

Article • 09/15/2021 • 2 minutes to read

'format specifier' is not a valid format specifier

This error occurs in the debugger. When you type a variable name into one of the debugger windows, you can follow it with a comma, and then a format specifier. Examples are: `myInt, h` or `myString,nq`. This error arises when the compiler does not recognize the [Format Specifiers in C#](#).

# Compiler Error CS0727

Article • 09/15/2021 • 2 minutes to read

## Invalid format specifier

This error occurs in the debugger. When you type a variable name into one of the debugger windows, you can follow it with a comma, and then a format specifier. Examples are: myInt, h; or myString,nq. This error arises when the compiler is completely unable to parse what you typed in. To resolve this error, retype the variable name, optionally followed by a comma and a [valid Format Specifier](#).

# Compiler Error CS0729

Article • 09/15/2021 • 2 minutes to read

Type 'type' is defined in this assembly, but a type forwarder is specified for it

You cannot use a type forwarder for a type defined in the same assembly.

## Example

The following sample generates CS0729.

C#

```
// CS0729.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:TypeForwardedTo(typeof(TestClass))]    // CS0729
class TestClass {}
```

# Compiler Error CS0730

Article • 09/15/2021 • 2 minutes to read

Cannot forward type 'type' because it is a nested type of 'type'

This error is generated when you try to forward a nested class.

## Example

The following sample generates CS0730. It consists of two source files. First, compile the library file `CS0730a.cs`, and then compile the file `CS0730.cs` referencing the library file.

C#

```
// CS0730a.cs
// compile with: /t:library
public class Outer
{
    public class Nested {}
}
```

C#

```
// CS0730.cs
// compile with: /t:library /r:CS0730a.dll
using System.Runtime.CompilerServices;

[assembly:TypeForwardedToAttribute(typeof(Outer.Nested))] // CS0730

[assembly:TypeForwardedToAttribute(typeof(Outer))] // OK
```

# Compiler Error CS0731

Article • 09/15/2021 • 2 minutes to read

The type forwarder for type 'type' in assembly 'assembly' causes a cycle

This error can only occur with improperly formed imported metadata. It cannot occur with only C# source.

## Example

The following sample generates CS0731. The example consists of three files:

1. Circular.il
2. Circular2.il
3. CS0731.cs

First compile the .IL files as libraries, and then compile the .cs code referencing the two files.

```
il

// Circular.il
// compile with: /DLL /out=Circular.dll
.assembly extern circular2
{
    .ver 0:0:0:0
}
.assembly extern circular3
{
    .ver 0:0:0:0
}
.assembly extern mscorelib
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )    // .z\V.4..
    .ver 2:0:0:0
}
.assembly Circular
{
    .custom instance void
[mscorelib]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::.
ctor(int32) = ( 01 00 08 00 00 00 00 00 )
    .hash algorithm 0x00008004
    .ver 0:0:0:0
}
.class extern forwarder Circular.Referenced.TypeForwarder
{
```

```
.assembly extern circular2
}
.module Circular.dll
// MVID: {880C2329-C915-42A0-83E9-9D10C3E6DBD0}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003      // WINDOWS_CUI
.corflags 0x00000001    // ILOONLY
// Image base: 0x04E40000
// ===== CLASS MEMBERS DECLARATION =====
.class public abstract auto ansi sealed beforefieldinit User
    extends [mscorlib]System.Object
{
    .method public hidebysig static class
[circular2]Circular.Referenced.TypeForwarder
        F() cil managed
    {
        .maxstack 1
        newobj     instance void
[circular2]Circular.Referenced.TypeForwarder::ctor()
        ret
    }
}
```

il

```
// Circular2.il
// compile with: /DLL /out=Circular2.dll
.assembly extern mscorel
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )    // .z\V.4..
    .ver 2:0:0:0
}
.assembly extern Circular
{
    .ver 0:0:0:0
}
.assembly circular2
{
    .custom instance void
[mscorlib]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::.
ctor(int32) = ( 01 00 08 00 00 00 00 00 )
    .hash algorithm 0x00008004
    .ver 0:0:0:0
}
.class extern forwarder Circular.Referenced.TypeForwarder
{
    .assembly extern Circular
}
.module circular2.dll
// MVID: {8B3BE5C8-DBE1-49C4-BC72-DF35F0387C21}
.imagebase 0x00400000
```

```
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003      //  WINDOWS_CUI
.corflags 0x00000001   //  ILOONLY
// Image base: 0x04E40000
```

C#

```
// CS0731.cs
// compile with: /reference:circular.dll /reference:circular2.dll
// CS0731 expected
class A {
    public static void Main() {
        User.F();
    }
}
```

# Compiler Error CS0733

Article • 09/15/2021 • 2 minutes to read

Cannot forward generic type, 'GenericType<>'

## Example

The following example generates CS0733. Compile the first file as a library, and then reference it when you compile the second file.

C#

```
// CS0733a.cs
// compile with: /target:library
public class GenericType<T>
{
}
```

C#

```
// CS0733.cs
// compile with: /target:library /r:CS0733a.dll
[assembly:
System.Runtime.CompilerServices.TypeForwardedTo(typeof(GenericType<int>))]
// CS0733
```

# Compiler Error CS0734

Article • 09/15/2021 • 2 minutes to read

The /moduleassemblyname option may only be specified when building a target type of 'module'

The compiler option **ModuleAssemblyName** should only be used when building a .netmodule. See [ModuleAssemblyName \(C# Compiler Option\)](#) for more information.

For more information on building a .netmodule, see the **module** element for the [TargetType \(C# Compiler Options\)](#).

## Example

The following sample generates CS0734. To resolve, add

<TargetType>module</TargetType> to the compilation.

C#

```
// CS0734.cs
// compile with: /moduleassemblyname:A
// CS0734 expected
public class Test {}
```

# Compiler Error CS0735

Article • 09/15/2021 • 2 minutes to read

Invalid type specified as an argument for TypeForwardedTo attribute

The following sample generates CS0735.

C#

```
// CS735.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:TypeForwardedTo(typeof(int[]))]    // CS0735
[assembly:TypeForwardedTo(typeof(string))]   // OK
```

# Compiler Error CS0736

Article • 09/15/2021 • 2 minutes to read

'type name' does not implement interface member 'member name'. 'method name' cannot implement an interface member because it is static.

This error is generated when a static method is either implicitly or explicitly declared as an implementation of an interface member.

## To correct this error

- Remove the `static` modifier from the method declaration.
- Change the name of the interface method.
- Redefine the containing type so that it does not inherit from the interface.

## Example

The following code generates CS0736 because `Program.testMethod` is declared as static:

```
C#  
  
// cs0736.cs  
namespace CS0736  
{  
  
    interface ITest  
    {  
        int testMethod(int x);  
    }  
  
    class Program : ITest // CS0736  
    {  
        public static int testMethod(int x) { return 0; }  
        // Try the following line instead.  
        // public int testMethod(int x) { return 0; }  
        public static void Main() { }  
    }  
}
```

## See also

- [Interfaces](#)



# Compiler Error CS0737

Article • 09/15/2021 • 2 minutes to read

'type name' does not implement interface member 'member name'. 'method name' cannot implement an interface member because it is not public.

A method that implements an interface member must have public accessibility. All interface members are `public`.

## To correct this error

1. Add the `public` access modifier to the method.

## Example

The following code generates CS0737:

```
C#  
  
// cs0737.cs  
interface ITest  
{  
    // Default access of private with no modifier.  
    int Return42();  
    // Try the following line instead.  
    // public int Return42();  
}  
  
struct Struct1 : ITest // CS0737  
{  
    int Return42() { return (42); }  
}  
  
public class Test  
{  
    public static int Main(string[] args)  
    {  
        Struct1 s1 = new Struct1();  
  
        return (1);  
    }  
}
```

## See also

- [Interfaces](#)

# Compiler Error CS0738

Article • 09/15/2021 • 2 minutes to read

'type name' does not implement interface member 'member name'. 'method name' cannot implement 'interface member' because it does not have the matching return type of ' type name'.

The return value of an implementing method in a class must match the return value of the interface member that it implements.

## To correct this error

1. Change the return type of the method to match that of the interface member.

## Example

The following code generates CS0738 because the class method returns `void` and the interface member of the same name returns `int`:

```
C#  
  
using System;  
  
interface ITest  
{  
    int TestMethod();  
}  
public class Test: ITest  
{  
    public void TestMethod() { } // CS0738  
    // Try the following line instead.  
    // public int TestMethod();  
}
```

## See also

- [Interfaces](#)



# Compiler Error CS0739

Article • 09/15/2021 • 2 minutes to read

'type name' duplicate TypeForwardedToAttribute.

An assembly can have no more than one [TypeForwardedToAttribute](#) to an external type.

## To correct this error

1. Locate and remove the duplicate [TypeForwardedToAttribute](#).

## Example

The following code generates CS0739:

C#

```
// CS0739.cs
// CS0739
// Assume that a class Test is declared in a separate dll
// with a namespace that is named cs739dll.
[assembly:
System.Runtime.CompilerServices.TypeForwardedTo(typeof(cs739dll.Test))]
[assembly:
System.Runtime.CompilerServices.TypeForwardedTo(typeof(cs739dll.Test))]
namespace cs0739
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

## See also

- [TypeForwardedToAttribute](#)



# Compiler Error CS0742

Article • 09/15/2021 • 2 minutes to read

A query body must end with a select clause or a group clause

A query expression must terminate with either a `select` clause or a `group` clause without a continuation.

## To correct this error

1. Add a `select clause` or `group clause` to the query.

## Example

The following code generates CS0742:

C#

```
// cs0742.cs
using System.Linq;
public class Test
{
    public static int Main()
    {
        int[] array = { 1, 2, 3 };
        var query = from num in array; // CS0742
        return 1;
    }
}
```

If the `group` clause uses the `into` keyword to store the results of the grouping into a temporary identifier, it cannot be the last clause in a query. A `select` or second `group` clause is still required.

## See also

- [LINQ Query Expressions](#)



# Compiler Error CS0743

Article • 09/15/2021 • 2 minutes to read

Expected contextual keyword 'on'

The pattern for a `join` clause is `join...in...on...equals`, as shown in this example:

C#

```
var query = from x in array1
             join y in array2 on x equals y
             select x;
```

## To correct this error

1. Add the `on` keyword to the `join` clause.

## Example

The following code generates CS0743:

C#

```
// cs0743.cs
using System;
using System.Linq;

public class C
{
    public static int Main()
    {
        int[] array1 = { 1, 2, 3, 4, 5, 6 };
        int[] array2 = { 5, 6, 7, 8, 9 };
        var c = from x in array1
                join y in array2 x equals y // CS0743
                select x;
        return 1;
    }
}
```

## See also

- [LINQ Query Expressions](#)

- join clause

# Compiler Error CS0744

Article • 09/15/2021 • 2 minutes to read

Expected contextual keyword 'equals'

The pattern for a `join` clause is `join...in...on...equals`, as shown in this example:

C#

```
var query = from x in array1
             join y in array2 on x equals y
             select x;
```

## To correct this error

1. Add the `equals` keyword to the `join` clause.

## Example

The following code generates CS0744:

C#

```
// cs0744.cs
using System;
using System.Linq;

public class C
{
    public static int Main()
    {
        int[] array1 = { 1, 2, 3 ,4, 5, 6 };
        int[] array2 = { 5, 6, 7, 8, 9 };
        var c = from x in array1
                join y in array2 on x y // CS0744
                select x;
        return 1;
    }
}
```

## See also

- [LINQ Query Expressions](#)

- join clause

# Compiler Error CS0745

Article • 09/15/2021 • 2 minutes to read

Expected contextual keyword 'by'

The pattern for the `group` clause is `group...by` followed by an optional `into`, as shown in the following example:

C#

```
string[] names = { "Bob", "Bill", "Jonetta", "Mary" };

var query = from name in names
            group name by name[0];
```

or

C#

```
var query2 = from name in names
              group name by name[0] into g
              //...additional query clauses
```

## To correct this error

1. Add the `by` keyword to the `group` clause.

## Example

The following code generates CS0745:

C#

```
// cs0745.cs
using System;
using System.Linq;

public class C
{
    public static int Main()
    {
        string[] names = { "Bob", "Bill", "Jonetta", "Mary" };

        var query = from name in names
                    group name by name[0]; // CS0745
```

```
        return 1;
    }
}
```

## See also

- [LINQ Query Expressions](#)
- [group clause](#)

# Compiler Error CS0746

Article • 09/15/2021 • 2 minutes to read

Invalid anonymous type member declarator. Anonymous type members must be declared with a member assignment, simple name or member access.

An anonymous type must be declared with a member assignment, simple name, or member access.

## To correct this error

1. Ensure that your declaration uses only member assignment, simple names, or member access expressions.

## Example

The following code generates CS0746 in the declaration of `incorrect_1` and `incorrect_2`. The following declarations show two of the correct ways to declare an anonymous type.

C#

```
// cs0746.cs
public class C
{
    public static int Main()
    {
        int i = 100;
        string s = "Bottles of beer.";

        var incorrect_1 = new { a.b = 1 }; // CS0746
        var incorrect_2 = new {100, "Bottles of beer."}; // CS0746
        var correct_1 = new { i, s }; //OK
        var correct_2 = new {num = i, message = s}; // OK

        return 1;
    }
}
```

## See also

- [Anonymous Types](#)



# Compiler Error CS0747

Article • 09/15/2021 • 2 minutes to read

Invalid initializer member declarator.

An object initializer is used to assign values to properties or fields. Any expression which is not an assignment to a property or field is a compile-time error.

## To correct this error

1. Ensure that all expressions in the initializer are assignments to properties or fields of the type. In the following example, the second expression is an error because the value `1` is not assigned to any property or field of `List<int>`.

## Example

The following code generates CS0747:

```
C#  
  
// cs0747.cs  
using System.Collections.Generic;  
  
public class C  
{  
    public static int Main()  
    {  
        var t = new List<int> { Capacity = 2, 1 }; // CS0747  
        return 1;  
    }  
}
```

## See also

- [Object and Collection Initializers](#)



# Compiler Error CS0748

Article • 09/15/2021 • 2 minutes to read

Inconsistent lambda parameter usage; parameter types must be all explicit or all implicit.

If a lambda expression has multiple input parameters, some parameters cannot use implicit typing while others use explicit typing.

To correct this error, either omit all parameter type declarations or explicitly specify the type of all parameters.

## Example

The following code generates CS0748, because, in the lambda expression, only `alpha` is given an explicit type:

C#

```
class CS0748
{
    System.Func<int, int, int> d = (int alpha, beta) => beta / alpha;
```

## See also

- [Lambda expressions](#)



# Compiler Error CS0750

Article • 09/15/2021 • 2 minutes to read

A partial method cannot have access modifiers or the virtual, abstract, override, new, sealed, or extern modifiers.

Because of their special behavior, partial methods are subject to restrictions as to the modifiers they can accept.

## To correct this error

1. Remove the unauthorized modifier from the method declaration.

## Example

The following example generates CS0750:

```
C#  
  
// cs0750.cs  
using System;  
  
public class Base  
{  
    protected virtual void PartG()  
    {  
    }  
  
    protected void PartH()  
    {  
    }  
    protected virtual void PartI()  
    {  
    }  
}  
  
public partial class C:Base  
{  
    // All these partial method declarations  
    // will generate CS0750.  
    public partial void PartA();  
    private partial void PartB();  
    protected partial void PartC();  
    internal partial void PartD();  
    virtual partial void PartE();  
    abstract partial void PartF();  
    override partial void PartG();
```

```
new partial void PartH();
sealed override partial void PartI();
extern partial void PartJ();

public static int Main()
{
    return 1;
}
```

## See also

- [Partial Classes and Methods](#)

# Compiler Error CS0751

Article • 09/15/2021 • 2 minutes to read

A partial method must be declared in a partial class or partial struct

It is not possible to declare a [partial](#) method unless it is encapsulated inside a partial class or partial struct.

## To correct this error

1. Either remove the `partial` modifier from the method and provide an implementation, or else add the `partial` modifier to the enclosing class or struct.

## Example

The following example generates CS0751:

```
C#  
  
// cs0751.cs  
using System;  
  
public class C  
{  
    partial void Part(); // CS0751  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)



# Compiler Error CS0752

Article • 09/15/2021 • 2 minutes to read

A partial method cannot have out parameters

A partial method cannot have an `out` parameter. Out parameters are not allowed because if the partial method is removed by the compiler then there is no guarantee that the out parameter is ever assigned.

## To correct this error

1. Remove the `out` modifier from the parameter and use the return value of the method instead, or else remove the `partial` modifier from the method declaration.

## Example

The following code generates CS0752:

```
C#  
  
// cs0752.cs  
public partial class C  
{  
    partial void Part(out int num); // CS0752  
    // try the following line instead  
    // partial void Part(int num);  
  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)



# Compiler Error CS0753

Article • 09/15/2021 • 2 minutes to read

Only methods, classes, structs, or interfaces may be partial.

The [partial](#) modifier can only be used with classes, structs, interfaces, and methods.

## To correct this error

1. Remove the `partial` modifier from the variable or language construct.

## Example

The following code generates CS0753:

```
C#  
  
// cs0753.cs  
using System;  
  
public partial class C  
{  
    partial int num; // CS0753  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)



# Compiler Error CS0754

Article • 09/15/2021 • 2 minutes to read

A partial method may not explicitly implement an interface method.

A partial method cannot be declared as an explicit implementation of a method defined in an interface.

## To correct this error

1. Remove the explicit interface qualification from the method declaration.

## Example

The following code generates CS0754:

```
C#  
  
// cs0754.cs  
using System;  
  
public interface IF  
{  
    void Part();  
}  
  
public partial class C : IF  
{  
    partial void IF.Part(); //CS0754  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Explicit Interface Implementation](#)
- [Partial Classes and Methods](#)



# Compiler Error CS0755

Article • 09/15/2021 • 2 minutes to read

Both partial method declarations must be extension methods or neither may be an extension method.

A partial method consists of a defining declaration (signature) and an optional implementing declaration (body). If the defining declaration is an extension method, the implementing declaration, if one is defined, must also be an extension method. And if the defining method is not an extension method, the implementing must not be one either.

## To correct this error

1. Either remove the `this` modifier from one of the parts, or add it to the other.

## Example

The following example generates CS0755:

```
C#  
  
// cs0755.cs  
public static partial class Ext  
{  
    static partial void Part(this C c); //Extension method  
  
    // Typically the implementing declaration is in a separate file.  
    static partial void Part(C c) //CS0755  
    {  
    }  
}  
  
public partial class C  
{  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- Extension Methods

# Compiler Error CS0756

Article • 09/15/2021 • 2 minutes to read

A partial method may not have multiple defining declarations.

The defining declaration of a partial method is the part that specifies the method signature, but not the implementation (method body). A partial method must have exactly one defining declaration for each unique signature. Each overloaded version of a partial method must have its own defining declaration.

## To correct this error

1. Remove all except one defining declaration for the partial method.

## Example

C#

```
// cs0756.cs
using System;

public partial class C
{
    partial void Part();
    partial void Part(); // CS0756
    public static int Main()
    {
        return 1;
    }
}
```

## See also

- [Partial Classes and Methods](#)



# Compiler Error CS0757

Article • 09/15/2021 • 2 minutes to read

A partial method may not have multiple implementing declarations.

A partial method consists of exactly one defining declaration (signature) and one or zero implementing declarations (body). Multiple implementing declarations for the same identical defining declarations are not allowed. Partial methods may be overloaded, and each overloaded version may have one or zero implementing declarations.

## To correct this error

1. Remove all except one of the implementing declarations for the partial method.

## Example

The following example generates CS0757:

```
C#  
  
// cs0757.cs  
using System;  
  
public partial class C  
{  
    // Defining declaration.  
    partial void Part();  
  
    // Implementing declaration.  
    partial void Part()  
    {  
        //...Do something.  
    }  
  
    // Second implementing declaration.  
    partial void Part() // CS0757  
    {  
        //...Do something.  
    }  
  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)

# Compiler Error CS0758

Article • 09/15/2021 • 2 minutes to read

Both partial method declarations must use a `params` parameter or neither may use a `params` parameter.

If one part of a partial method specifies a `params` parameter, the other part must specify one also.

## To correct this error

1. Either add the `params` modifier in one part of the method, or remove it in the other.

## Example

The following code generates CS0758:

```
C#  
  
using System;  
  
public partial class C  
{  
    partial void Part(int i, params char[] array);  
    partial void Part(int i, char[] array) // CS0758  
    {  
    }  
  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)
- [params](#)



# Compiler Error CS0759

Article • 09/15/2021 • 2 minutes to read

No defining declaration found for implementing declaration of partial method 'method'.

A partial method must have a defining declaration that defines the signature (name, return type and parameters) of the method. The implementation or method body is optional.

## To correct this error

1. Provide a defining declaration for the partial method in the other part of a partial class or struct.

## Example

The following example generates CS0759:

```
C#  
  
// cs0759.cs  
using System;  
  
public partial class C  
{  
    partial void Part() // CS0759  
    {  
    }  
  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)



# Compiler Error CS0761

Article • 09/15/2021 • 2 minutes to read

Partial method declarations of 'method<T>' have inconsistent type parameter constraints.

If a partial method has an implementation, the generic type constraint must be identical to the constraint defined on the method signature.

## To correct this error

1. Make the generic type constraints identical on each part of the partial method.

## Example

The following code generates CS0761:

```
C#  
  
// cs0761.cs  
using System;  
  
public partial class C  
{  
    partial void Part<T>() where T : class;  
    partial void Part<T>() where T : struct // CS0761  
    {  
    }  
  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)
- [Constraints on Type Parameters](#)



# Compiler Error CS0762

Article • 09/15/2021 • 2 minutes to read

Cannot create delegate from method 'method' because it is a partial method without an implementing declaration

A partial method is not required to have an implementing declaration. However, a delegate does require that its encapsulated method have an implementation.

## To correct this error

1. Provide an implementation for the method that is used to initialize the delegate.

## Example

C#

```
public delegate void TestDel();

public partial class C
{
    partial void Part();

    public static int Main()
    {
        C c = new C();
        TestDel td = new TestDel(c.Part); // CS0762
        return 1;
    }
}
```

# Compiler Error CS0763

Article • 09/15/2021 • 2 minutes to read

Both partial method declarations must be static or neither may be static.

A partial method declaration cannot have one part static and the other part not static.

## To correct this error

1. Make both parts either static or non-static.

## Example

The following code generates CS0763:

C#

```
// cs0763.cs
using System;

public partial class C
{
    static partial void Part();
    partial void Part() // CS0763
    {

        public static int Main()
        {
            return 1;
        }
}
```

## See also

- [Partial Classes and Methods](#)
- [static](#)



# Compiler Error CS0764

Article • 09/15/2021 • 2 minutes to read

Both partial method declarations must be unsafe or neither may be unsafe

A partial method consists of a defining declaration (signature) and an optional implementing declaration (body). If the defining declaration has the `unsafe` modifier, the implementing declaration must also have it. Conversely, if the implementing declaration has the `unsafe` modifier, the defining declaration must also.

## To correct this error

1. Assuming that the defining declaration is correct, add or remove the `unsafe` modifier from the implementing declaration to match the defining declaration.

## Example

```
C#  
  
// cs0764.cs  
// Compile with: /target:library /unsafe  
public partial class C  
{  
    partial void Part();  
    unsafe partial void Part() //CS0764  
    {  
    }  
  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)



# Compiler Error CS0765

Article • 03/09/2023 • 2 minutes to read

Partial methods with only a defining declaration or removed conditional methods cannot be used in expression trees

Although a call to a removed partial method is an expression, it is not an acceptable expression in an expression tree.

## To correct this error

1. Add an implementing declaration for the partial method, or remove the code that is causing the conditional method to be excluded from compilation.

## Example

The following code generates CS0765 in two locations:

```
C#  
  
// cs0765.cs  
using System;  
using System.Collections;  
using System.Collections.Generic;  
using System.Diagnostics;  
using System.Linq;  
using System.Linq.Expressions;  
  
public delegate void delete();  
  
public class ConClass  
{  
    [Conditional("CONDITION")]  
    public static void TestMethod() { }  
}  
  
public partial class PartClass : IEnumerable  
{  
    List<object> list = new List<object>();  
  
    partial void Add(int x);  
  
    public IEnumerator GetEnumerator()  
    {  
        for (int i = 0; i < list.Count; i++)  
            yield return list[i];  
    }  
}
```

```
static void Main()
{
    Expression<Func<PartClass>> testExpr1 = () => new PartClass { 1, 2
}; // CS0765
    Expression<dele> testExpr2 = () => ConClass.TestMethod(); // CS0765
}
}
```

## See also

- [Partial Classes and Methods](#)
- [Expression Trees \(C#\)](#)

# Compiler Error CS0766

Article • 09/15/2021 • 2 minutes to read

Partial methods must have a void return type.

A partial method cannot return a value. Its return type must be void.

## To correct this error

1. Give the partial method a void return type, or else convert the method to a regular (not partial) method.

## Example

The following example generates CS0766:

```
C#  
  
// cs0766.cs  
using System;  
  
public partial class C  
{  
    partial int Part(); // CS0766  
  
    // Typically the implementing declaration  
    // is contained in a separate file.  
    partial int Part() //CS0766  
    {  
    }  
  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Partial Classes and Methods](#)



# Compiler Error CS0767

Article • 10/07/2022 • 2 minutes to read

Cannot inherit interface with the specified type parameters because it causes method to contain overloads which differ only on ref and out

## Example

The following sample generates CS0767:

C#

```
// CS0767.cs (0,0)

using System;

namespace Stuff
{
    interface ISomeInterface<T, U>
    {
        void Method(ref Func<U, T> _);
        void Method(out Func<T, U> _);
    }

    class Explicit : ISomeInterface<int, int>
    {
        void ISomeInterface<int, int>.Method(ref Func<int, int> v) { v = _ => throw new NotImplementedException(); }
        void ISomeInterface<int, int>.Method(out Func<int, int> v) { v = _ => throw new NotImplementedException(); }
    }
}
```

In this example, implementing `ISomeInterface<int,int>` creates two `Method` overloads with the same type of parameter but differs only by `ref/out`. This effectively declares a class that would otherwise raise error CS0663:

C#

```
class BadClass
{
    void Method(ref Func<int, int> v) { v = _ => throw new NotImplementedException(); }
    void Method(out Func<int, int> v) { v = _ => throw new NotImplementedException(); }
}
```

# To correct this error

The simplest way to correct this error is to use different type arguments for `ISomeInterface<T,U>`, for example:

C#

```
class Explicit : ISomeInterface<int, long>
{
    void ISomeInterface<int, long>.Method(ref Func<long, int> v) { v = _ 
=> throw new NotImplementedException(); }
    void ISomeInterface<int, long>.Method(out Func<int, long> v) { v = _ 
=> throw new NotImplementedException(); }
}
```

# Compiler Error CS0811

Article • 09/15/2021 • 2 minutes to read

The fully qualified name for 'name' is too long for debug information. Compile without '/debug' option.

There are size constraints on variable and type names in debug information.

# To correct this error

1. If modifying the name is not possible, the only alternative is to compile without the [DebugType](#) option.

## Example

The following code generates CS0811:

# Compiler Error CS0815

Article • 11/06/2021 • 2 minutes to read

Cannot assign 'expression' to an implicitly typed local

An expression that is used as the initializer for an implicitly typed variable must have a type. Because anonymous function expressions, method group expressions, and the null literal expression do not have a type, they are not appropriate initializers. An implicitly typed variable cannot be initialized with a null value in its declaration, although it can later be assigned a value of null. With C# version 10 Lambda expressions and method groups with natural types can be used as initializers in `var` declarations.

## To correct this error

1. Provide an explicit type for the variable.
2. Or specify natural types with C# version 10 and higher.

## Example

The following code generates CS0815:

```
C#  
  
// cs0815.cs  
class Test  
{  
    public static int Main()  
    {  
        var d = s => -1; // CS0815  
        var e = (string s) => 0; // CS0815 for C# versions before 10  
        var p = null; // CS0815  
        var del = delegate(string a) { return -1; }; // CS0815  
        return -1;  
    }  
}
```

## See also

- [Implicitly Typed Local Variables](#)



# Compiler Error CS0818

Article • 09/15/2021 • 2 minutes to read

Implicitly typed locals must be initialized

An implicitly typed local variable must be initialized with a value at the same time that it is declared.

## To correct this error

1. Assign a value to the variable or else give it an explicit type.

## Example

The following code generates CS0818:

```
C#  
  
// cs0818.cs  
class A  
{  
    public static int Main()  
    {  
        var a; // CS0818  
        return -1;  
    }  
}
```

## See also

- [Implicitly Typed Local Variables](#)



# Compiler Error CS0819

Article • 10/12/2021 • 2 minutes to read

Implicitly-typed variables cannot have multiple declarators.

Multiple declarators are allowed in explicit type declarations, but not with implicitly typed variables.

## To correct this error

There are three options:

1. If the variables are of the same type, use explicit declarations.
2. Declare and assign a value to each implicitly typed local variable on a separate line.
3. Declare a variable using [Tuple deconstruction](#) syntax. **Note:** this option will not work inside a `using` statement as `Tuple` does not implement `IDisposable`.

## Example 1

The following code generates CS0819:

```
C#  
  
// cs0819.cs  
class Program  
{  
    public static void Main()  
    {  
        var a = 3, b = 2; // CS0819  
  
        // First correction option.  
        //int a = 3, b = 2;  
  
        // Second correction option.  
        //var a = 3;  
        //var b = 2;  
  
        // Third correction option.  
        //var (a, b) = (3, 2);  
    }  
}
```

## Example 2

The following code generates CS0819:

```
C#  
  
// cs0819.cs  
class Program  
{  
    public static void Main()  
    {  
        using (var font1 = new Font("Arial", 10.0f),  
              font2 = new Font("Arial", 10.0f)) // CS0819  
        {  
        }  
  
        // First correction option.  
        //using (Font font1 = new Font("Arial", 10.0f),  
        //       font2 = new Font("Arial", 10.0f))  
        //{
/>        //}  
  
        // Second correction option.  
        //using (var font1 = new Font("Arial", 10.0f)  
        //{
/>        //    using (var font2 = new Font("Arial", 10.0f)  
        //    {  
        //    }  
        //}  
    }  
}
```

## See also

- [Implicitly Typed Local Variables](#)
- [Deconstructing Tuples and Other Types](#)

# Compiler Error CS0820

Article • 09/15/2021 • 2 minutes to read

Cannot assign array initializer to an implicitly typed local

An implicitly typed array is an array whose element type is inferred by the compiler. It must be initialized by using the `new[]` modifier as shown in the example code.

## To correct this error

- Use the `new[]` modifier with the array initializer.
- Do not use an implicitly typed local variable.

## Example

The following code generates CS0820 and shows how to correctly initialize an implicitly typed array:

C#

```
//cs0820.cs
class G
{
    public static int Main()
    {
        var a = { 1,2,3}; //CS0820
        // Try using one of the following lines instead.
        // var b = new[] { 1, 2, 3 };
        //int[] b = {1, 2, 3};
        return -1;
    }
}
```

## See also

- [Implicitly Typed Local Variables](#)



# Compiler Error CS0821

Article • 09/15/2021 • 2 minutes to read

Implicitly typed locals cannot be fixed

Implicitly typed local variables and anonymous types are not supported in the `fixed` context.

## To correct this error

1. Either remove the `fixed` modifier from the variable or else give the variable an explicit type.

## Example

The following code generates CS0821:

```
C#  
  
class A  
{  
    static int x;  
  
    public static int Main()  
    {  
        unsafe  
        {  
            fixed (var p = &x) { }  
        }  
        return -1;  
    }  
}
```

## See also

- [Implicitly Typed Local Variables](#)



# Compiler Error CS0822

Article • 09/15/2021 • 2 minutes to read

Implicitly typed locals cannot be const

Implicitly typed local variables are only necessary for storing anonymous types. In all other cases they are just a convenience. If the value of the variable never changes, just give it an explicit type. Attempting to use the `readonly` modifier with an implicitly typed local will generate CS0106.

## To correct this error

1. If you require the variable to be constant or `readonly`, give it an explicit type.

## Example

The following code generates CS0822:

```
C#  
  
// cs0822.cs  
class A  
{  
  
    public static int Main()  
    {  
        const var x = 0; // CS0822.cs  
        return -1;  
    }  
}
```

## See also

- [Implicitly Typed Local Variables](#)



# Compiler Error CS0825

Article • 09/15/2021 • 2 minutes to read

The contextual keyword 'var' may only appear within a local variable declaration.

## To correct this error

1. If the variable belongs at class scope, give it an explicit type. Otherwise move it inside the method where it will be used.

## Example

The following code generates CS0825 because `var` is used on a class field:

C#

```
// cs0825.cs
class Test
{
    // Both of these declarations trigger CS0825
    private var genreName;
    private var bookTitles = new List<string>();

    static int Main()
    {
        var totalBooks = 42; // var is OK here
        return -1;
    }
}
```

## See also

- [Implicitly Typed Local Variables](#)



# Compiler Error CS0826

Article • 09/15/2021 • 2 minutes to read

No best type found for implicitly typed array.

Array elements must all be the same type or implicitly convertible to the same type according to the type inference rules used by the compiler. The best type must be one of the types present in the array expression. Elements will not be converted to a new type such as `object`. For an implicitly typed array, the compiler must infer the array type based on the type of elements assigned to it.

## To correct this error

- Give the array an explicit type.
- Give all array elements the same type.
- Provide explicit casts on those elements that might be causing the problem.

## Example

The following code generates CS0826 because the array elements are not all the same type, and the compiler's type inference logic does not find a single best type:

C#

```
// cs0826.cs
public class C
{
    public static int Main()
    {
        var x = new[] { 1, "str" }; // CS0826

        char c = 'c';
        short s1 = 0;
        short s2 = -0;
        short s3 = 1;
        short s4 = -1;

        var array1 = new[] { s1, s2, s3, s4, c, '1' }; // CS0826
        return 1;
    }
}
```

## See also

- [Implicitly Typed Local Variables](#)

# Compiler Error CS0828

Article • 09/15/2021 • 2 minutes to read

Cannot assign 'expression' to anonymous type property.

An anonymous type cannot be initialized with a null value or an unsafe type, or a method group or anonymous function.

## To correct this error

1. Either add a type declaration to the left side of the assignment, or change the expression on the right side so that it has an acceptable type.

## Example

The following code generates CS0828 because a member of an anonymous type cannot be initialized with a null value.

```
C#  
  
// cs0828.cs  
using System;  
  
public class C  
{  
    public static int Main()  
    {  
        var a = 1;  
        var c = new { p1 = null }; // CS0828  
        return 1;  
    }  
}
```

## See also

- [Implicitly Typed Local Variables](#)



# Compiler Error CS0831

Article • 09/15/2021 • 2 minutes to read

An expression tree may not contain a base access.

A base access means to make a function call that would ordinarily be a virtual function call as a non-virtual function call on the base class method. This is not allowed in the expression tree itself, but you can invoke a helper method in your class that invokes the base class method.

## To correct this error

1. If you must access a base class method in this manner, create a helper method that calls into the base class and have the expression tree call the helper method. This technique is shown in the following code.

## Example

The following example generates CS0831:

```
C#  
  
// cs0831.cs  
using System;  
using System.Linq;  
using System.Linq.Expressions;  
  
public class A  
{  
    public virtual int BaseMethod() { return 1; }  
}  
public class C : A  
{  
    public override int BaseMethod() { return 2; }  
    public int Test(C c)  
    {  
        Expression<Func<int>> e = () => base.BaseMethod(); // CS0831  
  
        // Try the following line instead,  
        // along with the BaseAccessHelper method.  
        // Expression<Func<int>> e2 = () => BaseAccessHelper();  
        return 1;  
    }  
    // Uncomment to call from e2 expression above.  
    // int BaseAccessHelper()  
    // {
```

```
//      return base.BaseMethod();  
// }  
  
}
```

# Compiler Error CS0832

Article • 09/15/2021 • 2 minutes to read

An expression tree may not contain an assignment operator.

An expression tree does not preserve variable state or have any concept of a storage location.

## To correct this error

1. Remove the assignment operator from the lambda or query expression.

## Example

In the example code, as in all lambda expressions, `x` is just an input parameter being passed by value. Its value cannot be changed in an expression tree. It can be changed in a delegate lambda.

C#

```
// cs0843.cs
using System;
using System.Linq;
using System.Linq.Expressions;

public class C
{
    public static int Main()
    {
        Expression<Func<int, int>> e = x => x += 5; // CS0843
        return 1;
    }
}
```

# Compiler Error CS0833

Article • 09/15/2021 • 2 minutes to read

An anonymous type cannot have multiple properties with the same name.

An anonymous type, just like any type, cannot have two properties that have the same name.

## To correct this error

1. Give each property in the type a unique name.

## Example

The following example generates CS0833:

```
// cs0833.cs
using System;

public class C
{
    public static int Main()
    {
        var c = new { p1 = 1, p1 = 2 }; // CS0833
        return 1;
    }
}
```

## See also

- [Anonymous Types](#)



# Compiler Error CS0834

Article • 09/15/2021 • 2 minutes to read

A lambda expression must have an expression body to be converted to an expression tree.

Lambdas that are translated to expression trees must be expression lambdas; statement lambdas and anonymous methods can only be converted to delegate types.

## To correct this error

1. Remove the statement from the lambda expression.

## Example

The following example generates CS0834:

C#

```
// cs0834.cs
using System;
using System.Linq;
using System.Linq.Expressions;

public class C
{
    public static int Main()
    {
        Expression<Func<int, int>> e = x => { return x; }; // CS0834
    }
}
```

# Compiler Error CS0835

Article • 09/15/2021 • 2 minutes to read

Cannot convert lambda to an expression tree whose type argument 'type' is not a delegate type.

If a lambda expression is converted to an expression tree, the expression tree must have a delegate type for its argument. Furthermore, the lambda expression must be convertible to the delegate type.

## To correct this error

1. Change the type parameter from `int` to a delegate type, for example

```
Func<int,int>.
```

## Example

The following example generates CS0835:

```
C#  
  
// cs0835.cs  
using System;  
using System.Linq;  
using System.Linq.Expressions;  
  
public class C  
{  
    public static int Main()  
    {  
        Expression<int> e = x => x + 1; // CS0835  
  
        // Try the following line instead.  
        // Expression<Func<int,int>> e2 = x => x + 1;  
  
        return 1;  
    }  
}
```

# Compiler Error CS0836

Article • 09/15/2021 • 2 minutes to read

Cannot use anonymous type in a constant expression.

The only things allowed in a constant expression are named constants, literals, and mathematical expressions that combine constant expressions.

## To correct this error

1. Make the anonymous type a named type.

## Example

The following example shows one way to generate CS0836:

C#

```
// cs0836.cs
using System;
[AttributeUsage(AttributeTargets.Class, AllowMultiple = true, Inherited =
false)]
public class A : Attribute
{
    public A(object obj)
    {
    }
}

[A(new { })] // CS0836
public class B
{
}

public class Test
{
    public static int Main()
    {
        return 0;
    }
}
```

# Compiler Error CS0837

Article • 09/15/2021 • 2 minutes to read

The first operand of an 'is' or 'as' operator may not be a lambda expression, anonymous method, or method group.

Lambda expressions, anonymous methods and method groups may not be used on the left side of [is](#) or [as](#).

## To correct this error

- If the error involves the `is` operator, remember that `is` takes a value and a type and tells you whether the value can be made into that type by a reference, boxing, or unboxing conversion. Because lambdas are not values and have no reference, boxing, or unboxing conversions, lambdas are not candidates for `is`.
- If the code misuses `as`, the correction is probably to change it to a cast.

## Example

The following example generates CS0837:

```
C#  
  
// cs0837.cs  
namespace TestNamespace  
{  
    public delegate void Del();  
  
    class Test  
    {  
        static int Main()  
        {  
            bool b1 = ((() => { })) is Del; // CS0837  
            bool b2 = delegate() { } is Del; // CS0837  
            Del d1 = () => { } as Del; // CS0837  
            Del d2 = delegate() { } as Del; // CS0837  
            return 1;  
        }  
    }  
}
```

# Compiler Error CS0838

Article • 03/09/2023 • 2 minutes to read

An expression tree may not contain a multidimensional array initializer.

Multidimensional arrays in expression trees cannot be initialized by using an array initializer.

## To correct this error

1. Create and initialize the array before creating the expression tree.

## Example

The following example generates CS0838:

C#

```
// cs0838.cs
using System;
using System.Linq;
using System.Linq.Expressions;

namespace TestNamespace
{
    class Test
    {
        static int Main()
        {

            Expression<Func<int[,]>> expr =
                () => new int[2, 2] { { 1, 2 }, { 3, 4 } }; // CS0838

            // try the following 2 lines instead
            int[,] nums = new int[2, 2] { { 1, 2 }, { 3, 4 } };
            Expression<Func<int[,]>> expr2 = () => nums;

            return 1;
        }
    }
}
```

## See also

- [Expression Trees \(C#\)](#)



# Compiler Error CS0839

Article • 09/15/2021 • 2 minutes to read

Argument missing.

An argument is missing in the method call.

## To correct this error

1. Double check the signature of the method and locate and supply the missing argument.

## Example

The following example generates CS0839:

```
C#  
  
// cs0839.cs  
using System;  
  
namespace TestNamespace  
{  
    class Test  
    {  
        static int Add(int i, int j)  
        {  
            return i + j;  
        }  
  
        static int Main()  
        {  
            int i = Test.Add( , 5); // CS0839  
            return 1;  
        }  
    }  
}
```

# Compiler Error CS0840

Article • 09/15/2021 • 2 minutes to read

'Property name' must declare a body because it is not marked abstract or extern. Automatically implemented properties must define both get and set accessors.

Unless a regular property is marked as `abstract` or `extern`, or is a member of a `partial` type, it must supply a body. Auto-implemented properties do not provide accessor bodies, but they must specify both accessors. To create a read-only auto-implemented property, make the set accessor `private`.

## To correct this error

1. Supply the missing body or accessor or else use the `abstract`, `extern`, or `partial` (Type) modifiers on it and/or its enclosing type.

## Example

The following example generates CS0840:

```
C#  
  
// cs0840.cs  
// Compile with /target:library  
using System;  
class Test  
{  
    public int myProp { get; } // CS0840  
  
    // to create a read-only property  
    // try the following line instead  
    public int myProp2 { get; private set; }  
  
}
```

## See also

- [Auto-Implemented Properties](#)



# Compiler Error CS0841

Article • 02/04/2022 • 2 minutes to read

Cannot use local variable 'name' before it is declared.

A variable must be declared before it is used.

## Example of a variable used before declaration

The following example generates CS0841:

```
C#  
  
// cs0841.cs  
using System;  
  
public class Program  
{  
    public static void Main()  
    {  
        j = 5; // CS0841  
        int j;  
    }  
}
```

## Correct the error by moving the declaration before usage

Move the variable declaration before the line where the error occurs.

```
C#  
  
using System;  
  
public class Program  
{  
    public static void Main()  
    {  
        int j;  
        j = 5;  
    }  
}
```

# Example of a variable shadowing a type

In the following example, the intent was comparing `parameter` with `MyEnum.A`. Because a local variable declared later with the same type name, it shadows the type `MyEnum` and `MyEnum` in this method no longer refers to the `enum`, but refers to the declared local variable.

C#

```
using System;

public enum MyEnum
{
    A, B, C
}

public class C
{
    public void M(MyEnum parameter)
    {
        // error CS0841: Cannot use local variable 'MyEnum' before it is
        // declared
        if (parameter == MyEnum.A)
        {
            return;
        }

        // Change the variable to 'myEnum' to avoid shadowing the 'MyEnum'
        // type.
        // This change also aligns with the C# coding conventions.
        var MyEnum = parameter;
        Console.WriteLine(MyEnum.ToString());
    }
}
```

# Compiler Error CS0842

Article • 09/15/2021 • 2 minutes to read

Automatically implemented properties cannot be used inside a type marked with `StructLayout(LayoutKind.Explicit)`.

Auto-implemented properties have their backing fields provided by the compiler and the field is not accessible to source code. Therefore, they are not compatible with `LayoutKind.Explicit`.

## To correct this error

1. Make the property a regular property in which you provide the accessor bodies.

## Example

The following example generates CS0842:

```
C#  
  
// cs0842.cs  
using System;  
using System.Runtime.InteropServices;  
  
namespace TestNamespace  
{  
    [StructLayout(LayoutKind.Explicit)]  
    struct Str  
    {  
        public int Num // CS0842  
        {  
            get;  
            set;  
        }  
  
        static int Main()  
        {  
            return 1;  
        }  
    }  
}
```

# Compiler Error CS0844

Article • 09/15/2021 • 2 minutes to read

Cannot use local variable 'name' before it is declared. The declaration of the local variable hides the field 'name'.

An identifier can have only one meaning in a given block. Local variables that have the same name as class fields can hide the field by introducing a second meaning for the identifier. Therefore the compiler generates an error when you refer to a class field in a method, and then declare a local variable by the same name.

## To correct this error

- Use `this.num` to refer to the class field.
- Give the local variable a different name from the class field.

## Example

The following code generates CS0844:

```
C#  
  
class Test  
{  
    int num;  
    public void TestMethod()  
    {  
        num = 5; // CS0844  
        int num = 6;  
    }  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

Correct the error by using `this.num` to refer to the class field

```
C#
```

```
class Test
{
    int num;
    public void TestMethod()
    {
        this.num = 5; // Error fixed.
        int num = 6;
    }
    public static int Main()
    {
        return 1;
    }
}
```

**Correct the error by giving the local variable a different name from the class field**

C#

```
class Test
{
    int num;
    public void TestMethod()
    {
        num = 5; // Error fixed.
        int num2 = 6;
    }
    public static int Main()
    {
        return 1;
    }
}
```

# Compiler Error CS0845

Article • 09/15/2021 • 2 minutes to read

An expression tree lambda may not contain a coalescing operator with a null literal left-hand side.

Because null by itself does not have a type, the null coalescing operator cannot operate on it.

## To correct this error

1. Cast the null literal to an object.

## Example

The following code generates CS0845:

C#

```
// cs0845.cs
using System;
using System.Linq;
using System.Linq.Expressions;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Expression<Func<object>> e = () => null ?? null; // CS0845
            // Try the following line instead.
            // Expression<Func<object>> e = () => (object)null ?? null;
        }
    }
}
```

# Compiler Error CS0846

Article • 09/20/2022 • 2 minutes to read

A nested array initializer is expected

This error occurs when something other than an array initializer is used when creating an array.

## Example

The following sample generates CS0846:

```
C#  
  
// CS0846.cs (0,0)  
  
namespace Test  
{  
    public class Program  
    {  
        public void Goo()  
        {  
            var a3 = new[,]{ { { 3, 4 } }, 3, 4 };  
        }  
    }  
}
```

## To correct this error

This example contains a 3-dimensional array. The initializer does not represent a three-dimensional array, resulting in CS0846. The last two statements in the top-level array initializer `, 3, 4` are not an array initializer. Assuming the intent is to create a 3-dimensional array with 2, 1, and 2 elements per dimension, to correct this error, properly bracket the last two statements:

```
C#  
  
var a3 = new[,]{ { { 3, 4 } }, { { 3, 4 } } };
```

# Compiler Error CS0854

Article • 09/16/2022 • 2 minutes to read

An expression tree may not contain a call or invocation that uses optional arguments

## Example

The following sample generates CS0854:

C#

```
// CS0854.cs (10,48)
using System;
using System.Linq.Expressions;

public class Test
{
    public static int ModAdd2(int x = 3, int y = 4, params int[] b) { return
0; }

    static void Main()
    {
        Expression<Func<int>> testExpr = () => ModAdd2();
        Console.WriteLine(testExpr);
    }
}
```

The creation of an expression tree occurs at compile time, but that expression is evaluated and executed at run-time. The evaluation of optional method parameter values occurs at compile time, not during the execution of an expression. Although default parameters are currently required to be compile-time constants, a method declaration (and any default parameter value) may change after creating the expression (and before execution). Since the actual value used cannot be assured during the creation of the expression, it is considered a compile-time error.

## To correct this error

Make the created expression determinate and explicitly declare what value to use when the expression is evaluated and executed:

C#

```
Expression<Func<int>> testExpr = () => ModAdd2(3, 4);
```

# Compiler Error CS1001

Article • 04/02/2022 • 2 minutes to read

## Identifier expected

You did not supply an identifier. An identifier is the name of a class, struct, namespace, method, variable, and so on, that you provide.

The following example declares a simple class but does not give the class a name:

```
C#  
  
public class //CS1001  
{  
    public int Num { get; set; }  
    void MethodA() {}  
}
```

The following sample generates CS1001 because, when declaring an enum, you must specify members:

```
C#  
  
public class Program  
{  
    enum Colors  
    {  
        'a', 'b' // CS1001, 'a' is not a valid int identifier  
        // The following line shows examples of valid identifiers:  
        // Blue, Red, Orange  
    };  
  
    public static void Main()  
    {  
    }  
}
```

Parameter names are required even if the compiler doesn't use them, for example, in an interface definition. These parameters are required so that programmers who are consuming the interface know something about what the parameters mean.

```
C#  
  
interface IMyTest  
{  
    void TestFunc1(int, int); // CS1001  
    // Use the following line instead:
```

```
// void TestFunc1(int a, int b);  
}  
  
class CMyTest : IMyTest  
{  
    void IMyTest.TestFunc1(int a, int b)  
    {  
    }  
}
```

## See also

- [Operators and expressions](#)
- [Types](#)

# Compiler Error CS1002

Article • 09/15/2021 • 2 minutes to read

; expected

The compiler detected a missing semicolon. A semicolon is required at the end of every statement in C#. A statement may span more than one line.

The following sample generates CS1002:

```
C#  
  
// CS1002.cs  
namespace x  
{  
    abstract public class clk  
    {  
        int i    // CS1002, missing semicolon  
  
        public static int Main()  
        {  
            return 0;  
        }  
    }  
}
```

## See also

- [Statements](#)



# Compiler Error CS1003

Article • 09/15/2021 • 2 minutes to read

Syntax error, 'char' expected

The compiler will generate this error for any one of several error conditions. Review your code to find the syntax error.

The following sample generates CS1003:

C#

```
// CS1003.cs
public class b
{
    public static void Main()
    {
        int[] a;
        a[];    // CS1003
    }
}
```

# Compiler Error CS1004

Article • 09/15/2021 • 2 minutes to read

## Duplicate 'modifier' modifier

A duplicate modifier, such as an **access** modifier, was detected.

The following sample generates CS1004:

C#

```
// CS1004.cs
public class clk
{
    public public static void Main()    // CS1004, two public keywords
    {
    }
}
```

# Compiler Error CS1007

Article • 09/15/2021 • 2 minutes to read

Property accessor already defined

When you declare a [property](#), you must also declare its accessor methods. However, a property cannot have more than one `get` accessor method or more than one `set` accessor method.

## Example

The following sample generates CS1007:

C#

```
// CS1007.cs
public class clk
{
    public int MyProperty
    {
        get
        {
            return 0;
        }
        get // CS1007, this is the second get method
        {
            return 0;
        }
    }

    public static void Main() {}
}
```

# Compiler Error CS1008

Article • 09/15/2021 • 2 minutes to read

Type byte, sbyte, short, ushort, int, uint, long, or ulong expected

Certain data types, such as [enums](#), can only be declared to hold data of specified types.

The following sample generates CS1008:

C#

```
// CS1008.cs
abstract public class clk
{
    enum splitch : char // CS1008, char not valid type for enums
    {
        x, y, z
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1009

Article • 09/15/2021 • 2 minutes to read

## Unrecognized escape sequence

An unexpected character follows a backslash (\) in a [string](#). The compiler expects one of the valid escape characters. For more information, see [Character Escapes](#).

The following sample generates CS1009.

```
C#  
  
// CS1009-a.cs  
class MyClass  
{  
    static void Main()  
    {  
        // The following line causes CS1009.  
        string a = "\m";  
        // Try the following line instead.  
        // string a = "\t";  
    }  
}
```

A common cause of this error is using the backslash character in a file name, as the following example shows.

```
C#  
  
string filename = "c:\myFolder\myFile.txt";
```

To resolve this error, use "\\\" or the @-quoted string literal, as the following example shows.

```
C#  
  
// CS1009-b.cs  
class MyClass  
{  
    static void Main()  
    {  
        // The following line causes CS1009.  
        string filename = "c:\myFolder\myFile.txt";  
        // Try one of the following lines instead.  
        // string filename = "c:\\myFolder\\\\myFile.txt";  
        // string filename = @"c:\myFolder\myFile.txt";  
    }  
}
```

```
    }  
}
```

## See also

- [string](#)

# Compiler Error CS1010

Article • 09/15/2021 • 2 minutes to read

Newline in constant

A [string](#) was not properly delimited.

The following sample generates CS1010:

C#

```
// CS1010.cs
class Sample
{
    static void Main()
    {
        string a = "Hello World;    // CS1010
        // Use the following line, with end quote before semicolon:
        string a = "Hello World"; //
    }
}
```

## See also

- [Strings \(C# Programming Guide\)](#)



# Compiler Error CS1011

Article • 09/15/2021 • 2 minutes to read

## Empty character literal

A `char` was declared and initialized to a null. The initialization of a `char` must specify a character.

The following sample generates CS1011:

C#

```
// CS1011.cs
class Sample
{
    public char CharField = '';    // CS1011
}
```

# Compiler Error CS1012

Article • 09/15/2021 • 2 minutes to read

Too many characters in character literal

An attempt was made to initialize a `char` constant with more than one character.

CS1012 can also occur when doing data binding. For example the following line will give an error:

```
<%# DataBinder.Eval(Container.DataItem, 'doctitle') %>
```

Try the following line instead:

```
<%# DataBinder.Eval(Container.DataItem, "doctitle") %>
```

The following sample generates CS1012:

```
C#  
  
// CS1012.cs  
class Sample  
{  
    static void Main()  
    {  
        char a = 'xx';    // CS1012  
        char a2 = 'x';   // OK  
        System.Console.WriteLine(a2);  
    }  
}
```

# Compiler Error CS1013

Article • 09/15/2021 • 2 minutes to read

## Invalid number

The compiler detected a malformed number. For more information on integral types, see the [Integral Types Table](#).

## Example

The following sample generates CS1013:

C#

```
// CS1013.cs
class Sample
{
    static void Main()
    {
        int i = 0x;    // CS1013
    }
}
```

# Compiler Error CS1014

Article • 09/15/2021 • 2 minutes to read

A get or set accessor expected

A method declaration was found in a property declaration. You can only declare `get` and `set` methods in a property.

For more information on properties, see [Using Properties](#).

## Example

The following sample generates CS1014.

```
C#  
  
// CS1014.cs  
// compile with: /target:library  
class Sample  
{  
    public int TestProperty  
    {  
        get  
        {  
            return 0;  
        }  
        int z; // CS1014 not get or set  
    }  
}
```

# Compiler Error CS1015

Article • 10/27/2021 • 2 minutes to read

An object, string, or class type expected

An attempt was made to pass a predefined data type into a `catch` block. Only data types that derive from `System.Exception` can be passed into a `catch` block. For more information on exceptions, see [Exceptions and Exception Handling](#).

## Example

The following sample generates CS1015:

C#

```
// CS1015.cs
class Sample
{
    static void Main()
    {
        try
        {
        }
        catch(int) // CS1015, int is not derived from System.Exception
        {
        }
    }
}
```

# Compiler Error CS1016

Article • 09/15/2021 • 2 minutes to read

Named attribute argument expected

Unnamed attribute arguments must appear before the named arguments.

## Example

The following sample generates CS1016:

C#

```
// CS1016.cs
using System;

[AttributeUsage(AttributeTargets.Class)]
public class HelpAttribute : Attribute
{
    public HelpAttribute(string url)    // url is a positional parameter
    {
        m_url = url;
    }

    public string Topic = null;        // Topic is a named parameter
    private string m_url = null;
}

[HelpAttribute(Topic="Samples", "http://intranet/inhouse")]    // CS1016
// try the following line instead
//[HelpAttribute("http://intranet/inhouse", Topic="Samples")]
public class MainClass
{
    public static void Main ()
    {
    }
}
```

# Compiler Error CS1017

Article • 10/27/2021 • 2 minutes to read

Catch clauses cannot follow the general catch clause of a try statement

A `catch` block that does not take any parameters must be the last in a series of `catch` blocks. For more information on exceptions, see [Exceptions and Exception Handling](#).

## Example

The following sample generates CS1017:

C#

```
// CS1017.cs
using System;

namespace x
{
    public class b : Exception
    {
    }

    public class a
    {
        public static void Main()
        {
            try
            {

                catch // CS1017, must be last catch
                {
                }

                catch(b)
                {
                    throw;
                }
            }
        }
    }
}
```

# Compiler Error CS1018

Article • 09/15/2021 • 2 minutes to read

Keyword 'this' or 'base' expected

The compiler encountered an incomplete constructor declaration.

## Example

The following example generates CS1018, and suggests several ways to resolve the error:

C#

```
// CS1018.cs
public class C
{
}

public class a : C
{
    public a(int i)
    {

    }

    public a () : // CS1018
    // possible resolutions:
    // public a () resolves by removing the colon
    // public a () : base() calls C's parameterless constructor
    // public a () : this(1) calls the assignment constructor of class a
    {

    }

    public static int Main()
    {
        return 1;
    }
}
```

# Compiler Error CS1019

Article • 09/15/2021 • 2 minutes to read

## Overloadable unary operator expected

Something that looks like an overloaded unary operator has been declared, but the operator is missing or is in the wrong location in the signature.

A *unary operator* is an operator that operates on a single operand. For example, `++` is a unary operator. You can overload some unary operators by using the `operator` keyword and specifying a single parameter of the type that the operator operates on. For example, if you want to overload the operator `++` for a user-defined class `Temperature` so that you can write `Temperature++`, you can define it in this way:

C#

```
public static Temperature operator ++ (Temperature temp)
{
    temp.Degrees++;
    return temp;
}
```

When you receive this error, you have declared something that looks like an overloaded unary operator, except that the operator itself is missing or is in the wrong location in the signature. If you remove the `++` from the signature in the previous example, you will generate CS1019.

The following code generates CS1019:

C#

```
// CS1019.cs
public class ii
{
    int i
    {
        get
        {
            return 0;
        }
    }
}

public class a
{
    public int i;
```

```
// Generates CS1019: "ii" is not a unary operator.  
public static a operator ii(a aa)  
  
    // Use the following line instead:  
    //public static a operator ++(a aa)  
    {  
        aa.i++;  
        return aa;  
    }  
  
public static void Main()  
{  
}  
}
```

## See also

- [C# operators](#)

# Compiler Error CS1020

Article • 09/15/2021 • 2 minutes to read

Overloadable binary operator expected

An attempt was made to define an operator overload, but the operator was not an overloadable binary operator, which takes two parameters. For the list of overloadable operators, see the [Overloadable operators](#) section of the [Operator overloading](#) article.

The following sample generates CS1020:

C#

```
// CS1020.cs
public class iii
{
    public static int operator ++(iii aa, int bb)    // CS1020, change ++ to +
    {
        return 0;
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1021

Article • 09/15/2021 • 2 minutes to read

Integral constant is too large

A value represented by an integer literal is greater than [UInt64.MaxValue](#).

The following sample generates CS1021:

C#

```
// CS1021.cs
class Program
{
    static void Main(string[] args)
    {
        int a = 18_446_744_073_709_552_000;
    }
}
```

The following code also generates CS1021:

C#

```
using System.Numerics;

class Program
{
    static void Main(string[] args)
    {
        var a = new BigInteger(18_446_744_073_709_552_000);
    }
}
```

For information about how to instantiate a [System.Numerics.BigInteger](#) instance whose value exceeds the range of the built-in numeric types, see the [Instantiating a BigInteger Object](#) section of the [BigInteger](#) reference page.

# Compiler Error CS1022

Article • 09/15/2021 • 2 minutes to read

Type or namespace definition, or end-of-file expected

A source-code file does not have a matching set of braces.

The following sample generates CS1022:

C#

```
// CS1022.cs
namespace x
{
}
} // CS1022
```

# Compiler Error CS1023

Article • 09/15/2021 • 2 minutes to read

Embedded statement cannot be a declaration or labeled statement

An embedded statement, such as the statements following an `if` statement, can contain neither declarations nor labeled statements.

The following sample generates CS1023 twice:

C#

```
// CS1023.cs
public class a
{
    public static void Main()
    {
        if (1)
            int i;      // CS1023, declaration is not valid here

        if (1)
            xx : i++; // CS1023, labeled statement is not valid here
    }
}
```

# Compiler Error CS1024

Article • 09/15/2021 • 2 minutes to read

Preprocessor directive expected

A line began with the pound symbol (#), but the subsequent string was not a valid [preprocessor directive](#).

The following sample generates CS1024:

C#

```
// CS1024.cs
#import System // CS1024
```

# Compiler Error CS1025

Article • 09/15/2021 • 2 minutes to read

Single-line comment or end-of-line expected

A line with a [preprocessor directive](#) cannot have a multiline comment.

The following sample generates CS1025:

C#

```
#if true /* hello
*/
// CS1025
#endif // this is a good comment
```

CS1025 could also occur if you attempt some invalid preprocessor directive, as follows:

C#

```
// CS1025.cs
#define a

class Sample
{
    static void Main()
    {
        #if a 1 // CS1025, invalid syntax
            System.Console.WriteLine("Hello, World!");
        #endif
    }
}
```

# Compiler Error CS1026

Article • 09/15/2021 • 2 minutes to read

) expected

An incomplete statement was found.

A common cause of this error is placing a statement, rather than an expression, within an inline expression in an ASP.NET page. For example, the following is incorrect:

ASP.NET (C#)

```
<%=new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days;%>
```

The following is correct:

ASP.NET (C#)

```
<%=new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days %>
```

It is interpreted as follows:

ASP.NET (C#)

```
<% Response.Write(new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days); %>
```

The following example generates CS1026:

C#

```
// CS1026.cs
#if (a == b    // CS1026, add closing )
#endif

class x
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS1027

Article • 09/15/2021 • 2 minutes to read

#endif directive expected

A matching `#endif` [preprocessor directive](#) was not found for a specified `#if` directive. Or, the compiler may have found a `#endregion` directive when there was no matching `#region` directive inside a `#if` block.

The following sample generates CS1027:

C#

```
// CS1027.cs
#if true    // CS1027, uncomment next line to resolve
// #endif

namespace x
{
    public class clk
    {
        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS1028

Article • 09/15/2021 • 2 minutes to read

Unexpected preprocessor directive

A [preprocessor directive](#) was found but not expected.

For example, a `#endif` was found with no preceding `#if`.

The following sample generates CS1028:

C#

```
// CS1028.cs
#endif // CS1028, no matching #if
namespace x
{
    public class clk
    {
        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS1029

Article • 09/15/2021 • 2 minutes to read

```
#error: 'text'
```

Displays the text of an error defined with the `#error` directive.

The following sample shows how to create a user-defined error:

C#

```
// CS1029.cs
class Sample
{
    static void Main()
    {
        #error Let's give an error here // CS1029
    }
}
```

Compilation produces the following output:

Console

```
example.cs(9,8): error CS1029: #error: 'Let's give an error here' // CS1029
'
```

# Compiler Error CS1031

Article • 09/15/2021 • 2 minutes to read

Type expected

A type parameter is expected.

## Example

The following sample generates CS1031:

```
C#  
  
// CS1031.cs  
namespace x  
{  
    public class ii  
    {  
    }  
  
    public class a  
    {  
        public static operator +(a aa)    // CS1031  
        // try the following line instead  
        // public static ii operator +(a aa)  
        {  
            return new ii();  
        }  
  
        public static void Main()  
        {  
            e = new base;    // CS1031, not a type  
            e = new this;   // CS1031, not a type  
            e = new ();      // CS1031, not a type  
        }  
    }  
}
```

# Compiler Error CS1032

Article • 09/15/2021 • 2 minutes to read

Cannot define/undefine preprocessor symbols after first token in file

The `#define` and `#undef` [preprocessor directives](#) must be used at the beginning of a program, before any other keywords, such as those used in the namespace declaration.

The following sample generates CS1032:

C#

```
// CS1032.cs
namespace x
{
    public class clk
    {
        #define a    // CS1032, put before namespace
        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS1033

Article • 09/15/2021 • 2 minutes to read

Source file has exceeded the limit of 16,707,565 lines representable in the PDB; debug information will be incorrect

A source-code file exceeded the maximum allowable number of lines that the compiler can process. To resolve this error, create two or more source-code files from the original file. The maximum number of lines is 268,435,454 lines. If you are using `/debug`, using more than 16,707,566 will result in corrupted debug information.

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Error CS1034

Article • 09/15/2021 • 2 minutes to read

Compiler limit exceeded: Line cannot exceed 'number' characters

The limit for the number of characters allowed on a line is 16,777,214.

# Compiler Error CS1035

Article • 09/15/2021 • 2 minutes to read

End-of-file found, '\*' expected

An opening comment delimiter was not matched with a closing delimiter.

The following sample generates CS1035:

C#

```
// CS1035.cs
public class a
{
    public static void Main()
    {
    }
}
/* // CS1035, needs closing comment
```

# Compiler Error CS1036

Article • 09/15/2021 • 2 minutes to read

( or . expected

The XML in a [DocumentationFile](#) comment was badly formed.

# Compiler Error CS1037

Article • 09/15/2021 • 2 minutes to read

Overloadable operator expected

When specifying a comment with [DocumentationFile](#), the compiler encountered an invalid link.

# Compiler Error CS1038

Article • 09/15/2021 • 2 minutes to read

#endregion directive expected

A [#region](#) directive did not have a matching [#endregion](#) directive.

The following sample generates CS1038:

C#

```
// CS1038.cs
#region testing

public class clk
{
    public static void Main()
    {
    }
}
// CS1038
// uncomment the next line to resolve
// #endregion
```

# Compiler Error CS1039

Article • 09/15/2021 • 2 minutes to read

Unterminated string literal

The compiler detected an ill-formed [string](#) literal.

## Example

The following sample generates CS1039. To resolve the error, add the terminating quotation mark.

C#

```
// CS1039.cs
public class MyClass
{
    public static void Main()
    {
        string b = @"hello, world; // CS1039
    }
}
```

# Compiler Error CS1040

Article • 09/15/2021 • 2 minutes to read

Preprocessor directives must appear as the first non-white-space character on a line

A [preprocessor directive](#) was found on a line and was not the first token on the line. A directive must be the first token on the line.

The following sample generates CS1040:

C#

```
// CS1040.cs
/* Define a symbol, X */ #define X    // CS1040

// try the following two lines instead
// /* Define a symbol, X */
// #define X

public class MyClass
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS1041

Article • 09/15/2021 • 2 minutes to read

Identifier expected, 'keyword' is a keyword

A reserved word for the C# language was found where an identifier was expected.

Replace the [keyword](#) with a user-specified identifier.

## Example 1

The following sample generates CS1041:

```
C#  
  
// CS1041a.cs  
class MyClass  
{  
    public void f(int long)    // CS1041  
    // Try the following instead:  
    // public void f(int i)  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

## Example 2

When you are importing from another programming language that does not have the same set of reserved words, you can modify the reserved identifier with the @ prefix, as demonstrated in the following sample.

An identifier with an @ prefix is called a verbatim identifier.

```
C#  
  
// CS1041b.cs  
class MyClass  
{  
    public void f(int long)    // CS1041  
    // Try the following instead:  
    // public void f(int @long)  
    {
```

```
}

public static void Main()
{
}

}
```

# Compiler Error CS1043

Article • 09/15/2021 • 2 minutes to read

{ or ; expected

A property accessor was declared incorrectly. For more information, see [Using Properties](#).

## Example

The following sample generates CS1043.

C#

```
// CS1043.cs
// compile with: /target:library
public class MyClass
{
    public int DoSomething
    {
        get return 1;    // CS1043
        set {}
    }

    // OK
    public int DoSomething2
    {
        get { return 1;}
    }
}
```

# Compiler Error CS1044

Article • 09/15/2021 • 2 minutes to read

Cannot use more than one type in a for, using, fixed, or declaration statement

The compiler found an invalid statement.

The following sample generates CS1044:

C#

```
// CS1044.cs
using System;

public class MyClass : IDisposable
{
    public void Dispose()
    {
        Console.WriteLine("Res1.Dispose()");
    }

    public static void Main()
    {
        using (MyClass mc1 = new MyClass(),
              MyClass mc2 = new MyClass()) // CS1044, remove an
instantiation
        {
        }
    }
}
```

# Compiler Error CS1055

Article • 09/15/2021 • 2 minutes to read

An add or remove accessor expected

If your `event` is not declared as a field, it must define both `add` and `remove` accessor functions.

The following sample generates CS1055:

C#

```
// CS1055.cs
delegate void del();
class Test
{
    public event del MyEvent
    {
        int i; // CS1055
        // uncomment accessors and delete previous line to resolve
        // add
        // {
        //     MyEvent += value;
        // }
        // remove
        // {
        //     MyEvent -= value;
        // }
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1056

Article • 09/15/2021 • 2 minutes to read

Unexpected character 'character'

The C# compiler encountered an unexpected character, and is unable to identify the token currently being processed. For example, if the compiler encounters a Euro-character in the middle of processing an identifier, it will be unable to classify the identifier, since a Euro-character would only be valid inside a string, and the compiler knows it is not processing a string.

# Compiler Error CS1057

Article • 09/15/2021 • 2 minutes to read

'member': static classes cannot contain protected members

This error is generated by declaring a protected member inside a static class.

## Example

The following example generates CS1057.

```
C#  
  
// CS1057.cs  
using System;  
  
static class Class1  
{  
    protected static int x;    // CS1057  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS1059

Article • 09/15/2021 • 2 minutes to read

The operand of an increment or decrement operator must be a variable, property or indexer.

This error is raised when you try to increment or decrement a constant value. It can also occur if you try to increment an expression such as `(a+b)++`.

## To correct this error

- Make the variable non-const.
- Remove the increment or decrement operator.
- Store the expression in a variable, and then increment the variable.

## Example

The following example generates CS1059 because `i` is a constant, not a variable, and `E` is an `Enum` type, whose elements are also constant values.

C#

```
// CS1059.cs
class Program
{
    public enum E : sbyte
    {
        a = 1,
        b = 2
    }

    static void Main(string[] args)
    {
        const int i = 0;
        i++;           // CS1059
        E test = E.a++; // CS1059
    }
}
```

## See also

- Constants

# Compiler Error CS1061

Article • 10/27/2021 • 2 minutes to read

'type' does not contain a definition for 'name' and no accessible extension method 'name' accepting a first argument of type 'type' could be found (are you missing a using directive or an assembly reference?).

This error occurs when you try to call a method or access a class member that does not exist.

## Example

The following example generates CS1061 because `Person` does not have a `DisplayName` method. It does have a method that is called `WriteName`. Perhaps that is what the author of this source code meant to write.

C#

```
public class Person
{
    private string _name;

    public Person(string name) => _name = name;

    // Person has one method, called WriteName.
    public void WriteName()
    {
        System.Console.WriteLine(_name);
    }
}

public class Program
{
    public static void Main()
    {
        var p = new Person("PersonName");

        // The following call fails because Person does not have
        // a method called DisplayName.
        p.DisplayName(); // CS1061
    }
}
```

## To correct this error

1. Make sure you typed the member name correctly.
2. If you have access to modify this class, you can add the missing member and implement it.
3. If you don't have access to modify this class, you can add an [extension method](#).

## See also

- [The C# type system](#)
- [Extension Methods](#)

# Compiler Error CS1063

Article • 10/07/2022 • 2 minutes to read

The best overloaded `Add` method for the collection initializer element is obsolete.

This error occurs when a type that implements `IEnumerable` implements one or more `Add` methods but the best matching overload is attributed with `ObsoleteAttribute` and the `ObsoleteAttribute.IsError` property is initialized to `true`.

## Example

The following sample generates CS1063:

```
C#  
  
// CS1063.cs (9,38)  
using System;  
using System.Collections;  
  
class Test  
{  
    public static void Main()  
    {  
        B coll = new B { "a" };  
    }  
}  
  
public class B : IEnumerable  
{  
    [Obsolete("Don't use this overload", true)]  
    public void Add(string s)  
    {  
        //...  
    }  
    public void Add(int i)  
    {  
        //...  
    }  
    IEnumerator IEnumerable.GetEnumerator()  
    {  
        return null;  
    }  
}
```

## To correct this error

If you own the enumerable type code and want to utilize a collection initializer, remove the `ObsoleteAttribute`.

C#

```
public class B : IEnumerable
{
    public void Add(string s)
    {
        //...
    }
    public void Add(int i)
    {
        //...
    }
    IEnumerator IEnumerable.GetEnumerator()
    {
        return null;
    }
}
```

If you do not own the enumerable type code, you will have to choose another means to initialize the collection.

# Compiler Error CS1065

Article • 09/20/2022 • 2 minutes to read

Default values are not valid in this context.

Default values are used when declaring optional arguments. Optional arguments are not supported when declaring an anonymous method with the delegate operator. The delegate operator creates an anonymous method.

## Example

The following sample generates CS1065:

```
C#  
  
// CS1065.cs (5,27)  
  
class A  
{  
    delegate void D(int x);  
    D d1 = delegate(int x = 42) { };  
}
```

## To correct this error

Removing the default value corrects this error:

```
C#  
  
class A  
{  
    delegate void D(int x);  
    D d1 = delegate(int x) { };  
}
```

# Compiler Error CS1100

Article • 09/15/2021 • 2 minutes to read

Method 'name' has a parameter modifier 'this' which is not on the first parameter.

The `this` modifier is allowed only on the first parameter of a method, which indicates to the compiler that the method is an extension method.

## To correct this error

1. Remove the `this` modifier from all except the first parameter of the method.

## Example

The following code generates CS1100 because a `this` parameter is modifying the second parameter:

```
C#  
  
// cs1100.cs  
static class Test  
{  
    static void ExtMethod(int i, this Test c) // CS1100  
    {  
    }  
}
```

## See also

- [Extension Methods](#)



# Compiler Error CS1101

Article • 09/15/2021 • 2 minutes to read

The parameter modifier 'ref' cannot be used with 'this'.

When the `this` keyword modifies the first parameter of a static method, it signals to the compiler that the method is an extension method. No other modifiers are needed or allowed on the first parameter of an extension method.

## Example

The following example generates CS1101:

```
C#  
  
// cs1101.cs  
// Compile with: /target:library  
public static class Extensions  
{  
    // No type parameters.  
    public static void Test(ref this int i) {} // CS1101  
  
    // Single type parameter.  
    public static void Test<T>(ref this T t) {} // CS1101  
  
    // Multiple type parameters.  
    public static void Test<T,U,V>(ref this U u) {} // CS1101  
}
```

## See also

- [Extension Methods](#)
- [this](#)
- [ref](#)



# Compiler Error CS1102

Article • 09/15/2021 • 2 minutes to read

The parameter modifier 'out' cannot be used with 'this'.

When the `this` keyword modifies the first parameter of a static method, it signals to the compiler that the method is an extension method. No other modifiers are needed or allowed on the first parameter of an extension method.

## To correct this error

1. Remove the unauthorized modifiers from the first parameter.

## Example

The following example generates CS1102:

```
C#  
  
// cs1102.cs  
// Compile with: /target:library.  
public static class Extensions  
{  
    // No type parameters.  
    public static void Test(this out int i) {} // CS1102  
  
    //Single type parameter  
    public static void Test<T>(this out T t) {}// CS1102  
  
    //Multiple type parameters  
    public static void Test<T,U,V>(this out U u) {}// CS1102  
}
```

## See also

- [Extension Methods](#)
- [this](#)
- [out](#)



# Compiler Error CS1103

Article • 09/15/2021 • 2 minutes to read

The first parameter of an extension method cannot be of type 'type'.

The first parameter of an extension method cannot be a pointer type.

## Example

The following example generates CS1103:

C#

```
// cs1103.cs
public static class Extensions
{
    public unsafe static char* Test(this char* charP) { return charP; } // CS1103
}
```

## See also

- [Extension Methods](#)
- [unsafe](#)



# Compiler Error CS1104

Article • 09/15/2021 • 2 minutes to read

A parameter array cannot be used with 'this' modifier on an extension method.

The first parameter of an extension method cannot be a params array.

## To correct this error

1. Remember that the first parameter of an extension method definition specifies which type the method will "extend". It is not an input parameter. Therefore, it makes no sense to have a params array in this location. If you do have to pass in a params array, make it the second parameter.

## Example

The following example generates CS1104:

```
C#  
  
// cs1104.cs  
// Compile with: /target:library  
public static class Extensions  
{  
    public static void Test<T>(this params T[] tArr) {} // CS1104  
}
```

## See also

- [Extension Methods](#)
- [params](#)



# Compiler Error CS1105

Article • 09/15/2021 • 2 minutes to read

Extension methods must be static.

Extension methods must be declared as static methods in a non-generic static class.

## Example

The following example generates CS1105 because `Test` is not static:

C#

```
// cs1105.cs
// Compile with: /target:library
public class Extensions
{
    // Single type parameter.
    public void Test<T>(this System.String s) {} //CS1105
}
```

## See also

- [Extension Methods](#)
- [static](#)



# Compiler Error CS1106

Article • 05/20/2022 • 2 minutes to read

Extension methods must be defined in a non generic static class.

Extension methods must be defined as static methods in a non-generic static class.

## Example

The following example generates CS1106 because the class `Extensions` is not defined as

`static`:

C#

```
// cs1106.cs
public class Extensions // CS1106
{
    public static void Test<T>(this System.String s) { }
```

## See also

- [Extension Methods](#)
- [static](#)



# Compiler Error CS1107

Article • 07/07/2022 • 2 minutes to read

A parameter can only have one 'modifier name' modifier.

It is an error for parameter modifiers such as `this`, `ref`, `in`, and `out` to appear more than one time in a parameter definition.

## Example

The following example generates CS1107:

C#

```
// cs1107.cs
public static class Test
{
    // Extension methods.
    public static void TestMethod(this this int t) { } // CS1107

}

public class TestTwo
{
    // Regular Instance Method
    public void TestMethod(ref ref int i) { } // CS1107

    // Regular Instance Method
    public void TestMethod(in in double d) { } // CS1107
}
```

# Compiler Error CS1108

Article • 09/15/2021 • 2 minutes to read

A parameter cannot have all the specified modifiers; there are too many modifiers on the parameter.

Certain combinations of parameter modifiers, such as `in`, `ref`, and `out`, are not allowed because they have mutually exclusive meanings for the compiler.

## Example

The following example generates CS1108:

C#

```
// cs1108.cs
// Compile with: /target:library
public class Test
{
    // Regular Instance Method.
    public void TestMethod(ref out int i) {} // CS1108

}
```

# Compiler Error CS1109

Article • 09/15/2021 • 2 minutes to read

Extension Methods must be defined on top level static classes, 'name' is a nested class.

Extension methods cannot be defined in nested classes.

## Example

The following example generates CS1109 because the class `Extension` is nested inside the class `Out`:

C#

```
// cs1109.cs
public class Test
{
}
static class Out
{
    static class Extension
    {
        static void ExtMethod(this Test c) // CS1109
        {
        }
    }
}
```

## See also

- [Extension Methods](#)



# Compiler Error CS1110

Article • 09/15/2021 • 2 minutes to read

Cannot use 'this' modifier on first parameter of method declaration without a reference to System.Core.dll. Add a reference to System.Core.dll or remove 'this' modifier from the method declaration.

Extension methods are supported on version 3.5 and later of .NET Framework. Extension methods generate metadata which marks the method with an attribute. The attribute class is in system.core.dll.

## To correct this error

1. As the message states, add a reference to System.Core.dll or remove the `this` modifier from the method declaration.

## Example

The following example generates CS1110 if the file is not compiled with a reference to System.Core.dll:

```
C#  
  
// cs1110.cs  
// CS1110  
// Compile with: /target:library  
public static class Extensions  
{  
    public static bool Test(this bool b) { return b; }  
}
```

## See also

- [Extension Methods](#)



# Compiler Error CS1112

Article • 09/15/2021 • 2 minutes to read

Do not use 'System.Runtime.CompilerServices.ExtensionAttribute'. Use the 'this' keyword instead.

This error is generated when the [ExtensionAttribute](#) is used on a non-static class that contains extension methods. If this attribute is used on a static class, another error, such as CS0708: "Cannot declare instance members in a static class," might occur.

In C#, extension methods must be defined in a static class and the first parameter of the method is modified with the `this` keyword. Do not use the attribute at all in the source code. For more information, see [Extension Methods](#).

## To correct this error

1. Remove the attribute and apply the `this` modifier to the first parameter of the method.

## Example

The following example generates CS1112:

C#

```
// cs1112.cs
[System.Runtime.CompilerServices.ExtensionAttribute] // CS1112
public class Extensions
{
    public bool A(bool b) { return b; }
}

class A { }
```

# Compiler Error CS1113

Article • 09/15/2021 • 2 minutes to read

Extension methods 'name' defined on value type 'name' cannot be used to create delegates.

Extension methods that are defined for class types can be used to create delegates.  
Extension methods that are defined for value types cannot.

## To correct this error

1. Associate the extension method with a class type.
2. Make the method a regular method on the struct.

## Example

The following example generates CS1113:

```
C#  
  
// cs1113.cs  
using System;  
public static class Extensions  
{  
    public static S ExtMethod(this S s)  
    {  
        return s;  
    }  
}  
  
public struct S  
{  
}  
  
public class Test  
{  
    static int Main()  
    {  
        Func<S> f = new S().ExtMethod; // CS1113  
        return 1;  
    }  
}
```

## See also

- [Extension Methods](#)

# Compiler Error CS1501

Article • 09/15/2021 • 2 minutes to read

No overload for method 'method' takes 'number' arguments

A call was made to a class method, but no definition of the method takes the specified number of arguments.

## Example

The following sample generates CS1501.

C#

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            ExampleClass ec = new ExampleClass();
            ec.ExampleMethod();
            ec.ExampleMethod(10);
            // The following line causes compiler error CS1501 because
            // ExampleClass does not contain an ExampleMethod that takes
            // two arguments.
            ec.ExampleMethod(10, 20);
        }
    }

    // ExampleClass contains two overloads for ExampleMethod. One of them
    // has no parameters and one has a single parameter.
    class ExampleClass
    {
        public void ExampleMethod()
        {
            Console.WriteLine("Zero parameters");
        }

        public void ExampleMethod(int i)
        {
            Console.WriteLine("One integer parameter.");
        }

        //// To fix the error, you must add a method that takes two
        // arguments.
        //public void ExampleMethod (int i, int j)
```

```
//{
//    Console.WriteLine("Two integer parameters.");
//}
}
```

# Compiler Error CS1502

Article • 09/15/2021 • 2 minutes to read

The best overloaded method match for 'name' has some invalid arguments

This error occurs when the argument types being passed to the method do not match the parameter types of that method. If the called method is overloaded, then none of the overloaded versions has a signature that matches the argument types being passed.

To resolve this problem, do one of the following:

- Double-check the types of the arguments being passed. Make sure that they match the arguments of the method being called.
- If appropriate, convert any mismatched parameters using the [Convert](#) class.
- If appropriate, cast any mismatched parameters to match the type that the method is expecting.
- If appropriate, define another overloaded version of the method to match the parameter types that are being sent.

The following sample generates CS1502:

```
C#  
  
// CS1502.cs  
namespace x  
{  
    public class a  
    {  
        public a(char i)  
        // try the following constructor instead  
        // public a(int i)  
        {  
        }  
  
        public static void Main()  
        {  
            a aa = new a(2222);    // CS1502  
        }  
    }  
}
```

# Compiler Error CS1503

Article • 09/15/2021 • 2 minutes to read

Argument 'number' cannot convert from TypeA to TypeB

The type of one argument in a method does not match the type that was passed when the class was instantiated. This error typically appears along with CS1502. See [CS1502](#) for a discussion of how to resolve this error.

# Compiler Error CS1504

Article • 09/15/2021 • 2 minutes to read

Source file 'file' could not be opened ('reason')

A source file could not be opened or read by the compiler. The file may be locked by another application, or there could be some other operating system problem. The message includes the operating system's reason for why the compiler could not open or read the file.

# Compiler Error CS1507

Article • 09/15/2021 • 2 minutes to read

Cannot link resource file 'file' when building a module

[LinkResources](#) was used in the same compilation with the **module** option on the [TargetType](#), which is not allowed. For example, the following options would generate CS1507:

Console

```
csc /linkresource:rf.resource /target:module in.cs
```

Embedding resources ([Resources](#)), however, is allowed.

# Compiler Error CS1508

Article • 09/15/2021 • 2 minutes to read

Resource identifier 'identifier' has already been used in this assembly

In a compilation, the same identifier (*identifier*) was passed to two or more [Resources](#) or [LinkResources](#) compiler options.

For example, the following options would generate CS1508:

Console

```
/resource:anyfile.bmp,DuplicateIdent /linkresource:a.bmp,DuplicateIdent
```

# Compiler Error CS1509

Article • 09/15/2021 • 2 minutes to read

Referenced file 'file' is not an assembly; use **AddModules** option instead

An output file (output file 1), produced in a compilation that used the **module** element of the **TargetType** (does not have an assembly manifest), was specified to **References**. So, rather than appending an assembly to the assembly for the current program, the metadata information in output file 1 will be added to the assembly for the current program.

# Compiler Error CS1510

Article • 09/15/2021 • 2 minutes to read

A ref or out argument must be an assignable variable

Only a variable can be passed as a `ref` parameter in a method call. A `ref` value is like passing a pointer.

## Example

The following sample generates CS1510:

C#

```
// CS1510.cs
public class C
{
    public static int j = 0;

    public static void M(ref int j)
    {
        j++;
    }

    public static void Main ()
    {
        M (ref 2);    // CS1510, can't pass a number as a ref parameter
        // try the following to resolve the error
        // M (ref j);
    }
}
```

# Compiler Error CS1511

Article • 09/15/2021 • 2 minutes to read

Keyword 'base' is not available in a static method

The `base` keyword was used in a `static` method. `base` can only be called in an instance constructor, instance method, or instance accessor.

## Example

The following sample generates CS1511.

C#

```
// CS1511.cs
// compile with: /target:library
public class A
{
    public int j = 0;
}

class C : A
{
    public void Method()
    {
        base.j = 3;    // base allowed here
    }

    public static int StaticMethod()
    {
        base.j = 3;    // CS1511
        return 1;
    }
}
```

# Compiler Error CS1512

Article • 09/15/2021 • 2 minutes to read

Keyword 'base' is not available in the current context

The `base` keyword was used outside of a method, property, or constructor.

The following example generates CS1512:

C#

```
// CS1512.cs
using System;

class Base {}

class CMyClass : Base
{
    private String xx = base.ToString();    // CS1512
    // Try putting this initialization in the constructor instead:
    // public CMyClass()
    // {
    //     xx = base.ToString();
    // }

    public static void Main()
    {
        CMyClass z = new CMyClass();
    }
}
```

# Compiler Error CS1513

Article • 09/15/2021 • 2 minutes to read

} expected

The compiler expected a closing curly brace (}) that was not found.

The following sample generates CS1513:

C#

```
// CS1513
namespace y    // CS1513, no close curly brace
{
    class x
    {
        public static void Main()
        {
        }
    }
}
```

# Compiler Error CS1514

Article • 09/15/2021 • 2 minutes to read

{ expected

The compiler expected an opening curly brace ({) that was not found.

The following sample generates CS1514:

C#

```
// CS1514.cs
namespace x
// CS1514, no open curly brace
```

# Compiler Error CS1515

Article • 09/15/2021 • 2 minutes to read

'in' expected

In a `foreach`, `in` statement, the "in" part is missing.

## Example

The following sample generates CS1515:

```
C#  
  
using System;  
  
class Driver  
{  
    static void Main()  
    {  
        int[] arr = new int[] {1, 2, 3};  
  
        // try the following line instead  
        // foreach (int x in arr)  
        foreach (int x arr)      // CS1515, "in" is missing  
        {  
            Console.WriteLine(x);  
        }  
    }  
}
```

# Compiler Error CS1517

Article • 09/15/2021 • 2 minutes to read

## Invalid preprocessor expression

The compiler encountered an invalid preprocessor expression.

For more information, see [Preprocessor Directives](#).

The following sample shows some valid and invalid preprocessor expressions:

C#

```
// CS1517.cs
#if symbol      // OK
#endif
#if !symbol     // OK
#endif
#if (symbol)    // OK
#endif
#if true        // OK
#endif
#if false       // OK
#endif
#if 1           // CS1517
#endif
#if ~symbol     // CS1517
#endif
#if *           // CS1517
#endif

class x
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS1518

Article • 09/15/2021 • 2 minutes to read

Expected class, delegate, enum, interface, or struct

A declaration was found that is not supported in a [namespace](#). Inside a namespace, the compiler accepts only classes, structs, enums, interfaces, namespaces, and delegates.

## Example

The following sample generates CS1518:

C#

```
// CS1518.cs
namespace x
{
    sealed class c1 {};          // OK
    namespace f2 {};              // OK
    sealed f3 {};                // CS1518
}
```

# Compiler Error CS1519

Article • 09/15/2021 • 2 minutes to read

Invalid token 'token' in class, struct, or interface member declaration

This error is generated whenever a token is encountered in a location where it does not belong. A *token* is a keyword; an identifier (the name of a class, struct, method, and so on); a string, character, or numeric literal value such as 108, "Hello", or 'A'; or an operator or punctuator such as `==` or `;`.

Any [class](#), struct, or interface member declaration that contains invalid modifiers before the type will generate this error. To fix the error, remove the invalid modifiers.

The following sample generates CS1519 in five places because tokens are placed in locations where they are not valid:

C#

```
// CS1519.cs
// Generates CS1519 because a class name cannot be a number:
class Test 42
{
    // Generates CS1519 because of 'j' following 'I'
    // with no comma between them:
    int i j;
    // Generates CS1519 because of "checked" on void method:
    checked void f4();

    // Generates CS1519 because of "num":
    void f5(int a num){}

    // Generates CS1519 because of namespace inside class:
    namespace;

}
```

## See also

- [Classes](#)
- [Structure types](#)
- [Interfaces](#)
- [Methods](#)



# Compiler Error CS1520

Article • 07/06/2022 • 2 minutes to read

Method must have a return type

A method that is declared in a class, struct, or interface must have an explicit return type. In the following example, the `IntToString` method has a return value of `string`:

C#

```
class Test
{
    string IntToString(int i)
    {
        return i.ToString();
    }
}
```

The following sample generates CS1520:

C#

```
public class x
{
    // Method declaration missing a return type before the name of MyMethod
    // Note: the method is empty for the purposes of this example so as to
    // not add confusion.
    MyMethod() { }
}
```

And can be fixed by adding a return type to the method, such as adding `void` in the example below:

C#

```
public class x
{
    // MyMethod no longer throws an error, because it has a return type --
    // "void" in this case.
    void MyMethod() { }
}
```

Alternatively, this error might be encountered when the case of a constructor's name differs from that of the class or struct declaration, as in the following sample. Because the name is not exactly the same as the class name, the compiler interprets it as a regular method, not a constructor, and produces the error:

C#

```
public class Class1
{
    // Constructor should be called Class1, not class1
    public class1()    // CS1520
    {
    }
}
```

## See also

- [Methods](#)
- [Constructors](#)

# Compiler Error CS1521

Article • 09/15/2021 • 2 minutes to read

Invalid base type

A [base](#) class specification was ill formed.

The following sample generates CS1521:

```
C#  
  
// CS1521.cs  
class CMyClass  
{  
    public static void Main()  
    {  
    }  
}  
  
class CMyClass1 : CMyClass // OK  
{  
}  
  
class CMyClass2 : CMyClass[] // CS1521  
{  
}  
  
class CMyClass3 : CMyClass* // CS1521  
{  
}
```

# Compiler Error CS1524

Article • 10/27/2021 • 2 minutes to read

Expected catch or finally

A `try` block must be followed by a `catch` or `finally` block.

For more information on exceptions, see [Exceptions and Exception Handling](#).

## Example

The following sample generates CS1524:

C#

```
// CS1524.cs
class x
{
    public static void Main()
    {
        try
        {
            // Code here
        }
        catch
        {
        }
        try
        {
            // Code here
        }
        finally
        {
        }
        try
        {
            // Code here
        }
    }      // CS1524, missing catch or finally
}
```

# Compiler Error CS1525

Article • 09/15/2021 • 2 minutes to read

Invalid expression term 'character'

The compiler detected an invalid character in an expression.

The following sample generates CS1525:

C#

```
// CS1525.cs
class x
{
    public static void Main()
    {
        int i = 0;
        i = i + // OK - identifier
        'c' + // OK - character
        (5) + // OK - parenthesis
        [ + // CS1525, operator not a valid expression element
        throw + // CS1525, keyword not allowed in expression
        void; // CS1525, void not allowed in expression
    }
}
```

An empty label can also generate CS1525, as in the following sample:

C#

```
// CS1525b.cs
using System;
public class MyClass
{
    public static void Main()
    {
        goto FoundIt;
        FoundIt: // CS1525
        // Uncomment the following line to resolve:
        // System.Console.WriteLine("Hello");
    }
}
```

# Compiler Error CS1526

Article • 09/15/2021 • 2 minutes to read

A new expression requires (), [], or {} after type

The `new` operator, used to dynamically allocate memory for an object, was not specified correctly.

## Example

The following sample shows how to use `new` to allocate space for an array and an object.

C#

```
// CS1526.cs
public class y
{
    public static int i = 0;
    public int myi = 0;
}

public class z
{
    public static void Main()
    {
        y py = new y;    // CS1526
        y[] aoys = new y[10];   // Array of Ys

        for (int i = 0; i < aoys.Length; i++)
            aoys[i] = new y();   // an object of type y
    }
}
```

# Compiler Error CS1527

Article • 09/15/2021 • 2 minutes to read

Elements defined in a namespace cannot be explicitly declared as private, protected, protected internal or private protected.

Type declarations in a namespace can have either [public](#) or [internal](#) access. If no accessibility is specified, [internal](#) is the default.

The following sample generates CS1527:

```
C#  
  
// CS1527.cs  
namespace Sample  
{  
    private class C1 {}          // CS1527  
    protected class C2 {}        // CS1527  
    protected internal class C3 {} // CS1527  
    private protected class C4 {} // CS1527  
}
```

The following example generates CS1527 because when no namespace is explicitly declared in your program code, all type declarations are located implicitly within the global namespace.

```
C#  
  
//cs1527_2.cs  
using System;  
  
protected class C4 {}  
private struct S1 {}
```

## See also

- [Namespaces](#)
- [:: Operator](#)
- [Accessibility Domain](#)
- [Access Modifiers](#)



# Compiler Error CS1528

Article • 09/15/2021 • 2 minutes to read

Expected ; or = (cannot specify constructor arguments in declaration)

A reference to a class was formed as if an object to the class was being created. For example, there was an attempt to pass a variable to a constructor. Use the [new](#) operator to create an object of a class.

The following sample generates CS1528:

```
C#  
  
// CS1528.cs  
using System;  
  
public class B  
{  
    public B(int i)  
    {  
        _i = i;  
    }  
  
    public void PrintB()  
    {  
        Console.WriteLine(_i);  
    }  
  
    private int _i;  
}  
  
public class mine  
{  
    public static void Main()  
    {  
        B b(3); // CS1528, reference is not an object  
        // try one of the following  
        // B b;  
        // or  
        // B bb = new B(3);  
        // bb.PrintB();  
    }  
}
```

# Compiler Error CS1529

Article • 10/06/2022 • 2 minutes to read

A using clause must precede all other elements defined in the namespace except extern alias declarations

A [using](#) clause must appear first in a namespace.

## Example

The following sample generates CS1529:

```
C#  
  
// CS1529.cs  
namespace X  
{  
    namespace Subspace  
    {  
        using Microsoft;  
  
        class SomeClass  
        {  
        };  
  
        using Microsoft;      // CS1529, place before class definition  
    }  
  
    using System.Reflection; // CS1529, place before namespace 'Subspace'  
}  
  
using System;           // CS1529, place at the beginning of the file
```

# Compiler Error CS1530

Article • 09/15/2021 • 2 minutes to read

Keyword 'new' is not allowed on elements defined in a namespace

It is not necessary to specify the [new](#) keyword on any construct that is in a [namespace](#).

The following sample generates CS1530:

C#

```
// CS1530.cs
namespace a
{
    new class i    // CS1530
    {

        // try the following instead
        class ii
        {
            public static void Main()
            {
            }
        }
    }
}
```

# Compiler Error CS1534

Article • 09/15/2021 • 2 minutes to read

Overloaded binary operator 'operator' takes two parameters

The definition of a binary [operator](#) must take two parameters.

The following sample generates CS1534:

C#

```
// CS1534.cs
class MyClass
{
    public static MyClass operator - (MyClass MC1, MyClass MC2, MyClass MC3)
// CS1534
    // try the following line instead
    // public static MyClass operator - (MyClass MC1, MyClass MC2)
    {
        return new MyClass();
    }

    public static int Main()
    {
        return 1;
    }
}
```

# Compiler Error CS1535

Article • 09/15/2021 • 2 minutes to read

Overloaded unary operator 'operator' takes one parameter

The definition of a unary [operator](#) must take one parameter.

## Example

The following sample generates CS1535:

C#

```
// CS1535.cs
class MyClass
{
    // uncomment the method parameter to resolve CS1535
    public static MyClass operator ++ /*MyClass MC1*/ // CS1535
    {
        return new MyClass();
    }

    public static int Main()
    {
        return 1;
    }
}
```

# Compiler Error CS1536

Article • 09/15/2021 • 2 minutes to read

Invalid parameter type void

It is not necessary or valid to specify a [void](#) parameter other than a void pointer.

The following sample generates CS1536:

C#

```
// CS1536.cs
class a
{
    public static int x( void )    // CS1536
    // try the following line instead
    // public static int x()
    {
        return 0;
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1537

Article • 09/15/2021 • 2 minutes to read

The using alias 'alias' appeared previously in this namespace

You defined a symbol twice as an alias for a namespace. A symbol can only be defined once.

The following sample generates CS1537:

C#

```
// CS1537.cs
namespace x
{
    using System;
    using Object = System.Object;
    using Object = System.Object; // CS1537, delete this line to resolve
    using System = System;
}
```

# Compiler Error CS1540

Article • 09/15/2021 • 2 minutes to read

Cannot access protected member 'member' via a qualifier of type 'type1'; the qualifier must be of type 'type2' (or derived from it)

A derived `class` cannot access protected members of its base class through an instance of the base class. An instance of the base class declared in the derived class might, at run time, be an instance of another type that is derived from the same base but is not otherwise related to the derived class. Because protected members can be accessed only by derived types, any attempts to access protected members that might not be valid at run time are marked by the compiler as not valid.

In the `Employee` class in the following example, `emp2` and `emp3` both have type `Person` at compile time, but `emp2` has type `Manager` at run time. Because `Employee` is not derived from `Manager`, it cannot access the protected members of the base class, `Person`, through an instance of the `Manager` class. The compiler cannot determine what the run-time type of the two `Person` objects will be. Therefore, both the call from `emp2` and the call from `emp3` cause compiler error CS1540.

```
C#  
  
using System;  
using System.Text;  
  
namespace CS1540  
{  
    class Program1  
    {  
        static void Main()  
        {  
            Employee.PreparePayroll();  
        }  
    }  
  
    class Person  
    {  
        protected virtual void CalculatePay()  
        {  
            Console.WriteLine("CalculatePay in Person class.");  
        }  
    }  
  
    class Manager : Person  
    {  
        protected override void CalculatePay()  
        {  
            Console.WriteLine("CalculatePay in Manager class.");  
        }  
    }  
}
```

```
        Console.WriteLine("CalculatePay in Manager class.");

    }

}

class Employee : Person
{
    public static void PreparePayroll()
    {
        Employee emp1 = new Employee();
        Person emp2 = new Manager();
        Person emp3 = new Employee();
        // The following line calls the method in the Employee base
class,
        // Person.
        emp1.CalculatePay();

        // The following lines cause compiler error CS1540. The compiler
        // cannot determine at compile time what the run-time types of
        // emp2 and emp3 will be.
        //emp2.CalculatePay();
        //emp3.CalculatePay();

    }
}
}
```

## See also

- Inheritance
- Polymorphism
- Access Modifiers
- Abstract and Sealed Classes and Class Members

# Compiler Error CS1541

Article • 09/15/2021 • 2 minutes to read

Invalid reference option: 'symbol' — cannot reference directories

The compiler detected an attempt to specify a directory rather than a specific file. For example, when you use the [References](#) compiler option, you must specify a file; it is not possible to specify a directory.

For example, passing `/reference:c:\` to the compiler would generate CS1541.

# Compiler Error CS1542

Article • 09/15/2021 • 2 minutes to read

'dll' cannot be added to this assembly because it already is an assembly; use '/R' option instead

The file that was referenced with the [AddModules](#) compiler option was not built with **module** element of the [TargetType](#) compiler option; use [References](#) to reference the file in this compilation.

# Compiler Error CS1545

Article • 09/15/2021 • 2 minutes to read

Property, indexer, or event 'property' is not supported by the language; try directly calling accessor methods 'set accessor' or 'get accessor'

The code is consuming an object that has a non-default [indexer](#) and tried to use the indexed syntax. To resolve this error, call the property's `get` or `set` accessor method.

## Example 1

C++

```
// CPP1545.cpp
// compile with: /clr /LD
// a Visual C++ program
using namespace System;
public ref struct Employee {
    Employee( String^ s, int d ) {}

    property String^ name {
        String^ get() {
            return nullptr;
        }
    }
};

public ref struct Manager {
    property Employee^ Report [String^] {
        Employee^ get(String^ s) {
            return nullptr;
        }

        void set(String^ s, Employee^ e) {}
    }
};
```

## Example 2

The following sample generates CS1545.

C#

```
// CS1545.cs
// compile with: /r:CPP1545.dll
```

```
class x {
    public static void Main() {
        Manager Ed = new Manager();
        Employee Bob = new Employee("Bob Smith", 12);
        Ed.Report[ Bob.name ] = Bob;    // CS1545
        Ed.set_Report( Bob.name, Bob);  // OK
    }
}
```

# Compiler Error CS1546

Article • 09/15/2021 • 2 minutes to read

Property, indexer, or event 'property' is not supported by the language; try directly calling accessor method 'accessor'

Your code is consuming an object that has a default indexed property and tried to use the indexed syntax. To resolve this error, call the property's accessor method. For more information on indexers and properties, see [Indexers](#).

The following sample generates CS1546.

## Example 1

This code sample consists of a .cpp file, which compiles to a .dll, and a .cs file, which uses that .dll. The following code is for the .dll file and defines a property to be accessed by the code in the .cs file.

C++

```
// CPP1546.cpp
// compile with: /clr /LD
using namespace System;
public ref class MCPP
{
public:
    property int Prop [int,int]
    {
        int get( int i, int b )
        {
            return i;
        }
    }
};
```

## Example 2

This is the C# file.

C#

```
// CS1546.cs
// compile with: /r:CPP1546.dll
using System;
public class Test
```

```
{  
    public static void Main()  
    {  
        int i = 0;  
        MCPP mcpp = new MCPP();  
        i = mcpp.Prop(1,1); // CS1546  
        // Try the following line instead:  
        // i = mcpp.get_Prop(1,1);  
    }  
}
```

# Compiler Error CS1547

Article • 09/15/2021 • 2 minutes to read

Keyword 'void' cannot be used in this context

The compiler detected an invalid use of the `void` keyword.

The following sample generates CS1547:

C#

```
// CS1547.cs
public class MyClass
{
    void BadMethod()
    {
        void i;    // CS1547, cannot have variables of type void
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1548

Article • 09/15/2021 • 2 minutes to read

Cryptographic failure while signing assembly 'assembly' — 'reason'

CS1548 occurs when assembly signing fails. This is usually due to an invalid key file name, an invalid key file path, or a corrupt key file.

To fully sign an assembly, you must provide a valid key file that contains information about the public and private keys. To delay sign an assembly, you must select the **Delay sign only** check box and provide a valid key file that contains information about the public key information. The private key is not necessary when an assembly is delay-signed.

For more information, see [How to: Sign an Assembly with a Strong Name](#), [KeyFile \(C# Compiler Options\)](#) and [DelaySign \(C# Compiler Options\)](#).

When creating an assembly, the C# compiler calls into a utility called al.exe. If there is a failure in the assembly creation, the reason for the failure is reported by al.exe. See [Al.exe Tool Errors and Warnings](#) and search that topic for the text reported by the compiler in 'reason'.

## See also

- [How to: Sign an Assembly with a Strong Name](#)



# Compiler Error CS1551

Article • 09/15/2021 • 2 minutes to read

Indexers must have at least one parameter

An [indexer](#) that takes no arguments was declared.

The following sample generates CS1551:

C#

```
// CS1551.cs
public class MyClass
{
    int intI;

    int this[] // CS1551
    // try the following line instead
    // int this[int i]
    {
        get
        {
            return intI;
        }
        set
        {
            intI = value;
        }
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1552

Article • 09/15/2021 • 2 minutes to read

Array type specifier, [], must appear before parameter name

The position of the array type specifier is after the variable name in the array declaration.

## Example

The following sample generates CS1552:

C#

```
// CS1552.cs
public class C
{
    public static void Main(string args[])    // CS1552
    // try the following line instead
    // public static void Main(string [] args)
    {
    }
}
```

# Compiler Error CS1553

Article • 09/15/2021 • 2 minutes to read

Declaration is not valid; use 'modifier operator <dest-type> (...) instead

The return type for a [conversion operator](#) must immediately precede the parameter list, and *modifier* is either `implicit` or `explicit`.

The following sample generates CS1553:

C#

```
// CS1553.cs
class MyClass
{
    public static int implicit operator (MyClass f)    // CS1553
    // try the following line instead
    // public static implicit operator int (MyClass f)
    {
        return 6;
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1554

Article • 09/15/2021 • 2 minutes to read

Declaration is not valid; use '<type> operator op (...) instead

The return type of an [overloaded operator](#) must appear before the `operator` keyword.

The following sample generates CS1554:

C#

```
// CS1554.cs
class MyClass
{
    public static operator ++ MyClass (MyClass f)      // CS1554
    // try the following line instead
    // public static MyClass operator ++ (MyClass f)
    {
        return new MyClass ();
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1555

Article • 09/15/2021 • 2 minutes to read

Could not find 'class' specified for Main method

A class was specified to the [MainEntryPoint](#) compiler option, but the class name was not found in the source code.

# Compiler Error CS1556

Article • 09/15/2021 • 2 minutes to read

'construct' specified for Main method must be a valid class or struct

The [MainEntryPoint](#) compiler option was passed an identifier that was not a class name.

# Compiler Error CS1557

Article • 09/15/2021 • 2 minutes to read

Cannot use 'class' for Main method because it is in a different output file

The [MainEntryPoint](#) compiler option was specified for one output file in a multi-output file compilation. However, the class was not found in the source code for the /main compilation; it was found in a source code file for one of the other output files in the compilation.

# Compiler Error CS1558

Article • 09/15/2021 • 2 minutes to read

'class' does not have a suitable static Main method

The [MainEntryPoint](#) compiler option specified a class in which to look for a [Main](#) method. However, the [Main](#) method was not defined correctly.

The following example generates CS1558 because of invalid return type.

C#

```
// CS1558.cs
// compile with: /main:MyNamespace.MyClass

namespace MyNamespace
{
    public class MyClass
    {
        public static float Main()
        {
            return 0.0; // CS1558 because the return type is a float.
        }
    }
}
```

# Compiler Error CS1559

Article • 09/15/2021 • 2 minutes to read

Cannot use 'object' for Main method because it is imported

An invalid class was specified to the [StartupObject](#) compiler option; the class cannot be used as a location for the [Main](#) method.

# Compiler Error CS1560

Article • 09/15/2021 • 2 minutes to read

Invalid filename specified for preprocessor directive. Filename is too long or not a valid filename

The file name that was specified with `#line` exceeded `_MAX_PATH` (256 characters) or the line on which `#line` was found exceeded 2000 characters.

## Example

The following sample generates CS1560.

C#

```
// cs1560.cs
using System;
class MyClass
{
    public static void Main()
    {
        Console.WriteLine("Normal line #1.");
        #line 21
"MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile12345
67890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1
234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyF
ile1234567890MyFile1234567890.txt"    // CS1560
    }
}
```

# Compiler Error CS1561

Article • 09/15/2021 • 2 minutes to read

Output filename is too long or invalid

The output file name cannot be longer than 256 characters, and may not contain invalid characters, such as the Euro symbol, a question mark, or a backslash, among others.

# Compiler Error CS1562

Article • 09/15/2021 • 2 minutes to read

Outputs without source must have the /out option specified

The compilation could create an output file, but there was no source code file as input from which the name of the output file could be implied. For example, you may be trying to compile a metadata- or resource-only file.

Use the [OutputAssembly](#) compiler option to specify the name of the output file.

# Compiler Error CS1563

Article • 09/15/2021 • 2 minutes to read

Output 'output file' does not have any source files

The [OutputAssembly](#) compiler option was specified, but no source code files follow. You should either not use [OutputAssembly](#), or you should specify the source code files following [OutputAssembly](#).

# Compiler Error CS1564

Article • 09/15/2021 • 2 minutes to read

Conflicting options specified: Win32 resource file; Win32 manifest.

If you use the **/Win32res** compiler option, you must include the custom Win32 manifest (if it is required) in the resource file. You cannot provide a custom Win32 manifest separately from a Win32 resource file. Use the **/win32manifest** option only if you are not specifying a win32 resource file.

## To correct this error

1. Add the win32 manifest to the win32 resource file and remove the **/win32manifest** compiler option.

## Example

The following example produces CS1564 if it is compiled with the **/Win32res** option and no manifest is included in the resource file.

```
C#  
  
// cs1564.cs  
// Compile with: /Win32res  
public class Test  
{  
    static int Main(string[] args)  
    {  
        return 1;  
    }  
}
```

The C# 3.0 compiler adds a default win32Manifest to all binaries.

## See also

- [Win32Manifest \(C# Compiler Options\)](#)
- [Win32Resource \(C# Compiler Options\)](#)



# Compiler Error CS1565

Article • 09/15/2021 • 2 minutes to read

Conflicting options specified: Win32 resource file; Win32 icon

It is not valid to specify both the [Win32Resource](#) compiler option and the [Win32Icon](#) compiler option in the same compilation.

# Compiler Error CS1566

Article • 09/15/2021 • 2 minutes to read

Error reading resource file 'file' — 'reason'

The compiler had trouble with the file name passed to the [Resources](#) compiler option.

# Compiler Error CS1567

Article • 09/15/2021 • 2 minutes to read

Error generating Win32 resource: 'file'

Your compilation either used the [Win32Icon](#) compiler option or did not use [Win32Resource](#), which causes the compiler to generate a file that contains resource information, but the compiler was unable to create the file due to insufficient disk space or some other error.

If you are unable to resolve the file-generation problem, you could use [Win32Resource](#), which does not generate a file that contains resource information.

# Compiler Error CS1569

Article • 09/15/2021 • 2 minutes to read

Error generating XML documentation file 'Filename' ('reason')

When attempting to write the XML documentation to the file named in the message, an error occurred for the reason specified. The reason may be something like "network drive not found," or "access denied." Often, the reason will suggest what needs to be done to correct the error. For example, if the error says "access denied," you would verify that you have write permission on the file.

## Example 1

```
C#  
  
// 1569a.cs  
// compile with: /doc:CS1569.xml  
// post-build command: attrib +r CS1569.xml  
class Test  
{  
    /// <summary>Test.</summary>  
    public static void Main() {}  
}
```

## Example 2

The previous sample generated an .xml file that was then set to read only. This sample attempts to write to the same file. The following sample generates CS1569.

```
C#  
  
// CS1569.cs  
// compile with: /doc:CS1569.xml  
// CS1569 expected  
class Test  
{  
    /// <summary>Test.</summary>  
    public static void Main() {}  
}
```

# Compiler Error CS1575

Article • 09/15/2021 • 2 minutes to read

A `stackalloc` expression requires `[]` after type

The size of the requested allocation, with `stackalloc`, must be specified in square brackets.

The following sample generates CS1575:

C#

```
// CS1575.cs
// compile with: /unsafe
public class MyClass
{
    unsafe public static void Main()
    {
        int *p = stackalloc int (30);    // CS1575
        // try the following line instead
        // int *p = stackalloc int [30];
    }
}
```

# Compiler Error CS1576

Article • 09/15/2021 • 2 minutes to read

The line number specified for #line directive is missing or invalid

The compiler detected an error with the value passed to the [#line](#) directive.

The following sample generates CS1576:

C#

```
// CS1576.cs
public class MyClass
{
    static void Main()
    {
        #line "abc.sc"          // CS1576
        // try the following line instead
        //#line 101 "abc.sc"
        intt i;   // error will be reported on line 101
    }
}
```

# Compiler Error CS1577

Article • 09/15/2021 • 2 minutes to read

## Assembly generation failed —reason

The assembly-generation part of the compilation failed after the C# compiler successfully compiled the source code. See the errors documentation for the ALink utility ([Al.exe Tool Errors and Warnings](#)) for more information.

## See also

- [Al.exe \(Assembly Linker\)](#)



# Compiler Error CS1578

Article • 09/15/2021 • 2 minutes to read

Filename, single-line comment or end-of-line expected

After a `#line` directive, only a file name (in double quotation marks) or a single-line comment is allowed.

The following sample generates CS1578:

C#

```
// CS1578.cs
class MyClass
{
    static void Main()
    {
        #line 101 abc.cs    // CS1578
        // try the following line instead
        //#line 101 "abc.cs"
        intt i;           // error will be reported on line 101
    }
}
```

# Compiler Error CS1579

Article • 09/15/2021 • 2 minutes to read

foreach statement cannot operate on variables of type 'type1' because 'type2' does not contain a public definition for 'identifier'

To iterate through a collection using the `foreach` statement, the collection must meet the following requirements:

- Its type must include a public parameterless `GetEnumerator` method whose return type is either class, struct, or interface type.
- The return type of the `GetEnumerator` method must contain a public property named `Current` and a public parameterless method named `MoveNext` whose return type is `Boolean`.

## Example

The following sample generates CS1579 because the `MyCollection` class doesn't contain the public `GetEnumerator` method:

```
C#  
  
// CS1579.cs  
using System;  
public class MyCollection  
{  
    int[] items;  
    public MyCollection()  
    {  
        items = new int[5] {12, 44, 33, 2, 50};  
    }  
  
    // Delete the following line to resolve.  
    MyEnumerator GetEnumerator()  
  
    // Uncomment the following line to resolve:  
    // public MyEnumerator GetEnumerator()  
    {  
        return new MyEnumerator(this);  
    }  
  
    // Declare the enumerator class:  
    public class MyEnumerator  
    {  
        int nIndex;  
        MyCollection collection;
```

```
public MyEnumerator(MyCollection coll)
{
    collection = coll;
    nIndex = -1;
}

public bool MoveNext()
{
    nIndex++;
    return (nIndex < collection.items.Length);
}

public int Current => collection.items[nIndex];
}

public static void Main()
{
    MyCollection col = new MyCollection();
    Console.WriteLine("Values in the collection are:");
    foreach (int i in col) // CS1579
    {
        Console.WriteLine(i);
    }
}
```

# Compiler Error CS1583

Article • 09/15/2021 • 2 minutes to read

'file' is not a valid Win32 resource file

This error occurs when you specify a filename with the [Win32Resource](#) compiler option that is not a valid or correctly formatted Win32 resource file. In Visual Studio, the filename is specified in the Application pane of the Project Designer.

## See also

- [Application Page, Project Designer \(C#\)](#)



# Compiler Error CS1585

Article • 09/15/2021 • 2 minutes to read

Member modifier 'keyword' must precede the member type and name

The access specifier in a method signature did not occur in the correct location.

The following sample generates CS1585:

C#

```
// CS1585.cs
public class Class1
{
    public void static Main(string[] args)    // CS1585
    // try the following line instead
    // public static void Main(string[] args)
    {
    }
}
```

# Compiler Error CS1586

Article • 09/15/2021 • 2 minutes to read

Array creation must have array size or array initializer

An array was declared incorrectly.

The following sample generates CS1586:

C#

```
// CS1586.cs
using System;
class MyClass
{
    public static void Main()
    {
        int[] a = new int[]; // CS1586
        // try the following line instead
        int[] b = new int[5];
    }
}
```

## See also

- [Arrays](#)



# Compiler Error CS1588

Article • 09/15/2021 • 2 minutes to read

Cannot determine common language runtime directory -- 'reason'

The compiler cannot retrieve the directory that the .NET runtime is installed in. This error indicates that the common language runtime is improperly installed.

# Compiler Error CS1593

Article • 09/15/2021 • 2 minutes to read

Delegate 'del' does not take 'number' arguments

The number of arguments passed to a [delegate](#) invocation does not agree with the number of parameters in the delegate declaration.

The following sample generates CS1593:

C#

```
// CS1593.cs
using System;
delegate string func(int i); // declare delegate

class a
{
    public static void Main()
    {
        func dt = new func(z);
        x(dt);
    }

    public static string z(int j)
    {
        Console.WriteLine(j);
        return j.ToString();
    }

    public static void x(func hello)
    {
        hello(8, 9); // CS1593
        // try the following line instead
        // hello(8);
    }
}
```

# Compiler Error CS1594

Article • 09/15/2021 • 2 minutes to read

Delegate 'delegate' has some invalid arguments

The type of an argument passed to a [delegate](#) invocation does not agree with the type of the parameter in the delegate declaration.

The following sample generates CS1594:

C#

```
// CS1594.cs
using System;
delegate string func(int i); // declare delegate

class a
{
    public static void Main()
    {
        func dt = new func(z);
        x(dt);
    }

    public static string z(int j)
    {
        Console.WriteLine(j);
        return j.ToString();
    }

    public static void x(func hello)
    {
        hello("8"); // CS1594
        // try the following line instead
        // hello(8);
    }
}
```

# Compiler Error CS1597

Article • 09/15/2021 • 2 minutes to read

Semicolon after method or accessor block is not valid

Semicolons are not needed (or allowed) to end a method or accessor block.

The following sample generates CS1597:

C#

```
// CS1597.cs
class TestClass
{
    public static void Main()
    {
        }; // CS1597, remove semicolon
}
```

# Compiler Error CS1599

Article • 09/15/2021 • 2 minutes to read

Method or delegate cannot return type 'type'

Some types in the .NET class library, for example, [TypedReference](#), [RuntimeArgumentHandle](#) and [ArgIterator](#) cannot be used as return types because they can potentially be used to perform unsafe operations.

The following sample generates CS1599:

```
C#  
  
// CS1599.cs  
using System;  
  
class MyClass  
{  
    public static void Main()  
    {  
    }  
  
    public TypedReference Test1()    // CS1599  
    {  
        return null;  
    }  
  
    public ArgIterator Test2()    // CS1599  
    {  
        return null;  
    }  
}
```

# Compiler Error CS1600

Article • 09/15/2021 • 2 minutes to read

Compilation cancelled by user

A compilation with the C# compiler was cancelled while using the Visual Studio IDE.

# Compiler Error CS1601

Article • 09/15/2021 • 2 minutes to read

Method or delegate parameter cannot be of type 'type'

Some types in the .NET class library, for example, [TypedReference](#), [RuntimeArgumentHandle](#) and [ArgIterator](#) cannot be used as `in`, `ref` or `out` parameters because they could potentially be used to perform unsafe operations.

The following sample generates CS1601:

```
C#  
  
// CS1601.cs  
using System;  
  
class MyClass  
{  
    public void Test1(in TypedReference t)    // CS1601  
    {  
    }  
  
    public void Test2(ref TypedReference t)    // CS1601  
    {  
    }  
  
    public void Test3(out ArgIterator t)    // CS1601  
    {  
        t = default;  
    }  
}
```

# Compiler Error CS1604

Article • 09/15/2021 • 2 minutes to read

Cannot assign to 'variable' because it is read-only

An assignment was made to a read-only variable. To avoid this error, do not assign or attempt to modify to this variable in this context.

# Compiler Error CS1605

Article • 09/15/2021 • 2 minutes to read

Cannot pass 'var' as a `ref` or `out` argument because it is read-only

A variable passed as a `ref` or `out` parameter is expected to be modified in the called method. Therefore, it is not possible to pass a read-only parameter as `ref` or `out`.

# Compiler Error CS1606

Article • 09/15/2021 • 2 minutes to read

Assembly signing failed; output may not be signed -- reason

The assembly was produced, but when the compiler attempted to finish signing it, there was a failure.

# Compiler Error CS1608

Article • 09/15/2021 • 2 minutes to read

The Required attribute is not permitted on C# types

[RequiredAttributeAttribute](#) is not allowed on types defined in C# code.

# Compiler Error CS1609

Article • 09/15/2021 • 2 minutes to read

Modifiers cannot be placed on event accessor declarations

Modifiers can only be placed on event declarations and not on the event accessor declarations. For more information, see [Using Properties](#).

## Example

The following sample generates CS1609.

C#

```
// CS1609.cs
// compile with: /target:library
delegate int Del();
class A
{
    public event Del MyEvent
    {
        private add {}    // CS1609
        // try the following line instead
        // add {}
        remove {}
    }
}
```

# Compiler Error CS1611

Article • 09/15/2021 • 2 minutes to read

The params parameter cannot be declared as in ref or out

The keywords [in](#), [ref](#) or [out](#) cannot be used with the [params](#) keyword.

The following sample generates CS1611:

C#

```
// CS1611.cs
public class MyClass
{
    public static void Test(params ref int[] a) // CS1611, remove ref
    {

    }

    public static void Main()
    {
        Test(1);
    }
}
```

# Compiler Error CS1612

Article • 03/30/2023 • 2 minutes to read

Cannot modify the return value of 'expression' because it is not a variable

An attempt was made to modify a value type that is produced as the result of an intermediate expression but is not stored in a variable. This error can occur when you attempt to directly modify a struct in a generic collection, as shown in the following example:

C#

```
List<MyStruct> list = {...};  
list[0].Name = "MyStruct42"; //CS1612
```

To modify the struct, first assign it to a local variable, modify the variable, then assign the variable back to the item in the collection.

C#

```
List<MyStruct> list = {...};  
MyStruct ms = list[0];  
ms.Name = "MyStruct42";  
list[0] = ms;
```

This error occurs because value types are copied on assignment. When you retrieve a value type from a property or indexer, you are getting a copy of the object, not a reference to the object itself. The copy that is returned is not stored by the property or indexer because they are actually methods, not storage locations (variables). You must store the copy into a variable that you declare before you can modify it.

The error does not occur with reference types because a property or indexer in that case returns a reference to an existing object, which is a storage location.

If you are defining the class or struct, you can resolve this error by modifying your property declaration to provide access to the members of a struct. If you are writing client code, you can resolve the error by creating your own instance of the struct, modifying its fields, and then assigning the entire struct back to the property. As a third alternative, you can change your struct to a class.

## Example

CS1612 also occurs when you attempt to access the member of a struct through a property on an enclosing class that is returning the entire struct, as shown in the following example:

```
C#  
  
// CS1612.cs  
using System;  
  
public struct MyStruct  
{  
    public int Width;  
}  
  
public class ListView  
{  
    public MyStruct Size { get; set; }  
}  
  
public class MyClass  
{  
    public MyClass()  
    {  
        ListView lvi;  
        lvi = new ListView();  
        lvi.Size.Width = 5; // CS1612  
  
        // You can use the following lines instead.  
        // MyStruct ms;  
        // ms.Width = 5;  
        // lvi.Size = ms;  
    }  
  
    public static void Main()  
    {  
        MyClass mc = new MyClass();  
        // Keep the console open in debug mode.  
        Console.WriteLine("Press any key to exit.");  
        Console.ReadKey();  
    }  
}
```

## See also

- [Structure types](#)
- [Value types](#)
- [Reference types](#)

# Compiler Error CS1613

Article • 09/15/2021 • 2 minutes to read

The managed coclass wrapper class 'class' for interface 'interface' cannot be found (are you missing an assembly reference?)

An attempt was made to instantiate a COM object from an interface. The interface has the **ComImport** and **CoClass** attributes, but the compiler cannot find the type given for the **CoClass** attribute.

To resolve this error, you can try one of the following:

- Add a reference to the assembly that has the coclass (most of the time the interface and coclass should be in the same assembly). See [References](#) or [Add Reference Dialog Box](#) for information.
- Fix the **CoClass** attribute on the interface.

The following sample demonstrates correct usage of **CoClassAttribute**:

```
C#  
  
// CS1613.cs  
using System;  
using System.Runtime.InteropServices;  
  
[Guid("1FFD7840-E82D-4268-875C-80A160C23296")]  
[ComImport()]  
[CoClass(typeof(A))]  
public interface IA{}  
public class A : IA {}  
  
public class AA  
{  
    public static void Main()  
    {  
        IA i;  
        i = new IA(); // This is equivalent to new A().  
                      // because of the CoClass attribute on IA  
    }  
}
```

# Compiler Error CS1614

Article • 09/15/2021 • 2 minutes to read

'name' is ambiguous between 'name' and 'nameAttribute'; use either '@name' or 'nameAttribute'.

The compiler has encountered an ambiguous attribute specification.

For convenience, the C# compiler allows you to specify **ExampleAttribute** as just `[Example]`. However, ambiguity arises if an attribute class named `Example` exists along with **ExampleAttribute**, because the compiler cannot tell if `[Example]` refers to the `Example` attribute or the **ExampleAttribute** attribute. To clarify, use `[@Example]` for the `Example` attribute and `[ExampleAttribute]` for **ExampleAttribute**.

The following sample generates CS1614:

```
C#  
  
// CS1614.cs  
using System;  
  
// Both of the following classes are valid attributes with valid  
// names (MySpecial and MySpecialAttribute). However, because the lookup  
// rules for attributes involves auto-appending the 'Attribute' suffix  
// to the identifier, these two attributes become ambiguous; that is,  
// if you specify MySpecial, the compiler can't tell if you want  
// MySpecial or MySpecialAttribute.  
  
public class MySpecial : Attribute {  
    public MySpecial() {}  
}  
  
public class MySpecialAttribute : Attribute {  
    public MySpecialAttribute() {}  
}  
  
class MakeAWarning {  
    [MySpecial()] // CS1614  
        // Ambiguous: MySpecial or MySpecialAttribute?  
    public static void Main() {  
    }  
  
    [@MySpecial()] // This isn't ambiguous, it binds to the first attribute  
    above.  
    public static void Nowarning() {  
    }  
  
    [MySpecialAttribute()] // This isn't ambiguous, it binds to the second  
    attribute above.
```

```
public static void NoWarning2() {  
}  
  
[@MySpecialAttribute()] // This is also legal.  
public static void NoWarning3() {  
}  
}
```

# Compiler Error CS1615

Article • 09/15/2021 • 2 minutes to read

Argument 'number' should not be passed with the 'keyword' keyword

One of the keywords `ref` or `out` was used when the function did not take a `ref` or `out` parameter for that argument. To resolve this error, remove the incorrect keyword and use the appropriate keyword that matches the function declaration, if any.

The following sample generates CS1615:

C#

```
// CS1615.cs
class C
{
    public void f(int i) {}
    public static void Main()
    {
        int i = 1;
        f(ref i); // CS1615
    }
}
```

# Compiler Error CS1617

Article • 11/10/2021 • 2 minutes to read

Invalid option 'option' for **LangVersion**. Use `<LangVersion>?</LangVersion>` to list supported values.

This error occurs if you used the **LangVersion** command line switch or project setting but didn't specify a valid language option. To resolve this error, check the command line syntax or project setting and change it to one of the listed options.

For example, compiling with `csc -langversion:ISO` will generate error CS1617.

## Valid values for `-langversion`

The valid values for the language versions depend on the .NET version you are using. See [the language version rules](#) for more information on which language version is available with which version of .NET. If you are receiving this error while attempting to use a newer language version, either downgrade to a lower language version or update your .NET SDK to a version that supports the language version.

The following table specifies the current valid values for `-langversion`:

Value	Meaning
<code>preview</code>	The compiler accepts all valid language syntax from the latest preview version.
<code>latest</code>	The compiler accepts syntax from the latest released version of the compiler (including minor version).
<code>latestMajor</code> or <code>default</code>	The compiler accepts syntax from the latest released major version of the compiler.
<code>11.0</code>	The compiler accepts only syntax that is included in C# 11 or lower.
<code>10.0</code>	The compiler accepts only syntax that is included in C# 10 or lower.
<code>9.0</code>	The compiler accepts only syntax that is included in C# 9 or lower.
<code>8.0</code>	The compiler accepts only syntax that is included in C# 8.0 or lower.
<code>7.3</code>	The compiler accepts only syntax that is included in C# 7.3 or lower.
<code>7.2</code>	The compiler accepts only syntax that is included in C# 7.2 or lower.
<code>7.1</code>	The compiler accepts only syntax that is included in C# 7.1 or lower.

Value	Meaning
7	The compiler accepts only syntax that is included in C# 7.0 or lower.
6	The compiler accepts only syntax that is included in C# 6.0 or lower.
5	The compiler accepts only syntax that is included in C# 5.0 or lower.
4	The compiler accepts only syntax that is included in C# 4.0 or lower.
3	The compiler accepts only syntax that is included in C# 3.0 or lower.
ISO-2 or 2	The compiler accepts only syntax that is included in ISO/IEC 23270:2006 C# (2.0).
ISO-1 or 1	The compiler accepts only syntax that is included in ISO/IEC 23270:2003 C# (1.0/1.2).

# Compiler Error CS1618

Article • 09/15/2021 • 2 minutes to read

Cannot create delegate with 'method' because it has a Conditional attribute

You cannot create a delegate with a conditional method because the method might not exist in some builds.

The following sample generates CS1618:

C#

```
// CS1618.cs
using System;
using System.Diagnostics;

delegate void del();

class MakeAnError {
    public static void Main() {
        del d = new del(ConditionalMethod);    // CS1618
        // Invalid because on builds where DEBUG is not set,
        // there will be no "ConditionalMethod".
    }
    // To fix the error, remove the next line:
    [Conditional("DEBUG")]
    public static void ConditionalMethod()
    {
        Console.WriteLine("Do something only in debug");
    }
}
```

# Compiler Error CS1619

Article • 09/15/2021 • 2 minutes to read

Cannot create temporary file 'filename' -- reason

The compiler was unable to create a temporary file for the given reason (for example, the disk is full).

# Compiler Error CS1620

Article • 09/15/2021 • 2 minutes to read

Argument 'number' must be passed with the 'keyword' keyword

This error occurs if you are passing an argument to a function that takes a [ref](#) or [out](#) parameter and you don't include the `ref` or `out` keyword at the point of call, or you include the wrong keyword. The error text indicates the appropriate keyword to use and which argument caused the failure.

The following sample generates CS1620:

```
C#  
  
// CS1620.cs  
class C  
{  
    void f(ref int i) {}  
    public static void Main()  
    {  
        int x = 1;  
        f(out x); // CS1620 - f takes a ref parameter, not an out parameter  
        // Try this line instead:  
        // f(ref x);  
    }  
}
```

# Compiler Error CS1621

Article • 09/29/2022 • 2 minutes to read

The yield statement cannot be used inside an anonymous method or lambda expression

You cannot use the [yield](#) statement in an [anonymous method](#) or a [lambda expression](#).

The following sample generates CS1621:

```
C#  
  
// CS1621.cs  
  
using System.Collections;  
  
delegate object MyDelegate();  
  
class C : IEnumerable  
{  
    public IEnumerator GetEnumerator()  
    {  
        MyDelegate d = delegate  
        {  
            yield return this; // CS1621  
            return this;  
        };  
        d();  
        // Try this instead:  
        // MyDelegate d = delegate { return this; };  
        // yield return d();  
    }  
  
    public static void Main()  
    {  
    }  
}
```

## See also

- [Iterators](#)



# Compiler Error CS1622

Article • 09/15/2021 • 2 minutes to read

Cannot return a value from an iterator. Use the yield return statement to return a value, or yield break to end the iteration.

An iterator is a special function that returns a value via the yield statement rather than the return statement. For more information, see [iterators](#).

The following sample generates CS1622:

```
C#  
  
// CS1622.cs  
// compile with: /target:library  
using System.Collections;  
  
class C : IEnumerable  
{  
    public IEnumerator GetEnumerator()  
    {  
        return (IEnumerator) this; // CS1622  
        yield return this; // OK  
    }  
}
```

# Compiler Error CS1623

Article • 09/15/2021 • 2 minutes to read

Iterators cannot have in ref or out parameters

This error occurs if an iterator method takes an `in`, `ref`, or `out` parameter. To avoid this error, remove the `in`, `ref`, or `out` keyword from the method signature.

## Example

The following sample generates CS1623:

C#

```
// CS1623.cs
using System.Collections;

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        yield return 0;
    }

    // To resolve the error, remove in
    public IEnumerator GetEnumerator(in short i) // CS1623
    {
        yield return i;
    }
    // To resolve the error, remove ref
    public IEnumerator GetEnumerator(ref int i) // CS1623
    {
        yield return i;
    }

    // To resolve the error, remove out
    public IEnumerator GetEnumerator(out float f) // CS1623
    {
        f = 0.0F;
        yield return f;
    }
}
```

# Compiler Error CS1624

Article • 09/15/2021 • 2 minutes to read

The body of 'accessor' cannot be an iterator block because 'type' is not an iterator interface type

This error occurs if an iterator accessor is used but the return type is not one of the iterator interface types: [IEnumerable](#), [IEnumerable<T>](#), [IEnumerator](#), [IEnumerator<T>](#).

To avoid this error, use one of the iterator interface types as a return type.

## Example

The following sample generates CS1624:

C#

```
// CS1624.cs
using System;
using System.Collections;

class C
{
    public int Iterator
    // Try this instead:
    // public IEnumerable Iterator
    {
        get // CS1624
        {
            yield return 1;
        }
    }
}
```

# Compiler Error CS1625

Article • 09/15/2021 • 2 minutes to read

Cannot yield in the body of a finally clause

A yield statement is not allowed in the body of a finally clause. To avoid this error, move the yield statement out of the finally clause.

The following sample generates CS1625:

C#

```
// CS1625.cs
using System.Collections;

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        try
        {
        }
        finally
        {
            yield return this; // CS1625
        }
    }

    public class CMain
    {
        public static void Main() { }
```

# Compiler Error CS1626

Article • 09/15/2021 • 2 minutes to read

Cannot yield a value in the body of a try block with a catch clause

A yield statement is not allowed in a try block if there is a catch clause associated with the try block. To avoid this error, either move the yield statement out of the try/catch/finally block, or remove the catch block.

The following sample generates CS1626:

```
C#  
  
// CS1626.cs  
using System.Collections;  
  
class C : IEnumerable  
{  
    public IEnumerator GetEnumerator()  
    {  
        try  
        {  
            yield return this; // CS1626  
        }  
        catch  
        {  
  
        }  
        finally  
        {  
  
        }  
    }  
  
    public class CMain  
    {  
        public static void Main() { }  
    }  
}
```

# Compiler Error CS1627

Article • 09/15/2021 • 2 minutes to read

Expression expected after yield return

This error occurs if `yield` is used without an expression. To avoid this error, insert the appropriate expression in the statement.

The following sample generates CS1627:

```
C#  
  
// CS1627.cs  
using System.Collections;  
  
class C : IEnumerable  
{  
    public IEnumerator GetEnumerator()  
    {  
        yield return;    // CS1627  
        // To resolve, add the following line:  
        // yield return 0;  
    }  
}  
  
public class CMain  
{  
    public static void Main() { }  
}
```

# Compiler Error CS1628

Article • 09/15/2021 • 2 minutes to read

Cannot use in ref or out parameter 'parameter' inside an anonymous method, lambda expression, or query expression

This error occurs if you use an `in`, `ref`, or `out` parameter inside an anonymous method block. To avoid this error, use a local variable or some other construct.

The following sample generates CS1628:

C#

```
// CS1628.cs

delegate int MyDelegate();

class C
{
    public static void F(ref int i)
    {
        MyDelegate d = delegate { return i; }; // CS1628
        // Try this instead:
        // int tmp = i;
        // MyDelegate d = delegate { return tmp; };
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1629

Article • 09/15/2021 • 2 minutes to read

Unsafe code may not appear in iterators

The C# language specification does not allow unsafe code in iterators.

The following sample generates CS1629:

C#

```
// CS1629.cs
// compile with: /unsafe
using System.Collections.Generic;
class C
{
    IEnumerator<int> IteratorMeth() {
        int i;
        unsafe // CS1629
        {
            int *p = &i;
            yield return *p;
        }
    }
}
```

# Compiler Error CS1630

Article • 09/15/2021 • 2 minutes to read

Invalid option 'option' for **ErrorReport**; must be **prompt**, **send**, **queue**, or **none**

The command line option **ErrorReport** must be followed by **prompt**, **send**, **queue**, or **none**, specifying what action you want taken when an internal compiler error occurs.

{!NOTE} This compiler option is no longer supported.



# Compiler Error CS1631

Article • 09/15/2021 • 2 minutes to read

Cannot yield a value in the body of a catch clause

The yield statement is not allowed from within the body of a catch clause. To avoid this error, move the yield statement outside the body of the catch clause.

The following sample generates CS1631:

C#

```
// CS1631.cs
using System;
using System.Collections;

public class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        try
        {
        }
        catch(Exception e)
        {
            yield return this; // CS1631
        }
    }

    public static void Main()
    {
    }
}
```

# Compiler Error CS1632

Article • 09/15/2021 • 2 minutes to read

Control cannot leave the body of an anonymous method or lambda expression

This error occurs if a jump statement (`break`, `goto`, `continue`, etc.) attempts to move control out of an anonymous method block. An anonymous method block is a function body and can only be exited by a return statement or by reaching the end of the block.

The following sample generates CS1632:

C#

```
// CS1632.cs
// compile with: /target:library
delegate void MyDelegate();
class MyClass
{
    public void Test()
    {
        for (int i = 0 ; i < 5 ; i++)
        {
            MyDelegate d = delegate {
                break; // CS1632
            };
        }
    }
}
```

# Compiler Error CS1637

Article • 09/15/2021 • 2 minutes to read

Iterators cannot have unsafe parameters or yield types

Check the argument list of the iterator and the type of any yield statements to verify that you are not using any unsafe types.

## Example

The following sample generates CS1637:

C#

```
// CS1637.cs
// compile with: /unsafe
using System.Collections;

public unsafe class C
{
    public IEnumerator Iterator1(int* p) // CS1637
    {
        yield return null;
    }
}
```

# Compiler Error CS1638

Article • 04/12/2022 • 2 minutes to read

'identifier' is a reserved identifier and cannot be used when ISO language version mode is used

When the ISO language compatibility option is specified by the `/langversion` compiler switch, any identifier with double underscores anywhere in the identifier will produce this error. To avoid this error, eliminate any identifiers with double underscores, or do not use the ISO-1 language version option.

## Example

The following sample generates CS1638:

C#

```
// CS1638.cs
// compile with: /langversion:ISO-1
class bad__identifier // CS1638 (double underscores are not ISO compliant)
{
}

// Try this instead:
//class GoodIdentifier
//{
//}

class CMain
{
    public static void Main() { }
```

## See also

- [LangVersion \(C# Compiler Options\)](#)



# Compiler Error CS1639

Article • 09/15/2021 • 2 minutes to read

The managed coclass wrapper class signature 'signature' for interface 'interface' is not a valid class name signature

This error occurs if the metadata for the interface is invalid. This should never occur if the interface was generated using C# or a supported .NET language. Check with the vendor of the interface or verify that the interface assembly was generated successfully.

# Compiler Error CS1640

Article • 09/15/2021 • 2 minutes to read

foreach statement cannot operate on variables of type 'type' because it implements multiple instantiations of 'interface', try casting to a specific interface instantiation

The type inherits from two or more instances of `IEnumerable<T>`, which means there is not a unique enumeration of the type that `foreach` could use. Specify the type of `IEnumerable<T>` or use another looping construct.

## Example

The following sample generates CS1640:

C#

```
// CS1640.cs

using System;
using System.Collections;
using System.Collections.Generic;

public class C : IEnumerable, IEnumerable<int>, IEnumerable<string>
{
    IEnumerator<int> IEnumerable<int>.GetEnumerator()
    {
        yield break;
    }

    IEnumerator<string> IEnumerable<string>.GetEnumerator()
    {
        yield break;
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return (IEnumerator)((IEnumerable<string>)this).GetEnumerator();
    }
}

public class Test
{
    public static int Main()
    {
        foreach (int i in new C()){} // CS1640

        // Try specifying the type of IEnumerable<T>
        // foreach (int i in (IEnumerable<int>)new C()){}
    }
}
```

```
    return 1;
}
}
```

# Compiler Error CS1641

Article • 09/15/2021 • 2 minutes to read

A fixed size buffer field must have the array size specifier after the field name

Unlike regular arrays, fixed size buffers require a constant size to be specified at the declaration point. To resolve this error, add a positive integer literal or a constant positive integer and put the square brackets after the identifier.

The following sample generates CS1641:

```
C#  
  
// CS1641.cs  
// compile with: /unsafe /target:library  
unsafe struct S {  
    fixed int [] a; // CS1641  
  
    // OK  
    fixed int b [10];  
    const int c = 10;  
    fixed int d [c];  
}
```

# Compiler Error CS1642

Article • 09/15/2021 • 2 minutes to read

Fixed size buffer fields may only be members of structs.

This error occurs if you use a fixed size buffer field in a `class`, instead of a `struct`. To resolve this error, change the `class` to a `struct` or declare the field as an ordinary array.

## Example

The following sample generates CS1642.

C#

```
// CS1642.cs
// compile with: /unsafe /target:library
unsafe class C
{
    fixed int a[10];    // CS1642
}

unsafe struct D
{
    fixed int a[10];
}

unsafe class E
{
    public int[] a = null;
}
```

# Compiler Error CS1643

Article • 09/15/2021 • 2 minutes to read

Not all code paths return a value in method of type 'type'!

This error occurs if a delegate body does not have a return statement, or has a return statement that the compiler is unable to verify will be reached. In the example below, the compiler does not attempt to predict the result of the branching condition in order to verify that the anonymous method block always returns a value.

## Example

The following sample generates CS1643:

```
C#  
  
// CS1643.cs  
delegate int MyDelegate();  
  
class C  
{  
    static void Main()  
    {  
        MyDelegate d = delegate  
        {  
            int i = 0;  
            if (i == 0)  
                return 1;  
        };  
    }  
}
```

# Compiler Error CS1644

Article • 09/15/2021 • 2 minutes to read

Feature 'feature' is not part of the standardized ISO C# language specification, and may not be accepted by other compilers

This error occurs if you specified the [LangVersion](#) option ISO-1 and the code you are compiling uses features that are not part of the ISO 1.0 standard. To resolve this error, do not use any of the C# 2.0 compiler features such as generics or anonymous methods with the ISO-1 compatibility option.

The following sample generates CS1644:

C#

```
// CS1644.cs
// compile with: /langversion:ISO-1 /target:library
class <T> {} // CS1644
```

# Compiler Error CS1646

Article • 09/15/2021 • 2 minutes to read

Keyword, identifier, or string expected after verbatim specifier: @

See string literals for the usage of the verbatim specifier '@'. The verbatim specifier is only allowed before a string, keyword or identifier. To resolve this error, remove the @ symbol from any inappropriate place or add the intended string, keyword or identifier.

The following sample generates CS1646:

```
C#  
  
// CS1646  
class C  
{  
    int i = @5; // CS1646  
    // Try this line instead:  
    // int i = 5;  
}
```

# Compiler Error CS1647

Article • 09/15/2021 • 2 minutes to read

An expression is too long or complex to compile near 'code'

There was a stack overflow in the compiler processing your code. To resolve this error, simplify your code. If your code is valid, get help at the [Developer Community](#) site or [StackOverflow](#).

# Compiler Error CS1648

Article • 09/15/2021 • 2 minutes to read

Members of readonly field 'identifier' cannot be modified (except in a constructor or a variable initializer)

This error occurs when you attempt to modify a member of a field which is readonly where it is not allowed to be modified. To resolve this error, limit assignments to readonly fields to the constructor or variable initializer, or remove the readonly keyword from the declaration of the field.

## Example

The following sample generates CS1648:

```
C#  
  
// CS1648.cs  
public struct Inner  
{  
    public int i;  
}  
  
class Outer  
{  
    public readonly Inner inner = new Inner();  
}  
  
class D  
{  
    static void Main()  
    {  
        var outer = new Outer();  
        outer.inner.i = 1; // CS1648  
    }  
}
```

# Compiler Error CS1649

Article • 09/15/2021 • 2 minutes to read

Members of readonly field 'identifier' cannot be passed ref or out (except in a constructor)

This error occurs if you pass a variable to a function that is a member of a `readonly` field as a `ref` or `out` argument. Since `ref` and `out` parameters may be modified by the function, this is not allowed. To resolve this error, remove the `readonly` keyword on the field, or do not pass the members of the `readonly` field to the function. For example, you might try creating a temporary variable which can be modified and passing the temporary as a `ref` argument, as shown in the following example.

## Example

The following sample generates CS1649:

```
C#  
  
// CS1649.cs  
public struct Inner  
{  
    public int i;  
}  
  
class Outer  
{  
    public readonly Inner inner = new Inner();  
}  
  
class D  
{  
    static void f(ref int iref)  
    {  
    }  
  
    static void Main()  
    {  
        Outer outer = new Outer();  
        f(ref outer.inner.i); // CS1649  
        // Try this code instead:  
        // int tmp = outer.inner.i;  
        // f(ref tmp);  
    }  
}
```

# Compiler Error CS1650

Article • 09/15/2021 • 2 minutes to read

Fields of static readonly field 'identifier' cannot be assigned to (except in a static constructor or a variable initializer)

This error occurs when you attempt to modify a member of a field which is readonly and static where it is not allowed to be modified. To resolve this error, limit assignments to readonly fields to the constructor or variable initializer, or remove the `readonly` keyword from the declaration of the field.

C#

```
// CS1650.cs
public struct Inner
{
    public int i;
}

class Outer
{
    public static readonly Inner inner = new Inner();
}

class D
{
    static void Main()
    {
        Outer.inner.i = 1; // CS1650
    }
}
```

# Compiler Error CS1651

Article • 09/15/2021 • 2 minutes to read

Fields of static readonly field 'identifier' cannot be passed ref or out (except in a static constructor)

This error occurs if you pass a variable to a function that is a member of a static readonly field as a ref argument. Since ref parameters may be modified by the function, this is not allowed. To resolve this error, remove the `readonly` keyword on the field, or do not pass the members of the readonly field to the function. For example, you might try creating a temporary variable which can be modified and passing the temporary as a ref argument, as shown in the following example.

The following sample generates CS1651:

```
C#  
  
// CS1651.cs  
public struct Inner  
{  
    public int i;  
}  
  
class Outer  
{  
    public static readonly Inner inner = new Inner();  
}  
  
class D  
{  
    static void f(ref int iref)  
    {  
    }  
  
    static void Main()  
    {  
        f(ref Outer.inner.i); // CS1651  
        // Try this instead:  
        // int tmp = Outer.inner.i;  
        // f(ref tmp);  
    }  
}
```

# Compiler Error CS1654

Article • 09/15/2021 • 2 minutes to read

Cannot modify members of 'variable' because it is a 'read-only variable type'

This error occurs when you try to modify members of a variable which is read-only because it is in a special construct.

One common area that this occurs is within `foreach` loops. It is a compile-time error to modify the value of the collection elements. Therefore, you cannot make any modifications to elements that are [value types](#), including [structs](#). In a collection whose elements are [reference types](#), you can modify accessible members of each element, but any try to add or remove or replace complete elements will generate [Compiler Error CS1654](#).

## Example

The following example generates error CS1654 because `Book` is a [struct](#). To fix the error, change the `struct` to a [class](#).

```
C#  
  
using System.Collections.Generic;  
using System.Text;  
  
namespace CS1654  
{  
  
    struct Book  
{  
        public string Title;  
        public string Author;  
        public double Price;  
        public Book(string t, string a, double p)  
        {  
            Title=t;  
            Author=a;  
            Price=p;  
  
        }  
    }  
  
    class Program  
{  
        List<Book> list;  
        static void Main(string[] args)  
        {
```

```
//Use a collection initializer to initialize the list
Program prog = new Program();
prog.list = new List<Book>();
prog.list.Add(new Book ("The C# Programming Language",
                      "Hejlsberg, Wiltamuth, Golde",
                      29.95));
prog.list.Add(new Book ("The C++ Programming Language",
                      "Stroustrup",
                      29.95));
prog.list.Add(new Book ("The C Programming Language",
                      "Kernighan, Ritchie",
                      29.95));
foreach(Book b in prog.list)
{
    //Compile error if Book is a struct
    //Make Book a class to modify its members
    b.Price +=9.95; // CS1654
}

}
}
```

# Compiler Error CS1655

Article • 03/14/2023 • 2 minutes to read

Cannot pass fields of 'variable' as a ref or out argument because it is a 'readonly variable type'

This error occurs if you are attempting to pass a member of a `foreach` variable, a `using` variable, or a `fixed` variable to a function as a ref or out argument. Because these variables are considered read-only in these contexts, this is not allowed.

The following sample generates CS1655:

```
C#  
  
// CS1655.cs  
struct S  
{  
    public int i;  
}  
  
class CMain  
{  
    static void f(ref int iref)  
    {  
    }  
  
    public static void Main()  
    {  
        S[] sa = new S[10];  
        foreach(S s in sa)  
        {  
            CMain.f(ref s.i); // CS1655  
        }  
    }  
}
```

# Compiler Error CS1656

Article • 03/14/2023 • 2 minutes to read

Cannot assign to 'variable' because it is a 'read-only variable type'

This error occurs when an assignment to variable occurs in a read-only context. Read-only contexts include `foreach` iteration variables, `using` variables, and `fixed` variables. To resolve this error, avoid assignments to a statement variable in `using` blocks, `foreach` statements, and `fixed` statements.

## Example 1

The following example generates error CS1656 because it tries to replace complete elements of a collection inside a `foreach` loop. One way to work around the error is to change the `foreach` loop to a `for` loop. Another way, not shown here, is to modify the members of the existing element; this is possible with classes, but not with structs.

C#

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;

namespace CS1656_2
{

    class Book
    {
        public string Title;
        public string Author;
        public double Price;
        public Book(string t, string a, double p)
        {
            Title=t;
            Author=a;
            Price=p;
        }
    }

    class Program
    {
        private List<Book> list;
        static void Main(string[] args)
        {
            Program prog = new Program();
```

```

prog.list = new List<Book>();
prog.list.Add(new Book ("The C# Programming Language",
                      "Hejlsberg, Wiltamuth, Golde",
                      29.95));
prog.list.Add(new Book ("The C++ Programming Language",
                      "Stroustrup",
                      29.95));
prog.list.Add(new Book ("The C Programming Language",
                      "Kernighan, Ritchie",
                      29.95));
foreach(Book b in prog.list)
{
    // Cannot modify an entire element in a foreach loop
    // even with reference types.
    // Use a for or while loop instead
    if (b.Title == "The C Programming Language")
        // Cannot assign to 'b' because it is a 'foreach
        // iteration variable'
        b = new Book("Programming Windows, 5th Ed.", "Petzold",
29.95); //CS1656
}

//With a for loop you can modify elements
//for(int x = 0; x < prog.list.Count; x++)
//{
//    if(prog.list[x].Title== "The C Programming Language")
//        prog.list[x] = new Book("Programming Windows, 5th
Ed.", "Petzold", 29.95);
//}
//foreach(Book b in prog.list)
//    Console.WriteLine(b.Title);

}
}
}

```

## Example 2

The following sample demonstrates how CS1656 can be generated in other contexts besides a `foreach` loop:

```

C#

// CS1656.cs
// compile with: /unsafe
using System;

class C : IDisposable
{
    public void Dispose() { }
}
```

```
class CMain
{
    unsafe public static void Main()
    {
        using (C c = new C())
        {
            // Cannot assign to 'c' because it is a 'using variable'
            c = new C(); // CS1656
        }

        int[] ary = new int[] { 1, 2, 3, 4 };
        fixed (int* p = ary)
        {
            // Cannot assign to 'p' because it is a 'fixed variable'
            p = null; // CS1656
        }
    }
}
```

# Compiler Error CS1657

Article • 03/14/2023 • 2 minutes to read

Cannot pass 'parameter' as a ref or out argument because 'reason'

This error occurs when a variable is passed as a `ref` or `out` argument in a context in which that variable is readonly. Readonly contexts include `foreach` iteration variables, `using` variables, and `fixed` variables. To resolve this error, do not call functions that take the `foreach`, `using` or `fixed` variable as a `ref` or `out` parameter in `using` blocks, `foreach` statements, and `fixed` statements.

## Example 1

The following sample generates CS1657:

```
C#  
  
// CS1657.cs  
using System;  
class C : IDisposable  
{  
    public int i;  
    public void Dispose() {}  
}  
  
class CMain  
{  
    static void f(ref C c)  
    {  
    }  
    static void Main()  
    {  
        using (C c = new C())  
        {  
            f(ref c); // CS1657  
        }  
    }  
}
```

## Example 2

The following code illustrates the same problem in a `fixed` statement:

```
C#
```

```
// CS1657b.cs
// compile with: /unsafe
unsafe class C
{
    static void F(ref int* p)
    {
    }

    static void Main()
    {
        int[] a = new int[5];
        fixed(int* p = a) F(ref p); // CS1657
    }
}
```

# Compiler Error CS1660

Article • 09/15/2021 • 2 minutes to read

Cannot convert anonymous method block to type 'type' because it is not a delegate type

This error occurs if you try to assign or otherwise convert an anonymous method block to a type which is not a delegate type.

The following sample generates CS1660:

```
C#  
  
// CS1660.cs  
delegate int MyDelegate();  
class C {  
    static void Main()  
    {  
        int i = delegate { return 1; }; // CS1660  
        // Try this instead:  
        // MyDelegate myDelegate = delegate { return 1; };  
        // int i = myDelegate();  
    }  
}
```

# Compiler Error CS1661

Article • 09/15/2021 • 2 minutes to read

Cannot convert anonymous method block to delegate type 'delegate type' because the specified block's parameter types do not match the delegate parameter types

This error occurs if, in an anonymous method definition, the parameter types of the anonymous method do not match the delegate parameter types. Check the number of parameters, the parameter types, and any ref or out parameters and verify an exact match.

The following sample generates CS1661:

C#

```
// CS1661.cs

delegate void MyDelegate(int i);

class C
{
    public static void Main()
    {
        MyDelegate d = delegate(string s) { }; // CS1661
    }
}
```

# Compiler Error CS1662

Article • 09/15/2021 • 2 minutes to read

Cannot convert anonymous method block to delegate type 'delegate type' because some of the return types in the block are not implicitly convertible to the delegate return type

This error occurs if the anonymous method block's return statement had a type that was not implicitly convertible to the return type of the delegate.

The following sample generates CS1662:

C#

```
// CS1662.cs

delegate int MyDelegate(int i);

class C
{

    public static void Main()
    {
        MyDelegate d = delegate(int i) { return 1.0; }; // CS1662
        // Try this instead:
        // MyDelegate d = delegate(int i) { return (int)1.0; };
    }
}
```

# Compiler Error CS1663

Article • 09/15/2021 • 2 minutes to read

Fixed size buffer type must be one of the following: bool, byte, short, int, long, char, sbyte, ushort, uint, ulong, float or double

A fixed sized buffer may not be any type other than those listed. To avoid this error, use another type or do not use a fixed array.

## Example

The following sample generates CS1663.

```
C#  
  
// CS1663.cs  
// compile with: /unsafe /target:library  
  
unsafe struct C  
{  
    fixed string ab[10]; // CS1663  
}
```

# Compiler Error CS1664

Article • 09/15/2021 • 2 minutes to read

Fixed size buffer of length 'length' and type 'type' is too big

The maximum size of a fixed-size buffer (as determined by the length multiplied by the element size) is  $2^{31} = 268435455$ .

# Compiler Error CS1665

Article • 09/15/2021 • 2 minutes to read

Fixed size buffers must have a length greater than zero

This error occurs if a fixed size buffer is declared with a zero or negative size. The length of a fixed size buffer must be a positive integer.

## Example

The following sample generates CS1665.

C#

```
// CS1665.cs
// compile with: /unsafe /target:library
struct S
{
    public unsafe fixed int A[0];    // CS1665
}
```

# Compiler Error CS1666

Article • 09/15/2021 • 2 minutes to read

You cannot use fixed size buffers contained in unfixed expressions. Try using the `fixed` statement.

This error occurs if you use the fixed sized buffer in an expression involving a class that is not itself fixed. The runtime is free to move the unfixed class around to optimize memory access, which could lead to errors when using the fixed sized buffer. To avoid this error, use the `fixed` keyword on the statement.

## Example

The following sample generates CS1666.

C#

```
// CS1666.cs
// compile with: /unsafe /target:library
unsafe struct S
{
    public fixed int buffer[1];
}

unsafe class Test
{
    S field = new S();

    private bool example1()
    {
        return (field.buffer[0] == 0);    // CS1666 error
    }

    private bool example2()
    {
        // OK
        fixed (S* p = &field)
        {
            return (p->buffer[0] == 0);
        }
    }

    private bool example3()
    {
        S local = new S();
        return (local.buffer[0] == 0);
    }
}
```



# Compiler Error CS1667

Article • 09/15/2021 • 2 minutes to read

Attribute 'attribute' is not valid on property or event accessors. It is valid on 'declaration type' declarations only.

This error occurs if you use an attribute on a property or event accessor, when it should be on the property or event itself. This error could occur with the attributes [CLSCompliantAttribute](#), [ConditionalAttribute](#), and [ObsoleteAttribute](#).

## Example

The following sample generates CS1670:

```
C#  
  
// CS1667.cs  
using System;  
  
public class C  
{  
    private int i;  
  
    //Try this instead:  
    //#[Obsolete]  
    public int ObsoleteProperty  
    {  
        [Obsolete] // CS1667  
        get { return i; }  
        set { i = value; }  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS1670

Article • 09/15/2021 • 2 minutes to read

params is not valid in this context

A number of C# features are incompatible with variable argument lists, and do not allow the `params` keyword, including the following:

- Parameter lists of anonymous methods
- Overloaded operators

## Example

The following sample generates CS1670:

C#

```
// CS1670.cs
public class C
{
    public bool operator +(params int[] paramsList) // CS1670
    {
        return false;
    }

    static void Main()
    {
    }
}
```

# Compiler Error CS1671

Article • 09/15/2021 • 2 minutes to read

A namespace declaration cannot have modifiers or attributes

Modifiers are not meaningful when applied to a namespace, so they are not allowed.

The following sample generates CS1671:

C#

```
// CS1671.cs
public namespace NS // CS1671
{
}
```

# Compiler Error CS1672

Article • 09/15/2021 • 2 minutes to read

Invalid option 'option' for /platform; must be anycpu, x86, Itanium or x64

The options specify the processor type; use one of the forms listed.

# Compiler Error CS1673

Article • 09/15/2021 • 2 minutes to read

Anonymous methods, lambda expressions, and query expressions inside structs cannot access instance members of 'this'. Consider copying 'this' to a local variable outside the anonymous method, lambda expression or query expression and using the local instead.

The following sample generates CS1673:

```
C#  
  
// CS1673.cs  
delegate int MyDelegate();  
  
public struct S  
{  
    int member;  
  
    public int F(int i)  
    {  
        member = i;  
        // Try assigning to a local variable  
        // S s = this;  
        MyDelegate d = delegate()  
        {  
            i = this.member; // CS1673  
            // And use the local variable instead of "this"  
            // i = s.member;  
            return i;  
  
        };  
        return d();  
    }  
}  
  
class CMain  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS1674

Article • 03/14/2023 • 2 minutes to read

'T': type used in a using statement must be implicitly convertible to 'System.IDisposable'

The [using statement](#) is intended to be used to ensure the disposal of an object at the end of the `using` block, thus, only types which are disposable may be used in such a statement. For example, value types are not disposable, and type parameters which are not constrained to be classes may not be assumed to be disposable.

## Example 1

The following sample generates CS1674.

```
C#  
  
// CS1674.cs  
class C  
{  
    public static void Main()  
    {  
        int a = 0;  
        a++;  
  
        using (a) {} // CS1674  
    }  
}
```

## Example 2

The following sample generates CS1674.

```
C#  
  
// CS1674_b.cs  
using System;  
class C {  
    public void Test() {  
        using (C c = new C()) {} // CS1674  
    }  
}  
  
// OK  
class D : IDisposable {  
    void IDisposable.Dispose() {}
```

```
public void Dispose() {}

public static void Main() {
    using (D d = new D()) {}
}

}
```

## Example 3

The following case illustrates the need for a class type constraint to guarantee that an unknown type parameter is disposable. The following sample generates CS1674.

C#

```
// CS1674_c.cs
// compile with: /target:library
using System;
public class C<T>
// Add a class type constraint that specifies a disposable class.
// Uncomment the following line to resolve.
// public class C<T> where T : IDisposable
{
    public void F(T t)
    {
        using (t) {} // CS1674
    }
}
```

# Compiler Error CS1675

Article • 09/15/2021 • 2 minutes to read

Enums cannot have type parameters

To resolve this error, remove the type parameter from the `enum` declaration.

## Example

The following sample generates CS1675:

C#

```
// CS1675.cs
enum E<T> // CS1675
{
}

class CMain
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS1676

Article • 09/15/2021 • 2 minutes to read

Parameter 'number' must be declared with the 'keyword' keyword

This error occurs when the parameter type modifier in an anonymous method is different from that used in the declaration of the delegate you are casting the method to.

The following sample generates CS1676:

```
C#  
  
// CS1676.cs  
delegate void E(ref int i);  
class Errors  
{  
    static void Main()  
    {  
        E e = delegate(out int i) { };    // CS1676  
        // To resolve, use the following line instead:  
        // E e = delegate(ref int i) { };  
    }  
}
```

# Compiler Error CS1677

Article • 09/15/2021 • 2 minutes to read

Parameter 'number' should not be declared with the 'keyword' keyword

This error occurs when the parameter type modifier in an anonymous method does not match that used in the declaration of the delegate, to which you are casting the method.

## Example

The following sample generates CS1677:

C#

```
// CS1677.cs
delegate void D(int i);
class Errors
{
    static void Main()
    {
        D d = delegate(out int i) { };    // CS1677
        // To resolve, use the following line instead:
        // D d = delegate(int i) { };

        D d = delegate(ref int j){}; // CS1677
        // To resolve, use the following line instead:
        // D d = delegate(int j){};
    }
}
```

# Compiler Error CS1678

Article • 09/15/2021 • 2 minutes to read

Parameter 'number' is declared as type 'type1' but should be 'type2'

This error occurs when the parameter type in an anonymous method is different from the declaration of the delegate you are casting the method to.

The following sample generates CS1678:

C#

```
// CS1678
delegate void D(int i);
class Errors
{
    static void Main()
    {
        D d = delegate(string s) { };    // CS1678
        // To resolve, use the following line instead:
        // D d = delegate(int s) { };
    }
}
```

# Compiler Error CS1679

Article • 09/15/2021 • 2 minutes to read

Invalid extern alias for '/reference'; 'identifier' is not a valid identifier

When using the external assembly alias feature of the `/reference` option, the text that follows `/reference:` and that precedes the '=' must be a valid C# identifier or keyword according to the C# Language Specification.

To correct this error, change text before the "=" to a valid C# identifier or keyword.

## Example

The following example generates CS1679.

C#

```
// CS1679.cs
// compile with: /reference:123$BadIdentifier%System.dll
class TestClass {
    static void Main()
    {
    }
}
```

# Compiler Error CS1680

Article • 09/15/2021 • 2 minutes to read

Invalid reference alias option: 'alias=' -- missing filename.

This error occurs when you use the `alias` feature with the `/reference` compiler option without specifying a valid file name.

The following sample generates CS1680.

C#

```
// CS1680.cs
// compile with: /reference:alias=
// CS1680 expected
// To resolve, specify the name of a file with an assembly manifest
class MyClass {}
```

# Compiler Error CS1681

Article • 09/15/2021 • 2 minutes to read

You cannot redefine the global extern alias

The global alias is already defined to include all unaliased references and therefore cannot be redefined.

## Example

The following sample generates CS1681.

C#

```
// CS1681.cs
// compile with: /reference:global=System.dll
// CS1681 expected

// try this instead: /reference:System.dll
class A
{
    static void Main() {}
}
```

# Compiler Error CS1686

Article • 09/15/2021 • 2 minutes to read

Local 'variable' or its members cannot have their address taken and be used inside an anonymous method or lambda expression

This error is generated when you use a variable, and attempt to take its address, and one of these actions is done inside an anonymous method.

## Example

The following sample generates CS1686.

C#

```
// CS1686.cs
// compile with: /unsafe /target:library
class MyClass
{
    public unsafe delegate int * MyDelegate();

    public unsafe int * Test()
    {
        int j = 0;
        MyDelegate d = delegate { return &j; };    // CS1686
        return &j;    // OK
    }
}
```

# Compiler Error CS1688

Article • 09/15/2021 • 2 minutes to read

Cannot convert anonymous method block without a parameter list to delegate type 'delegate' because it has one or more out parameters

The compiler allows parameters to be omitted from an anonymous method block in most cases. This error arises when the anonymous method block does not have a parameter list, but the delegate has an `out` parameter. The compiler does not allow this situation because it would need to ignore the presence of the `out` parameter, which is unlikely to be the correct behavior.

## Example

The following code generates error CS1688.

```
C#  
  
// CS1688.cs  
using System;  
delegate void OutParam(out int i);  
class ErrorCS1676  
{  
    static void Main()  
{  
        OutParam o;  
        o = delegate // CS1688  
        // Try this instead:  
        // o = delegate(out int i)  
        {  
            Console.WriteLine("");  
        };  
    }  
}
```

# Compiler Error CS1689

Article • 09/15/2021 • 2 minutes to read

Attribute 'Attribute Name' is only valid on methods or attribute classes

This error only occurs with the **ConditionalAttribute** attribute. As the message states, this attribute can only be used on methods or attribute classes. For example, trying to apply this attribute to a class will generate this error.

## Example

The following sample generates CS1689.

C#

```
// CS1689.cs
// compile with: /target:library
[System.Diagnostics.Conditional("A")]    // CS1689
class MyClass {}
```

# Compiler Error CS1703

Article • 09/15/2021 • 2 minutes to read

An assembly with the same simple name 'name' has already been imported. Try removing one of the references or sign them to enable side-by-side.

The compiler removes references that have the same path and file name, but it is possible that the same file exists in two places, or that you forgot to change the version number. This error points out that two references have the same assembly identity and thus the compiler has no way of distinguishing between them in metadata. Either remove one of the redundant references, or make the references unique somehow, such as by incrementing the assembly version number.

The following code generates error CS1703.

## Example 1

This code creates assembly A in the .\bin1 directory.

Save this example in a file named CS1703a1.cs, and compile it with the following flags:

```
/t:library /out:.\\bin1\\cs1703.dll /keyfile:key.snk
```

C#

```
using System;
public class A { }
```

## Example 2

This code creates a copy of assembly A in the .\bin2 directory.

Save this example in a file named CS1703a2.cs, and compile it with the following flags:

```
/t:library /out:.\\bin2\\cs1703.dll /keyfile:key.snk
```

C#

```
using System;
public class A { }
```

## Example 3

This code references the assembly A in the two prior modules.

Save this example in a file named CS1703ref.cs, and compile it with the following flags:

```
/t:library /r:A2=.\bin2\cs1703.dll /r:A1=.\bin1\cs1703.dll
```

C#

```
extern alias A1;
extern alias A2;
```

# Compiler Error CS1704

Article • 09/15/2021 • 2 minutes to read

An assembly with the same simple name 'Assembly Name' has already been imported. Try removing one of the references or sign them to enable side-by-side.

This error points out that two references have the same assembly identity because the assemblies in question lack strong names, they were not signed, and thus the compiler has no way of distinguishing between them in metadata. Thus, the run time ignores the version and culture assembly name properties. The user should remove the redundant reference, rename one of the references, or provide a strong name for them.

## Example 1

This sample creates an assembly and saves it to the root directory.

C#

```
// CS1704_a.cs
// compile with: /target:library /out:c:\\cs1704.dll
public class A {}
```

## Example 2

This sample creates an assembly with the same name as the previous sample, but saves it to a different location.

C#

```
// CS1704_b.cs
// compile with: /target:library /out:cs1704.dll
public class A {}
```

## Example 3

This sample attempts to reference both assemblies. The following sample generates CS1704.

C#

```
// CS1704_c.cs
// compile with: /target:library /r:A2=cs1704.dll /r:A1=c:\\cs1704.dll
// CS1704 expected
extern alias A1;
extern alias A2;
```

# Compiler Error CS1705

Article • 09/15/2021 • 4 minutes to read

Assembly 'AssemblyName1' uses 'TypeName' which has a higher version than referenced assembly 'AssemblyName2'

You are accessing a type that has a higher version number than the version number in a referenced assembly. Typically, this error is caused by the accidental use of two versions of the same assembly.

For example, suppose that you have two assemblies, Asmb1 and Asmb2. Assembly Asmb1 references version 1.0 of assembly Asmb2. Assembly Asmb1 also uses a class `MyClass` that was added to assembly Asmb2 in version 2.0. The compiler has unification rules for binding references, and a reference to `MyClass` in version 2.0 cannot be satisfied by version 1.0.

## Examples

The following more detailed example consists of four code modules:

- Two DLLs that are identical except for a version attribute.
- A third DLL that references the first two.
- A client that references only version 1.0 of the identical DLLs, but accesses a class from version 2.0.

The following code creates the first of the identical DLLs. For information about how to generate a key file, see [KeyFile \(C# Compiler Options\)](#).

C#

```
// CS1705a.cs

// Compile by using the following command:
//      csc /target:library /out:C:\\CS1705.dll /keyfile:mykey.snk
CS1705a.cs

// The DLL is created in the C:\\ directory.

// The AssemblyVersion attribute specifies version 1.0 for this DLL.

[assembly:System.Reflection.AssemblyVersion("1.0")]
public class Class1
{
```

```
    public void Method1() {}  
}
```

The following code defines version 2.0 of the assembly, as specified by the [AssemblyVersionAttribute](#) attribute.

C#

```
// CS1705b.cs  
  
// Compile by using the following command:  
//      csc /target:library /out:CS1705.dll /keyfile:mykey.snk CS1705b.cs  
  
// The DLL is created in the current directory.  
  
// The AssemblyVersion attribute specifies version 2.0 for this DLL.  
  
[assembly:System.Reflection.AssemblyVersion("2.0")]  
public class Class1  
{  
    public void Method1() { }  
}
```

The following code references the two DLL versions that are defined in the preceding code. `AssemblyA` refers to the DLL created by `CS1705a.cs` (version 1.0). `AssemblyB` refers to the DLL created by `CS1705b.cs` (version 2.0). In `ClassC`, two methods are defined. The first, `Return1A`, returns an object of type `Class1A`, which is an alias for `Class1` from version 1.0 of the DLL. The second, `Return1B`, returns an object of type `Class1B`, which is an alias for `Class1` from version 2.0 of the DLL. The definition of `Return1A` creates a dependency on version 1.0; the definition of `Return1B` creates a dependency on version 2.0.

C#

```
// CS1705c.cs  
  
// Compile by using the following command. AssemblyA refers to the DLL  
// created by  
// CS1705a.cs (version 1.0). AssemblyB refers to the DLL created by  
// CS1705b.cs  
// (version 2.0).  
//      csc /target:library /r:AssemblyA=C:\\CS1705.dll  
//      /r:AssemblyB=CS1705.dll CS1705c.cs  
  
extern alias AssemblyA;  
extern alias AssemblyB;  
  
// Class1A is an alias for type Class1 from VS1705a.cs, which is in version
```

```

1.0
// of the assembly. Class1B is an alias for type Class1 from CS1705b.cs,
which
// is in version 2.0 of the assembly.

using Class1A = AssemblyA::Class1;
using Class1B = AssemblyB::Class1;

// Method Return1A in ClassC returns an object of type Class1A, which is
// Class1 from version 1.0 of the DLL. Method Return1B returns an object
// of type Class1B, which is Class1 from version 2.0 of the DLL.

public class ClassC
{
    // The following line creates a dependency on version 1.0 of CS1705.dll.
    // This is not the source of the problem when ClassC is accessed from
    // CS1705d.cs because CS1705d.cs references version 1.0 of the DLL.
    // Therefore, type Class1A and the assembly have the same version.
    public static Class1A Return1A() { return new Class1A(); }

    // The following line creates a dependency on version 2.0 of CS1705.dll.
    // This causes compiler error CS1705 when ClassC is accessed from
    // CS1705d.cs, because CS1705d.cs does not reference version 2.0 of the
    // DLL. Class1B is the alias for Class1 in version 2.0, and CS1705d.cs
    // references version 1.0.
    public static Class1B Return1B() { return new Class1B(); }
}

```

The following code generates compiler error CS1705. It references the DLL created by CS1705a.cs (version 1.0). However, in the `Main` method, the code accesses `ClassC` from CS1705c.cs. `ClassC` uses a type that is defined in CS1705b.cs (version 2.0). This causes compiler error CS1705 because the type has a version number for CS1705.dll that is higher than the referenced version of CS1705.dll.

C#

```

// CS1705d.cs

// Compile by using the following command:
//      csc /reference:C:\\CS1705.dll /reference:CS1705c.dll CS1705d.cs

// C:\\CS1705.dll is version 1.0 of the assembly.

class Tester
{
    static void Main()
    {
        // Return1A returns a type defined in version 1.0.
        ClassC.Return1A().Method1();
        // Return1B returns a type defined in version 2.0.
        ClassC.Return1B().Method1();
    }
}

```

```
    }  
}
```

You can resolve the error in one of the following ways:

- Update the code so that all files use the same version of the DLL.
- Add a reference to version 2.0 of the DLL to CS1705d.cs by using the following command to compile:

```
csc /reference:C:\\CS1705.dll /reference:CS1705.dll /reference:CS1705c.dll  
CS1705d.cs
```

Although the program compiles when you use this command, it still does not run. To enable the program to run, you can provide an application configuration file that includes a [`<dependentAssembly>` element](#) that uses [`<assemblyIdentity>`](#) and [`<codeBase>`](#) child elements to specify the location of version 1.0 of the DLL. For more information about configuration files, see [Configuring Apps](#).

## See also

- [extern alias](#)
- [:: Operator](#)

# Compiler Error CS1706

Article • 09/15/2021 • 2 minutes to read

Expression cannot contain anonymous methods or lambda expressions

You cannot insert an anonymous method inside an expression.

## To correct this error

1. Use a regular `delegate` in the expression.

## Example

The following example generates CS1706.

```
C#  
  
// CS1706.cs  
using System;  
  
delegate void MyDelegate();  
class MyAttribute : Attribute  
{  
    public MyAttribute(MyDelegate d) { }  
}  
  
// Anonymous Method in Attribute declaration is not allowed.  
[MyAttribute(delegate{/* anonymous Method in Attribute declaration */})] //  
CS1706  
class Program  
{  
}
```

# Compiler Error CS1708

Article • 09/15/2021 • 2 minutes to read

Fixed size buffers can only be accessed through locals or fields

A new feature in C# 2.0 is the ability to define an in-line array inside of a `struct`. Such arrays can only be accessed via local variables or fields, and may not be referenced as intermediate values on the left-hand side of an expression. Also, the arrays cannot be accessed by fields that are `static` or `readonly`.

To resolve this error, define an array variable, and assign the in-line array to the variable. Or, remove the `static` or `readonly` modifier from the field representing the in-line array.

## Example

The following sample generates CS1708.

```
C#  
  
// CS1708.cs  
// compile with: /unsafe  
using System;  
  
unsafe public struct S  
{  
    public fixed char name[10];  
}  
  
public unsafe class C  
{  
    public S UnsafeMethod()  
    {  
        S myS = new S();  
        return myS;  
    }  
  
    static void Main()  
    {  
        C myC = new C();  
        myC.UnsafeMethod().name[3] = 'a'; // CS1708  
        // Uncomment the following 2 lines to resolve:  
        // S myS = myC.UnsafeMethod();  
        // myS.name[3] = 'a';  
  
        // The field cannot be static.  
        C._s1.name[3] = 'a'; // CS1708
```

```
// The field cannot be readonly.  
myC._s2.name[3] = 'a'; // CS1708  
}  
  
static readonly S _s1;  
public readonly S _s2;  
}
```

# Compiler Error CS1713

Article • 09/15/2021 • 2 minutes to read

Unexpected error building metadata name for type Typename1—'Reason'

This error is often caused by an internal compiler error. Making some minor changes to your code, such as shortening the length of the name, and then recompiling, may resolve this error.

# Compiler Error CS1714

Article • 09/15/2021 • 2 minutes to read

The base class or interface of TypeName1 could not be resolved or is invalid

You are implementing TypeName1 from a class or interface that either could not be resolved, for example, the compiler was unable to locate it, or is invalid. The solution is to determine which of these two cases it is, and either more correctly specify the location of the type, or fix any compiler errors in the base class.

# Compiler Error CS1715

Article • 09/15/2021 • 2 minutes to read

'Type1': type must be 'Type2' to match overridden member 'MemberName'

This error is the same as [Compiler Error CS0508](#), except that CS0508 now only applies to methods that have return types, while CS1715 applies to properties and indexers that only have 'types' instead of 'return types'.

## Example

The following code generates CS1715.

C#

```
// CS1715.cs
abstract public class Base
{
    abstract public int myProperty
    {
        get;
        set;
    }
}

public class Derived : Base
{
    int myField;
    public override double myProperty // CS1715
    // try the following line instead
    // public override int myProperty
    {
        get { return myField; }
        set { myField+= value; }
    }

    public static void Main()
    {
        Derived d = new Derived();
        d.myProperty = 5;
    }
}
```

# Compiler Error CS1716

Article • 09/15/2021 • 2 minutes to read

Do not use 'System.Runtime.CompilerServices.FixedBuffer' attribute. Use the 'fixed' field modifier instead.

This error arises in an unsafe code section that contains a fixed-size array declaration similar to a field declaration. Do not use this attribute. Instead, use the keyword `fixed`.

## Example

The following example generates CS1716.

C#

```
// CS1716.cs
// compile with: /unsafe
using System;
using System.Runtime.CompilerServices;

public struct UnsafeStruct
{
    [FixedBuffer(typeof(int), 4)] // CS1716
    unsafe public int aField;
    // Use this single line instead of the above two lines.
    // unsafe public fixed int aField[4];
}

public class TestUnsafe
{
    static int Main()
    {
        UnsafeStruct us = new UnsafeStruct();
        unsafe
        {
            if (us.aField[0] == 0)
                return us.aField[1];
            else
                return us.aField[2];
        }
    }
}
```

# Compiler Error CS1719

Article • 09/15/2021 • 2 minutes to read

Error reading Win32 resource file 'File Name' -- 'reason'

An attempt to read the Win32 resource file failed for the reason given in the error, typically something like "file not found" or "access denied." This error is resolved by correcting the problem described by the reason.

# Compiler Error CS1721

Article • 09/15/2021 • 2 minutes to read

Class 'class' cannot have multiple base classes: 'class\_1' and 'class\_2'

The most common cause of this error message is attempting to use multiple inheritance. A class in C# may only inherit directly from one class. However, a class can implement any number of interfaces.

## Example

The following example shows one way in which CS1721 is generated:

```
C#  
  
// CS1721.cs  
public class A {}  
public class B {}  
public class MyClass : A, B {} // CS1721
```

## To correct this error

The following are different ways to correct this error:

- Make class `B` inherit from `A`, and `MyClass` inherit from `B`:

```
C#  
  
public class A {}  
public class B : A {}  
public class MyClass : B {}
```

- Declare `B` as an interface. Make `MyClass` inherit from the interface `B`, and the class `A`:

```
C#  
  
public class A {}  
public interface B {}  
public class MyClass : A, B {}
```

## See also

- [Polymorphism](#)
- [Interfaces](#)

# Compiler Error CS1722

Article • 09/15/2021 • 2 minutes to read

Base class 'class' must come before any interfaces

When specifying a class to inherit from and interfaces to implement, the class name must be specified first.

## Example

The following sample generates CS1722.

C#

```
// CS1722.cs
// compile with: /target:library
public class A {}
interface I {}

public class MyClass : I, A {}    // CS1722
public class MyClass2 : A, I {}   // OK
```

# Compiler Error CS1724

Article • 09/15/2021 • 2 minutes to read

Value specified for the argument to  
'System.Runtime.InteropServices.DefaultCharsetAttribute' is not valid

This error is generated by an invalid argument to the [DefaultCharsetAttribute](#) class.

## Example

The following example generates CS1724.

C#

```
// CS1724.cs
using System.Runtime.InteropServices;
// To resolve, replace 42 with a valid CharSet value.
[module:DefaultCharsetAttribute((CharSet)42)] // CS1724
class C {

    [DllImport("F.Dll")]
    extern static void FW1Named();

    static void Main() {}
}
```

# Compiler Error CS1725

Article • 09/15/2021 • 2 minutes to read

Friend assembly reference 'reference' is invalid. InternalsVisibleTo declarations cannot have a version, culture, public key token, or processor architecture specified.

You cannot add a version culture in a friend assembly reference. Partial classes should be visible to friend assemblies.

## Example

The following sample generates CS1725.

C#

```
// CS1725.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("partial01,version=1.1.0.0")]    // CS1725
// try the following line instead
// [assembly:InternalsVisibleTo("partial01")]

partial class TestClass
{
    public static string strBar = "my string";
}
```

## See also

- [How to: Create Signed Friend Assemblies](#)



# Compiler Error CS1726

Article • 09/15/2021 • 2 minutes to read

Friend assembly reference 'reference' is invalid. Strong-name signed assemblies must specify a public key in their `InternalsVisibleTo` declarations.

A strong name signed assembly can only grant friend assembly access, made with the `InternalsVisibleToAttribute`, to other strongly signed assemblies.

To resolve CS1726, either sign (give a strong name to) the assembly to which you want to grant friend access, or don't grant friend access.

For more information, see [Friend Assemblies](#).

## Example

The following sample generates CS1726.

```
C#  
  
// Save this code as CS1726.cs  
  
// Run the following command to create CS1726.key:  
//     sn -k CS1726.key  
  
// Then compile by using the following command:  
//     csc /keyfile:CS1726.key /target:library CS1726.cs  
  
using System.Runtime.CompilerServices;  
  
// The following line causes compiler error CS1726.  
[assembly: InternalsVisibleTo("UnsignedAssembly")]  
  
// To get rid of the error, try the following line instead.  
//[assembly: InternalsVisibleTo("SignedAssembly,  
PublicKey=002400000480000094000000060200000024000052534131000400000100010003  
1d7b6f3abc16c7de526fd67ec2926fe68ed2f9901afbc5f1b6b428bf6cd9086021a0b38b76bc  
340dc6ab27b65e4a593fa0e60689ac98dd71a12248ca025751d135df7b98c5f9d09172f7b62d  
abdd302b2a1ae688731ff3fc7a6ab9e8cf39fb73c60667e1b071ef7da5838dc009ae0119a9cb  
ff2c581fc0f2d966b77114b2c4")]  
  
class A { }
```

## See also

- How to: Create Signed Friend Assemblies

# Compiler Error CS1727

Article • 09/15/2021 • 2 minutes to read

Cannot send error report automatically without authorization. Please visit " to authorize sending error report.

The Web site listed in the error text explains how to enable automatic error reporting for Visual Studio 2005 command line tools.

## Example

The following sample generates CS1727.

C#

```
// CS1727.cs
// compile with: /errorreport:send
// CS1727 expected
class Test
{
    static void Main(){}
}
```

# Compiler Error CS1728

Article • 09/15/2021 • 2 minutes to read

Cannot bind delegate to 'member' because it is a member of 'type'

You cannot bind delegates to members of `Nullable` value types.

## Example

The following sample generates CS1728:

C#

```
// CS1728.cs
class Test
{
    delegate T GetT<T>();
    delegate T GetT1<T>(T t);

    delegate bool E(object o);
    delegate int I();
    delegate string S();

    static void Main()
    {
        int? x = null;
        int? y = 5;

        GetT<int> d1 = x.GetValueOrDefault; // CS1728
        GetT<int> d2 = y.GetValueOrDefault; // CS1728
        GetT1<int> d3 = x.GetValueOrDefault; // CS1728
        GetT1<int> d4 = y.GetValueOrDefault; // CS1728
    }
}
```

# Compiler Error CS1729

Article • 09/15/2021 • 2 minutes to read

'type' does not contain a constructor that takes 'number' arguments.

This error occurs when you either directly or indirectly invoke the constructor of a class but the compiler cannot find any constructors with the same number of parameters. In the following example, the `test` class has no constructors that take any arguments. It therefore has only a parameterless constructor that takes zero arguments. Because in the second line in which the error is generated, the derived class declares no constructors of its own, the compiler provides a parameterless constructor. That constructor invokes a parameterless constructor in the base class. Because the base class has no such constructor, CS1729 is generated.

## To correct this error

1. Adjust the number of parameters in the call to the constructor.
2. Modify the class to provide a constructor with the parameters you must call.
3. Provide a parameterless constructor in the base class.

## Example

The following example generates CS1729:

```
C#  
  
// cs1729.cs  
class Test  
{  
    static int Main()  
    {  
        // Class Test has only a parameterless constructor, which takes no  
        // arguments.  
        Test test1 = new Test(2); // CS1729  
        // The following line resolves the error.  
        Test test2 = new Test();  
  
        // Class Parent has only one constructor, which takes two int  
        // parameters.  
        Parent exampleParent1 = new Parent(10); // CS1729  
        // The following line resolves the error.  
        Parent exampleParent2 = new Parent(10, 1);
```

```
        return 1;
    }
}

public class Parent
{
    // The only constructor for this class has two parameters.
    public Parent(int i, int j) { }

}

// The following declaration causes a compiler error because class Parent
// does not have a constructor that takes no arguments. The declaration of
// class Child2 shows how to resolve this error.
public class Child : Parent { } // CS1729

public class Child2 : Parent
{
    // The constructor for Child2 has only one parameter. To access the
    // constructor in Parent, and prevent this compiler error, you must
    // provide
    // a value for the second parameter of Parent. The following example
    provides 0.
    public Child2(int k)
        : base(k, 0)
    {
        // Add the body of the constructor here.
    }
}
```

## See also

- [Inheritance](#)
- [Constructors](#)

# Compiler Error CS1730

Article • 03/15/2023 • 2 minutes to read

Assembly and module attributes must precede all other elements defined in a file except using clauses and extern alias declarations.

An attribute applied at the assembly level cannot appear after any type definitions.

## To correct this error

1. Move the attribute to the top of the file, but below the `using` directives and `extern alias` declarations.

## Example

The following code generates CS1730:

```
C#  
  
// cs1730.cs  
class Test  
{  
}  
[assembly: System.Attribute] // CS1730
```

## See also

- [Attributes](#)

# Compiler Error CS1731

Article • 09/15/2021 • 2 minutes to read

Cannot convert 'expression' to delegate because some of the return types in the block are not implicitly convertible to the delegate return type.

This error is generated when a lambda expression or anonymous method has a return type that is not compatible with the delegate's return type.

## To correct this error

1. Change the return type of either the delegate or the expression.

## Example

The following code generates CS1731:

C#

```
class CS1731
{
    delegate double D();
    D d = () => { return "Who knows the real sword of Gryffindor?"; };
}
```

# Compiler Error CS1732

Article • 09/15/2021 • 2 minutes to read

Expected parameter.

This error is produced when a lambda expression contains a comma following an input parameter but does not specify the following parameter.

## To correct this error

1. Either remove the comma, or add the input parameter that the compiler expects to find after the comma.

## Example

The following example produces CS1732:

```
C#  
  
// cs1732.cs  
// compile with: /target:library  
class Test  
{  
    delegate void D(int x, int y);  
    static void Main()  
    {  
        D d = (x,) => { }; // CS1732  
    }  
}
```

# Compiler Error CS1733

Article • 09/15/2021 • 2 minutes to read

Expected expression.

This error is produced whenever the compiler is expecting an expression on the line where the error occurred. In the following example, the trailing comma in the initializer indicates to the compiler that another expression will follow.

## To correct this error

- Provide the missing expression.
- Remove the tokens that are causing the compiler to expect an expression.

## Example

The following example produces CS1733 because of the trailing comma:

C#

```
// cs1733.cs
using System.Collections.Generic;
public class Test
{
    public static void Main()
    {
        List<int> list = new List<int>() {{ 1, 2, }};// CS1733
    }
}
```

# Compiler Error CS1736

Article • 09/17/2022 • 2 minutes to read

Default parameter value for must be a compile-time constant

## Example

The following sample generates CS1736:

```
C#  
  
// CS1736.cs  
  
public unsafe class C  
{  
    static void F(int i = G())  
    {  
        // ...  
    }  
    static int G() => 0;
```

A default parameter value is evaluated upon the invocation of the method. What a value may be when the method is eventually invoked cannot be pre-determined at declaration-time unless that value is constant at compile-time.

## To correct this error

If a dynamically evaluated value is required, consider using a compile-time constant as a marker value which is then checked at run-time:

```
C#  
  
static void F(int i = -1)  
{  
    if(i == -1) i = G();  
    //...  
}
```

# Compiler Error CS1737

Article • 10/07/2022 • 2 minutes to read

Optional parameters must appear after all required parameters

The compiler does not support optional parameters being declared before required parameters. All optional parameters must be after all required parameters.

## Example

The following sample generates CS1737:

```
C#  
  
// CS1737.cs (7,45)  
class C  
{  
    static void F(object? x)  
    {  
        G(y: x);  
    }  
    static void G(object? x = null, object y)  
    {  
    }  
}
```

## To correct this error

The signature for this method may be changed without effecting existing code that calls the method because a value for the optional parameter has not been used. For example:

```
C#  
  
// CS1737.cs (7,45)  
class C  
{  
    static void F(object? x)  
    {  
        G(y: x);  
    }  
    static void G(object y, object? x = null)  
    {  
    }  
}
```

# Compiler Error CS1739

Article • 09/20/2022 • 2 minutes to read

The best overload for does not have a parameter named

## Example

The following sample generates CS1739:

```
C#  
  
// CS1739.cs (11,31)  
using System;  
  
public class A  
{  
    public int this[Range range] => 42;  
}  
public class C  
{  
    public static void Main()  
    {  
        Console.Write(new A()[param: 1..^1]);  
    }  
}
```

## To correct this error

Use the name of the parameter as it is declared in the member to correct this error.

```
C#  
  
public static void Main()  
{  
    Console.Write(new A()[range: 1..^1]);  
}
```

# Compiler Error CS1740

Article • 09/20/2022 • 2 minutes to read

Named argument cannot be specified multiple times

## Example

The following sample generates CS1740:

The compiler does not support passing more than one value for a named argument.

C#

```
// CS1740.cs (9,17)
class C
{
    static void M(params int[] x)
    {
    }
    static void Main()
    {
        M(x: 1, x: 2);
    }
}
```

## To correct this error

Pick which value should be passed as the argument and remove the other:

C#

```
M(x: 1);
```

# Compiler Error CS1741

Article • 09/20/2022 • 2 minutes to read

A `ref` or `out` parameter cannot have a default value

Using `ref` or `out` in a method signature causes arguments to be passed by reference, making the parameter an alias for the argument. Since the parameter must be a variable, should the default value be used, no variable would exist as the alias for the argument.

## Example

The following sample generates CS1741:

```
C#  
  
// CS1741.cs (6,21)  
class Program  
{  
    static void Main(string[] args)  
    {  
        void RefOut(ref int x = 2)  
        {  
            x++;  
        }  
        int y = 2;  
        RefOut(ref y);  
    }  
}
```

## To correct this error

In this example, removing the `ref` modifier from the method signature would be a logic error--the method's body would have no visible side effects when executed. To correct this error, remove the unnecessary default value from the method signature:

```
C#  
  
static void Main(string[] args)  
{  
    void RefOut(ref int x)  
    {  
        x++;  
    }  
    int y = 2;
```

```
    RefOut(ref y);  
}
```

# Compiler Error CS1742

Article • 09/20/2022 • 2 minutes to read

An array access may not have a named argument specifier

## Example

The following sample generates CS1742:

```
C#  
  
// CS1742.cs (0,0)  
public class B  
{  
    static void Main()  
    {  
        int[] arr = { };  
        int s = arr[arr: 1];  
    }  
}
```

An array may not be declared with a named argument. This code generates CS1742 because using the name `arr` to refer to an argument when accessing the array is not syntactically correct.

## To correct this error

Remove the use of named arguments when accessing an array to correct this error:

```
C#  
  
static void Main()  
{  
    int[] arr = { };  
    int s = arr[1];  
}
```

# Compiler Error CS1750

Article • 10/07/2022 • 2 minutes to read

A value of type cannot be used as a default parameter because there are no standard conversions to type

## Example

The following sample generates CS1750:

C#

```
public struct S
{
    public override string ToString() { return "S::ToString"; }
}
public class A
{
    public static S Goo(S p = 42) { return p; }
}
```

There is no standard conversion between `int` and the newly declared struct `S`, using an `int` compile-time constant to initialize an instance of struct `S` results in CS1750. Adding a user-defined conversion operator (e.g., `public static implicit operator S(int n) => ...`) will not correct this error because that does not add a *standard conversion*.

# Compiler Error CS1751

Article • 10/27/2022 • 2 minutes to read

Cannot specify a default value for a parameter array.

## Example

The following sample generates CS1751:

```
C#  
  
// CS1751.cs  
void Method(params object[] values = null)  
{  
}
```

## Solution

```
C#  
  
// Explicitly passing null  
object[] values = null;  
Method(values);  
  
void Method(params object[] values)  
{  
    if (values == null)  
    {  
    }  
}
```

For more information, see [Params](#).

# Compiler Error CS1763

Article • 09/20/2022 • 2 minutes to read

A default parameter value of a reference type other than string can only be initialized with null

## Example

The following sample generates CS1763:

```
C#  
  
// CS1763.cs (0,0)  
class Program  
{  
    public void Goo<T, U>(T t = default(U)) where U : T  
    {  
    }  
    static void Main(string[] args)  
    {  
    }  
}
```

This example generates CS1763 because the `Goo<T,U>` parameter is declared with a default value of `default(U)` when the type of the parameter is `T`, despite the constraint that `U` derive from base class `T`.

## To correct this error

Changing `default(U)` to use the corresponding type argument corrects this error:

```
C#  
  
public void Goo<T, U>(T t = default(T)) where U : T  
{  
}
```

# Compiler Error CS1900

Article • 03/11/2022 • 2 minutes to read

Warning level must be in the range 0-4

The [WarningLevel](#) compiler option can only take one of five possible values (0, 1, 2, 3, or 4). Any other value passed to `/warn` will result in CS1900.

The following sample generates CS1900:

C#

```
// CS1900.cs
// compile with: /W:5
// CS1900 expected
class x
{
    public static void Main()
    {
    }
}
```

## ⓘ Note

The compiler no longer generates this error. Starting with C# 9, values greater than 4 represent "warning waves" and are valid, even if no warnings are defined for that wave yet.

# Compiler Error CS1902

Article • 09/15/2021 • 2 minutes to read

Invalid option 'option' for /debug; must be full or pdbonly

An invalid option was passed to the [DebugType](#) compiler option.

The following sample generates CS1902:

C#

```
// CS1902.cs
// compile with: /debug:x
// CS1902 expected
class x
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS1906

Article • 09/15/2021 • 2 minutes to read

Invalid option 'option'; Resource visibility must be either 'public' or 'private'

This error indicates an invalid [Resources \(Embed Resource File to Output\)](#) or [LinkResources \(Link to .NET Framework Resource\)](#) command line option. Check the syntax of the `/resource` or `/linkresource` command line option, and make sure that the accessibility modifier used is either `public` or `private`.

# Compiler Error CS1908

Article • 09/15/2021 • 2 minutes to read

The type of the argument to the `DefaultValue` attribute must match the parameter type.

This error is generated when you use the wrong argument for the `DefaultValueAttribute` attribute value. Use a value that matches the parameter type.

## Example

The following sample generates CS1908.

C#

```
// CS1908.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface ISomeInterface
{
    void Bad([Optional] [DefaultValue("true")] bool b);    // CS1908

    void Good([Optional] [DefaultValue(true)] bool b);    // OK
}
```

# Compiler Error CS1909

Article • 09/15/2021 • 2 minutes to read

The `DefaultValue` attribute is not applicable on parameters of type 'type'

CS1909 is generated when you use a `DefaultValue` attribute that is not applicable on the parameter type.

## Example

The following sample generates CS1909.

C#

```
// CS1909.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface ISomeInterface
{
    void Test1([DefaultParameterValue(new int[] {1, 2})] int[] arr1);    // CS1909

    void Test2([DefaultParameterValue("Test String")] string s);    // OK
}
```

# Compiler Error CS1910

Article • 09/15/2021 • 2 minutes to read

Argument of type 'type' is not applicable for the DefaultValue attribute

For parameters whose type is object, the argument of the `DefaultParameterValueAttribute` must be `null`, an integral type, a floating point, `bool`, `string`, `enum`, or `char`. The argument can not be of type `Type` or any array type.

## Example

The following sample generates CS1910.

C#

```
// CS1910.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface MyI
{
    void Test([DefaultParameterValue(typeof(object))] object o); // CS1910
}
```

# Compiler Error CS1912

Article • 09/15/2021 • 2 minutes to read

Duplicate initialization of member 'name'.

An object initializer can initialize each member only one time.

## To correct this error

1. Remove the second initialization of the member in the object initializer.

## Example

The following code generates CS1912 because `memberA` is initialized two times:

```
C#  
  
// cs1912.cs  
using System.Linq;  
  
public class TestClass  
{  
    public int memberA { get; set; }  
    public int memberB { get; set; }  
}  
  
public class Test  
{  
    static void Main()  
    {  
        TestClass tc = new TestClass() { memberA = 5, memberA = 6, memberB =  
2}; // CS1912  
    }  
}
```

## See also

- [Object and Collection Initializers](#)



# Compiler Error CS1913

Article • 10/27/2021 • 2 minutes to read

Member 'name' cannot be initialized. It is not a field or property.

Object initializers can only be used to initialize accessible fields or properties.

## To correct this error

1. Initialize the class member in a regular constructor or other initialization method.

## Example

The following example generates CS1913:

```
C#  
  
// cs1912.cs  
class A  
{  
    public delegate void D();  
    public event D myEvent;  
}  
  
public class Test  
{  
    static void Main()  
{  
        A a = new A() {myEvent = M}; // CS1913  
    }  
  
    public void M(){}
}
```

## See also

- [The C# type system](#)



# Compiler Error CS1914

Article • 09/15/2021 • 2 minutes to read

Static field 'name' cannot be assigned in an object initializer

Object initializers by definition initialize objects, or instances, of classes. They cannot be used to initialize a `static` field of a type. No matter how many instances of a class are created, there is only one copy of a `static` field.

## To correct this error

1. Either change the field to an instance field in the type, or remove the attempt to initialize it from the object initializer.

## Example

The following code generates CS1914 because the initializer tries to initialize the `TestClass.Number` field, which is `static`:

C#

```
// cs1914.cs
using System.Linq;
public class TestClass
{
    public string Message { get; set; }
    public static int Number { get; set; }
}
class Test
{
    static void Main()
    {
        TestClass b = new TestClass() { Message = "Hello", Number = "555-
1212" }; // CS1914
    }
}
```

# Compiler Error CS1917

Article • 09/15/2021 • 2 minutes to read

Members of read-only field 'name' of type 'struct name' cannot be assigned with an object initializer because it is of a value type.

Read-only fields that are value types can only be assigned in a constructor.

## To correct this error

- Change the struct to a class type.
- Initialize the struct with a constructor.

## Example

The following code generates CS1917:

C#

```
// cs1917.cs
class CS1917
{
    public struct TestStruct
    {
        public int i;
    }
    public class C
    {
        public readonly TestStruct str = new TestStruct();
        public static int Main()
        {
            C c = new C { str = { i = 1 } }; // CS1917
            return 0;
        }
    }
}
```

# Compiler Error CS1918

Article • 09/15/2021 • 2 minutes to read

Members of property 'name' of type 'type' cannot be assigned with an object initializer because it is of a value type.

This error occurs when you try to use an object initializer to initialize the properties of a struct type that is itself a property of the class that is being initialized.

## To correct this error

1. If you must fully initialize the fields of the property in the object initializer, change the struct to a class type. Otherwise, initialize the struct members in a separate method call after you create the object by using the object initializer.

## Example

The following example generates CS1918:

```
C#  
  
// cs1918.cs  
public struct MyStruct  
{  
    public int i;  
  
}  
public class Test  
{  
    private MyStruct str = new MyStruct();  
    public MyStruct Str  
    {  
        get  
        {  
            return str;  
        }  
    }  
    public static int Main()  
    {  
        Test t = new Test { Str = { i = 1 } }; // CS1918  
        return 0;  
    }  
}
```

## See also

- [Object and Collection Initializers](#)

# Compiler Error CS1919

Article • 02/25/2023 • 2 minutes to read

Unsafe type 'type name' cannot be used in object creation.

The `new` operator creates objects only on the managed heap. However, you can create objects in unmanaged memory indirectly by using the interoperability capabilities of the language to call native methods that return pointers.

## To correct this error

1. Use a safe type in the new object creation expression. For example, use `char` or `int` instead of `char*` or `int*`.
2. If you must create objects in unmanaged memory, use a Win32 or COM method or else write your own function in C or C++ and call it from C#.

## Example

The following example generates CS1919 because a pointer type is unsafe:

```
C#  
  
// cs1919.cs  
// Compile with: /unsafe  
unsafe public class C  
{  
    public static int Main()  
    {  
        var col1 = new int* { }; // CS1919  
        var col2 = new char* { }; // CS1919  
        return 1;  
    }  
}
```

## See also

- [Interoperability](#)

# Compiler Error CS1920

Article • 09/15/2021 • 2 minutes to read

Element initializer cannot be empty.

A collection initializer consists of a sequence of element initializers. The element initializers do not have to be enclosed in braces unless they contain an assignment expression. However, if you do supply braces, they cannot be empty. If the element initializer is an object initializer, the braces may be empty as long as the initializer contains a new object creation expression.

## To correct this error

- Add the missing expression between the braces.
- If the expression is intended to be an object initializer, add the new object creation expression in front of the braces.

## Example

The following example generates CS1920:

C#

```
// cs1920.cs
using System.Collections.Generic;
public class Test
{
    public static int Main()
    {
        // Error. Empty initializer
        // for inner list.
        List<List<int>> collection =
            new List<List<int>>() { { } }; // CS1920

        // OK. No initializer for inner list.
        List<List<int>> collection2 =
            new List<List<int>>() { };

        // OK. Inner list is initialized
        // to one List<int> with zero elements.
        List<List<int>> collection3 =
            new List<List<int>>() { new List<int> { } };
        return 0;
    }
}
```

## See also

- [Object and Collection Initializers](#)

# Compiler Error CS1921

Article • 09/15/2021 • 2 minutes to read

The best overloaded method match for 'method' has wrong signature for the initializer element. The initializable Add must be an accessible instance method.

This error is generated when you try to use a collection initializer with a class that has no public non-static `Add` method. If the `Add` method is not accessible because of its protection level (`private`, `protected`, `internal`) then you will get CS0122, so that this error probably means that the method is defined as `static`.

## Example

The following example generates CS1921:

C#

```
// cs1921.cs
using System.Collections;
public class C : CollectionBase
{
    public static void Add(int i)
    {
    }
}
public class Test
{
    public static void Main()
    {
        var collection = new C { 1, 2, 3 }; // CS1921
    }
}
```

## See also

- [Object and Collection Initializers](#)



# Compiler Error CS1922

Article • 09/15/2021 • 2 minutes to read

Collection initializer requires its type 'type' to implement System.Collections.IEnumerable.

In order to use a collection initializer with a type, the type must implement `IEnumerable`. This error can occur if you accidentally use collection initializer syntax when you meant to use an object initializer.

## To correct this error

- If the type does not represent a collection, use object initializer syntax instead of collection initializer syntax.
- If the type does represent a collection, modify it to implement `IEnumerable` before you can use collection initializers to initialize objects of that type.
- If the type represents a collection and you do not have access to the source code, just initialize its elements by using their class constructors or other initialization methods.

## Example

The following code produces CS1922:

C#

```
// cs1922.cs
public class Test
{
    public static void Main()
    {
        // Collection initializer.
        var tc = new TestClass {1, "hello"}; // CS1922

        // Object initializer.
        var tc2 = new TestClass { memberA = 1, memberB = "hello" }; // OK
    }
}

public class TestClass
{
    public int memberA { get; set; }
```

```
    public string memberB { get; set; }  
}
```

## See also

- [Object and Collection Initializers](#)

# Compiler Error CS1925

Article • 09/15/2021 • 2 minutes to read

Cannot initialize object of type 'type' with a collection initializer.

Collection initializers are only allowed for collection classes that meet certain criteria. For more information, see [Object and Collection Initializers](#). This error is also produced when you try to use the short form of an array initializer nested inside a collection initializer.

## To correct this error

1. Initialize the object by calling its constructors and methods.

## Example

The following code generates CS1925:

```
C#  
  
// cs1925.cs  
public class Student  
{  
    public int[] Scores;  
}  
  
class Test  
{  
    static void Main(string[] args)  
    {  
        Student student = new Student { Scores = { 1, 2, 3 } }; // CS1925  
    }  
}
```

# Compiler Error CS1926

Article • 09/15/2021 • 2 minutes to read

Error reading Win32 manifest file 'filename' -- 'error'.

This error is generated when the following conditions are true:

1. The **/win32manifest** option is specified either on the command line or by right-clicking the **Project** icon in **Solution Explorer**, pointing to **Add**, clicking **New Item**, and then clicking **Application Manifest File**.
2. The file is either corrupted or missing.

## To correct this error

1. Remove the option.
2. Replace, repair, or regenerate the file.

## Example

The following example generates CS1926 when it is compiled with a corrupted or missing win32 manifest file:

```
C#  
  
// cs1926.cs  
// Compile with: /win32manifest: ../../app.manifest  
// CS1926  
class Test  
{  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Win32Manifest \(C# Compiler Options\)](#)



# Compiler Error CS1928

Article • 09/15/2021 • 2 minutes to read

'Type' does not contain a definition for 'method' and the best extension method overload 'method' has some invalid arguments.

This error is produced when the compiler cannot find a class member with the name of the method you have called. It can find an extension method with that name, but not with a signature that matches the types you passed in with your method call.

## To correct this error

1. Pass in types that match an existing extension method or class method.

## Example

The following code generates CS1928:

```
C#  
  
// cs1928.cs  
class Test  
{  
    static void Main()  
    {  
        Test t = new Test();  
        t.M("hi"); // CS1928  
    }  
}  
static class Ext  
{  
    public static void M(this Test t, int i)  
    {  
    }  
}
```

This error is often accompanied by CS1503: Argument 'n': cannot convert from 'typeA' to 'typeB'.

## See also

- [Extension Methods](#)



# Compiler Error CS1929

Article • 09/15/2021 • 2 minutes to read

Instance argument: cannot convert from 'typeA' to 'typeB'.

This error is generated when you try to invoke an extension method from a class that it does not extend. In the example shown here, the extension method is defined for the derived class **A**, but not for the base class **B**.

## To correct this error

1. Create a new extension method for the type where you have to invoke it, or else move the call into an object of the type that the existing method extends.

## Example

The following code generates CS1928 and CS1929:

```
C#  
  
// cs1929.cs  
using System.Linq;  
using System.Collections;  
  
static class Ext  
{  
    public static void ExtMethod(this A a)  
    {  
    }  
}  
  
class A : B  
{  
}  
  
class B  
{  
    static void Main()  
    {  
        B b = new B();  
        b.ExtMethod(); // CS1929  
    }  
}
```

## See also

- [Extension Methods](#)

# Compiler Error CS1930

Article • 09/15/2021 • 2 minutes to read

The range variable 'name' has already been declared

The range variable in a query expression is in scope until the query expression terminates. It must therefore have a unique identifier.

## To correct this error

1. Give a unique name to each range variable that is introduced in a query expression.

## Example

The following example generates CS1930 because the identifier `num` is used for the range variable in the `from` clause and for the range variable introduced by the `let` clause.

```
C#  
  
// cs1930.cs  
using System.Linq;  
class Program  
{  
    static void Main()  
    {  
        int[] nums = { 0, 1, 2, 3, 4, 5 };  
        var query = from num in nums  
                    let num = 3 // CS1930  
                    select num;  
    }  
}
```

## See also

- [LINQ Query Expressions](#)



# Compiler Error CS1931

Article • 09/15/2021 • 2 minutes to read

The range variable 'variable' conflicts with a previous declaration of 'variable'.

The declaration of a range variable, just like every other declaration, must have an identifier which is unique within the variable's declaration space.

## To correct this error

1. Give the range variable a unique name.

## Example

The following code generates CS1931 because the identifier `x` is used both as a local variable in `Main` and as the range variable in the query expression:

```
C#  
  
// cs1931.cs  
class Test  
{  
    static void Main()  
    {  
        int x = 1;  
        var y = from x in Enumerable.Range(1, 100) // CS1931  
                select x;  
    }  
}
```

## See also

- [LINQ Query Expressions](#)



# Compiler Error CS1932

Article • 09/15/2021 • 2 minutes to read

Cannot assign 'expression' to a range variable.

The compiler must be able to infer the type of a range variable, whether it is introduced in a `from` clause or a `let` clause. It cannot be null because null is not a type, and it cannot be assigned with an expression of an unsafe type.

## To correct this error

- Remove the assignment that is not valid.
- Explicitly cast the expression to an allowed type

## Example

The following code generates CS1932 because the type of the range variable cannot be inferred. Cast the value to the intended type to fix the error, as shown in the following example.

C#

```
// CS1932.cs
using System.Linq;
class Test
{
    static void Main()
    {

        var x = from i in Enumerable.Range(1, 100)
                let k = null // CS1932
                // Try the following line instead.
                let k = (string) null
                select i;
    }
}
```

## See also

- [LINQ Query Expressions](#)



# Compiler Error CS1933

Article • 09/15/2021 • 2 minutes to read

Expression cannot contain query expressions

Some variables cannot be initialized with a query expression. Constants cannot be initialized with query expressions because constants may only be initialized with some combination of literals, named constants, and mathematical operators.

## To correct this error

1. Remove the modifier from the query variable.

## Example

The following example generates CS1933:

```
C#  
  
// cs1933.cs  
using System.Linq;  
using System.Collections;  
  
class Program  
{  
    const IEnumerable e = from x in new[] { 1, 2, 3 } select x; // CS1933  
    static int Main()  
    {  
        return 1;  
    }  
}
```

### ⓘ Note

This compiler error is no longer used in Roslyn. The previous example generates CS0133 when compiled with Roslyn.

## See also

- [LINQ in C#](#)



# Compiler Error CS1934

Article • 09/15/2021 • 2 minutes to read

Could not find an implementation of the query pattern for source type 'type'. 'method' not found. Consider explicitly specifying the type of the range variable 'name'.

This error is produced if a query expression specifies a data source for which no standard query operators are implemented. One way to produce this error is to specify an `ArrayList` without giving an explicit type for the range variable.

## To correct this error

1. In the following example, the solution is to just specify the type of the range variable:

C#

```
var q = from int x in list
```

## Example

The following example shows one way to produce CS1934:

C#

```
// cs1934.cs
using System.Linq;
using System.Collections;
static class Test
{
    public static void Main()
    {
        var list = new ArrayList { 0, 1, 2, 3, 4, 5 };
        var q = from x in list // CS1934
                select x + 1;
    }
}
```

## See also

- [How to query an `ArrayList` with LINQ](#)



# Compiler Error CS1935

Article • 09/15/2021 • 2 minutes to read

Could not find an implementation of the query pattern for source type 'type'. 'method' not found. Are you missing a using directive for 'System.Linq'?

The source type in a query must be `IEnumerable`, `IEnumerable<T>`, or a derived type, or a type for which you or someone else has implemented the standard query operators. If the source type is an `IEnumerable` or `IEnumerable<T>`, you must add a `using` directive for the `System.Linq` namespace in order to bring the standard query operator extension methods into scope. Custom implementations of the standard query operators must be brought into scope in the same way, with a `using` directive and, if necessary, a reference to the assembly.

## To correct this error

Add the required using directives and references to the project.

## Example

The following code generates CS1935 because the `using` directive for `System.Linq` is commented out:

```
C#  
  
// cs1935.cs  
// CS1935  
using System;  
using System.Collections.Generic;  
// using System.Linq;  
  
class Test  
{  
    static int Main()  
    {  
        int[] nums = { 0,1,2,3,4,5 };  
        IEnumerable<int> e = from n in nums  
                             where n > 3  
                             select n;  
        return 0;  
    }  
}
```

## See also

- [Standard Query Operators Overview](#)

# Compiler Error CS1936

Article • 09/15/2021 • 2 minutes to read

Could not find an implementation of the query pattern for source type 'type'. 'method' not found.

In order to query a source type, that type must implement the standard query operator methods that you are invoking in the query. The implementation can be either in the form of class members or extension methods that are brought into scope with the appropriate `using` directive.

## To correct this error

- Make sure that you are querying a collection of objects, not an individual object.
- Make sure that you have specified the necessary `using` directives.

## Example

The following example produces CS1936:

```
C#  
  
// cs1936.cs  
using System.Collections;  
using System.Linq;  
class Test  
{  
    static int Main()  
    {  
        object obj;  
        IEnumerable e = from x in obj // CS1936  
                        select x;  
        return 0;  
    }  
}
```

This error typically occurs when you accidentally try to query an object of some type instead of a collection of those objects.

## See also

- [Standard Query Operators Overview](#)



# Compiler Error CS1937

Article • 09/15/2021 • 2 minutes to read

The name 'name' is not in scope on the left side of 'equals'. Consider swapping the expressions on either side of 'equals'.

The `equals` keyword is a special operator that is used in a `join` clause to determine equality between two expressions. The range variable for the left side source sequence is in scope on the left side of equals, and the range variable for the right side source is only in scope on the left side of equals. You can verify this by experimenting with IntelliSense in the following code example.

## To correct this error

1. Swap the position of the two range variables as shown in the commented line in the following example:

## Example

The following example generates CS1937.

C#

```
// cs1937.cs
using System.Linq;
class Test
{
    static void Main()
    {
        int[] sourceA = { 1, 2, 3, 4, 5 };
        int[] sourceB = { 3, 4, 5, 6, 7 };

        var query = from a in sourceA
                    join b in sourceB on b equals a // CS1937
                    // Try the following line instead.
                    //join b in sourceB on a equals b
                    select new { a, b };
    }
}
```

The left side is generally called the "outer" side and the right is generally called the "inner" side.

## See also

- [join clause](#)

# Compiler Error CS1938

Article • 09/15/2021 • 2 minutes to read

The name 'name' is not in scope on the right side of 'equals'. Consider swapping the expressions on either side of 'equals'.

The `equals` keyword is a special operator that is used in a `join` clause to determine equality between two expressions. The range variable for the left side source sequence is in scope on the left side of equals, and the range variable for the right side source is only in scope on the left side of equals. You can verify this by experimenting with IntelliSense in the following code example.

## To correct this error

1. Swap the position of the two range variables as shown in the commented line in the following example:

## Example

The following code generates CS1938:

```
C#  
  
// cs1938.cs  
using System.Linq;  
class Test  
{  
    static void Main()  
    {  
        int[] sourceA = { 1, 2, 3, 4, 5 };  
        int[] sourceB = { 3, 4, 5, 6, 7 };  
  
        var query = from a in sourceA  
                    join b in sourceB on b equals a // CS1938  
                    // Try the following line instead.  
                    // join b in sourceB on a equals b  
                    select new { a, b };  
    }  
}
```

## See also

- [join clause](#)



# Compiler Error CS1939

Article • 09/15/2021 • 2 minutes to read

Cannot pass the range variable 'name' as an out or ref parameter.

A range variable is a read-only variable that is introduced in a query expression and serves as an identifier for each successive element in a source sequence. Because it cannot be modified in any way, there is no point in passing it by `ref` or `out`. Therefore, both operations are not valid.

## To correct this error

1. Pass the range variable by value.

## Example

The following example generates CS1939:

```
C#  
  
// cs1939.cs  
using System.Linq;  
class Test  
{  
    public static void F(ref int i)  
    {  
    }  
    public static void Main()  
    {  
        var list = new int[] { 0, 1, 2, 3, 4, 5 };  
        var q = from x in list  
                let k = x  
                select Test.F(ref x); // CS1939  
    }  
}
```

# Compiler Error CS1940

Article • 09/15/2021 • 2 minutes to read

Multiple implementations of the query pattern were found for source type 'type'.  
Ambiguous call to 'method'.

This error is generated when multiple implementations of a query method are defined and the compiler cannot disambiguate which one is best to use for the query. In the following example, both versions of `Select` have the same signature, because they both accept one `int` as an input parameter and have `int` as a return value.

## To correct this error

1. Provide only one implementation for each method.

## Example

The following code generates CS1940:

```
C#  
  
// cs1940.cs  
using System; //must include explicitly for types defined in 3.5  
class Test  
{  
    public delegate int Dele(int x);  
    int num = 0;  
    public int Select(Func<int, int> d)  
    {  
        return d(this.num);  
    }  
    public int Select(Dele d) // CS1940  
    {  
        return d(this.num) + 1;  
    }  
    public static void Main()  
    {  
        var q = from x in new Test()  
                select x;  
    }  
}
```

## See also

- Standard Query Operators Overview

# Compiler Error CS1941

Article • 09/15/2021 • 2 minutes to read

The type of one of the expressions in the 'clause' clause is incorrect. Type inference failed in the call to 'method'.

Type inference in query expressions flows from the type of the elements in the data source(s).

## To correct this error

1. If it is not immediately obvious why the error is occurring, examine the query carefully and trace the type of the result of each clause from the data source to the point where the error is occurring.

## Example

The following code generates CS1941 because the `equals` operator is being asked to compare an `int` to a `string`.

C#

```
// cs1941.cs
using System.Collections;
using System.Linq;
class Test
{
    static int Main()
    {
        var nums = new[] { 1, 2, 3, 4, 5, 6 };
        var words = new string[] { "lake", "mountain", "sky" };
        IEnumerable e = from n in nums
                        join w in words on n equals w // CS1941
                        select w;
        return 0;
    }
}
```

The method in which type inference fails is the method that the query clause is translated to at compile time.

## See also

- LINQ in C#
- Type Relationships in LINQ Query Operations

# Compiler Error CS1942

Article • 09/21/2022 • 2 minutes to read

The type of the expression in the 'clause' clause is incorrect. Type inference failed in the call to 'method'.

This error is typically generated when the range variable has been given an incorrect explicit type.

## To correct this error

1. If the range variable is explicitly typed, make sure that the type is either the same as, or implicitly convertible from, the type of the elements in the collection it iterates. If the range variable is preceded with the `var` keyword, remove `var`.

## Example

The following code generates CS1942:

```
C#  
  
// cs1942.cs  
class Program  
{  
    static void Main(string[] args)  
    {  
        var x = from var i in Enumerable.Range(1, 100) // CS1949  
                select i; //CS1942  
    }  
}
```

CS1942 is related to CS1949 because the use of `var` with a range variable causes the underlying `Cast<T>` operation to fail because `var` is not a type.

## See also

- [var](#)
- [LINQ in C#](#)



# Compiler Error CS1943

Article • 09/15/2021 • 2 minutes to read

An expression of type 'type' is not allowed in a subsequent from clause in a query expression with source type 'type'. Type inference failed in the call to 'method'.

All range variables must represent queryable types.

## To correct this error

1. Make sure that the type is a queryable type that implements `IEnumerable`, `IEnumerable<T>` or a derived interface, or any other type which has a query pattern defined for it.
2. If the type is a non-generic `IEnumerable`, provide an explicit type on the range variable.

## Example

The following code generates CS1943:

```
C#  
  
// cs1943.cs  
using System.Linq;  
class Test  
{  
    class TestClass  
    { }  
    static void Main()  
    {  
        int[] nums = { 0, 1, 2, 3, 4, 5 };  
        TestClass tc = new TestClass();  
  
        var x = from n in nums  
                from s in tc // CS1943  
                select n + s;  
    }  
}
```

# Compiler Error CS1944

Article • 09/15/2021 • 2 minutes to read

An expression tree may not contain an unsafe pointer operation

Expression trees do not support pointer types because the [Expression<TDelegate>.Compile](#) method is only allowed to produce verifiable code. See comments.

## To correct this error

1. Do not use pointer types when you are trying to create an expression tree.

## Example

The following example generates CS1944:

```
C#  
  
// cs1944.cs  
// Compile with: /unsafe  
using System.Linq.Expressions;  
unsafe class Test  
{  
    public delegate int* D(int i);  
    static void Main()  
    {  
        Expression<D> tree = x => &x; // CS1944  
    }  
}  
  
using System.Linq.Expressions;  
unsafe class Test  
{  
    public delegate int* D(int i);  
    static void Main()  
    {  
        Expression<D> tree = x => &x; // CS1944  
    }  
}
```

In some situations it is okay to have pointers in expression trees. For example, consider the following code:

```
Expression<Func<int*[], int*[]>> e = (int*[] i)=>i;
```

This code is a valid expression tree because no type arguments are pointer types. They are arrays of pointers, and arrays are not pointer types. Also, the body of the expression tree does not do anything dangerous with any pointer.

## See also

- [unsafe](#)

# Compiler Error CS1945

Article • 03/09/2023 • 2 minutes to read

An expression tree may not contain an anonymous method expression.

Expression trees can only contain expressions. Anonymous methods can only represent statements.

## To correct this error

1. Do not try to create an expression tree with a statement.

## Example

The following code generates CS1945:

```
C#  
  
// cs1945.cs  
using System;  
using System.Linq.Expressions;  
  
public delegate void D();  
class Test  
{  
    static void Main()  
    {  
        Expression<Func<int, Func<int, bool>>> tree = (x => delegate(int i)  
{ return true; }); // CS1945  
    }  
}
```

## See also

- [Expression Trees \(C#\)](#)
- [Operators and expressions](#)
- [Expression-bodied members](#)

# Compiler Error CS1946

Article • 03/09/2023 • 2 minutes to read

An anonymous method expression cannot be converted to an expression tree.

An [anonymous method](#) represents a set of statements but an expression tree must not contain a statement. Therefore an anonymous method cannot be represented by an expression tree.

To correct this error, change the anonymous method to a [lambda expression](#).

## Example

The following example generates CS1946:

```
C#  
  
// cs1946.cs  
using System;  
using System.Linq.Expressions;  
  
public delegate void D();  
  
class Test  
{  
    static void Main()  
    {  
        Expression<D> tree = delegate() { }; //CS1946  
        // Try using a lambda expression instead.  
        // Expression<D> tree = (x) => x + 1;  
    }  
}
```

## See also

- [Expression Trees](#)

# Compiler Error CS1947

Article • 09/15/2021 • 2 minutes to read

Range variable 'variable name' cannot be assigned to -- it is read only.

A range variable is like an iteration variable in a `foreach` statement. It cannot be assigned to in a query expression.

## To correct this error

1. Remove the assignment to the range variable.
2. If necessary, introduce a new range variable by using the `let` clause and use it to store the value.

## Example

The following code generates CS1947:

C#

```
// cs1947.cs
using System.Linq;
class Test
{
    static void Main()
    {
        int[] array = new int[] { 1, 2, 3, 4, 5 };
        var x = from i in array
                let k = i
                select i = 5; // CS1947
        x.ToList();
    }
}
```

## See also

- [LINQ Query Expressions](#)



# Compiler Error CS1948

Article • 09/15/2021 • 2 minutes to read

The range variable 'name' cannot have the same name as a method type parameter

The same declaration space cannot contain two declarations of the same identifier.

## To correct this error

1. Change the name of the range variable or the type parameter.

## Example

The following example generates CS1948 because the identifier `T` is used for the range variable and for the type parameter on method `TestMethod`:

C#

```
// cs1948.cs
using System.Linq;
class Test
{
    public void TestMethod<T>(T t)
    {
        var x = from T in Enumerable.Range(1, 100) // CS1948
                 select T;
    }
}
```

# Compiler Error CS1949

Article • 09/21/2022 • 2 minutes to read

The contextual keyword 'var' cannot be used in a range variable declaration.

A range variable is implicitly typed by the compiler. There is no need to use `var` with a range variable.

## To correct this error

1. Remove the `var` keyword from in front of the range variable.

## Example

The following example generates CS1949:

C#

```
// cs1949.cs
using System;
using System.Linq;
class Test
{
    static void Main()
    {
        var x = from var i in Enumerable.Range(1, 100) // CS1949
            select i;
    }
}
```

## See also

- [LINQ Query Expressions](#)
- [Introduction to LINQ Queries \(C#\)](#)



# Compiler Error CS1950

Article • 09/15/2021 • 2 minutes to read

The best overloaded Add method 'name' for the collection initializer has some invalid arguments.

To support collection initializers, a class must implement `IEnumerable` and have a public `Add` method. To initialize the type by using a collection initializer, the input parameter of the `Add` method must be compatible with the type of the object you are trying to add.

## To correct this error

- Use a compatible type in the collection initializer.
- Modify the input parameter and/or accessibility of the `Add` method in the collection type.
- Add a new `Add` method with an input parameter that matches what you are passing in.
- Make your collection class generic so that it can have an `Add` method that accepts any type you pass in.

## Example

The following example generates CS1950:

```
C#  
  
// cs1950.cs  
using System.Collections;  
class TestClass : CollectionBase  
{  
    public void Add(int c)  
    {  
    }  
}  
  
class Test  
{  
    static void Main()  
    {  
        TestClass t = new TestClass { "hi" }; // CS1950  
    }  
}
```

## See also

- [Object and Collection Initializers](#)

# Compiler Error CS1951

Article • 03/09/2023 • 2 minutes to read

An expression tree lambda may not contain an in, out, or ref parameter.

An expression tree just represents expressions as data structures. There is no way to represent specific memory locations as is required when you pass a parameter by reference.

## To correct this error

1. The only option is to remove the modifier in the delegate definition and pass the parameter by value.

## Example

The following example generates CS1951:

```
C#  
  
// cs1951.cs  
using System.Linq;  
public delegate int TestDelegate(ref int i);  
class Test  
{  
    static void Main()  
    {  
        System.Linq.Expressions.Expression<TestDelegate> tree1 = (ref int x)  
=> x; // CS1951  
    }  
}
```

## See also

- [Expression Trees \(C#\)](#)

# Compiler Error CS1952

Article • 09/15/2021 • 2 minutes to read

An expression tree lambda may not contain a method with variable arguments

The unsupported `__arglist` keyword is not allowed in lambda expressions that compile to expression trees.

## To correct this error

1. Forget that you ever heard of `__arglist`.

## Example

The following code produces CS1952:

C#

```
// cs1952.cs
using System;
using System.Linq.Expressions;

class Test
{
    public static int M(__arglist)
    {
        return 1;
    }

    static int Main()
    {
        Expression<Func<int, int>> f = x => Test.M(__arglist(x)); // CS1952
        return 1;
    }
}
```

# Compiler Error CS1953

Article • 09/15/2021 • 2 minutes to read

An expression tree lambda may not contain a method group.

A method call requires the `()` operator. The method name without that operator refers to the method group, which is the set of all the overloaded methods with that name.

## To correct this error

1. If you meant to call the method, add the `()` operator.

## Example

The following example generates CS1953:

C#

```
// cs1953.cs
using System;
using System.Linq.Expressions;
class CS1953
{
    public static void Main()
    {
        double num = 10;
        Expression<Func<bool>> testExpr =
            () => num.GetType is int; // CS1953
    }
}
```

# Compiler Error CS1954

Article • 09/15/2021 • 2 minutes to read

The best overloaded method match 'method' for the collection initializer element cannot be used. Collection initializer 'Add' methods cannot have ref or out parameters.

For a collection class to be initialized by using a collection initializer, the class must have a public `Add` method that is not a `ref` or `out` parameter.

## To correct this error

- If you can modify the source code of the collection class, provide an `Add` method that does not take a `ref` or `out` parameter.
- If you cannot modify the collection class, initialize it with the constructors it provides. You cannot use a collection initializer with it.

## Example

The following example produces CS1954 because the only available overload of the `Add` list in `MyList` has a `ref` parameter.

C#

```
// cs1954.cs
using System.Collections.Generic;
class MyList<T> : IEnumerable<T>
{
    List<T> _list;
    public void Add(ref T item)
    {
        _list.Add(item);
    }

    public System.Collections.Generic.IEnumerator<T> GetEnumerator()
    {
        int index = 0;
        T current = _list[index];
        while (current != null)
        {
            yield return _list[index++];
        }
    }
}

System.Collections.IEnumerable
```

```
System.Collections.IEnumerable.GetEnumerator()
{
    return GetEnumerator();
}
}

public class MyClass
{
    public string tree { get; set; }
}
class Program
{
    static void Main(string[] args)
    {
        MyList<MyClass> myList = new MyList<MyClass> { new MyClass { tree =
"maple" } }; // CS1954
    }
}
```

## See also

- [Object and Collection Initializers](#)

# Compiler Error CS1955

Article • 09/15/2021 • 2 minutes to read

Non-invocable member 'name' cannot be used like a method.

Only methods and delegates can be invoked. This error is generated when you try to use empty parentheses to call something other than a method or delegate.

## To correct this error

1. Remove the parentheses from the expression.

## Example

The following code generates CS1955 because the code is trying to invoke a field and a property by using the [invocation expression \(\)](#). You cannot call a field or a property. Use the [member access expression .](#) to access the value it stores.

C#

```
// cs1955.cs
class A
{
    public int x = 0;
    public int X
    {
        get { return x; }
        set { x = value; }
    }
}

class Test
{
    static int Main()
    {
        A a = new A();
        a.x(); // CS1955
        a.X(); // CS1955
        // Try this line instead:
        // int num = a.x;
    }
}
```

# Compiler Error CS1958

Article • 09/08/2022 • 2 minutes to read

Object and collection initializer expressions may not be applied to a delegate creation expression.

A delegate has no members like a class or struct has, and so there is nothing for an object initializer to initialize. If you encounter this error, it is probably because there are braces after the delegate creation expression. Just remove the braces and this error will disappear.

## To correct this error

1. Remove the braces.

## Example

The following code produces CS1958:

C#

```
// cs1958.cs
public class MemberInitializerTest
{
    delegate void D<T>();
    public static void GenericMethod<T>() { }
    public static void Run()
    {
        D<int> genD = new D<int>(GenericMethod<int>) { }; // CS1958
        // Try the following line instead
        // D<int> genD = new D<int>(GenericMethod<int>);
    }
}
```

# Compiler Error CS1959

Article • 09/15/2021 • 2 minutes to read

'name' is of type 'type'. The type specified in a constant declaration must be sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double, decimal, bool, string, an enum-type, or a reference-type.

The types permitted in a const declaration are limited to those described in this message.

## To correct this error

1. Declare the constant with an allowed type.

## Example

The following code produces CS1959 because `null` is not a type.

```
C#  
  
// cs1959.cs  
class Program  
{  
    static void Test<T>() where T : class  
    {  
        const T x = null; // CS1959  
    }  
}
```

## See also

- [Constants](#)
- [null](#)



# Compiler Error CS1983

Article • 10/04/2022 • 2 minutes to read

The return type of an `async` method must be `void`, `Task`, `Task<T>`, a task-like type, `IEnumerable<T>`, or `IAsyncEnumerable<T>`

## Example

The following sample generates CS1983:

```
C#  
  
// CS1983.cs (4,62)  
using System.Collections.Generic;  
  
class C  
{  
    static async IEnumerable<int> M()  
    {  
        yield return await Task.FromResult(1);  
    }  
}
```

Since `IEnumerable.GetEnumerator` returns an `IEnumerator<T>` whose `MoveNext` method does not return a value of `Task<T>`, it is not compatible with an `async` method.

## To correct this error

Use an interface that results in the invocation of method that returns a type of `Task<T>`, e.g., `IAsyncEnumerable<T>`:

```
C#  
  
using System.Collections.Generic;  
  
class C  
{  
    static async IAsyncEnumerable<int> M()  
    {  
        yield return await Task.FromResult(1);  
    }  
}
```

# Compiler Error CS1986

Article • 09/17/2022 • 2 minutes to read

'await' requires that the type have a suitable 'GetAwaiter' method

## Example

The following sample generates CS1986:

C#

```
using System.Runtime.CompilerServices;
using System;
using System.Threading.Tasks;

class Program
{
    static async Task M(MyTask<int> x)
    {
        var z = await x;
        System.Console.WriteLine(z);
    }
}

public class MyTask<TResult>
{
    readonly MyTaskAwaiter<TResult> awainer;
    public MyTask(TResult value)
    {
        this.awainer = new MyTaskAwaiter<TResult>(value);
    }
    public static MyTaskAwaiter<TResult> GetAwaiter() => throw new
NotImplementedException();
}

public class MyTaskAwaiter<TResult> : INotifyCompletion
{
    TResult value;
    public MyTaskAwaiter(TResult value)
    {
        this.value = value;
    }
    public bool IsCompleted { get => true; }
    public TResult GetResult() => value;
    public void OnCompleted(Action continuation) => throw new
NotImplementedException();
}
```

A `GetAwaiter` method must be a non-static method named `GetAwaiter` and return an instance of an object that implements `INotifyCompletion`.

A `GetAwaiter` needs to implement the `INotifyCompletion` interface (and optionally the `ICriticalNotifyCompletion` interface) and return a type that itself exposes three members [1]:

C#

```
bool IsCompleted { get; }
void OnCompleted(Action continuation);
TResult GetResult(); // TResult can also be void
```

## To correct this error

The reason CS1986 is raised in the example is that the `GetAwaiter` method is `static`. To correct this error, remove the `static` modifier (and correctly implement the method):

C#

```
public class MyTask<TResult>
{
    readonly MyTaskAwaiter<TResult> awaiter;
    public MyTask(TResult value)
    {
        this.awaiter = new MyTaskAwaiter<TResult>(value);
    }
    public MyTaskAwaiter<TResult> GetAwaiter() => awaiter;
}
```

## See also

[1] [await anything](#); ↗

# Compiler Error CS1988

Article • 09/17/2022 • 2 minutes to read

Async methods cannot have `ref`, `in` or `out` parameters

`ref` parameters are not supported in `async` methods because the method may not have completed when control returns to the calling code. Any changes to the referenced variables will not be visible to the calling code, resulting in a CS1988 error.

## Example

The following sample generates CS1988:

```
C#  
  
class C  
{  
    async Task M(ref int left, ref int right)  
    {  
        await Task.Run(() =>  
        {  
            left = 1;  
            right = 2;  
        });  
    }  
}
```

## To correct this error

One reason to use the `ref` keyword is that a method conceptually has more than one result but implemented with more than one `ref` parameter. In this case, correcting the error can be achieved by transforming the method to return a single result through the use of a tuple:

```
C#  
  
async Task<(int,int)> M(int left, int right)  
{  
    await Task.Run(() =>  
    {  
        left = 1;  
        right = 2;  
    });  
}
```

```
    return (left, right);  
}
```

# Compiler Error CS1994

Article • 09/17/2022 • 2 minutes to read

The 'async' modifier can only be used in methods that have a body.

## Example

The following sample generates CS1994:

```
C#  
  
interface IInterface  
{  
    async void F();  
}
```

## To correct this error

A non-concrete method declaration in an interface declaration has no method body. In order to support the `async` modifier, the compiler subsumes the method body logic in a state machine. Without a method body, the compiler cannot emit this state machine. In addition, the logic of a method body must contain an `await` operator to signify a continuation the state machine must manage. Without that `await` operator, a state machine has nothing to manage.

In the case of a non-concrete method, if deferring the implementation of a method body to a class that implements the interface, simply removing the `async` modifier will correct the error:

```
C#  
  
interface IInterface  
{  
    void F();  
}
```

Alternatively, a concrete default method (introduced in C# 8.0) could be declared within the interface:

```
C#
```

```
interface IInterface
{
    async void F()
    {
        await Task.Run(() =>
        {
            /* do something useful*/
        });
    }
}
```

# Compiler Error CS1996

Article • 09/17/2022 • 2 minutes to read

Cannot await in the body of a lock statement

## Example

The following sample generates CS1996:

```
C#  
  
public class C  
{  
    private readonly Dictionary<string, string> keyValuePairs = new();  
  
    public async Task<string> ReplaceValueAsync(string key, HttpClient  
httpClient)  
    {  
        lock (keyValuePairs)  
        {  
            var newValue = await httpClient.GetStringAsync(string.Empty);  
            if (keyValuePairs.ContainsKey(key)) keyValuePairs[key] =  
newValue;  
            else keyValuePairs.Add(key, newValue);  
            return newValue;  
        }  
    }  
}
```

## To correct this error

Asynchronous code within a `lock` statement block is hard to implement reliably and even harder to implement in a general sense. The C# compiler doesn't support doing this to avoid emitting code that will be prone to deadlocks. Extracting the asynchronous code from the `lock` statement block will correct this error. For example:

```
C#  
  
public class C  
{  
    private readonly Dictionary<string, string> keyValuePairs = new();  
  
    public async Task<string> ReplaceValueAsync(string key, HttpClient  
httpClient)  
    {
```

```
    var newValue = await httpClient.GetStringAsync(string.Empty);
    lock (keyValuePairs)
    {
        if (keyValuePairs.ContainsKey(key)) keyValuePairs[key] =
newValue;
        else keyValuePairs.Add(key, newValue);
        return newValue;
    }
}
```

# Compiler Error CS1997

Article • 09/17/2022 • 2 minutes to read

Since is an `async` method that returns `Task`, a `return` keyword must not be followed by an object expression. Did you intend to return `Task<T>?`

## Example

The following sample generates CS1997:

C#

```
using System.Threading.Tasks;
class C
{
    public static async Task F1()
    {
        return await Task.Factory.StartNew(() => 1);
    }
}
```

## To correct this error

A `return` statement in an `async` method returns the result of an awaitable statement. If the awaitable statement does not have a result, the state machine emitted by the compiler encapsulates returning the non-generic `Task`, eliminating the need for a `return` statement. Encountering error CS1995 means the referenced code includes a `return` statement that conflicts with the `async` modifier and the method's `return` type. The error indicates that the current method's implementation does not align with its initial intent. The simplest way to correct the error is to remove the `return` statement:

C#

```
public static async Task F1()
{
    await Task.Factory.StartNew(() => 1);
}
```

But, the resulting implementation no longer needs the `async` modifier or the `await` operator. A more accurate way of correcting this error is not to remove the `return` statement, but to remove the `async` modifier and the `await` operator:

C#

```
public static Task F1()
{
    return Task.Factory.StartNew(() => 1);
}
```

# Compiler Error CS2001

Article • 09/15/2021 • 2 minutes to read

Source file 'file' could not be found

A source file name was passed to the compiler, but could not be located. Check the spelling of the file name and the location of the file.

# Compiler Error CS2003

Article • 09/15/2021 • 2 minutes to read

Response file 'file' included multiple times

A response file was passed to the compiler more than once. A response file can only be passed to the compiler once per output file.

For more on response files, see [ResponseFiles \(Specify Response File\)](#).

# Compiler Error CS2005

Article • 09/15/2021 • 2 minutes to read

Missing file specification for 'option' option

A [compiler option](#) was only partially specified.

For example, when using `-recurse`, you must specify the file to search:  
`/recurse:filename.cs`.

## Example

The following sample generates CS2005.

```
C#  
  
// CS2005.cs  
// compile with: /recurse:  
// CS2005 expected  
class x  
{  
    public static void Main() {}  
}
```

# Compiler Error CS2006

Article • 09/15/2021 • 2 minutes to read

Command-line syntax error: Missing 'text' for 'option' option

The syntax for *option* requires additional text. For information, see [Compiler Options](#).

# Compiler Error CS2007

Article • 09/15/2021 • 2 minutes to read

Unrecognized command-line option: 'option'

The compiler was passed a string that was not a [compiler option](#), even though it began with a forward slash (/).

The following sample generates CS2007:

```
C#  
  
// CS2007.cs  
// compile with: /recur  
// CS2007 expected  
class x  
{  
    public static void Main() {}  
}
```

# Compiler Error CS2008

Article • 09/15/2021 • 2 minutes to read

## No inputs specified

The compiler was invoked and compiler options were specified, but no source-code files were passed.

# Compiler Error CS2011

Article • 09/15/2021 • 2 minutes to read

Unable to open response file 'file'

A response file was specified in a compilation, but the compiler was unable to locate and open the file.

For more on response files, see [ResponseFiles \(Specify Response File\)](#).

# Compiler Error CS2012

Article • 09/15/2021 • 2 minutes to read

Cannot open 'file' for writing

While using the `-bugreport:`, `file` compiler option, the file could not be opened for writing. Make sure you specified a valid file name and that the file is not read-only.

 **Important**

The `bugreport` option is no longer supported.

# Compiler Error CS2013

Article • 09/15/2021 • 2 minutes to read

Invalid image base number 'value'

An invalid value (not a number) was passed to the [BaseAddress](#) compiler option.

The following sample generates CS2013:

C#

```
// CS2013.cs
// compile with: /target:library /baseaddress:x
// CS2013 expected
class MyClass
{}
```

# Compiler Error CS2015

Article • 09/15/2021 • 2 minutes to read

'file' is a binary file instead of a text file

A file was passed to the compiler that was a binary file. The compiler expects a source code file.

# Compiler Error CS2016

Article • 09/15/2021 • 2 minutes to read

Code page 'codepage' is invalid or not installed

The [CodePage](#) compiler option was passed an invalid value.

The following sample generates CS2016:

C#

```
// CS2016.cs
// compile with: /codepage:x
// CS2016 expected
class MyClass
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS2017

Article • 09/15/2021 • 2 minutes to read

Cannot specify /main if building a module or library

You cannot specify a main entry point when you are building a **library TargetType**.

The following sample generates CS2017:

C#

```
// CS2017.cs
// compile with: /main:MyClass /target:library
// CS2017 expected
class MyClass
{
    public static void Main()
    {
    }
}
```

# Compiler Error CS2018

Article • 09/15/2021 • 2 minutes to read

Unable to find messages file 'cscmsgs.dll'

The .dll file that contains the compiler's error and warning messages was not found. This file must be present in the same directory as the other compiler support files.

# Compiler Error CS2019

Article • 09/15/2021 • 2 minutes to read

Invalid target type for /target: must specify 'exe', 'winexe', 'library', or 'module'

The [TargetType](#) compiler option was used, but an invalid parameter was passed. To resolve this error, recompile the program using the form of the [/target](#) option that is appropriate to your output file.

The following sample generates CS2017:

```
C#  
  
// CS2019.cs  
// compile with: /target:libra  
// CS2019 expected  
class MyClass  
{  
}
```

# Compiler Error CS2020

Article • 09/15/2021 • 2 minutes to read

Only the first set of input files can build a target other than 'module'

In a multi-output compilation, the first output file must be built with [-target:exe](#), [-target:winexe](#), or [-target:library](#). Any subsequent output files must be built with [-target:module](#).

# Compiler Error CS2021

Article • 09/08/2022 • 2 minutes to read

File name 'file' is too long or invalid

All file names passed to the C# compiler must be no longer than `_MAX_PATH` (defined in a Windows header file). The compiler will give this error in the following situations:

- A file name (including the path) is longer than `_MAX_PATH`.
- The file name contains invalid characters.
- The file name contains wildcards where wildcards are not allowed (such as in resource file names).



# Compiler Error CS2022

Article • 03/18/2022 • 2 minutes to read

Options '/out' and '/target' must appear before source file names

The [-out \(Set Output Filename\)](#) and [-target \(Specify Output File Format\)](#) compiler options, when specified on the command line, must precede the source code files.

# Compiler Error CS2024

Article • 09/15/2021 • 2 minutes to read

Invalid file section alignment number '#'

An invalid value was passed to the [FileAlignment](#) compiler option.

## Example

The following sample generates CS2024:

```
C#  
  
// CS2024.cs  
// compile with: /filealign:ex  
// CS2024 expected  
class MyClass  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Error CS2032

Article • 09/15/2021 • 2 minutes to read

Character 'character' is not allowed on the command-line or in response files

Response files and the command line options for csc.exe cannot contain ASCII control characters in the range 0-31 or the pipe (|) character.

Compiler error CS2032 is difficult to demonstrate from the command line because the command line processor and the integrated development environment (IDE) filter out characters that are not valid. The following procedure generates the error by using a [response file](#).

## To generate this error

1. In the *My Documents* folder, create a text file that is named *CS2032.rsp*, and then enter the following compiler options in it:

```
Console  
/target:exe /out:cs|2032.exe cs2032.cs
```

2. In the *My Documents* folder, create a text file that's named *cs2032.cs* and that contains whatever you want.
3. Open [Visual Studio Developer Command Prompt](#) or [Visual Studio Developer PowerShell](#).
4. Change the current directory to `C:\Users\<YourUsername>\Documents`.
5. Run the following command from the command prompt: `csc @cs2032.rsp`
6. The CS2032 error message appears because the response file, *CS2032.rsp*, contains a pipe character.



# Compiler Error CS2033

Article • 09/15/2021 • 2 minutes to read

Cannot create short filename 'filename' when a long filename with the same short filename already exists

Compile any C# file with a name longer than eight characters. Then compile another file with the short version of the preceding file name, such as the first six characters of the name plus "~1." The second compile will generate this error.

To resolve this error, rename the short file name to one that does not conflict with the long file name.

# Compiler Error CS2034

Article • 09/15/2021 • 2 minutes to read

A /reference option that declares an extern alias can only have one filename. To specify multiple aliases or filenames, use multiple /reference options.

To specify two aliases and/or file names, use two /reference options, like this:

## Example

The following code will generate error CS2034.

C#

```
// CS2034.cs
// compile with: /r:A1=cs2034a1.dll;A2=cs2034a2.dll
// to fix, compile with: /r:A1=cs2034a1.dll /r:A2=cs2034a2.dll
// CS2034
extern alias A1;
extern alias A2;
using System;
```

# Compiler Error CS2035

Article • 09/15/2021 • 2 minutes to read

Command-line syntax error: Missing ':<number>' for 'compiler\_option' option

Some compiler options require a value.

## Example

The following sample generates CS2035.

C#

```
// CS2035.cs
// compile with: /baseaddress
// CS2035 expected
```

# Compiler Error CS2036

Article • 09/15/2021 • 2 minutes to read

The `/pdb` option requires that the `/debug` option also be used.

Program database files are only generated for debug builds. The `/pdb` option is therefore meaningless in a retail build.

## To correct this error

- Add the `/debug` compiler option.
- Remove the `/pdb` compiler option.

## Example

The following example generates CS2036 when it is compiled with the `/pdb` option but not the `/debug` option:

```
C#  
  
// cs2036.cs  
// Compile with: /pdb  
// CS2036  
class Test  
{  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [PdbFile \(C# Compiler Options\)](#)
- [DebugType \(C# Compiler Options\)](#)



# Compiler Error CS4004

Article • 09/17/2022 • 2 minutes to read

Cannot await in an unsafe context

## Example

The following sample generates CS4004:

```
C#  
  
using System.Threading.Tasks;  
  
public static class C  
{  
    public static unsafe async Task<string> ReverseTextAsync(string text)  
    {  
        return await Task.Run(() =>  
        {  
            if (string.IsNullOrEmpty(text))  
            {  
                return text;  
            }  
  
            fixed (char* pText = text)  
            {  
                char* pStart = pText;  
                char* pEnd = pText + text.Length - 1;  
                for (int i = text.Length / 2; i >= 0; i--)  
                {  
                    char temp = *pStart;  
                    *pStart++ = *pEnd;  
                    *pEnd-- = temp;  
                }  
  
                return text;  
            }  
        });  
    }  
}
```

The `ReverseText` method naively uses a background task to asynchronously create a new string in reverse order of a given string.

## To correct this error

Separating the unsafe code from the awaitable code will correct this error. One separation technique is creating a new method for the unsafe code and then calling it from the awaitable code. For example:

C#

```
public static class C
{
    public static async Task<string> ReverseTextAsync(string text)
    {
        return await Task.Run(() => ReverseTextUnsafe(text));
    }

    private static unsafe string ReverseTextUnsafe(string text)
    {
        if (string.IsNullOrEmpty(text))
        {
            return text;
        }

        fixed (char* pText = text)
        {
            char* pStart = pText;
            char* pEnd = pText + text.Length - 1;
            for (int i = text.Length / 2; i >= 0; i--)
            {
                char temp = *pStart;
                *pStart++ = *pEnd;
                *pEnd-- = temp;
            }

            return text;
        }
    }
}
```

# Compiler Error CS4008

Article • 09/16/2022 • 2 minutes to read

Cannot await 'void'

## Example

The following sample generates CS4008:

```
C#  
  
// CS4008.cs (7,33)  
  
using System.Threading.Tasks;  
  
class Test  
{  
    public async void goo()  
    {  
        await Task.Factory.StartNew(() => { });  
    }  
  
    public async void bar()  
    {  
        await goo();  
    }  
  
    public static void Main() { }  
}
```

## To correct this error

Although this error can be corrected by changing the signature of `goo`:

```
C#  
  
public async Task goo()  
{  
    await Task.Factory.StartNew(() => { });  
}
```

Simply adding `Task` to the method's signature needlessly perpetuates a compiler-created state machine when it is not needed. The `goo` method does not require an

`await`, nor does it need to be asynchronous. Instead, consider simply returning the `Task` created by `Task.Factory`:

C#

```
public Task goo()
{
    return Task.Factory.StartNew(() => { });
}
```

# Compiler Error CS4009

Article • 11/20/2021 • 2 minutes to read

'Type.Method': an entry point cannot be marked with the `async` modifier.

You cannot use the `async` keyword in the application entry point (typically the `Main` method).

## ⓘ Important

Starting with C# 7.1, the `Main` method can have an `async` modifier. For more information, see [Async main return values](#). For information about how to select the C# language version, see the [Select the C# language version](#) article.

## Example

The following example produces CS4009:

```
C#  
  
using System;  
using System.Threading.Tasks;  
  
public class Example  
{  
    public static async void Main()  
    {  
        Console.WriteLine("About to wait two seconds");  
        await WaitTwoSeconds();  
        Console.WriteLine("About to exit the program");  
    }  
  
    private static async Task WaitTwoSeconds()  
    {  
        await Task.Delay(2000);  
        Console.WriteLine("Returning from an asynchronous method");  
    }  
}
```

## To correct this error

Update the [C# language version](#) used by the project to 7.1 or higher.

If you use C# 7.0 or lower, remove the `async` keyword from the signature of the application entry point. Also remove any `await` keywords you've used to await asynchronous methods in your application entry point.

However, you still have to wait for the asynchronous method to complete before your entry point resumes execution. Otherwise, compilation generates compiler warning CS4014, and the application will terminate before the asynchronous operation completes. The following example illustrates this problem:

```
C#  
  
using System;  
using System.Threading.Tasks;  
  
public class Example  
{  
    public static void Main()  
    {  
        Console.WriteLine("About to wait two seconds");  
        WaitTwoSeconds();  
        Console.WriteLine("About to exit the program");  
    }  
  
    private static async Task WaitTwoSeconds()  
    {  
        await Task.Delay(2000);  
        Console.WriteLine("Returning from an asynchronous method");  
    }  
}  
  
// The example displays the following output:  
//     About to wait two seconds  
//     About to exit the program
```

To await a method that returns a `Task`, call its `Wait` method, as the following example illustrates:

```
C#  
  
using System;  
using System.Threading.Tasks;  
  
public class Example  
{  
    public static void Main()  
    {  
        Console.WriteLine("About to wait two seconds");  
        WaitTwoSeconds().Wait();  
        Console.WriteLine("About to exit the program");  
    }  
}
```

```
private static async Task WaitTwoSeconds()
{
    await Task.Delay(2000);
    Console.WriteLine("Returning from an asynchronous method");
}
// The example displays the following output:
//     About to wait two seconds
//     Returning from an asynchronous method
//     About to exit the program
```

To await a method that returns a `Task<TResult>`, retrieve the value of its `Result` property, as the following example does:

C#

```
using System;
using System.Threading.Tasks;

public class Example
{
    public static void Main()
    {
        Console.WriteLine("About to wait two seconds");
        int value = WaitTwoSeconds().Result;
        Console.WriteLine($"Value returned from the aynnc operation:
{value}");
        Console.WriteLine("About to exit the program");
    }

    private static async Task<int> WaitTwoSeconds()
    {
        await Task.Delay(2000);
        Console.WriteLine("Returning from an asynchronous method");
        return 100;
    }
}
// The example displays the following output:
//     About to wait two seconds
//     Returning from an asynchronous method
//     Value returned from the aynnc operation: 100
//     About to exit the program
```

## See also

- [async keyword](#)
- [await operator](#)

# Compiler Error CS4013

Article • 10/07/2022 • 2 minutes to read

Instance of type cannot be used inside a nested function, query expression, iterator block or async method

## Example

The following sample generates CS4013:

```
C#  
  
public class C  
{  
    public static IEnumerable<string> Lines(char[] text)  
    {  
        ReadOnlySpan<char> chars = text;  
        var index = chars.IndexOf('\n');  
        while (index > 0)  
        {  
            yield return chars[..index].ToString();  
            chars = chars[(index + 1)..];  
            index = chars.IndexOf('\n');  
        }  
  
        yield return chars.ToString();  
    }  
}
```

This enumerator method extracts lines of text from a character array. It naively tries to use `ReadOnlySpan<T>` to improve performance.

## To correct this error

`Lines(char[] text)` is an enumerator function. An enumerator function compiles the method's body into a state machine that manages the sequence of states the iterator function goes through while processing. That state machine is implemented as a generated class, and the state is implemented as variables within that class. That captured local state is forced from a stack context to a heap context. Since `ref structs` like `ReadOnlySpan<T>` cannot be stored in the heap, the CS4013 error is raised. To continue to use a `ReadOnlySpan<T>`, to correct this error, the method must be re-implemented as a non-iterator function, for example:

C#

```
public static IEnumerable<string> Lines2(char[] text)
{
    ReadOnlySpan<char> chars = text;

    var lines = new List<string>();
    var index = chars.IndexOf('\n');
    while (index > 0)
    {
        lines.Add(chars[..index].ToString());
        chars = chars[(index+1)..];
        index = chars.IndexOf('\n');
    }

    lines.Add(chars.ToString());
    return lines;
}
```

To continue to use an iterator function, to correct this error, the method must be re-implemented to avoid using `ReadOnlySpan<T>`, for example:

C#

```
public static IEnumerable<string> Lines2(char[] chars)
{
    var startIndex = 0;
    var index = Array.IndexOf(chars, '\n');
    while (index > 0)
    {
        yield return new string(chars, startIndex, index);
        startIndex = index+1;
        index = Array.IndexOf(chars, '\n', startIndex);
    }
    yield return new string(chars, startIndex, chars.Length - startIndex);
}
```

# Compiler Error CS4032

Article • 10/07/2022 • 2 minutes to read

The 'await' operator can only be used within an async method. Consider marking this method with the 'async' modifier and changing its return type to 'Task<T>'.

## Example

The following sample generates CS4032:

```
C#  
  
// CS4032.cs (7,9)  
using System.Collections.Generic;  
using System.Threading.Tasks;  
  
class C  
{  
    static IAsyncEnumerator<int> M(int value)  
    {  
        yield return value;  
        await Task.CompletedTask;  
    }  
}
```

## To correct this error

To correct this error, change the signature of method `M` to make it asynchronous:

```
C#  
  
using System.Collections.Generic;  
using System.Threading.Tasks;  
  
class C  
{  
    static async IAsyncEnumerator<int> M(int value)  
    {  
        yield return value;  
        await Task.CompletedTask;  
    }  
}
```

# Compiler Error CS4033

Article • 10/07/2022 • 2 minutes to read

The 'await' operator can only be used within an async method. Consider marking this method with the 'async' modifier and changing its return type to 'Task'.

## Example

The following sample generates CS4033:

```
C#  
  
// CS4033.cs (7,9)  
  
using System.Collections.Generic;  
class C  
{  
    void M(IAsyncEnumerable<int> collection)  
    {  
        await foreach (var i in collection)  
        {  
        }  
    }  
}
```

## To correct this error

To correct this error, change the signature of method `M` to make it asynchronous:

```
C#  
  
using System.Collections.Generic;  
class C  
{  
    async void M(IAsyncEnumerable<int> collection)  
    {  
        await foreach (var i in collection)  
        {  
        }  
    }  
}
```

# Compiler Error CS5001

Article • 11/16/2021 • 2 minutes to read

Program does not contain a static 'Main' method suitable for an entry point

This error occurs when no static `Main` method with a correct signature is found in the code that produces an executable file. It also occurs if the entry point function, `Main`, is defined with the wrong case, such as lower-case `main`. For information about the rules that apply to the `Main` method, see [Main\(\) and Command-Line Arguments](#).

If the `Main` method has an `async` modifier, make sure that the [selected C# language version](#) is 7.1 or higher and to use `Task` or `Task<int>` as the return type.

The `Main` method is only required when compiling an executable file, that is, when the `exe` or `winexe` element of the `TargetType` compiler option is specified. The following Visual Studio project types specify one of these options by default:

- Console application
- ASP.NET Core application
- WPF application
- Windows Forms application

## Example

The following example generates CS5001:

```
C#  
  
// CS5001.cs  
// CS5001 expected when compiled with -target:exe or -target:winexe  
public class Program  
{  
    // Uncomment the following line to resolve.  
    // static void Main() {}  
}
```

# Compiler Error CS7000

Article • 09/14/2022 • 2 minutes to read

Unexpected use of an aliased name

## Examples

The following sample generates CS7000 because the alias qualifier ( :: ) is not supported in a namespace declaration:

```
C#  
  
using N = ClassLibrary1;  
  
namespace N::A  
{  
    public class Goo  
    {  
    }  
}
```

## To correct this error

Either remove the unsupported use of the alias in the namespace declaration:

```
C#  
  
namespace A  
{  
    void Goo()  
    {  
    }  
}
```

Or, replace the alias with the fully qualified name:

```
C#  
  
namespace ClassLibrary1.A  
{  
    void Goo()  
    {  
    }  
}
```

# Compiler Error CS7003

Article • 09/15/2021 • 2 minutes to read

Unexpected use of an unbound generic name

This error occurs if you use a generic type needing one parameter type without passing any generic parameter type name between the angle brackets. This use may be a variable declaration, or an object instantiation.

## To correct this error

Provide one parameter type name in angle brackets when using a generic type.

## Example

The following example generates CS7003:

```
C#  
  
// CS7003.cs  
class Program  
{  
    static void Main(string[] args)  
    {  
        var myVar1 = new MyGenericClass<>(); //CS7003  
  
        MyGenericClass<> var2; //CS7003  
    }  
}  
  
public class MyGenericClass<T> { }
```

## See also

- [Generics](#)



# Compiler Error CS8124

Article • 10/04/2022 • 2 minutes to read

Tuple must contain at least two elements.

## Example

The following sample generates CS8124:

C#

```
// CS8124.cs (4,20)

class C
{
    void M(int x, () y, (int a) z) { }
```

## To correct this error

A valid tuple declaration contains two or more elements, ensuring tuple declarations have two or more elements corrects this error:

C#

```
class C
{
    void M(int x, (int, int) y, (int a, int b) z) { }
```

# Compiler Error CS8125

Article • 10/04/2022 • 2 minutes to read

Tuple element name is only allowed at position.

## Example

The following sample generates CS8125:

```
C#  
  
// CS8125.cs (2,15)  
  
public class C  
{  
    public void Method()  
    {  
        var tuple3 = (Item2: 2, Item1: 1);  
    }  
}
```

## To correct this error

If tuple element names `Item1`, `Item2`, etc. are used, ensuring the correct order corrects this error:

```
C#  
  
public void Method()  
{  
    var tuple3 = (Item1: 2, Item2: 1);  
}
```

# Compiler Error CS8127

Article • 10/04/2022 • 2 minutes to read

Tuple element names must be unique.

## Example

The following sample generates CS8127:

```
C#  
  
// CS8127.cs (7,0)  
  
internal struct NewStruct  
{  
    public int a;  
    public int b;  
  
    public static implicit operator (int a, int b)(NewStruct value)  
    {  
        return (value.a, value.b);  
    }  
}
```

## To correct this error

Ensuring the names of elements within a tuple declaration are unique corrects this error:

```
C#  
  
public static implicit operator (int a, int b)(NewStruct value)  
{  
    return (value.a, value.b);  
}
```

# Compiler Error CS8129

Article • 10/04/2022 • 2 minutes to read

No suitable 'Deconstruct' instance or extension method was found for type, with `out` parameters and a `void` return type.

## Example

The following sample generates CS8129:

```
C#  
  
// CS8129.cs (11,52)  
  
class C  
{  
    static void Main()  
    {  
        long x;  
        string y;  
        (x, y) = new C();  
    }  
  
    public int Deconstruct(out int a, out string b)  
    {  
        a = 1;  
        b = "hello";  
        return 42;  
    }  
}
```

## To correct this error

A valid `Deconstruct` method returns `void` and has two or more `out` parameters that match the type of a tuple that would be deconstructed. Implementing a valid `Deconstruct` method corrects this error:

```
C#  
  
public void Deconstruct(out int a, out string b)  
{  
    a = 1;  
    b = "hello";  
}
```

# Compiler Error CS8130

Article • 10/07/2022 • 2 minutes to read

Cannot infer the type of implicitly-typed deconstruction variable.

## Example

The following sample generates CS8130:

C#

```
// CS8130.cs (5,14)
class Program
{
    static void Main()
    {
        var (x2, y2) = () => { };
    }
}
```

The compiler cannot convert a delegate (`Action`) to a two-element tuple and thus is unable to infer the type of each element of the tuple.

## To correct this error

To assign a value to a tuple, ensuring the right-hand-side expression is a tuple with the same number of elements as that on the left-hand-side corrects this error:

C#

```
static void Main()
{
    var (x2, y2) = (1, 2);
}
```

# Compiler Error CS8131

Article • 10/07/2022 • 2 minutes to read

Deconstruct assignment requires an expression with a type on the right-hand-side.

## Example

The following sample generates CS8131:

C#

```
// CS8131.cs (5,24)
class Program
{
    static void Main()
    {
        var (x2, y2) = () => { };
    }
}
```

The compiler cannot convert a delegate (`Action`) to a two-element tuple, resulting in CS8131.

## To correct this error

To assign a value to a tuple, ensure the right-hand-side expression is the same type of tuple as that on the left-hand-side:

C#

```
static void Main()
{
    var (x2, y2) = (1, 2);
}
```

# Compiler Error CS8132

Article • 10/04/2022 • 2 minutes to read

Cannot deconstruct a tuple of elements into variables.

## Example

The following sample generates CS8132:

```
C#  
  
// CS8132.cs (5,9)  
class Program  
{  
    static void F(object x, object y, object? z)  
    {  
        (object? a, object? b) = (x, y, z = 3);  
        Console.WriteLine(a);  
        Console.WriteLine(b);  
        Console.WriteLine(z);  
    }  
}
```

## To correct this error

To deconstruct to a tuple with a smaller number of elements, using the discard variables will correct this error:

```
C#  
  
class Program  
{  
    static void F(object x, object y, object? z)  
    {  
        (object? a, object? b, object _) = (x, y, z = 3);  
        Console.WriteLine(a);  
        Console.WriteLine(b);  
        Console.WriteLine(z);  
    }  
}
```

# Compiler Error CS8139

Article • 10/04/2022 • 2 minutes to read

cannot change tuple element names when overriding inherited member

## Example

The following sample generates CS8139:

```
C#  
  
// CS8139.cs (9,38)  
  
public class Base  
{  
    public virtual (object a, object b) M((object c, object d) x) { return  
x; }  
}  
  
class C : Base  
{  
    public override (object, object) M((object c, object d) y) { return y; }  
}
```

## To correct this error

Ensuring the tuple element names in the overriding member match those in the virtual member corrects this error:

```
C#  
  
class C : Base  
{  
    public override (object a, object b) M((object c, object d) y) { return  
y; }  
}
```

# Compiler Error CS8140

Article • 10/04/2022 • 2 minutes to read

is already listed in the interface list on type with different tuple element names, as.

## Example

The following sample generates CS8140:

```
C#  
  
// CS8140.cs (11,7)  
  
interface I<T>  
{  
    T GetValue();  
}  
  
interface I2 : I<(int c, int d)>  
{  
}  
  
class C : I<(int a, int b)>, I2  
{  
    public (int c, int d) GetValue()  
    {  
        return (1, 2);  
    }  
}
```

## To correct this error

Renaming the tuple element names to match the interface declaration corrects this error:

```
C#  
  
interface I<T>  
{  
    T GetValue();  
}  
  
interface I2 : I<(int c, int d)>  
{  
}
```

```
class C : I<(int c, int d)>, I2
{
    public (int c, int d) GetValue()
    {
        return (1, 2);
    }
}
```

# Compiler Error CS8141

Article • 11/03/2022 • 2 minutes to read

The tuple element names in the signature of method must match the tuple element names of interface method (including on the return type).

## Example

The following sample generates CS8141:

```
C#  
  
// CS8141.cs (10,27)  
using System.Collections;  
  
public interface IGrabber<out T>  
{  
    T GetOne();  
}  
  
class SomeGrabber : IGrabber<(int, int)>  
{  
    public (int a, int b) GetOne()  
    {  
        return (1, 2);  
    }  
}
```

## To correct this error

Changing the signature of the `GetOne` method to return an unnamed tuple, matching the unnamed tuple in the interface, will correct this error:

```
C#  
  
public (int, int) GetOne()  
{  
    return (1, 2);  
}
```

# Compiler Error CS8145

Article • 09/24/2022 • 2 minutes to read

Auto-implemented properties cannot return by reference

Auto-implemented properties are not guaranteed to have a member or variable that can be referenced and thus do not support return by reference.

## Example

The following sample generates CS8145:

C#

```
// CS8145.cs (4,13)

public class C
{
    public ref int Property1 { get; }
```

## To correct this error

If the property can be implemented through a backing field, then refactoring to use a backing field and `ref`-returning the field will correct this error:

C#

```
public class C
{
    private int property1;

    public ref int Property1 => ref property1;
```

If the property cannot be implemented through a backing field, then removing the `ref` modifier from the property corrects this error:

C#

```
public class C
{
    public int Property1 { get; }
```



# Compiler Error CS8146

Article • 09/27/2022 • 2 minutes to read

Properties which return by reference must have a get accessor

## Example

The following sample generates CS8146:

```
C#  
  
// CS8146.cs (5,18)  
  
public class C  
{  
    private ref int number;  
  
    private ref int Number { init => number = value; }  
}
```

## To correct this error

Ensuring a property that returns by reference has a get accessor will correct this error:

```
C#  
  
public class C  
{  
    private ref int number;  
  
    private ref int Number => ref number;  
}
```

# Compiler Error CS8147

Article • 10/07/2022 • 2 minutes to read

Properties which return by reference cannot have set accessors

## Example

The following sample generates CS8147:

C#

```
// CS8147.cs (6,44)

public class C
{
    private ref int number;

    ref int Number { get => ref number; init => number = value; }
}
```

## To correct this error

Removing any set accessor for a property that returns by reference will correct this error:

C#

```
public class C
{
    private ref int number;

    ref int Number => ref number;
}
```

# Compiler Error CS8148

Article • 09/27/2022 • 2 minutes to read

must match by reference return of overridden member

## Example

The following sample generates CS8148:

```
C#  
  
// CS8148.cs (11,29)  
  
public class Base  
{  
    public virtual int GetNumber() { return 0; }  
}  
  
public class Derived : Base  
{  
    private int number;  
  
    public override ref int GetNumber() { return ref number; }  
}
```

## To correct this error

Ensuring that base members that return by value are implemented by members that do not return by reference corrects this error:

```
C#  
  
public class Base  
{  
    public virtual int GetNumber() { return 0; }  
}  
  
public class Derived : Base  
{  
    private int number;  
  
    public override int GetNumber() { return number; }  
}
```

# Compiler Error CS8149

Article • 09/27/2022 • 2 minutes to read

By-reference returns may only be used in methods that return by reference

## Example

The following sample generates CS8149:

```
C#  
  
// CS8149.cs (9,33)  
  
delegate int E();  
  
class C  
{  
    static int i;  
    static void M()  
    {  
        var e = new E(() => ref i);  
    }  
}
```

## To correct this error

Ensuring that methods and delegates declared as returning by value do not return by reference corrects this error:

```
C#  
  
delegate int E();  
  
class C  
{  
    static int i;  
    static void M()  
    {  
        var e = new E(() => i);  
    }  
}
```

# Compiler Error CS8150

Article • 09/24/2022 • 2 minutes to read

By-value returns may only be used in methods that return by value

## Example

The following sample generates CS8150:

```
C#  
  
// CS8150.cs (6,9)  
  
class C  
{  
    ref int M()  
    {  
        return new();  
    }  
}
```

## To correct this error

To return by value, refactor the method from being by reference. For example:

```
C#  
  
// CS8150.cs (6,9)  
  
class C  
{  
    int M()  
    {  
        return new();  
    }  
}
```

# Compiler Error CS8151

Article • 09/27/2022 • 2 minutes to read

The return expression must be of type because this method returns by reference

## Example

The following sample generates CS8151:

```
C#  
  
// CS8151.cs (6,20)  
  
class Program  
{  
    ref int M(ref long i)  
    {  
        return ref i;  
    }  
}
```

## To correct this error

Ensuring that a `ref` return matches the type of the referenced variable regardless of any implicit conversions corrects this error:

```
C#  
  
class Program  
{  
    ref long M(ref long i)  
    {  
        return ref i;  
    }  
}
```

# Compiler Error CS8152

Article • 09/24/2022 • 2 minutes to read

Type does not implement interface member. cannot implement because it does not have matching return by reference.

## Example

The following sample generates CS8152:

To implement an interface with a method that returns by reference, the implementation of the method must also return by reference and not by value.

```
C#  
  
// CS8152.cs (6,21)  
  
public interface ITest  
{  
    ref readonly int M();  
}  
public class Test : ITest  
{  
    public int M() => 0;  
}
```

## To correct this error

Ensure interface methods that return by reference do not return by value. For example:

```
C#  
  
public class Test : ITest  
{  
    int m;  
    public ref readonly int M() => ref m;  
}
```

# Compiler Error CS8153

Article • 09/24/2022 • 2 minutes to read

An expression tree lambda may not contain a call to a method, property, or indexer that returns by reference

## Example

The following sample generates CS8153:

C#

```
// CS8153.cs (11,46)

using System;
using System.Linq.Expressions;

namespace RefPropCrash
{
    class Program
    {
        static void Main(string[] args)
        {
            TestExpression(() => new Model { Value = 1 });
        }

        static void TestExpression(Expression<Func<Model>> expression)
        {
        }
    }

    class Model
    {
        int value;
        public ref int Value => ref value;
    }
}
```

## To correct this error

To use a method within an expression, ensure it is not by reference. For example:

C#

```
class Model
{
```

```
    public int Value { get; set; }  
}
```

# Compiler Error CS8154

Article • 09/24/2022 • 2 minutes to read

The body cannot be an iterator block because it returns by reference

## Example

The following sample generates CS8154:

C#

```
// CS8154.cs (12,17)

class TestClass
{
    int x = 0;
    ref int TestFunction()
    {
        if (true)
        {
            yield return x;
        }

        ref int localFunction()
        {
            if (true)
            {
                yield return x;
            }
        }

        yield return localFunction();
    }
}
```

## To correct this error

Iterator methods cannot return by reference, refactoring to return by value corrects this error:

C#

```
class TestClass
{
    int x = 0;
    IEnumerable<int> TestFunction()
```

```
{  
    if (true)  
    {  
        yield return x;  
    }  
  
    IEnumerable<int> localFunction()  
    {  
        if (true)  
        {  
            yield return x;  
        }  
    }  
  
    foreach (var item in localFunction())  
        yield return item;  
}  
}
```

# Compiler Error CS8155

Article • 09/24/2022 • 2 minutes to read

Lambda expressions that return by reference cannot be converted to expression trees

## Example

The following sample generates CS8155:

```
C#  
  
// CS8155.cs (11,51)  
  
using System.Linq.Expressions;  
class TestClass  
{  
    int x = 0;  
  
    delegate ref int RefReturnIntDelegate(int y);  
  
    void TestFunction()  
    {  
        Expression<RefReturnIntDelegate> lambda = (y) => ref x;  
    }  
}
```

## To correct this error

To be convertible to an expression tree, refactoring to return by value corrects this error:

```
C#  
  
class TestClass  
{  
    int x = 0;  
  
    delegate int RefReturnIntDelegate(int y);  
  
    void TestFunction()  
    {  
        Expression<RefReturnIntDelegate> lambda = (y) => x;  
    }  
}
```

# Compiler Error CS8156

Article • 09/24/2022 • 2 minutes to read

An expression cannot be used in this context because it may not be passed or returned by reference

## Example

The following sample generates CS8156:

C#

```
// CS8156.cs (7,27)

class Test
{
    delegate ref int D1();

    void Test1()
    {
        D1 d1 = () => ref 2 + 2;
    }
}
```

## To correct this error

When not using referenceable variables, refactoring to return by value corrects this error:

C#

```
class Test
{
    delegate int D1();

    void Test1()
    {
        D1 d1 = () => 2 + 2;
    }
}
```

# Compiler Error CS8157

Article • 09/24/2022 • 2 minutes to read

Cannot return by reference because it was initialized to a value that cannot be returned by reference

## Example

The following sample generates CS8157:

```
C#  
  
// CS8157.cs (8,21)  
  
class C  
{  
    ref int M()  
    {  
        int x = 0;  
        ref int rx = ref x;  
        return ref (rx = ref (new int[1])[0]);  
    }  
}
```

## To correct this error

To return a value that cannot be returned by reference, refactoring to return by value corrects this error:

```
C#  
  
class C  
{  
    int M()  
    {  
        int x = 0;  
        ref int rx = ref x;  
        return rx = ref (new int[1])[0];  
    }  
}
```

# Compiler Error CS8158

Article • 09/24/2022 • 2 minutes to read

Cannot return by reference a member because it was initialized to a value that cannot be returned by reference

## Example

The following sample generates CS8158:

C#

```
// CS8158.cs (11,14)

public class Test
{
    public struct S1
    {
        public char x;
    }

    ref char Test1(char arg1, S1 arg2)
    {
        ref S1 r = ref arg2;
        return ref r.x;
    }
}
```

## To correct this error

To return members initialized to a value that cannot be returned by reference, refactoring to return by value corrects this error:

C#

```
char Test1(char arg1, S1 arg2)
{
    ref S1 r = ref arg2;
    return r.x;
}
```

# Compiler Error CS8159

Article • 09/24/2022 • 2 minutes to read

Cannot return the range variable by reference

## Example

The following sample generates CS8159:

C#

```
// CS8159.cs (7,74)

using System.Linq;
class TestClass
{
    delegate ref char RefCharDelegate();
    void TestMethod()
    {
        var x = from c in "TestValue" select (RefCharDelegate)(() => ref c);
    }
}
```

## To correct this error

To return a range variable, refactoring to return by value correct this error:

C#

```
using System.Linq;
class TestClass
{
    delegate char RefCharDelegate();
    void TestMethod()
    {
        var x = from c in "TestValue" select (RefCharDelegate)(() => c);
    }
}
```

# Compiler Error CS8160

Article • 09/24/2022 • 2 minutes to read

A readonly field cannot be returned by writable reference

## Example

The following sample generates CS8160:

```
C#  
  
// CS8160.cs (8,20)  
  
class Program  
{  
    readonly int i = 0;  
  
    ref int M()  
    {  
        return ref i;  
    }  
}
```

## To correct this error

To return a `readonly` field, refactoring to return by value corrects this error:

```
C#  
  
class Program  
{  
    readonly int i = 0;  
  
    int M()  
    {  
        return i;  
    }  
}
```

# Compiler Error CS8161

Article • 09/24/2022 • 2 minutes to read

A static readonly field cannot be returned by writable reference

## Example

The following sample generates CS8161:

```
C#  
  
// CS8161.cs (12,14)  
public class Test  
{  
    public struct S1  
    {  
        public char x;  
    }  
  
    public static readonly char s1;  
  
    ref char Test2()  
    {  
        return ref s1;  
    }  
}
```

## To correct this error

To return the value of a `static readonly` field, refactor to return by value:

```
C#  
  
public class Test  
{  
    public struct S1  
    {  
        public char x;  
    }  
  
    public static readonly char s1;  
  
    char Test2()  
    {  
        return s1;  
    }  
}
```



# Compiler Error CS8162

Article • 09/24/2022 • 2 minutes to read

Members of readonly field cannot be returned by writable reference

## Example

The following sample generates CS8162:

```
C#  
  
// CS8162.cs (12,14)  
public class Test  
{  
    public struct S1  
    {  
        public char x;  
    }  
  
    public readonly S1 i2;  
  
    ref char Test1()  
    {  
        return ref i2.x;  
    }  
}
```

## To correct this error

To return the value of a `readonly` field, refactoring to return by value corrects this error:

```
C#  
  
public class Test  
{  
    public struct S1  
    {  
        public char x;  
    }  
  
    public readonly S1 i2;  
    char Test1()  
    {  
        return i2.x;  
    }  
}
```

# Compiler Error CS8163

Article • 09/24/2022 • 2 minutes to read

Fields of static readonly field cannot be returned by writable reference

## Example

The following sample generates CS8163:

C#

```
// CS8163.cs (12,14)
public class Test
{
    public struct S1
    {
        public char x;
    }

    public static readonly S1 s2;

    char Test2()
    {
        return s2.x;
    }
}
```

## To correct this error

To return the value of a `static readonly` field, refactoring to return by value corrects this error:

C#

```
public class Test
{
    public struct S1
    {
        public char x;
    }

    public static readonly S1 s2;

    char Test2()
    {
        return s2.x;
    }
}
```

```
    }  
}
```

# Compiler Error CS8166

Article • 09/24/2022 • 2 minutes to read

Cannot return a parameter by reference because it is not a ref parameter

## Example

The following sample generates CS8166:

```
C#  
  
// CS8166.cs (11,20)  
  
public class Test  
{  
    public struct S1  
    {  
        public char x;  
    }  
  
    ref char Test1(char arg1, S1 arg2)  
    {  
        return ref arg1;  
    }  
}
```

## To correct this error

To return a parameter that is not passed by reference, refactoring to use return by value will correct this error:

```
C#  
  
public class Test  
{  
    public struct S1  
    {  
        public char x;  
    }  
  
    char Test1(char arg1, S1 arg2)  
    {  
        return arg1;  
    }  
}
```

# Compiler Error CS8167

Article • 09/24/2022 • 2 minutes to read

Cannot return by reference a member of parameter because it is not a ref or out parameter

## Example

The following sample generates CS8167:

C#

```
// CS8167.cs (11,20)

public class Test
{
    public struct S1
    {
        public char x;
    }

    ref char Test1(char arg1, S1 arg2)
    {
        return ref arg1;
    }
}
```

## To correct this error

To return a parameter that is not declared with `out` or `ref`, refactoring to return by value corrects this error:

C#

```
public class Test
{
    public struct S1
    {
        public char x;
    }

    char Test1(char arg1, S1 arg2)
    {
        return arg1;
    }
}
```



# Compiler Error CS8168

Article • 09/27/2022 • 2 minutes to read

Cannot return local by reference because it is not a ref local

## Example

The following sample generates CS8168:

C#

```
// CS8168.cs (8,14)

public class Test
{
    ref char Test1()
    {
        char l = default(char);

        return ref l;
    }
}
```

The scope of `l` is within the body of the method. If a reference to `l` were to leave the scope of this method, that reference would outlive the object to which it refers. The compiler's scope rules cause error CS8168 to be generated in this example.

## To correct this error

To return the value of a local, refactoring to return by value will correct this error:

C#

```
// CS8168.cs (8,14)

public class Test
{
    char Test1()
    {
        char l = default(char);

        return l;
    }
}
```

# Compiler Error CS8169

Article • 09/21/2022 • 2 minutes to read

Cannot return a member of local by reference because it is not a ref local

## Example

The following sample generates CS8169:

```
C#  
  
// CS8169.cs (17,15)  
  
public class Test  
{  
    public struct S1  
    {  
        public char x;  
    }  
  
    ref char Test1(char arg1, ref S1 arg2)  
    {  
        S1 l = default(S1);  
        // valid  
        ref char r = ref l.x;  
  
        // invalid  
        return ref l.x;  
    }  
}
```

Using a `ref` to `S1.x` within the `Test1` method is valid because use of the reference does not leave the scope of the value type `l`. Upon returning from `Test`, `l` would become invalid along with any references to its properties.

# Compiler Error CS8170

Article • 09/21/2022 • 2 minutes to read

Struct members cannot return 'this' or other instance members by reference

## Example

The following sample generates CS8170:

Value types (i.e. `structs`), are most commonly allocated on the stack. A value type allocated on the stack becomes invalid leaving the scope in which it was declared. The compiler avoids a reference to a variable that becomes invalid leaving the scope by generating this error.

C#

```
// CS8170.cs (8,14)

struct Program
{
    public int d;

    public ref int M()
    {
        return ref d;
    }
}

public class Other
{
    public void Method()
    {
        var p = new Program();
        var d = p.M();
    }
}
```

## To correct this error

Changing the method to not `ref`-return corrects this error:

C#

```
delegate void D();
```

```
struct Program
{
    public event D d;

    public D M()
    {
        return d;
    }
}
```

If a reference to a member is required, consider extending the scope of the value. For example:

C#

```
public struct Program
{
    public int d;
}

public static class Extensions
{
    public static ref readonly int RefD(this in Program program)
    {
        return ref program.d;
    }
}

public class Other
{
    public void Method()
    {
        var p = new Program();
        var d = p.RefD();
    }
}
```

# Compiler Error CS8171

Article • 09/21/2022 • 2 minutes to read

Cannot initialize a by-value variable with a reference

## Example

The following sample generates CS8171:

```
C#  
  
// CS8171.cs (8,13)  
  
class Test  
{  
    void A()  
    {  
        int a = 123;  
        ref int x = ref a;  
        var y = ref x;  
    }  
}
```

Remember that `var y = ref x` is implicitly `int y = ref x` where `int y` is a by-value variable.

## To correct this error

Removing the `ref` modifier from the right side of the assignment will correct this error:

```
C#  
  
class Test  
{  
    void A()  
    {  
        int a = 123;  
        ref int x = ref a;  
        var y = x;  
    }  
}
```

# Compiler Error CS8172

Article • 09/22/2022 • 2 minutes to read

Cannot initialize a by-reference variable with a value

## Example

The following sample generates CS8172:

```
C#  
  
// CS8172.cs (10,17)  
  
class C  
{  
    void M()  
    {  
        ref readonly int L() => ref (new int[1])[0];  
  
        ref readonly int x = ref L();  
        ref int y = x;  
    }  
}
```

## To correct this error

Assigning a reference to a variable to a by-reference variable corrects this error:

```
C#  
  
class C  
{  
    void M()  
    {  
        ref readonly int L() => ref (new int[1])[0];  
  
        ref readonly int x = ref L();  
        ref int y = ref x;  
    }  
}
```

# Compiler Error CS8173

Article • 09/21/2022 • 2 minutes to read

The expression must be of type because it is being assigned by reference

When assigning a reference to a variable, the type of the variables must match to be referenceable.

## Example

The following sample generates CS8173:

C#

```
// CS8173.cs (12,18)

class C
{
    void M()
    {
        string s = "s";
        object o = s;
        ref string rs = ref s;
        ref object ro = ref o;

        ro = ref s;
    }
}
```

## To correct this error

Assigning the reference to a variable of the correct type will correct this error:

C#

```
rs = ref s;
```

# Compiler Error CS8174

Article • 09/21/2022 • 2 minutes to read

A declaration of a by-reference variable must have an initializer

## Example

The following sample generates CS8174:

```
C#  
  
// CS8174.cs (7,22)  
  
class C  
{  
    void M()  
    {  
        int i = 0;  
        for (ref int rx; i < 5; i++) { }  
    }  
}
```

## To correct this error

Initializing the by-reference variable with a reference to a variable will correct this error:

```
C#  
  
class C  
{  
    void M()  
    {  
        int i = 0;  
        for (ref int rx = ref i; i < 5; i++) { }  
    }  
}
```

# Compiler Error CS8175

Article • 09/21/2022 • 2 minutes to read

Cannot use ref local inside an anonymous method, lambda expression, or query expression

Remember that expression capturing is a compile-time operation and by-reference refers to a run-time location.

## Example

The following sample generates CS8175:

```
C#  
  
// CS8175.cs (10,21)  
  
using System;  
class C  
{  
    void M()  
    {  
        ref readonly int x = ref (new int[1])[0];  
        Action a = () =>  
        {  
            int i = x;  
        };  
    }  
}
```

## To correct this error

Removing the use of by-reference variables corrects this error. In the example code, this can be done by first assigning the value of the referenced variable to a by-value variable:

```
C#  
  
using System;  
class C  
{  
    void M()  
    {  
        ref readonly int x = ref (new int[1])[0];  
    }  
}
```

```
int vx = x;
Action a = () =>
{
    int i = vx;
};
}
```

# Compiler Error CS8176

Article • 09/21/2022 • 2 minutes to read

Iterators cannot have by-reference locals

Iterator blocks use deferred execution, where the evaluation of an expression is delayed until its realized value is actually required. To manage that deferred execution state, iterator blocks use a state machine, capturing variable state in closures implemented in compiler-generated classes and properties. A local variable reference (on the stack) cannot be captured within the instance of a class in the heap, so the compiler issues an error.

## Example

The following sample generates CS8176:

```
C#  
  
// CS8176.cs (7,26)  
  
using System.Collections.Generic;  
class C  
{  
    IEnumerable<int> M()  
    {  
        ref readonly int x = ref (new int[1])[0];  
        int i = x;  
        yield return i;  
    }  
}
```

## To correct this error

Removing use of by-reference corrects this error:

```
C#  
  
class C  
{  
    IEnumerable<int> M()  
    {  
        int x = (new int[1])[0];  
        int i = x;  
        yield return i;  
    }  
}
```

```
    }  
}
```

# Compiler Error CS8177

Article • 09/21/2022 • 2 minutes to read

Async methods cannot have by-reference locals

To manage asynchronous state, `async` methods use a state machine, capturing variable state in closures implemented in compiler-generated classes and properties. A local variable reference (on the stack) cannot be captured within the instance of a class in the heap, so the compiler issues an error.

## Example

The following sample generates CS8177:

```
C#  
  
// CS8177.cs (20,26)  
  
using System.Threading.Tasks;  
  
class E  
{  
    public class Enumerator  
    {  
        public ref int Current => throw new  
System.NotImplementedException();  
        public bool MoveNext() => throw new  
System.NotImplementedException();  
    }  
  
    public Enumerator GetEnumerator() => new Enumerator();  
}  
  
class C  
{  
    public async static Task Test()  
    {  
        await Task.CompletedTask;  
  
        foreach (ref int x in new E())  
        {  
            System.Console.Write(x);  
        }  
    }  
}
```

# To correct this error

Changing the variable declaration to remove the `ref` modifier corrects this error:

C#

```
class C
{
    public async static Task Test()
    {
        await Task.CompletedTask;

        foreach (int x in new E())
        {
            System.Console.Write(x);
        }
    }
}
```

# Compiler Error CS8178

Article • 09/21/2022 • 2 minutes to read

'await' cannot be used in an expression containing a call to because it returns by reference

## Example

The following sample generates CS8178:

```
C#  
  
using System;  
using System.Threading.Tasks;  
  
class TestClass  
{  
    int x;  
    ref int Save(int y)  
    {  
        x = y;  
        return ref x;  
    }  
  
    async Task TestMethod()  
    {  
        Save(1) = await Task.FromResult(0);  
    }  
}
```

## To correct this error

Changing the use of the return by reference to be synchronous corrects the error:

```
C#  
  
async Task TestMethod()  
{  
    var x = await Task.FromResult(0);  
    Save(1) = x;  
}
```

# Compiler Error CS8210

Article • 10/28/2022 • 2 minutes to read

A tuple may not contain a value of type 'void'.

## Example

The following sample generates CS8210:

C#

```
// CS8210.cs
void Method()
{
}

void Test()
{
    var x = ("something", Method());
}
```

Your method returns `void`, so that method doesn't return a value. You can't use a method that returns `void` for a data member of a tuple.

## See also

- [Tuple](#)
- [Void](#)



# Compiler Error CS8333

Article • 10/01/2022 • 2 minutes to read

Cannot return by writable reference because it is a readonly variable

## Example

The following sample generates CS8333:

```
C#  
  
// CS8333.cs (3,36)  
class Program  
{  
    static ref T F4<T>(in T t) => ref t;  
}
```

## To correct this error

To return a reference to a readonly variable, refactoring to return `ref readonly` will correct this error:

```
C#  
  
class Program  
{  
    static ref readonly T F4<T>(in T t) => ref t;  
}
```

# Compiler Error CS8334

Article • 10/01/2022 • 2 minutes to read

Members of cannot be returned by writable reference because it is a readonly variable

## Example

The following sample generates CS8334:

C#

```
// CS8334.cs (5,14)

class Program
{
    static ref int M(in int arg1, in (int Alice, int Bob) arg2)
    {
        return ref arg2.Alice;
    }
}
```

## To correct this error

To return a reference to a read-only member, refactoring to return `ref readonly` will correct this error:

C#

```
class Program
{
    static ref readonly int M(in int arg1, in (int Alice, int Bob) arg2)
    {
        return ref arg2.Alice;
    }
}
```

# Compiler Error CS8354

Article • 10/01/2022 • 2 minutes to read

Cannot return 'this' by reference.

A `ref` argument must be an assignable variable or array element. `this` is not assignable and cannot be returned by reference.

## Example

The following sample generates CS8354:

C#

```
// CS8354.cs (6,20)

class Program
{
    ref Program M()
    {
        return ref this;
    }
}
```

# Compiler Error CS8373

Article • 10/01/2022 • 2 minutes to read

The left-hand side of a ref assignment must be a ref variable.

## Example

The following sample generates CS8373:

```
C#  
  
// CS8373.cs (6,90)  
  
public class C  
{  
    void M(int a, ref int b)  
    {  
        a = ref b;  
    }  
}
```

## To correct this error

To assign the value of a `ref` variable, removing the `ref` keyword corrects this error:

```
C#  
  
public class C  
{  
    void M(int a, ref int b)  
    {  
        a = b;  
    }  
}
```

# Compiler Error CS8374

Article • 10/01/2022 • 2 minutes to read

Cannot ref-assign to because has a narrower escape scope than.

## Example

The following sample generates CS8374 because the reference contained in `r1` is externally visible but the scope of `rx`'s value (its lifetime) is local:

C#

```
// CS8374.cs (6,9)

class C
{
    void M(ref int r1)
    {
        int x = 0;
        ref int rx = ref x;
        for (int i = 0; i < (r1 = ref rx); i++)
        {
        }
    }
}
```

# Compiler Error CS8403

Article • 09/15/2021 • 2 minutes to read

Method 'method' with an iterator block must be 'async' to return  
'{IAsyncEnumerable<T>|IAsyncEnumerator<T>}'

## To correct this error

Mark your method with the `async` modifier.

## See also

- [async \(C# Reference\)](#)



# Compiler Error CS8410

Article • 09/15/2021 • 2 minutes to read

'type': type used in an asynchronous using statement must be implicitly convertible to 'System.IAsyncDisposable' or implement a suitable 'DisposeAsync' method.

The expression inside an `await using` statement must have a `DisposeAsync` method.

## To correct this error

Remove the `await using` keywords, or implement a suitable `DisposeAsync` method.

## Example

C#

```
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        // error CS8410: 'Example': type used in an asynchronous using
        // statement
        // must be implicitly convertible to 'System.IAsyncDisposable' or
        // implement
        // a suitable 'DisposeAsync' method.
        await using var example = new Example();
    }
}

class Example
```

## See also

- [Implement a `DisposeAsync` method](#)



# Compiler Error CS8411

Article • 09/15/2021 • 2 minutes to read

Asynchronous foreach statement cannot operate on variables of type 'type' because 'type' does not contain a suitable public instance definition for 'GetAsyncEnumerator'

`await foreach` statement operates only on types having a definition of `GetAsyncEnumerator`, such as `IAsyncEnumerable<T>`.

## To correct this error

Replace `await foreach` with `foreach`.

## Example

C#

```
using System.Collections.Generic;
using System.Threading.Tasks;

class Program
{
    async Task Example(IAsyncEnumerable<int> enumerator)
    {
        // error CS8411: Asynchronous foreach statement cannot operate on
        // variables
        // of type 'IAsyncEnumerable<T>' because 'IAsyncEnumerable<T>' does
        // not
        // contain a suitable public instance definition for
        // 'GetAsyncEnumerator'
        await foreach (int i in enumerator)
        {
        }
    }
}
```

# Compiler Error CS8803

Article • 09/17/2022 • 2 minutes to read

Top-level statements must precede namespace and type declarations.

## Example

The following sample generates CS8803:

```
C#  
  
// CS8803.cs (0,0)  
  
public record Person  
{  
    public string? GivenName { get; set; }  
    public string? FamilyName { get; set; }  
}  
  
int i = 0;
```

In a file with top-level statements, top-level statements must occur prior to any type declarations.

## To correct this error

Move the code before the namespace declaration:

```
C#  
  
int i = 0;  
  
public record Person  
{  
    public string? GivenName { get; set; }  
    public string? FamilyName { get; set; }  
}
```

It is common that types are declared within their own file, which would also correct this error by separating the type declaration from the top-level statements.

# Compiler Error CS8812

Article • 09/17/2022 • 2 minutes to read

Cannot convert &method group to non-function pointer type.

## Example

The following sample generates CS8812:

```
C#  
  
// CS8812.cs (6,22)  
  
unsafe class C  
{  
    static void M()  
    {  
        void* ptr1 = &M;  
    }  
}
```

The address of an expression (e.g., `&M`) has no type and thus cannot be assigned to a non-function pointer variable.

## To correct this error

Explicitly convert the expression to the required type (e.g., a `void delegate`):

```
C#  
  
unsafe class C  
{  
    static void M()  
    {  
        void* ptr1 = (delegate*<void>)&M;  
    }  
}
```

## See also

- Shouldn't method addresses be implicitly convertible to `void*` and thus allowing direct casts to function pointers with mismatched signatures? ↗

- Casting function pointer directly to void and then a function pointer type causes crash. ↗

# Compiler Error CS8515

Article • 10/27/2022 • 2 minutes to read

Parentheses are required around the switch governing expression.

## Example

The following code snippets generate CS8515:

Missing parentheses: ( ):

```
C#  
  
// CS8515.cs  
int x = 5;  
switch x  
{  
    case 5:  
        break;  
}
```

Missing curly brackets: { }:

```
C#  
  
// CS8515.cs  
int x = 5;  
switch (x)  
    case 5:  
        break;
```

## The correct format

```
C#  
  
int x = 5;  
switch (x)  
{  
    case 5:  
        break;  
}
```

For more information, see [Switch](#).

# Compiler Error CS9043

Article • 10/01/2022 • 2 minutes to read

Ref returning properties cannot be required.

The `required` modifier specifies that a member is required to be set during object initialization (i.e., via an object initializer.) For a property to be set within an object initializer, it must have a `set` accessor (a setter). `ref`-returning properties cannot have a setter and thus cannot also include the `required` modifier.

## Example

The following sample generates CS9043:

```
C#  
  
// CS9043.cs (5,29)  
  
class C  
{  
    private int i;  
    public required ref readonly int Number => ref i;  
}
```

## To correct this error

To have a `required` property, refactoring the property to return by value corrects this error:

```
C#  
  
public required int Number  
{  
    get  
    {  
        return i;  
    }  
    set  
    {  
        i = value;  
    }  
}
```

# Compiler Error CS9050

Article • 09/21/2022 • 2 minutes to read

A `ref` field cannot refer to a `ref struct`.

The compiler does not support the `ref` modifier on a field within a struct (to declare a stack-allocated field) of a type already declared stack-allocated (`ref struct`).

## Example

The following sample generates CS9050:

```
C#  
  
// CS9050.cs (0,0)  
ref struct Color  
{  
    public float r, g, b;  
}  
ref struct Group  
{  
    public ref Color color;  
}
```

## To correct this error

In this example, it is most likely a typo to have included a `ref` modifier on a field of a `ref struct` type within the declaration of a `ref struct`. Removing the `ref` modifier corrects this error.

```
C#  
  
ref struct Color  
{  
    public float r, g, b;  
}  
ref struct Group  
{  
    public Color color;  
}
```

# Compiler Warning (level 1) CS0183

Article • 09/15/2021 • 2 minutes to read

The given expression is always of the provided ('type') type

If a conditional statement always evaluates to `true`, then you do not need a conditional statement. This warning occurs when you try to evaluate a type using the `is` operator. If the evaluation is a value type, then the check is unnecessary.

The following sample generates CS0183:

```
C#  
  
// CS0183.cs  
// compile with: /W:1  
using System;  
public class Test  
{  
    public static void F(Int32 i32, String str)  
    {  
        if (str is Object)          // OK  
            Console.WriteLine( "str is an object" );  
        else  
            Console.WriteLine( "str is not an object" );  
  
        if (i32 is Object)    // CS0183  
            Console.WriteLine( "i32 is an object" );  
        else  
            Console.WriteLine( "i32 is not an object" ); // never reached  
    }  
  
    public static void Main()  
    {  
  
        F(0, "CS0183");  
        F(120, null);  
    }  
}
```

# Compiler Warning (level 1) CS0184

Article • 09/15/2021 • 2 minutes to read

The given expression is never of the provided ('type') type

The expression can never be **true** because the variable you are testing is neither declared as *type* nor derived from *type*.

The following sample generates CS0184:

C#

```
// CS0184.cs
// compile with: /W:1
class MyClass
{
    public static void Main()
    {
        int i = 0;
        if (i is string) // CS0184
            i++;
    }
}
```

# Compiler Warning (level 1) CS0197

Article • 03/11/2022 • 2 minutes to read

Passing 'argument' as ref or out or taking its address may cause a runtime exception because it is a field of a marshal-by-reference class

Any class that derives, directly or indirectly, from [MarshalByRefObject](#) is a marshal-by-reference class. Such a class can be marshalled by reference across process and machine boundaries. Thus, instances of this class could be proxies to remote objects. You cannot pass a field of a proxy object as [ref](#) or [out](#). So, you cannot pass fields of such a class as [ref](#) or [out](#), unless the instance is [this](#), which can not be a proxy object.

## Example

The following sample generates CS0197.

```
C#  
  
// CS0197.cs  
// compile with: /W:1  
class X : System.MarshalByRefObject  
{  
    public int i;  
}  
  
class M  
{  
    public int i;  
    static void AddSeventeen(ref int i)  
    {  
        i += 17;  
    }  
  
    static void Main()  
    {  
        X x = new X();  
        x.i = 12;  
        AddSeventeen(ref x.i);    // CS0197  
  
        // OK  
        M m = new M();  
        m.i = 12;  
        AddSeventeen(ref m.i);  
    }  
}
```

# Compiler Warning (level 1) CS0420

Article • 09/15/2021 • 2 minutes to read

'identifier': a reference to a volatile field will not be treated as volatile

A volatile field should not normally be passed using a `ref` or `out` parameter, since it will not be treated as volatile within the scope of the function. There are exceptions to this, such as when calling an interlocked API. As with any warning, you may use the `#pragma warning` to disable this warning in those rare cases where you are intentionally using a volatile field as a reference parameter.

The following sample generates CS0420:

```
C#  
  
// CS0420.cs  
// compile with: /W:1  
using System;  
  
class TestClass  
{  
    private volatile int i;  
  
    public void TestVolatile(ref int ii)  
    {  
    }  
  
    public static void Main()  
    {  
        TestClass x = new TestClass();  
        x.TestVolatile(ref x.i);    // CS0420  
    }  
}
```

# Compiler Warning (level 1) CS0465

Article • 09/15/2021 • 2 minutes to read

Introducing a 'Finalize' method can interfere with destructor invocation. Did you intend to declare a destructor?

This warning occurs when you create a class with a method whose signature is `public virtual void Finalize.`

If such a class is used as a base class and if the deriving class defines a finalizer, the finalizer will override the base class `Finalize` method, not `Finalize`.

## Example

The following sample generates CS0465.

```
C#  
  
// CS0465.cs  
// compile with: /target:library  
class A  
{  
    public virtual void Finalize() {} // CS0465  
}  
  
// OK  
class B  
{  
    ~B() {}  
}
```

# Compiler Warning (level 1) CS0602

Article • 09/15/2021 • 2 minutes to read

The feature 'old\_feature' is deprecated. Please use 'new\_feature' instead

A language feature used in your code (*old\_feature*) is still supported, but that support may be removed in a future release. Instead, you should use the recommended syntax (*new\_feature*).

 **Note**

This compiler warning is no longer used in Roslyn.

# Compiler warning (level 1) CS0612

Article • 09/15/2021 • 2 minutes to read

'member' is obsolete

The class designer marked a member with the [Obsolete attribute](#). This means that the member might not be supported in a future version of the class.

The following sample shows how accessing an obsolete member generates CS0612:

C#

```
// CS0612.cs
// compile with: /W:1
using System;

class MyClass
{
    [Obsolete]
    public static void ObsoleteMethod()
    {
    }

    [Obsolete]
    public static int ObsoleteField;
}

class MainClass
{
    static public void Main()
    {
        MyClass.ObsoleteMethod();      // CS0612 here: method is deprecated
        MyClass.ObsoleteField = 0;     // CS0612 here: field is deprecated
    }
}
```

# Compiler Warning (level 1) CS0626

Article • 09/15/2021 • 2 minutes to read

Method, operator, or accessor 'method' is marked external and has no attributes on it. Consider adding a [DllImport](#) attribute to specify the external implementation.

A method marked `extern` should also be marked with an attribute, for example, the [DllImport](#) attribute.

The attribute specifies where the method is implemented. At run time, the program will need this information.

The following sample generates CS0626:

C#

```
// CS0626.cs
// compile with: /warnaserror
using System.Runtime.InteropServices;

public class MyClass
{
    static extern public void M(); // CS0626
    // try the following line
    // [DllImport("mydll.dll")] static extern public void M();

    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS0657

Article • 03/15/2023 • 2 minutes to read

'attribute modifier' is not a valid attribute location for this declaration. Valid attribute locations for this declaration are 'locations'. All attributes in this block will be ignored.

The compiler found an attribute modifier in an invalid location. See [Attribute Targets](#) for more information.

The following sample generates CS0657:

```
C#  
  
// CS0657.cs  
// compile with: /target:library  
public class TestAttribute : System.Attribute {}  
[return: Test] // CS0657 return not valid on a class  
class Class1 {}
```

# Compiler Warning (level 1) CS0658

Article • 03/15/2023 • 2 minutes to read

'attribute modifier' is not a recognized attribute location. All attributes in this block will be ignored.

An invalid attribute modifier was specified. See [Attribute Targets](#) for more information.

The following sample generates CS0658:

C#

```
// CS0658.cs
using System;
public class TestAttribute : Attribute {}
[badAttributeLocation: Test]    // CS0658, badAttributeLocation is invalid
class ClassTest
{
    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS0672

Article • 09/15/2021 • 2 minutes to read

Member 'member1' overrides obsolete member 'member2'. Add the `Obsolete` attribute to 'member1'

The compiler found an `override` to a method marked as `obsolete`. However, the overriding method was not itself marked as obsolete. The overriding method will still generate [CS0612](#), if called.

Review your method declarations and explicitly indicate whether a method (and all of its overrides) should be marked `obsolete`.

The following sample generates CS0672:

```
C#  
  
// CS0672.cs  
// compile with: /W:1  
class MyClass  
{  
    [System.Obsolete]  
    public virtual void ObsoleteMethod()  
    {  
    }  
}  
  
class MyClass2 : MyClass  
{  
    public override void ObsoleteMethod() // CS0672  
    {  
    }  
}  
  
class MainClass  
{  
    static public void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS0684

Article • 09/15/2021 • 2 minutes to read

'interface' interface marked with 'CoClassAttribute' not marked with 'ComImportAttribute'

If you specify **CoClassAttribute** on an interface, you must also specify **ComImportAttribute**.

The following sample generates CS0684:

```
C#  
  
// CS0684.cs  
// compile with: /W:1  
using System;  
using System.Runtime.InteropServices;  
  
[CoClass(typeof(C))] // CS0684  
// try the following line instead  
// [CoClass(typeof(C)), ComImport]  
interface I  
{  
}  
  
class C  
{  
    static void Main() {}  
}
```

# Compiler Warning (level 1) CS0688

Article • 09/15/2021 • 2 minutes to read

'method1' has a link demand, but overrides or implements 'method2' which does not have a link demand. A security hole may exist.

The link demand set up on the derived class method can easily be circumvented by calling the base class method. To close the security hole, the base class method needs to also use the link demand. For more information, see [Demand vs. LinkDemand](#).

## Example

The following sample generates CS0688. To resolve the warning without modifying the base class, remove the security attribute from the overriding method. This will not solve the security problem.

C#

```
// CS0688.cs
// compile with: /W:1
using System;
using System.Security.Permissions;

class Base
{
    //Uncomment the following line to close the security hole
    //#[FileIOPermission(SecurityAction.LinkDemand, All=@"C:\\")]
    public virtual void DoScaryFileStuff()
    {
    }
}

class Derived: Base
{
    #[FileIOPermission(SecurityAction.LinkDemand, All=@"C:\\")] // CS0688
    public override void DoScaryFileStuff()
    {
    }
    static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS0809

Article • 09/15/2021 • 2 minutes to read

Obsolete member 'memberA' overrides non-obsolete member 'memberB'.

Typically, a member that is marked as obsolete should not override a member that is not marked as obsolete. This warning is generated in Visual Studio 2008 but not in Visual Studio 2005.

## To correct this error

Remove the `Obsolete` attribute from the overriding member, or add it to the base class member.

## Example

```
C#  
  
// CS0809.cs  
public class Base  
{  
    public virtual void Test1()  
    {  
    }  
}  
public class C : Base  
{  
    [System.Obsolete()]  
    public override void Test1() // CS0809  
    {  
    }  
}
```

# Compiler Warning (level 1) CS0824

Article • 09/15/2021 • 2 minutes to read

Constructor 'name' is marked external.

A constructor may be marked as extern. However, the compiler cannot verify that the constructor actually exists. Therefore the warning is generated.

## To remove this warning

1. Use a pragma warning directive to ignore it.
2. Move the constructor inside the type.

## Example

The following code generates CS0824:

```
C#  
  
// cs0824.cs  
public class C  
{  
    extern C(); // CS0824  
    public static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [extern](#)
- [#pragma warning](#)



# Compiler Warning (level 1) CS1030

Article • 09/15/2021 • 2 minutes to read

```
#warning: 'text'
```

Displays the text of a warning defined with the `#warning` directive.

The following sample shows how to create a user-defined warning:

C#

```
// CS1030.cs
class Sample
{
    static void Main()
    {
        #warning Let's give a warning here
    }
}
```

Compilation produces the following output:

Console

```
example.cs(6,16): warning CS1030: #warning: 'Let's give a warning here'
```

# Compiler Warning (level 1) CS1058

Article • 09/15/2021 • 2 minutes to read

A previous catch clause already catches all exceptions. All exceptions thrown will be wrapped in a `System.Runtime.CompilerServices.RuntimeWrappedException`

This attribute causes CS1058 if a `catch()` block has no specified exception type after a `catch (System.Exception e)` block. The warning advises that the `catch()` block will not catch any exceptions.

A `catch()` block after a `catch (System.Exception e)` block can catch non-CLS exceptions if the `RuntimeCompatibilityAttribute` is set to false in the `AssemblyInfo.cs` file: `[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = false)]`. If this attribute is not set explicitly to false, all thrown non-CLS exceptions are wrapped as Exceptions and the `catch (System.Exception e)` block catches them. For more information, see [How to catch a non-CLS exception](#).

## Example

The following example generates CS1058.

```
C#  
  
// CS1058.cs  
// CS1058 expected  
using System.Runtime.CompilerServices;  
  
// the following attribute is set to true by default in the C# compiler  
// set to false in your source code to resolve CS1058  
[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = true)]  
  
class TestClass  
{  
    static void Main()  
    {  
        try {}  
  
        catch (System.Exception e) {  
            System.Console.WriteLine("Caught exception {0}", e);  
        }  
  
        catch {} // CS1058. This line will never be reached.  
    }  
}
```

# Compiler Warning (level 1) CS1060

Article • 11/05/2021 • 2 minutes to read

Use of possibly unassigned field 'name'. Struct instance variables are initially unassigned if struct is unassigned.

Struct members are initialized to their default value if you do not explicitly initialize them. The default value for class types (and other reference types) is null. If the class is not initialized before any attempt to access it, a `NullReferenceException` will be thrown at run time. The compiler cannot determine definitively whether the class member will be initialized or not, and so CS1060 is a warning and not an error.

## To correct this error

1. Provide a constructor for the `struct` that initializes all its members.

## Example

The following code generates CS1060 because the class type `U` is a member of the `struct S` but is never initialized.

```
C#  
  
// cs1060.cs  
namespace CS1060  
{  
    public class U  
    {  
        public int i;  
    }  
  
    public struct S  
    {  
        public U u;  
        // Add constructor to correct the error.  
        //public S(int val)  
        //{
/>        //    u = new U() { i = val };  
        //}  
    }  
    public class Test  
    {  
        static void Main()  
        {  
            S s;
```

```
s.u.i = 5; // CS1060

//Try these lines instead, and uncomment the constructor in S
// S s = new S(0);
// s.u.i = 5;

}

}
```

## See also

- [Structure types](#)

# Compiler Warning (level 1) CS1200

Article • 09/15/2021 • 2 minutes to read

The feature 'invalid feature' is deprecated. Please use 'valid feature' instead.

The feature you are attempting to use is now deprecated. Update your code to use the valid feature instead.

# Compiler Warning (level 1) CS1201

Article • 09/15/2021 • 2 minutes to read

The feature 'invalid feature' is deprecated. Please use 'valid feature' instead.

The feature you are attempting to use is now deprecated. Update your code to use the valid feature instead.

# Compiler Warning (level 1) CS1202

Article • 09/15/2021 • 2 minutes to read

The feature 'invalid feature' is deprecated. Please use 'valid feature' instead.

The feature you are attempting to use is now deprecated. Update your code to use the valid feature instead.

# Compiler Warning (level 1) CS1203

Article • 09/15/2021 • 2 minutes to read

The feature 'feature' is deprecated. Please use 'feature' instead.

The feature 'invalid feature' is deprecated. Please use 'valid feature' instead.

The feature you are attempting to use is now deprecated. Update your code to use the valid feature instead.

# Compiler Warning (level 1) CS1522

Article • 09/15/2021 • 2 minutes to read

## Empty switch block

The compiler detected a `switch` block with no `case` or `default` statement. A `switch` block must have one or more `case` or `default` statements.

The following sample generates CS1522:

C#

```
// CS1522.cs
// compile with: /W:1
using System;
class X
{
    public static void Main()
    {
        int i = 6;

        switch(i)    // CS1522
        {
            // add something to the switch block, for example:
            /*
            case (5):
                Console.WriteLine("5");
                return;
            default:
                Console.WriteLine("not 5");
                return;
            */
        }
    }
}
```

# Compiler Warning (level 1) CS1570

Article • 09/15/2021 • 2 minutes to read

XML comment on 'construct' has badly formed XML — 'reason'

When using [DocumentationFile](#), any comments in the source code must be in XML. Any error with your XML markup will generate CS1570. For example:

- If you are passing a string to a `cref`, such as in an `<exception>` tag, the string must be enclosed in double quotation marks.
- If you're using a tag that doesn't have a closing tag, such as `<seealso>`, you must specify a forward slash before the closing angle bracket.
- If you need to use a greater-than or less-than symbol in the text of description, you need to represent them with `&gt;` or `&lt;`. Alternatively, you can use CDATA.
- The file or path attribute on an `<include>` tag was missing or improperly formed.

The following sample generates CS1570:

C#

```
public static class CompareFive
{
    // the following line generates CS1570
    /// <summary> returns true if < 5 </summary>
    // try one of the following instead
    // /// <summary> returns true if &lt; 5 </summary>
    // /// <summary><![CDATA[ returns true if < 5 ]]></summary>
    public static bool LessThanFive(int x) => x < 5;
}
```

# Compiler Warning (level 1) CS1574

Article • 09/15/2021 • 2 minutes to read

XML comment on 'construct' has syntactically incorrect cref attribute 'name'

A string passed to a cref tag, for example, within an <exception> tag, referred to a member that is not available within the current build environment. The string that you pass to a cref tag must be the syntactically correct name of a member or field.

For more information, see [Recommended Tags for Documentation Comments](#).

The following sample generates CS1574:

C#

```
// CS1574.cs
// compile with: /W:1 /doc:x.xml
using System;

/// <exception cref="System.Console.WriteLine">An exception class.
</exception> // CS1574
// instead, uncomment and try the following line
// /// <exception cref="System.Console.WriteLine">An exception class.
// </exception>
class EClass : Exception
{
}

class TestClass
{
    public static void Main()
    {
        try
        {
        }
        catch(EClass)
        {
        }
    }
}
```

# Compiler Warning (level 1) CS1580

Article • 09/15/2021 • 2 minutes to read

Invalid type for parameter 'parameter number' in XML comment cref attribute

When attempting to reference an overload form of a method, the compiler detected a syntax error. Typically, this indicates that the parameter name, and not the type, was specified. A malformed line will appear in the generated XML file.

The following sample generates CS1580:

```
C#  
  
// CS1580.cs  
// compile with: /W:1 /doc:x.xml  
using System;  
  
/// <seealso cref="Test(i)"/> // CS1580  
// try the following line instead  
// /// <seealso cref="Test(int)"/>  
public class MyClass  
{  
    /// <summary>help text</summary>  
    public static void Main()  
    {  
    }  
  
    /// <summary>help text</summary>  
    public void Test(int i)  
    {  
    }  
  
    /// <summary>help text</summary>  
    public void Test(char i)  
    {  
    }  
}
```

# Compiler Warning (level 1) CS1581

Article • 09/15/2021 • 2 minutes to read

Invalid return type in XML comment cref attribute

When attempting to reference a method, the compiler detected an error due to an invalid return type.

## Example

The following sample generates CS1581:

C#

```
// CS1581.cs
// compile with: /W:1 /doc:x.xml

/// <summary>help text</summary>
public class MyClass
{
    /// <summary>help text</summary>
    public static void Main()
    {
    }

    /// <summary>help text</summary>
    public static explicit operator int(MyClass f)
    {
        return 0;
    }
}

/// <seealso cref="MyClass.explicit operator intt(MyClass)"/> // CS1581
// try the following line instead
// /// <seealso cref="MyClass.explicit operator int(MyClass)"/>
public class MyClass2
{}
```

# Compiler Warning (level 1) CS1584

Article • 09/15/2021 • 2 minutes to read

XML comment on 'member' has syntactically incorrect cref attribute 'invalid\_syntax'

One of the parameters passed to a tag for documentation comments has invalid syntax.  
For more information, see [Recommended Tags for Documentation Comments](#).

## Example

The following sample generates CS1584.

C#

```
// CS1584.cs
// compile with: /W:1 /doc:CS1584.xml
/// <remarks>Test class</remarks>
public class Test
{
    /// <remarks>Called in <see cref="Test.Mai()"/>.</remarks> // CS1584
    // try the following line instead
    // /// <remarks>Called in <see cref="Test.Main()"/>.</remarks>
    public static void Test2() {}

    /// <remarks>Main method</remarks>
    public static void Main() {}
}
```

# Compiler Warning (level 1) CS1589

Article • 09/15/2021 • 2 minutes to read

Unable to include XML fragment 'fragment' of file 'file' -- reason

The syntax (*fragment*) of a `<include>` tag, which referenced a file (`file`), was incorrect for the specified *reason*.

A malformed line will be placed in the generated XML file.

The following sample generates CS1589:

C#

```
// CS1589.cs
// compile with: /W:1 /doc:CS1589_out.xml

/// <include file='CS1589.doc' path='MyDocs/MyMembers[@name="test"]/*' />
// CS1589
// try the following line instead
// /// <include file='CS1589.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS1590

Article • 09/15/2021 • 2 minutes to read

Invalid XML include element -- Missing file attribute

A path or doc attribute, passed to the `<include>` tag, was missing or incomplete.

The following sample generates CS1590:

C#

```
// CS1590.cs
// compile with: /W:1 /doc:x.xml

/// <include path='MyDocs/MyMembers[@name="test"]/*' />    // CS1590
// try the following line instead
// /// <include file='x.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS1592

Article • 09/15/2021 • 2 minutes to read

Badly formed XML in included comments file -- 'reason'

A problem, reported as *reason*, was found in the file specified by the [`<include>`](#) tag.

# Compiler Warning (level 1) CS1598

Article • 09/15/2021 • 2 minutes to read

XML parser could not be loaded for the following reason: 'reason'. The XML documentation file 'file' will not be generated.

The [DocumentationFile](#) option was specified, but the compiler could not find and load msxml3.dll. Make sure that the file msxml3.dll is installed and registered.

# Compiler Warning (level 1) CS1607

Article • 09/03/2022 • 2 minutes to read

## Assembly generation -- reason

A warning was generated from the assembly-creation phase of the compilation.

If you are building a 64-bit application on a 32-bit operating system, you must ensure that 64-bit versions of all referenced assemblies are installed on the target operating system.

All x86-specific common language runtime (CLR) assemblies have 64-bit counterparts (every CLR assembly will exist on all operating systems). Therefore, you can safely ignore CS1607 for CLR assemblies.

You can ignore this warning if you encounter it when you create an [AssemblyInformationalVersionAttribute](#). The informational version is a string that attaches additional version information to an assembly; this information is not used at run time. Although you can specify any text, a warning message appears on compilation if the string is not in the format that is used by the assembly version number, or if it is in that format but contains wildcard characters. This warning is harmless.

For more information, see [Al.exe Tool Errors and Warnings](#).

# Compiler Warning (level 1) CS1616

Article • 09/15/2021 • 2 minutes to read

Option 'option' overrides attribute 'attribute' given in a source file or added module

This warning occurs if the assembly attributes [AssemblyKeyFileAttribute](#) or [AssemblyKeyNameAttribute](#) found in source conflict with the [KeyFile](#) or [KeyContainer](#) command line option or key file name or key container specified in the Project Properties.

For the example below, assume you have a key file named `cs1616.snk`. Generate this file with the command line:

Console

```
sn -k CS1616.snk
```

The following sample generates CS1616:

C#

```
// CS1616.cs
// compile with: /keyfile:cs1616.snk
using System.Reflection;

// To fix the error, remove the next line
[assembly: AssemblyKeyFile("cs1616b.snk")] // CS1616

class C
{
    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS1633

Article • 09/15/2021 • 2 minutes to read

Unrecognized #pragma directive

The pragma used was not one of the known pragmas supported by the C# compiler. To resolve this error, use only pragmas supported.

The following sample generates CS1633:

C#

```
// CS1633.cs
// compile with: /W:1
#pragma unknown // CS1633

class C
{
    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS1634

Article • 09/15/2021 • 2 minutes to read

Expected disable or restore

This error occurs if a #pragma warning clause is badly formed, such as if disable or restore was omitted. For more information, see the [#pragma warning](#) topic.

## Example

The following sample generates CS1634:

C#

```
// CS1634.cs
// compile with: /W:1

#pragma warning // CS1634
// Try this instead:
// #pragma warning disable 0219

class MyClass
{
    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS1635

Article • 09/15/2021 • 2 minutes to read

Cannot restore warning 'warning code' because it was disabled globally

This warning occurs if you use the `/nowarn` command line option or project setting to disable a warning for the entire compilation unit, but you use `#pragma warning restore` to attempt to restore that warning. To resolve this error, remove the `/nowarn` command line option or project setting, or remove the `#pragma warning restore` for any warnings you are disabling via the command line or project settings. For more information, see [#pragma warning](#).

The following sample generates CS1635:

```
C#  
  
// CS1635.cs  
// compile with: /w:1 /nowarn:162  
  
enum MyEnum {one=1,two=2,three=3};  
  
class MyClass  
{  
    public static void Main()  
    {  
        #pragma warning disable 162  
  
        if (MyEnum.three == MyEnum.two)  
            System.Console.WriteLine("Duplicate");  
  
        #pragma warning restore 162  
    }  
}
```

# Compiler Warning (level 1) CS1645

Article • 09/15/2021 • 2 minutes to read

Feature 'feature' is not part of the standardized ISO C# language specification, and may not be accepted by other compilers

The feature you are using is not part of the ISO standard. Code using this feature may not compile on other compilers.

C#

```
// CS1645.cs
// compile with: /W:1 /t:module /langversion:ISO-1
[assembly:System.CLSCompliant(false)]
// To suppress the warning use the switch: /nowarn:1645
[module:System.CLSCompliant(false)]    // CS1645
class Test
{
}
```

# Compiler Warning (level 1) CS1658

Article • 09/15/2021 • 2 minutes to read

'warning text'. See also error 'error code'

The compiler emits this warning when it overrides an error with a warning. For information about the problem, refer to the error mentioned. To find the appropriate error from within the Visual Studio IDE, use the index. For example, if the text above reads "See also error 'CS1037'," look for CS1037 in the index.

## Example

The following example generates CS1658.

```
C#  
  
// CS1658.cs  
// compile with: /doc:x.xml  
// CS1584 expected  
/// <summary>  
/// </summary>  
public class C  
{  
    /// <see cref="C.F(params object[])"/> // CS1658  
    public static void M()  
    {  
    }  
  
    /// <summary>  
    /// </summary>  
    public void F(params object[] o)  
    {  
    }  
  
    static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS1682

Article • 09/15/2021 • 2 minutes to read

Reference to type 'type' claims it is nested within 'nested type', but it could not be found

This error arises when you import references that do not agree with other references or with code you have written. A common way to get this error is to write code that refers to a class in metadata, and then you either delete that class or modify its definition.

## Example 1

```
C#  
  
// CS1682.cs  
// compile with: /target:library /keyfile:mykey.snk  
public class A {  
    public class N1 {}  
}
```

## Example 2

```
C#  
  
// CS1682_b.cs  
// compile with: /target:library /reference:CS1682.dll  
using System;  
public class Ref {  
  
    public static A A1() {  
        return new A();  
    }  
  
    public static A.N1 N1() {  
        return new A.N1();  
    }  
}
```

## Example 3

```
C#  
  
// CS1682_c.cs  
// compile with: /target:library /keyfile:mykey.snk /out:CS1682.dll
```

```
public class A {  
    public void M1() {}  
}
```

## Example 4

The following sample generates CS1682.

C#

```
// CS1682_d.cs  
// compile with: /reference:CS1682.dll /reference:CS1682_b.dll /W:1  
// CS1682 expected  
class Tester {  
    static void Main()  
    {  
        Ref.A1().M1();  
    }  
}
```

# Compiler Warning (level 1) CS1683

Article • 09/15/2021 • 2 minutes to read

Reference to type 'Type Name' claims it is defined in this assembly, but it is not defined in source or any added modules

This error can occur when you are importing an assembly that contains a reference back to the assembly you are currently compiling, but the assembly being compiled contains nothing matching the reference. One way to get to this situation is to compile your assembly, which initially does contain the member that the assembly being imported is referencing. Then you update your assembly, mistakenly removing the members that the imported assembly is referencing.

# Compiler Warning (level 1) CS1684

Article • 09/15/2021 • 2 minutes to read

Reference to type 'Type Name' claims it is defined in 'Namespace', but it could not be found

This error can be caused by a reference inside one namespace referring to a type that it says exists inside a second namespace, but the type does not exist. For example, mydll.dll says that type A exists inside yourdll.dll, but no such type exists inside yourdll.dll. One possible cause of this error is that the version of yourdll.dll you are using is too old and A has not yet been defined.

The following sample generates CS1684.

## Example 1

```
C#  
  
// CS1684_a.cs  
// compile with: /target:library /keyfile:CS1684.key  
public class A {  
    public void Test() {}  
}  
  
public class C2 {}
```

## Example 2

```
C#  
  
// CS1684_b.cs  
// compile with: /target:library /r:cs1684_a.dll  
// post-build command: del /f CS1684_a.dll  
using System;  
public class Ref  
{  
    public static A GetA() { return new A(); }  
    public static C2 GetC() { return new C2(); }  
}
```

## Example 3

We now rebuild the first assembly, leaving out the definition of the class C2 not to be defined in the recompilation.

```
C#  
  
// CS1684_c.cs  
// compile with: /target:library /keyfile:CS1684.key /out:CS1684_a.dll  
public class A {  
    public void Test() {}  
}
```

## Example 4

This module references the second module by means of the identifier Ref. But the second module contains a reference to the class C2, which no longer exists because of the compilation in the previous step, and therefore the CS1684 error message is returned from the compilation of this module.

```
C#  
  
// CS1684_d.cs  
// compile with: /reference:cs1684_a.dll /reference:cs1684_b.dll  
// CS1684 expected  
class Tester  
{  
    public static void Main()  
    {  
        Ref.GetA().Test();  
    }  
}
```

# Compiler Warning (level 1) CS1685

Article • 09/15/2021 • 2 minutes to read

The predefined type 'System.type name' is defined in multiple assemblies in the global alias; using definition from 'File Name'

This error occurs when a predefined system type such as System.int32 is found in two assemblies. One way this can happen is if you are referencing mscorelib from two different places, such as trying to run the .NET Framework versions 1.0 and 1.1 side-by-side.

The compiler will use the definition from only one of the assemblies. The compiler searches only global aliases, does not search libraries defined **/reference**. If you have specified **/nostdlib**, the compiler will search for [Object](#), and in the future start all searches for predefined types in the file where it found [Object](#).

# Compiler Warning (level 1) CS1687

Article • 09/15/2021 • 2 minutes to read

Source file has exceeded the limit of 16,707,565 lines representable in the PDB, debug information will be incorrect

The PDB and debugger have some limitations about how big a file can be. If the source file is too big, the debugger will not behave properly beyond that limit. The user should either not emit debug information for that file by possibly using `#line hidden`, or they should find a way to shrink the file, possibly by splitting the file into multiple files. They might want to use the `partial` keyword to split up a large class.

# Compiler Warning (level 1) CS1690

Article • 03/11/2022 • 2 minutes to read

Accessing a member on 'member' may cause a runtime exception because it is a field of a marshal-by-reference class

This warning occurs when you try to call a method, property, or indexer on a member of a class that derives from [MarshalByRefObject](#), and the member is a value type. Objects that inherit from `MarshalByRefObject` are typically intended to be marshalled by reference across an application domain. If any code ever attempts to directly access the value-type member of such an object across an application domain, a runtime [InvalidOperationException](#) will occur. To resolve the warning, first copy the member into a local variable and call the method on that variable.

The following sample generates CS1690:

```
C#  
  
// CS1690.cs  
using System;  
  
class WarningCS1690 : MarshalByRefObject  
{  
    int i = 5;  
  
    public static void Main()  
    {  
        AppDomain domain = AppDomain.CreateDomain("MyDomain");  
        Type t = typeof(WarningCS1690);  
        WarningCS1690 e =  
(WarningCS1690)domain.CreateInstanceAndUnwrap(t.Assembly.FullName, t.FullName  
);  
  
        e.i.ToString(); // CS1690  
  
        // OK  
        int i = e.i;  
        i.ToString();  
        e.i = i;  
    }  
}
```

# Compiler Warning (level 1) CS1691

Article • 09/15/2021 • 2 minutes to read

'number' is not a valid warning number

A number that was passed to the `#pragma warning` preprocessor directive was not a valid warning number. Verify that the number represents a warning, not an error or another sequence of characters.

## Example

The following example generates CS1691.

C#

```
// CS1691.cs
public class C
{
    int i = 1;
    public static void Main()
    {
        C myC = new C();
#pragma warning disable 151 // CS1691
// Try the following line instead:
// #pragma warning disable 1645
        myC.i++;
#pragma warning restore 151 // CS1691
// Try the following line instead:
// #pragma warning restore 1645
    }
}
```

# Compiler Warning (level 1) CS1692

Article • 09/15/2021 • 2 minutes to read

## Invalid number

A number of preprocessor directives, such as `#pragma` and `#line`, use numbers as parameters. One of these numbers is invalid because it is too big, in the wrong format, contains illegal characters, and so on. To correct this error, correct the number.

## Example

The following example generates CS1692.

```
C#  
  
// CS1692.cs  
  
#pragma warning disable a // CS1692  
// Try this instead:  
// #pragma warning disable 1691  
  
class A  
{  
    static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS1694

Article • 09/15/2021 • 2 minutes to read

Invalid filename specified for preprocessor directive. Filename is too long or not a valid filename.

This warning occurs when using the `#pragma checksum` preprocessor directive. The file name specified is longer than 256 characters. To resolve this warning, use a shorter file name.

## Example

The following sample generates CS1694.

C#

```
// cs1694.cs
#pragma checksum
"MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile12345
67890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1
234567890MyFile1234567890MyFile1234567890MyFile1234567890MyFile1234567890MyF
ile1234567890MyFile1234567890.txt" {00 01 02 03 04 05 06 07 08 09 0A 0B 0C
0D 0E 0F} // CS1694
class MyClass {}
```

# Compiler Warning (level 1) CS1695

Article • 09/15/2021 • 2 minutes to read

Invalid #pragma checksum syntax; should be #pragma checksum "filename" "  
{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}" "XXXX..."

You should rarely encounter this error since the checksum is generally inserted at run time if you are generating code by means of the Code Dom API.

However, if you were to type in this `#pragma` statement and mistype either the GUID or checksum, you would get this error. The syntax checking by the compiler does not validate that you typed in a correct GUID, but it does check for the right number of digits and delimiters, and that the digits are hexadecimal. Likewise, it verifies that the checksum contains an even number of digits, and that the digits are hexadecimal.

## Example

The following example generates CS1695.

```
C#  
  
// CS1695.cs  
  
#pragma checksum "12345" // CS1695  
  
public class Test  
{  
    static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS1696

Article • 09/15/2021 • 2 minutes to read

Single-line comment or end-of-line expected

The compiler requires a preprocessor directive to be followed by an end-of-line terminator or by a single-line comment. The compiler has finished processing a valid preprocessor directive, and has encountered something that violates this syntax constraint.

## Example

The following sample generates CS1696.

```
C#  
  
// CS1696.cs  
class Test  
{  
    public static void Main()  
    {  
        #pragma warning disable 1030;219    // CS1696  
        #pragma warning disable 1030    // OK  
    }  
}
```

# Compiler Warning (level 1) CS1697

Article • 09/15/2021 • 2 minutes to read

Different checksum values given for 'file name'

You have specified more than one checksum for a given file. The debugger uses the checksum value to determine which file to debug when there is more than one file in a project with the same name. Most users will not encounter this error, but if you are writing an application that generates code, you may run into it. To resolve this error, ensure that you generate the checksum only once for any given code file.

# Compiler Warning (level 1) CS1699

Article • 09/15/2021 • 2 minutes to read

Use command line option "compiler\_option" or appropriate project settings instead of "attribute\_name"

In order to sign an assembly, it is necessary to specify a key file. Prior to C# 2.0, you specified the key file using CLR attributes in source code. These attributes are now deprecated.

Beginning with C# 2.0, you should use the [Signing Page](#) of the [Project Designer](#) or the Assembly Linker to specify the key file.

The [Signing Page of the Project Designer](#) is the preferred method; for more information, see [Signing Page, Project Designer](#) and [Managing Assembly and Manifest Signing](#).

The [How to: Sign an Assembly with a Strong Name](#) uses the following compiler options:

- [KeyFile \(C# Compiler Options\)](#) instead of the [AssemblyKeyFileAttribute](#) attribute.
- [KeyContainer \(C# Compiler Options\)](#) instead of [AssemblyKeyNameAttribute](#).
- [DelaySign \(C# Compiler Options\)](#) instead of [AssemblyDelaySignAttribute](#).

These attributes have been deprecated for the following reasons:

- There were security issues due to the attributes being embedded in the binary files produced by the compiler. Everyone who had your binary also had the keys stored in it.
- There were usability issues due to the fact that the path specified in the attributes was relative to the current working directory, which could change in the integrated development environment (IDE), or to the output directory. Thus, most times the key file is likely to be ..\\..\\mykey.snk. Attributes also make it more difficult for the project system to properly sign satellite assemblies. When you use the compiler options instead of these attributes, you can use a fully qualified path and file name for the key without anything being embedded in the output file; the project system and source code control system can properly manipulate that full path when projects are moved around; the project system can maintain a project-relative path to the key file, and still pass a full path to the compiler; other build programs can more easily sign outputs by passing the proper path directly to the compiler instead of generating a source file with the correct attributes.

- Using attributes with friend assemblies can hamper compiler efficiency. When you use attributes, the compiler does not know what the key is when it has to decide whether or not to grant friendship and so it has to guess. At the end of compilation, the compiler is able to verify the guess once it finally knows the key. When the key file is specified with a compiler option, the compiler can immediately decide whether to grant friendship.

## Example

The following sample generates CS1699. To resolve the error, remove the attribute and compile with `/delaysign`.

C#

```
// CS1699.cs
// compile with: /target:library
[assembly:System.Reflection.AssemblyDelaySign(true)] // CS1699
```

## See also

- [Signing Page, Project Designer](#)
- [Managing Assembly and Manifest Signing](#)
- [How to: Sign an Assembly with a Strong Name](#)

# Compiler Warning (level 1) CS1707

Article • 09/15/2021 • 2 minutes to read

Delegate 'DelegateName' bound to 'MethodName1' instead of 'MethodName2' because of new language rules

C# 2.0 implements new rules for binding a delegate to a method. Additional information is considered that was not looked at in the past. This warning indicates that the delegate is now bound to a different overload of the method than it was previously bound to. You may wish to verify that the delegate really should be bound to 'MethodName1' instead of 'MethodName2'.

For a description of how the compiler determines which method to bind a delegate to, see [Using Variance in Delegates](#).

# Compiler Warning (level 1) CS1709

Article • 09/15/2021 • 2 minutes to read

Filename specified for preprocessor directive is empty

You have specified a preprocessor directive that includes a file name, but that file is empty. To resolve this warning, put the needed content into the file.

## Example

The following example generates CS1709.

C#

```
// CS1709.cs
class Test
{
    static void Main()
    {
        #pragma checksum "" "{406EA660-64CF-4C82-B6F0-42D48172A799}" ""
CS1709
    }
}
```

# Compiler Warning (level 1) CS1720

Article • 09/15/2021 • 2 minutes to read

Expression will always cause a System.NullReferenceException because the default value of 'generic type' is null

If you write an expression involving the default of a generic type variable that is a reference type (for example, a class), this error will occur. Consider the following expression:

```
C#
```

```
default(T).ToString()
```

Since `T` is a reference type, its default value is null, and so attempting to apply the `ToString` method to it will throw a `NullReferenceException`.

## Example

The class reference constraint on type `T` ensures that `T` is a reference type.

The following sample generates CS1720.

```
C#
```

```
// CS1720.cs
using System;
public class Tester
{
    public static void GenericClass<T>(T t1) where T : class
    {
        Console.WriteLine(default(T).ToString()); // CS1720
    }
    public static void Main() {}
}
```

# Compiler Warning (level 1) CS1723

Article • 03/22/2023 • 2 minutes to read

XML comment has cref attribute 'attribute' that refers to a type parameter

This error is generated for an XML comment in case of using a `<see/>` tag with cross reference (cref) to a type parameter instead of the existing type (whether user-defined or built-in) in the code. It's impossible to link to 'attribute' of generic types because at the moment of creating the documentation the future type given as 'attribute' is not yet known.

To solve this issue `<typeparamref/>` tag should be used.

## Example

The following example contains a comment generating CS1723 as well as a reference that can be linked correctly.

```
C#  
  
public class Point  
{  
}  
  
// compile with: /t:library /doc:filename.XML  
///<summary>A generic list class.</summary>  
///uses <see cref="T" />      // CS1723  
///and <see cref="Point" />    // No warning  
public class List<T, Point>  
{  
}
```

This example shows how to correctly link both generic type `T` as well as already known user-defined `Point`

```
C#  
  
public class Point  
{  
}  
  
// compile with: /t:library /doc:filename.XML  
///<summary>A generic list class.</summary>  
///uses <typeparamref name="T" /> // No warning  
///and <see cref="Point" />      // No warning  
public class List<T, Point>
```

{  
}

# Compiler Warning (level 1) CS1762

Article • 09/15/2021 • 2 minutes to read

A reference was created to embedded interop assembly '<assembly1>' because of an indirect reference to that assembly from assembly '<assembly2>'. Consider changing the 'Embed Interop Types' property on either assembly.

You have added a reference to an assembly (assembly1) that has the `Embed Interop Types` property set to `True`. This instructs the compiler to embed interop type information from that assembly. However, the compiler cannot embed interop type information from that assembly because another assembly that you have referenced (assembly2) also references that assembly (assembly1) and has the `Embed Interop Types` property set to `False`.

## ⓘ Note

Setting the `Embed Interop Types` property on an assembly reference to `True` is equivalent to referencing the assembly by using the `/link` option for the command-line compiler.

## To address this warning

- To embed interop type information for both assemblies, set the `Embed Interop Types` property on all references to assembly1 to `True`. For more information about how to set that property, see [Walkthrough: Embedding Types from Managed Assemblies](#).
- To remove the warning, you can set the `Embed Interop Types` property of assembly1 to `False`. In this case, a primary interop assembly (PIA) provides interop type information.

## See also

- [EmbedInteropAssembly \(C# Compiler Options\)](#)
- [Interoperating with Unmanaged Code](#)



# Compiler Warning (level 1) CS1911

Article • 09/15/2021 • 2 minutes to read

Access to member 'name' through a 'base' keyword from an anonymous method, lambda expression, query expression, or iterator results in unverifiable code. Consider moving the access into a helper method on the containing type.

Calling virtual functions with the `base` keyword inside the method body of an iterator or anonymous methods will result in unverifiable code. Unverifiable code will fail to run in a partial trust environment.

One resolution for CS1911 is to move the virtual function call to a helper function.

## Example

The following sample generates CS1911.

```
C#  
  
// CS1911.cs  
// compile with: /W:1  
using System;  
  
delegate void D();  
delegate D RetD();  
  
class B {  
    protected virtual void M() {  
        Console.WriteLine("B.M");  
    }  
}  
  
class Der : B {  
    protected override void M() {  
        Console.WriteLine("D.M");  
    }  
}  
  
void Test() { base.M(); }  
D Test2() { return new D(base.M); }  
  
public D CallBaseM() {  
    return delegate () { base.M(); }; // CS1911  
  
    // try the following line instead  
    // return delegate () { Test(); };  
}  
  
public RetD DelToBaseM() {
```

```
        return delegate () { return new D(base.M); }; // CS1911

        // try the following line instead
        // return delegate () { return Test2(); };
    }

}

class Program {
    public static void Main() {
        Der der = new Der();
        D d = der.CallBaseM();
        d();
        RetD rd = der.DelToBaseM();
        rd();
    }
}
```

# Compiler Warning (level 1) CS1956

Article • 11/05/2021 • 2 minutes to read

Member 'name' implements interface member 'name' in type 'type'. There are multiple matches for the interface member at run-time. It is implementation dependent which method will be called.

This warning can be generated when two interface methods are differentiated only by whether a particular parameter is marked with `ref` or with `out`. It is best to change your code to avoid this warning because it is not obvious or guaranteed which method is called at run time.

Although C# distinguishes between `out` and `ref`, the CLR sees them as the same. When deciding which method implements the interface, the CLR just picks one.

## To avoid this warning

1. Give the compiler some way to differentiate the methods. For example, you can give them different names or provide an additional parameter on one of them.

## Example

The following code generates CS1956 because the two `Test` methods in `Base` differ only by the `ref/out` modifier on the first parameter.

C#

```
// cs1956.cs
class Base<T, S>
{
    // This is the method that should be called.
    public virtual int Test(out T x) // CS1956
    {
        x = default(T);
        return 0;
    }

    // This is the "last" method and is the one that ends up being called
    public virtual int Test(ref S x)
    {
        return 1;
    }
}

interface IFace
```

```
{  
    int Test(out int x);  
}  
  
class Derived : Base<int, int>, IFace  
{  
    static int Main()  
    {  
        IFace x = new Derived();  
        int y;  
        return x.Test(out y);  
    }  
}
```

# Compiler Warning (Level 1) CS1957

Article • 09/15/2021 • 2 minutes to read

Member 'name' overrides 'method'. There are multiple override candidates at run-time. It is implementation dependent which method will be called.

Method parameters that vary only by whether they are `ref` or `out` cannot be differentiated at run-time.

## To avoid this warning

1. Give one of the methods a different name or different number of parameters.

## Example

The following code generates CS1957:

```
C#  
  
// cs1957.cs  
class Base<T, S>  
{  
    public virtual string Test(out T x) // CS1957  
    {  
        x = default(T);  
        return "Base.Test";  
    }  
    public virtual void Test(ref S x) {}  
}  
  
class Derived : Base<int, int>  
{  
    public override string Test(out int x)  
    {  
        x = 0;  
        return "Derived.Test";  
    }  
  
    static int Main()  
    {  
        int x;  
        if (new Derived().Test(out x) == "Derived.Test")  
            return 0;  
        return 1;  
    }  
}
```

# Compiler Warning (level 1) CS2002

Article • 09/15/2021 • 2 minutes to read

Source file 'file' specified multiple times

A source file name was passed to the compiler more than once. You can only specify a file once to the compiler to build an output file.

This warning cannot be suppressed by the [-nowarn](#) option.

The following sample generates CS2002:

```
C#  
  
// CS2002.cs  
// compile with: CS2002.cs  
public class A  
{  
    public static void Main(){}
}
```

To generate the error, compile the example with the command line:

```
Console  
  
csc CS2002.cs CS2002.cs
```

# Compiler Warning (level 1) CS2014

Article • 09/15/2021 • 2 minutes to read

Compiler option 'old option' is obsolete, please use 'new option' instead

The form of the compiler option is deprecated. See [C# Compiler Options](#) for more information.

# Compiler Warning (level 1) CS2023

Article • 05/13/2022 • 2 minutes to read

Ignoring /noconfig option because it was specified in a response file

The `-noconfig` compiler option was specified in a response file, which is not allowed.

This warning cannot be suppressed by the [NoWarn](#) option.

# Compiler Warning (level 1) CS2029

Article • 05/13/2022 • 2 minutes to read

Invalid value for '/define'; 'identifier' is not a valid identifier

This warning occurs if the value that is used in the [DefineConstants](#) option has some invalid characters.

This warning cannot be suppressed by the [NoWarn](#) option.

# Compiler Warning (level 1) CS3000

Article • 09/15/2021 • 2 minutes to read

Methods with variable arguments are not CLS-compliant

The arguments used in the method expose features that are not in the Common Language Specifications (CLS). For more information on CLS Compliance, see [Language independence and language-independent components](#).

The following example generates the warning CS3000.

C#

```
// CS3000.cs
// compile with: /target:library
// CS3000 expected
[assembly:System.CLSCompliant(true)]

public class Test
{
    public void AddABunchOfInts( __arglist ) {}    // CS3000
}
```

# Compiler Warning (level 1) CS3001

Article • 09/15/2021 • 2 minutes to read

Argument type 'type' is not CLS-compliant

A `public`, `protected`, or `protected internal` method must accept a parameter whose type is compliant with the Common Language Specification (CLS). For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3001:

C#

```
// CS3001.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public void bad(ushort i)    // CS3001
    {
    }

    private void OK(ushort i)    // OK, private method
    {
    }

    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS3002

Article • 09/15/2021 • 2 minutes to read

Return type of 'method' is not CLS-compliant

A `public`, `protected`, or `protected internal` method must return a value whose type is compliant with the Common Language Specification (CLS). For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3002:

C#

```
// CS3002.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public ushort bad()    // CS3002, public method
    {
        ushort a;
        a = ushort.MaxValue;
        return a;
    }

    private ushort OK()    // OK, private method
    {
        ushort a;
        a = ushort.MaxValue;
        return a;
    }

    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS3003

Article • 09/15/2021 • 2 minutes to read

Type of 'variable' is not CLS-compliant

A `public`, `protected`, or `protected internal` variable must be of a type that is compliant with the Common Language Specification (CLS). For more information on CLS Compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3003:

```
C#  
  
// CS3003.cs  
  
[assembly:System.CLSCompliant(true)]  
public class a  
{  
    public ushort a1;    // CS3003, public variable  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS3004

Article • 09/15/2021 • 2 minutes to read

Mixed and decomposed Unicode characters are not CLS-compliant

Only composed UNICODE characters are allowed in [public](#), [protected](#), or [protected internal](#) identifiers in order to be compliant with the Common Language Specification (CLS). For more information on CLS Compliance, see [Language independence and language-independent components](#).

# Compiler Warning (level 1) CS3005

Article • 09/15/2021 • 2 minutes to read

Identifier 'identifier' differing only in case is not CLS-compliant

A `public`, `protected`, or `protected internal` identifier, which differs from another `public`, `protected`, or `protected internal` identifier only in the case of one or more letters, is not compliant with the Common Language Specification (CLS). For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3003:

```
C#  
  
// CS3005.cs  
  
using System;  
  
[assembly:CLSClaimed(true)]  
public class a  
{  
    public static int a1 = 0;  
    public static int A1 = 1; // CS3005  
  
    public static void Main()  
    {  
        Console.WriteLine(a1);  
        Console.WriteLine(A1);  
    }  
}
```

# Compiler Warning (level 1) CS3006

Article • 09/08/2022 • 2 minutes to read

Overloaded method 'method' differing only in ref or out, or in array rank, is not CLS-compliant

A method cannot be overloaded based on the [ref](#) or [out](#) parameter and still comply with the Common Language Specification (CLS). For more information on CLS Compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3006. To resolve this warning, comment out the assembly-level attribute or remove one of the method definitions.

C#

```
// CS3006.cs

using System;

[assembly: CLSCompliant(true)]
public class MyClass
{
    public void f(int i)
    {
    }

    public void f(ref int i) // CS3006
    {
    }

    public static void Main()
    {
    }
}
```

# Compiler Warning (level 1) CS3007

Article • 09/15/2021 • 2 minutes to read

Overloaded method 'method' differing only by unnamed array types is not CLS-compliant

This error occurs if you have an overloaded method that takes a jagged array and the only difference between the method signatures is the element type of the array. To avoid this error, consider using a rectangular array rather than a jagged array; use an additional parameter to disambiguate the function call; rename one or more of the overloaded methods; or, if CLS Compliance is not needed, remove the [CLSCompliantAttribute](#) attribute. For more information on CLS Compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3007:

C#

```
// CS3007.cs
[assembly: System.CLSCompliant(true)]
public struct S
{
    public void F(int[][] array) { }
    public void F(byte[][] array) { } // CS3007
    // Try this instead:
    // public void F1(int[][] array) {}
    // public void F2(byte[][] array) {}
    // or
    // public void F(int[,] array) {}
    // public void F(byte[,] array) {}
}
```

# Compiler Warning (level 1) CS3008

Article • 09/15/2021 • 2 minutes to read

Identifier 'identifier' differing only in case is not CLS-compliant

A [public](#), [protected](#), or [protected internal](#) identifier breaks compliance with the Common Language Specification (CLS) if it begins with an underscore character (\_). For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3008:

```
C#  
  
// CS3008.cs  
  
using System;  
  
[assembly:CLSClaimed(true)]  
public class a  
{  
    public static int _a = 0; // CS3008  
    // OK, private  
    // private static int _a1 = 0;  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS3009

Article • 09/15/2021 • 2 minutes to read

'type': base type 'type' is not CLS-compliant

A base type was marked as not having to be compliant with the Common Language Specification (CLS) in an assembly that was marked as being CLS compliant. Either remove the attribute that specifies the assembly is CLS compliant or remove the attribute that indicates the type is not CLS compliant. For more information on CLS Compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3009:

```
C#  
  
// CS3009.cs  
  
using System;  
  
[assembly:CLSClaimed(true)]  
[CLSClaimed(false)]  
public class B  
{  
}  
  
public class C : B // CS3009  
{  
    public static void Main () {}  
}
```

# Compiler Warning (level 1) CS3010

Article • 09/15/2021 • 2 minutes to read

'member': CLS-compliant interfaces must have only CLS-compliant members

In an assembly marked with `[assembly:CLSCompliant(true)]`, an interface contains a member marked with `[CLSCompliant(false)]`. Remove one of the Common Language Specification (CLS) compliance attributes. For more information about CLS Compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3010:

```
C#  
  
// CS3010.cs  
  
using System;  
  
[assembly:CLSCompliant(true)]  
public interface I  
{  
    [CLSCompliant(false)]  
    int M(); // CS3010  
}  
  
public class C : I  
{  
    public int M()  
    {  
        return 1;  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS3011

Article • 09/15/2021 • 2 minutes to read

'member': only CLS-compliant members can be abstract

A class member cannot be both [abstract](#) and non-compliant with the Common Language Specification (CLS). The CLS specifies that all class members shall be implemented. For more information about CLS compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3011:

```
C#  
  
// CS3011.cs  
  
using System;  
  
[assembly:CLSCompliant(true)]  
public abstract class I  
{  
    [CLSCompliant(false)]  
    public abstract int M(); // CS3011  
  
    // OK  
    [CLSCompliant(false)]  
    public void M2()  
    {  
    }  
}  
  
public class C : I  
{  
    public override int M()  
    {  
        return 1;  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS3012

Article • 09/15/2021 • 2 minutes to read

You cannot specify the `CLSCCompliant` attribute on a module that differs from the `CLSCCompliant` attribute on the assembly

In order for a module to be compliant with the Common Language Specification (CLS) through `[module:System.CLSCCompliant(true)]`, it must be built with the `module` element of the `TargetType` compiler option. For more information on the CLS, see [Language independence and language-independent components](#).

## Example

The following example, when built without `/target:module`, generates CS3012:

```
C#  
  
// CS3012.cs  
// compile with: /W:1  
  
[module:System.CLSCCompliant(true)] // CS3012  
public class C  
{  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS3013

Article • 09/15/2021 • 2 minutes to read

Added modules must be marked with the `CLSCompliant` attribute to match the assembly

A module that was compiled with the `module` element of the `TargetType` compiler option was added to a compilation with `AddModule`. However, the module's compliance with the Common Language Specification (CLS) does not agree with the CLS state of the current compilation.

CLS compliance is indicated with the `module` attribute. For example,

`[module:CLSCompliant(true)]` indicates that the module is CLS compliant, and `[module:CLSCompliant(false)]` indicates that the module is not CLS compliant. The default is `[module:CLSCompliant(false)]`. For more information on the CLS, see [Language independence and language-independent components](#).

# Compiler Warning (level 1) CS3014

Article • 09/15/2021 • 2 minutes to read

'member' does not need a CLSCompliant attribute because the assembly does not have a CLSCompliant attribute

In a source code file where compliance with the Common Language Specification (CLS) was not specified, a construct in the file was marked as being CLS compliant. This is not allowed. To resolve this warning, add an assembly level CLS compliant attribute to the file (in the following example, uncomment the line that contains the assembly level attribute). For more information about CLS compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3014:

```
C#  
  
// CS3014.cs  
  
using System;  
  
// [assembly:CLSSCompliant(true)]  
public class I  
{  
    [CLSSCompliant(true)] // CS3014  
    public void M()  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS3015

Article • 09/15/2021 • 2 minutes to read

'method signature' has no accessible constructors which use only CLS-compliant types

To be compliant with the Common Language Specification (CLS), the argument list of an attribute class cannot contain an array. For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following sample generates C3015.

C#

```
// CS3015.cs
// compile with: /target:library
using System;

[assembly:CLSCCompliant(true)]
public class MyAttribute : Attribute
{
    public MyAttribute(int[] ai) {} // CS3015
    // try the following line instead
    // public MyAttribute(int ai) {}
}
```

# Compiler Warning (level 1) CS3016

Article • 09/15/2021 • 2 minutes to read

Arrays as attribute arguments is not CLS-compliant

It is not compliant with the Common Language Specification (CLS) to pass an array to an attribute. For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3016:

```
C#  
  
// CS3016.cs  
  
using System;  
  
[assembly : CLSCompliant(true)]  
[C(new int[] {1, 2})] // CS3016  
// try the following line instead  
// [C()]  
class C : Attribute  
{  
    public C()  
    {  
    }  
  
    public C(int[] a)  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS3017

Article • 09/15/2021 • 2 minutes to read

You cannot specify the `CLSCCompliant` attribute on a module that differs from the `CLSCCompliant` attribute on the assembly

This warning occurs if you have a assembly `CLSCCompliant` attribute that conflicts with a module `CLSCCompliant` attribute. An assembly that is CLS compliant cannot contain modules that are not CLS compliant. To resolve this warning, make sure the assembly and module `CLSCCompliant` attributes are either both true or both false, or remove one of the attributes. For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following example generates CS3017:

```
C#  
  
// CS3017.cs  
// compile with: /target:module  
  
using System;  
  
[module: CLSCCompliant(true)]  
[assembly: CLSCCompliant(false)] // CS3017  
// Try this line instead:  
// [assembly: CLSCCompliant(true)]  
class C  
{  
    static void Main() {}  
}
```

# Compiler Warning (level 1) CS3018

Article • 09/15/2021 • 2 minutes to read

'type' cannot be marked as CLS-Compliant because it is a member of non CLS-compliant type 'type'

This warning occurs if a nested class with the `CLSCompliant` attribute set to `true` is declared as a member of a class declared with the `CLSCompliant` attribute set to `false`. This is not allowed, since a nested class cannot be CLS-compliant if it is a member of an outer class that is not CLS-compliant. To resolve this warning, remove the `CLSCompliant` attribute from the nested class, or change it from `true` to `false`. For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following sample generates CS3018.

```
C#  
  
// CS3018.cs  
// compile with: /target:library  
using System;  
  
[assembly: CLSCompliant(true)]  
[CLSCompliant(false)]  
public class Outer  
{  
    [CLSCompliant(true)] // CS3018  
    public class Nested {}  
  
    // OK  
    public class Nested2 {}  
  
    [CLSCompliant(false)]  
    public class Nested3 {}  
}
```

# Compiler Warning (level 1) CS3022

Article • 09/15/2021 • 2 minutes to read

CLSCCompliant attribute has no meaning when applied to parameters. Try putting it on the method instead.

Method parameters are not checked for CLS Compliance, since the CLS Compliance rules apply to methods and type declarations.

## Example

The following sample generates CS3022:

```
C#  
  
// CS3022.cs  
// compile with: /W:1  
  
using System;  
  
[assembly: CLSCCompliant(true)]  
[CLSCCompliant(true)]  
public class C  
{  
    public void F([CLSCCompliant(true)] int i)  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 1) CS3023

Article • 09/15/2021 • 2 minutes to read

CLSCompliant attribute has no meaning when applied to return types. Try putting it on the method instead.

Function return types are not checked for CLS Compliance, since the CLS Compliance rules apply to methods and type declarations.

## Example

The following example generates warning CS3023:

```
C#  
  
// C3023.cs  
  
[assembly:System.CLSCompliant(true)]  
public class Test  
{  
    [return:System.CLSCompliant(true)] // CS3023  
    // Try this instead:  
    // [method:System.CLSCompliant(true)]  
    public static int Main()  
    {  
        return 0;  
    }  
}
```

# Compiler Warning (level 1) CS3024

Article • 09/15/2021 • 2 minutes to read

Constraint type 'type' is not CLS-compliant.

The compiler issues this warning because the use of a non-CLS-compliant type as a generic type constraint could make it impossible for code written in some languages to consume your generic class.

## To eliminate this warning

1. Use a CLS-compliant type for the type constraint.

## Example

The following example generates CS3024 in several locations:

```
C#  
  
// cs3024.cs  
// Compile with: /target:library  
[assembly: System.CLSCompliant(true)]  
  
[type: System.CLSCompliant(false)]  
public class TestClass // CS3024  
{  
    public ushort us;  
}  
[type: System.CLSCompliant(false)]  
public interface ITest // CS3024  
{  
}  
public interface I<T> where T : TestClass  
{  
}  
public class TestClass_2<T> where T : ITest  
{  
}  
public class TestClass_3<T> : I<T> where T : TestClass  
{  
}  
public class TestClass_4<T> : TestClass_2<T> where T : ITest  
{  
}  
public class Test  
{  
    public static int Main()  
    {  
        return 0;  
    }  
}
```

## See also

- [Constraints on Type Parameters](#)

# Compiler Warning (level 1) CS3026

Article • 09/15/2021 • 2 minutes to read

CLS-compliant field 'field' cannot be volatile

A volatile variable should not be CLS compliant.

## Example

The following example generates CS3026.

C#

```
// CS3026.cs
[assembly:System.CLSCompliant(true)]
public class Test
{
    public volatile int v0 =0;    // CS3026
    // To resolve remove the CLS-Compliant attribute.
    public static void Main() { }
}
```

# Compiler Warning (level 1) CS3027

Article • 09/15/2021 • 2 minutes to read

'type\_1' is not CLS-compliant because base interface 'type\_2' is not CLS-compliant

A non-CLS compliant type cannot be a base type for a type that is CLS compliant.

## Example 1

The following sample contains an interface with a method that uses a non-CLS compliant type in its signature, making the type non-CLS compliant.

C#

```
// CS3027.cs
// compile with: /target:library
public interface IBase
{
    void IMethod(uint i);
}
```

## Example 2

The following sample generates CS3027.

C#

```
// CS3027_b.cs
// compile with: /reference:CS3027.dll /target:library /W:1
[assembly:System.CLSCompliant(true)]
public interface IDerived : IBase {}
```

# Compiler Warning (level 1) CS4014

Article • 02/13/2023 • 4 minutes to read

Because this call is not awaited, execution of the current method continues before the call is completed. Consider applying the `await` operator to the result of the call.

The current method calls an async method that returns a [Task](#) or a [Task<TResult>](#) and doesn't apply the `await` operator to the result. The call to the async method starts an asynchronous task. However, because no `await` operator is applied, the program continues without waiting for the task to complete. In most cases, that behavior isn't what you expect. Usually other aspects of the calling method depend on the results of the call or, minimally, the called method is expected to complete before you return from the method that contains the call.

An equally important issue is what happens to exceptions that are raised in the called async method. An exception that's raised in a method that returns a [Task](#) or [Task<TResult>](#) is stored in the returned task. If you don't await the task or explicitly check for exceptions, the exception is lost. If you await the task, its exception is rethrown.

As a best practice, you should always await the call.

You should consider suppressing the warning only if you're sure that you don't want to wait for the asynchronous call to complete and that the called method won't raise any exceptions. In that case, you can suppress the warning by assigning the task result of the call to a variable.

The following example shows how to cause the warning, how to suppress it, and how to await the call.

C#

```
static async Task CallingMethodAsync(int millisecondsDelay)
{
    Console.WriteLine("  Entering calling method.");

    // Call #1.
    // Call an async method. Because you don't await it, its completion
    // isn't coordinated with the current method, CallingMethodAsync.
    // The following line causes warning CS4014.
    CalledMethodAsync(millisecondsDelay);

    // Call #2.
    // To suppress the warning without awaiting, you can assign the
    // returned task to a variable. The assignment doesn't change how
```

```

// the program runs. However, recommended practice is always to
// await a call to an async method.

// Replace Call #1 with the following line.
// Task delayTask = CalledMethodAsync(millisecondsDelay);

// Call #3
// To contrast with an awaited call, replace the unawaited call
// (Call #1 or Call #2) with the following awaited call. Best
// practice is to await the call.

// await CalledMethodAsync(millisecondsDelay);

Console.WriteLine("  Returning from calling method.");
}

static async Task CalledMethodAsync(int millisecondsDelay)
{
    Console.WriteLine("    Entering called method, starting and awaiting
Task.Delay.");

    await Task.Delay(millisecondsDelay);

    Console.WriteLine("    Task.Delay is finished--returning from called
method.");
}

```

In the example, if you choose Call #1 or Call #2, the unawaited async method `CalledMethodAsync` finishes after both its caller `CallingMethodAsync` and the caller's caller is complete. The last line in the following output shows you when the called method finishes. Entry to and exit from the event handler that calls `CallingMethodAsync` in the full example are marked in the output.

Console

```

Entering the Click event handler.
    Entering calling method.
        Entering called method, starting and awaiting Task.Delay.
        Returning from calling method.
    Exiting the Click event handler.
        Task.Delay is finished--returning from called method.

```

You can also suppress compiler warnings by using `#pragma warning` directives.

## Example

The following console application contains the methods from the previous example. The following steps set up the application.

1. Create a console application, and name it `AsyncWarning`.

2. In the Visual Studio Code Editor, choose the `Program.cs` file.

3. Replace the code in `Program.cs` with the following code.

C#

```
using System;
using System.Threading.Tasks;

namespace AsyncWarning
{
    class Program
    {
        static async Task Main()
        {
            Console.WriteLine("Entering Main() application entry point.");

            int millisecondsDelay = 2000;
            await CallingMethodAsync(millisecondsDelay);

            Console.WriteLine("Exiting Main() application entry point.");

            await Task.Delay(millisecondsDelay + 500);
        }

        static async Task CallingMethodAsync(int millisecondsDelay)
        {
            Console.WriteLine(" Entering calling method.");

            // Call #1.
            // Call an async method. Because you don't await it, its completion
            // isn't coordinated with the current method,
            CallingMethodAsync.
            // The following line causes warning CS4014.
            // CalledMethodAsync(millisecondsDelay);

            // Call #2.
            // To suppress the warning without awaiting, you can assign the
            // returned task to a variable. The assignment doesn't change how
            // the program runs. However, recommended practice is always to
            // await a call to an async method.

            // Replace Call #1 with the following line.
            //Task delayTask = CalledMethodAsync(millisecondsDelay);

            // Call #3
        }
    }
}
```

```

        // To contrast with an awaited call, replace the unawaited
call
        // (Call #1 or Call #2) with the following awaited call.
Best
        // practice is to await the call.

        // await CalledMethodAsync(millisecondsDelay);

        Console.WriteLine("  Returning from calling method.");
    }

    static async Task CalledMethodAsync(int millisecondsDelay)
{
    Console.WriteLine("      Entering called method, starting and
awaiting Task.Delay.");

    await Task.Delay(millisecondsDelay);

    Console.WriteLine("      Task.Delay is finished--returning
from called method.");
}
}

// Output with Call #1 or Call #2. (Wait for the last line to
appear.)

// Entering Main() application entry point.
//   Entering calling method.
//     Entering called method, starting and awaiting Task.Delay.
//     Returning from calling method.
// Exiting Main() application entry point.
//     Task.Delay is finished--returning from called method.

// Output with Call #3, which awaits the call to CalledMethodAsync.

// Entering Main() application entry point.
//   Entering calling method.
//     Entering called method, starting and awaiting Task.Delay.
//     Task.Delay is finished--returning from called method.
//     Returning from calling method.
// Exiting Main() application entry point.
}

```

4. Select the  key to run the program.

The expected output appears at the end of the code.

## See also

- [await](#)
- [Asynchronous programming with `async` and `await`](#)



# Compiler Warning (level 1) CS5000

Article • 09/15/2021 • 2 minutes to read

Unknown compiler option '/option'

An invalid [compiler option](#) was specified.

 **Note**

This compiler error is no longer used in Roslyn.

# Compiler Warning (level 2) CS0108

Article • 09/15/2021 • 2 minutes to read

'member1' hides inherited member 'member2'. Use the new keyword if hiding was intended.

A variable was declared with the same name as a variable in a base class. However, the [new](#) keyword was not used. This warning informs you that you should use [new](#); the variable is declared as if [new](#) had been used in the declaration.

The following sample generates CS0108:

```
C#  
  
// CS0108.cs  
// compile with: /W:2  
using System;  
  
namespace x  
{  
    public class clk  
    {  
        public int i = 1;  
    }  
  
    public class cly : clk  
    {  
        public static int i = 2;    // CS0108, use the new keyword  
        // Use the following line instead:  
        // public static new int i = 2;  
  
        public static void Main()  
        {  
            Console.WriteLine(i);  
        }  
    }  
}
```

## See also

- [new Modifier](#)



# Compiler Warning (level 2) CS0114

Article • 09/15/2021 • 2 minutes to read

'function1' hides inherited member 'function2'. To make the current method override that implementation, add the override keyword. Otherwise add the new keyword.

A declaration in a class conflicts with a declaration in a base class such that the base class member will be hidden.

For more information, see [base](#).

The following sample generates CS0114:

```
C#  
  
// CS0114.cs  
// compile with: /W:2 /warnaserror  
abstract public class clx  
{  
    public abstract void f();  
}  
  
public class cly : clx  
{  
    public void f() // CS0114, hides base class member  
    // try the following line instead  
    // override public void f()  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 2) CS0162

Article • 09/15/2021 • 2 minutes to read

Unreachable code detected

The compiler detected code that will never be executed.

## Example

The following example generates CS0162:

```
C#  
  
// CS0162.cs  
// compile with: /W:2  
public class Program  
{  
    public static void Main()  
    {  
        goto lab1;  
        {  
            // The following statements cannot be reached:  
            int i = 9;    // CS0162  
            i++;  
        }  
        lab1:  
        {  
        }  
    }  
}
```

Another common example where this error is generated is as follows:

```
C#  
  
public static class Class1  
{  
    public static string Method1()  
    {  
        string x = "a";  
        switch (x)  
        {  
            case "a":  
                return "a";  
                break;           // CS0162  
            }  
        return "";  
    }  
}
```

```
    }  
}
```

The `break` statement cannot be reached because it occurs after the `return` statement.  
The `return` statement ends the enclosing `case` branch.

# Compiler Warning (level 2) CS0164

Article • 09/15/2021 • 2 minutes to read

This label has not been referenced

A label was declared but never used.

The following sample generates CS0164:

C#

```
// CS0164.cs
// compile with: /W:2
public class Program
{
    public static void Main()
    {
        int i = 0;
l1: // CS0164
        i++;
        // Uncomment the following lines to resolve this warning.
        // if (i < 10)
        //     goto l1;
    }
}
```

# Compiler Warning (level 2) CS0251

Article • 09/15/2021 • 2 minutes to read

Indexing an array with a negative index (array indices always start at zero)

Do not use a negative number to index into an array.

The following sample generates CS0251:

C#

```
// CS0251.cs
// compile with: /W:2
class MyClass
{
    public static void Main()
    {
        int[] myarray = new int[] {1,2,3};
        try
        {
            myarray[-1]++;
            // try the following line instead
            // myarray[1]++;
        }
        catch (System.IndexOutOfRangeException e)
        {
            System.Console.WriteLine("{0}", e);
        }
    }
}
```

# Compiler Warning (level 2) CS0252

Article • 09/15/2021 • 2 minutes to read

Possible unintended reference comparison; to get a value comparison, cast the left hand side to type 'type'

The compiler is doing a reference comparison. If you want to compare the value of strings, cast the left side of the expression to `type`.

The following sample generates CS0252:

C#

```
// CS0252.cs
// compile with: /W:2
using System;

class MyClass
{
    public static void Main()
    {
        string s = "11";
        object o = s + s;

        bool b = o == s;    // CS0252
        // try the following line instead
        // bool b = (string)o == s;
    }
}
```

# Compiler Warning (level 2) CS0253

Article • 09/15/2021 • 2 minutes to read

Possible unintended reference comparison; to get a value comparison, cast the right hand side to type 'type'

The compiler is doing a reference comparison. If you want to compare the value of strings, cast the right side of the expression to `type`.

The following sample generates CS0253:

C#

```
// CS0253.cs
// compile with: /W:2
using System;
class MyClass
{
    public static void Main()
    {
        string s = "11";
        object o = s + s;

        bool c = s == o;    // CS0253
        // try the following line instead
        // bool c = s == (string)o;
    }
}
```

# Compiler Warning (level 2) CS0278

Article • 09/15/2021 • 2 minutes to read

'type' does not implement the 'pattern name' pattern. 'method name' is ambiguous with 'method name'.

There are several statements in C# that rely on defined patterns, such as `foreach` and `using`. For example, the [foreach statement](#) relies on the collection class implementing the "enumerable" pattern.

CS0278 can occur if the compiler is unable to make the match due to ambiguities. For example, the "enumerable" pattern requires that there be a method called `MoveNext`, and your code might contain two methods called `MoveNext`. The compiler will attempt to find an interface to use, but it is recommended that you determine and resolve the cause of the ambiguity.

## Example

The following sample generates CS0278.

C#

```
// CS0278.cs
using System.Collections.Generic;
public class myTest
{
    public static void TestForeach<W>(W w)
        where W: IEnumerable<int>, IEnumerable<string>
    {
        foreach (int i in w) {}    // CS0278
    }
}
```

# Compiler Warning (level 2) CS0279

Article • 09/15/2021 • 2 minutes to read

'type name' does not implement the 'pattern name' pattern. 'method name' is either static or not public.

There are several statements in C# that rely on defined patterns, such as `foreach` and `using`. For example, `foreach` relies on the collection class implementing the enumerable pattern. This error occurs when the compiler is unable to make the match due to a method being declared `static` or not `public`. Methods in patterns are required to be instances of classes, and to be public.

## Example

The following example generates CS0279:

```
C#  
  
// CS0279.cs  
  
using System;  
using System.Collections;  
  
public class myTest : IEnumerable  
{  
    IEnumerator IEnumerable.GetEnumerator()  
    {  
        yield return 0;  
    }  
  
    internal IEnumerator GetEnumerator()  
    {  
        yield return 0;  
    }  
  
    public static void Main()  
    {  
        foreach (int i in new myTest()) {} // CS0279  
    }  
}
```

# Compiler Warning (level 2) CS0280

Article • 09/15/2021 • 2 minutes to read

'type' does not implement the 'pattern name' pattern. 'method name' has the wrong signature.

Two statements in C#, **foreach** and **using**, rely on predefined patterns, "collection" and "resource" respectively. This warning occurs when the compiler cannot match one of these statements to its pattern due to a method's incorrect signature. For example, the "collection" pattern requires that there be a method called **MoveNext** which takes no parameters and returns a **boolean**. Your code might contain a **MoveNext** method that has a parameter or perhaps returns an object.

The "resource" pattern and **using** provide another example. The "resource" pattern requires the **Dispose** method; if you define a property with the same name, you will get this warning.

To resolve this warning, ensure that the method signatures in your type match the signatures of the corresponding methods in the pattern, and ensure that you have no properties with the same name as a method required by the pattern.

## Example

The following sample generates CS0280.

```
C#  
  
// CS0280.cs  
using System;  
using System.Collections;  
  
public class ValidBase: IEnumerable  
{  
    IEnumerator IEnumerable.GetEnumerator()  
    {  
        yield return 0;  
    }  
  
    internal IEnumerator GetGetEnumerator()  
    {  
        yield return 0;  
    }  
}  
  
class Derived : ValidBase  
{
```

```
// field, not method
new public int GetEnumerator;
}

public class Test
{
    public static void Main()
    {
        foreach (int i in new Derived()) {} // CS0280
    }
}
```

# Compiler Warning (level 2) CS0435

Article • 09/15/2021 • 2 minutes to read

The namespace 'namespace' in 'assembly' conflicts with the imported type 'type' in 'assembly'. Using the namespace defined in 'assembly'.

This warning is issued when a namespace in a source file (file\_2) conflicts with an imported type in file\_1. The compiler uses the one in the source file.

The following example generates CS0435:

Compile this file first:

```
C#  
  
// CS0435_1.cs  
// compile with: /t:library  
public class Util  
{  
    public class A { }  
}
```

Then, compile this file:

```
C#  
  
// CS0435_2.cs  
// compile with: /r:CS0435_1.dll  
  
using System;  
  
namespace Util  
{  
    public class A { }  
}  
  
public class Test  
{  
    public static void Main()  
    {  
        Console.WriteLine(typeof(Util.A)); // CS0435  
    }  
}
```

# Compiler Warning (level 2) CS0436

Article • 09/15/2021 • 2 minutes to read

The type 'type' in 'assembly' conflicts with the imported type 'type2' in 'assembly'. Using the type defined in 'assembly'.

This warning is issued when a type in a source file (file\_2) conflicts with an imported type in file\_1. The compiler uses the one in the source file.

## Example 1

```
C#  
  
// CS0436_a.cs  
// compile with: /target:library  
public class A {  
    public void Test() {  
        System.Console.WriteLine("CS0436_a");  
    }  
}
```

## Example 2

The following example generates CS0436.

```
C#  
  
// CS0436_b.cs  
// compile with: /reference:CS0436_a.dll  
// CS0436 expected  
public class A {  
    public void Test() {  
        System.Console.WriteLine("CS0436_b");  
    }  
}  
  
public class Test  
{  
    public static void Main()  
    {  
        A x = new A();  
        x.Test();  
    }  
}
```

# Compiler Warning (level 2) CS0437

Article • 09/15/2021 • 2 minutes to read

The type 'type' in 'assembly2' conflicts with the imported namespace 'namespace' in 'fassembly1'. Using the type defined in 'assembly'.

This warning is issued when a type in a source file, file\_2, conflicts with an imported namespace in file\_1. The compiler uses the type in the source file.

## Example 1

```
C#  
  
// CS0437_a.cs  
// compile with: /target:library  
namespace Util  
{  
    public class A {  
        public void Test() {  
            System.Console.WriteLine("CS0437_a.cs");  
        }  
    }  
}
```

## Example 2

The following sample generates CS0437.

```
C#  
  
// CS0437_b.cs  
// compile with: /reference:CS0437_a.dll /W:2  
// CS0437 expected  
class Util  
{  
    public class A {  
        public void Test() {  
            System.Console.WriteLine("CS0437_b.cs");  
        }  
    }  
}  
  
public class Test  
{  
    public static void Main()  
    {
```

```
    Util.A x = new Util.A();
    x.Test();
}
}
```

# Compiler Warning (level 2) CS0440

Article • 09/15/2021 • 2 minutes to read

Defining an alias named 'global' is ill-advised since 'global::' always references the global namespace and not an alias

This warning is issued when you define an alias named global.

## Example

The following example generates CS0440:

C#

```
// CS0440.cs
// Compile with: /W:1

using global = MyClass;    // CS0440
class MyClass
{
    static void Main()
    {
        // Note how global refers to the global namespace
        // even though it is redefined above.
        global::System.Console.WriteLine();
    }
}
```

# Compiler Warning (level 2) CS0444

Article • 09/15/2021 • 2 minutes to read

Predefined type 'type name 1' was not found in 'System namespace 1' but was found in 'System namespace 2'

A predefined object such as [Int32](#) was not found where the compiler expected to find it, but instead found it in 'System namespace 2'.

The error could indicate that .NET is installed incorrectly. To fix this, reinstall .NET.

If you are writing your own base class libraries, you might also encounter this error. In this case, to resolve the error, rebuild mscorelib.

## Note

This compiler warning is no longer used in Roslyn.

# Compiler Warning (level 2) CS0458

Article • 09/15/2021 • 2 minutes to read

The result of the expression is always 'null' of type 'type name'

This warning is caused by a nullable value type expression that always results in `null`.

The following code generates warning CS0458.

## Example

This example illustrates a number of the different operations with nullable value types that will cause this error.

C#

```
// CS0458.cs
using System;
public class Test
{
    public static void Main()
    {
        int a = 5;
        int? b = a + null;      // CS0458
        int? qa = 15;
        b = qa + null;         // CS0458
        b -= null;              // CS0458
        int? qa2 = null;
        b = qa2 + null;         // CS0458
        qa2 -= null;             // CS0458
    }
}
```

# Compiler Warning (level 2) CS0464

Article • 09/15/2021 • 2 minutes to read

Comparing with null of type 'type' always produces 'false'

This warning is produced when you perform a comparison between a nullable value type variable and null, and the comparison is not `==` or `!=`. To resolve this error, verify if you really want to check a value for `null`. A comparison like `i == null` can be either true or false. A comparison like `i > null` is always false.

## Example

The following sample generates CS0464.

C#

```
// CS0464.cs
class MyClass
{
    public static void Main()
    {
        int? i = 0;
        if (i < null) ;    // CS0464

        i++;
    }
}
```

# Compiler Warning (level 2) CS0467

Article • 09/15/2021 • 2 minutes to read

Ambiguity between method 'method' and non-method 'non-method'. Using method group.

Inherited members from different interfaces that have the same signature cause an ambiguity error.

## Example

The following example generates CS0467.

C#

```
// CS0467.cs
interface IList
{
    int Count { get; set; }
}

interface ICounter
{
    void Count(int i);
}

interface IListCounter : IList, ICounter {}

class Driver
{
    void Test(IListCounter x)
    {
        // The following line causes the warning. The assignment also
        // causes an error because you can't assign a value to a method.
        x.Count = 1;
        x.Count(3);
        // To resolve the warning, you can change the name of the method or
        // the property.

        // You can also disambiguate by specifying IList or ICounter.
        ((IList)x).Count = 1;
        ((ICounter)x).Count(3);
    }

    static void Main()
    {
    }
}
```

# Compiler Warning (level 2) CS0469

Article • 09/15/2021 • 2 minutes to read

The 'goto case' value is not implicitly convertible to type 'type'

When you use `goto case`, there must be an implicit conversion from the value of the `goto case` to the type of the switch.

## Example

The following sample generates CS0469.

C#

```
// CS0469.cs
// compile with: /W:2
class Test
{
    static void Main()
    {
        char c = (char)180;
        switch (c)
        {
            case (char)127:
                break;

            case (char)180:
                goto case 127;    // CS0469
                // try the following line instead
                // goto case (char) 127;
        }
    }
}
```

# Compiler Warning (level 2) CS0472

Article • 09/15/2021 • 2 minutes to read

The result of the expression is always 'value1' since a value of type 'value2' is never equal to 'null' of type 'value3'

The compiler should warn if you use an operator with a constant null value.

## Example

The following sample generates CS0472.

C#

```
public class Test
{
    public static int Main()
    {
        int i = 5;
        int counter = 0;

        // Comparison:
        if (i == null) // CS0472
        // To resolve, use a valid value for i.
            counter++;
        return counter;
    }
}
```

# Compiler Warning (level 2) CS0618

Article • 09/15/2021 • 2 minutes to read

'member' is obsolete: 'text'

A class member was marked with the `Obsolete` attribute, such that a warning will be issued when the class member is referenced. For more information, see [Common Attributes](#).

The following sample generates CS0618:

C#

```
// CS0618.cs
// compile with: /W:2
using System;

public class C
{
    [Obsolete("Use newMethod instead", false)] // warn if referenced
    public static void m2()
    {
    }

    public static void newMethod()
    {
    }
}

class MyClass
{
    public static void Main()
    {
        C.m2(); // CS0618
    }
}
```

# Compiler Warning (level 2) CS0652

Article • 09/15/2021 • 2 minutes to read

Comparison to integral constant is useless; the constant is outside the range of type 'type'

The compiler detected a comparison between a constant and a variable where the constant is out of the range of the variable.

The following sample generates CS0652:

C#

```
// CS0652.cs
// compile with: /W:2
public class Class1
{
    private static byte i = 0;
    public static void Main()
    {
        short j = 256;
        if (i == 256)    // CS0652, 256 is out of range for byte
            i = 0;
    }
}
```

# Compiler Warning (level 2) CS0728

Article • 09/15/2021 • 2 minutes to read

Possibly incorrect assignment to local 'variable' which is the argument to a using or lock statement. The Dispose call or unlocking will happen on the original value of the local.

There are several scenarios where `using` or `lock` blocks will result in a temporary leak of resources. Here is one example:

```
thisType f = null;

using (f)

{
    f = new thisType();

    ...

}
```

In this case, the original value, such as null, of the variable `thisType` will be disposed of when the `using` block finishes executing, but the `thisType` object created inside the block will not be, although it will eventually get garbage collected.

To resolve this error, use the following form:

```
using (thisType f = new thisType())

{
    ...

}
```

In this case, the newly allocated `thisType` object will be disposed of.

## Example

The following code will generate warning CS0728.

C#

```
// CS0728.cs

using System;
public class ValidBase : IDisposable
{
    public void Dispose() { }
}

public class Logger
{
    public static void dummy()
    {
        ValidBase vb = null;
        using (vb)
        {
            vb = null; // CS0728
        }
        vb = null;
    }
    public static void Main() { }
}
```

# Compiler Warning (level 2) CS1571

Article • 09/15/2021 • 2 minutes to read

XML comment on 'construct' has a duplicate param tag for 'parameter'

When using the [DocumentationFile](#) compiler option, multiple comments were found for the same method parameter. Remove one of the duplicate lines.

The following sample generates CS1571:

C#

```
// CS1571.cs
// compile with: /W:2 /doc:x.xml

/// <summary>help text</summary>
public class MyClass
{
    /// <param name='Int1'>Used to indicate status.</param>
    /// <param name='Char1'>An initial.</param>
    /// <param name='Int1'>Used to indicate status.</param> // CS1571
    public static void MyMethod(int Int1, char Char1)
    {
    }

    /// <summary>help text</summary>
    public static void Main ()
    {
    }
}
```

# Compiler Warning (level 2) CS1572

Article • 09/15/2021 • 2 minutes to read

XML comment on 'construct' has a param tag for 'parameter', but there is no parameter by that name

When using the [DocumentationFile](#) compiler option, a comment was specified for a parameter that does not exist for the method. Change the value passed to the name attribute or remove one of the comment lines.

The following sample generates CS1572:

```
C#  
  
// CS1572.cs  
// compile with: /W:2 /doc:x.xml  
  
/// <summary>help text</summary>  
public class MyClass  
{  
    /// <param name='Int1'>Used to indicate status.</param>  
    /// <param name='Char1'>Used to indicate status.</param>  
    /// <param name='Char2'>???</param> // CS1572  
    public static void MyMethod(int Int1, char Char1)  
    {  
    }  
  
    /// <summary>help text</summary>  
    public static void Main ()  
    {  
    }  
}
```

# Compiler Warning (level 2) CS1587

Article • 09/15/2021 • 2 minutes to read

XML comment is not placed on a valid language element

Recommended tags for documentation comments are not allowed on all language elements. For example, a tag is not allowed on a namespace. For more information on XML comments, see [Recommended Tags for Documentation Comments](#).

## Example

The following sample generates CS1587:

C#

```
// CS1587.cs
// compile with: /W:2 /doc:x.xml

/// <summary>test</summary> // CS1587, tag not allowed on namespace
namespace MySpace
{
    class MyClass
    {
        public static void Main()
        {
        }
    }
}
```

# Compiler Warning (level 2) CS1668

Article • 09/15/2021 • 2 minutes to read

Invalid search path 'path' specified in 'path string' -- 'system error message'

The path supplied to [AdditionalLibPaths](#) at the command line was not valid, or a path in the LIB environment variable is invalid. Check the path used to verify that it exists and can be accessed. The error message in single quotation marks is the error returned from the operating system.

## See also

- [C# Compiler Options](#)
- [C# Compiler Errors](#)



# Compiler Warning (level 2) CS1698

Article • 09/15/2021 • 2 minutes to read

Circular assembly reference 'AssemblyName1' does not match the output assembly name 'AssemblyName2'. Try adding a reference to 'AssemblyName1' or changing the output assembly name to match.

CS1698 occurs when an assembly reference is incorrect. This can happen if a referenced assembly is recompiled. To resolve, do not replace an assembly that itself is a dependency of an assembly you are referencing.

## Example 1

C#

```
// CS1698_a.cs
// compile with: /target:library /keyfile:mykey.snk
[assembly:System.Reflection.AssemblyVersion("2")]
public class CS1698_a {}
```

## Example 2

C#

```
// CS1698_b.cs
// compile with: /target:library /reference:CS1698_a.dll /keyfile:mykey.snk
public class CS1698_b : CS1698_a {}
```

## Example 3

The following sample generates CS1698.

C#

```
// CS1698_c.cs
// compile with: /target:library /out:cs1698_a.dll /reference:cs1698_b.dll
//keyfile:mykey.snk
// CS1698 expected
[assembly:System.Reflection.AssemblyVersion("3")]
public class CS1698_c : CS1698_b {}
public class CS1698_a {}
```

# Compiler Warning (level 2) CS1701

Article • 09/15/2021 • 2 minutes to read

Assuming assembly reference "Assembly Name #1" matches "Assembly Name #2", you may need to supply runtime policy

The two assemblies differ in release and/or version number. For unification to occur, you must specify directives in the application's .config file, and you must provide the correct strong name of an assembly, as demonstrated in the following example code.

## Example 1

The following multifile sample references an assembly using two different external aliases. This first sample builds the older version of the code that creates assembly CS1701\_d.

C#

```
// CS1701_a.cs
// compile with: /target:library /out:cs1701_d.dll /keyfile:mykey.snk
using System.Reflection;
[assembly: AssemblyVersion("1.0")]
public class A {
    public void M1() {}
}

public class C1 {}
```

## Example 2

This is the code that creates the newer version of assembly CS1701\_d. Note that it compiles into a different directory than the older version, necessary since the output files have the same names.

C#

```
// CS1701_b.cs
// compile with: /target:library /out:c:\\cs1701_d.dll /keyfile:mykey.snk
using System.Reflection;
[assembly: AssemblyVersion("2.0")]
public class A {
    public void M2() {}
    public void M1() {}
}
```

```
public class C2 {}
public class C1 {}
```

## Example 3

This sample sets up the external aliases A1 and A2.

C#

```
// CS1701_c.cs
// compile with: /target:library /reference:A2=c:\\cs1701_d.dll
/reference:A1=cs1701_d.dll

extern alias A1;
extern alias A2;
// using System;
using a1 = A1::A;
using a2 = A2::A;

public class Ref {
    public static a1 A1() { return new a1(); }
    public static a2 A2() { return new a2(); }

    public static A1::C1 M1() { return new A1::C1(); }
    public static A2::C2 M2() { return new A2::C2(); }
}
```

## Example 4

This sample calls methods using two different aliases of A. The following sample generates C1701.

C#

```
// CS1701_d.cs
// compile with: /reference:c:\\CS1701_d.dll /reference:CS1701_c.dll
// CS1701 expected
class Tester {
    public static void Main() {
        Ref.A1().M1();
        Ref.A2().M2();
    }
}
```

# Compiler Warning (level 2) CS1710

Article • 09/15/2021 • 2 minutes to read

XML comment on 'type' has a duplicate typeparam tag for 'parameter'

The documentation of a generic type includes a duplicate tag for the type parameter.

## Example

The following code will cause warning CS1710 to appear.

C#

```
// CS1710.cs
// compile with: /doc:cs1710.xml
// To resolve this warning, delete one of the duplicate <typeparam>'s.
using System;
class Stack<ItemType>
{
}

/// <typeparam name="MyType">can be an int</typeparam>
/// <typeparam name="MyType">can be an int</typeparam>
class MyStackWrapper<MyType>
{
    // Open constructed type Stack<MyType>.
    Stack<MyType> stack;
    public MyStackWrapper(Stack<MyType> s)
    {
        stack = s;
    }
}

class CMain
{
    public static void Main()
    {
        // Closed constructed type Stack<int>.
        Stack<int> stackInt = new Stack<int>();
        MyStackWrapper<int> MyStackWrapperInt =
            new MyStackWrapper<int>(stackInt);
    }
}
```

# Compiler Warning (level 2) CS1711

Article • 09/15/2021 • 2 minutes to read

XML comment on 'type' has a typeparam tag for 'parameter', but there is no type parameter by that name

The documentation of a generic type includes a tag for the type parameter that has the wrong name.

## Example

The following code generates warning CS1711.

C#

```
// cs1711.cs
// compile with: /doc:cs1711.xml
// CS1711 expected
using System;
///<typeparam name="WrongName">can be an int</typeparam>
class CMain
{
    public static void Main() { }
```

# Compiler Warning (Level 2) CS1927

Article • 09/15/2021 • 2 minutes to read

Ignoring /win32manifest for module because it only applies to assemblies.

A win32 manifest is only applied at the assembly level. Your module will compile but it will not have a manifest.

## To correct this error

1. Remove the /win32manifest option.
2. Compile the code as an assembly.

## Example

The following example generates CS1927 when it is compiled with both the /target:module and /win32manifest compiler options.

```
C#  
  
// cs1927.cs  
// Compile with: /target:module /win32manifest  
using System;  
  
class ManifestWithModule  
{  
    static int Main()  
    {  
        return 1;  
    }  
}
```

## See also

- [Win32Manifest \(C# Compiler Options\)](#)
- [TargetType module \(C# Compiler Options\)](#)



# Compiler Warning (level 2) CS3019

Article • 09/15/2021 • 2 minutes to read

CLS compliance checking will not be performed on 'type' because it is not visible from outside this assembly.

This warning occurs when a type or a member that has the [CLSCompliantAttribute](#) attribute is not visible from another assembly. To resolve this error, remove the attribute on any classes or members that are not visible from the other assembly, or make the type or members visible. For more information on CLS compliance, see [Language independence and language-independent components](#).

## Example

The following sample generates CS3019:

```
C#  
  
// CS3019.cs  
// compile with: /W:2  
  
using System;  
  
[assembly: CLSCompliant(true)]  
  
// To fix the error, remove the next line  
[CLSCompliant(true)] // CS3019  
class C  
{  
    [CLSCompliant(false)] // CS3019  
    void Foo()  
    {  
    }  
  
    static void Main()  
    {  
    }  
}
```

## See also

- [Language Independence and Language-Independent Components](#)



# Compiler Warning (level 2) CS3021

Article • 09/15/2021 • 2 minutes to read

'type' does not need a CLSCompliant attribute because the assembly does not have a CLSCompliant attribute

This warning occurs if `[CLSCompliant(false)]` appears on a class in an assembly which does not have an assembly-level CLSCompliant attribute set to true (that is, the line `[assembly: CLSCompliant(true)]`). Since the assembly is not declaring itself CLS compliant, there is no need for anything within the assembly to declare itself non-compliant, since it is assumed to be non-compliant. For more information on CLS compliance, see [Language independence and language-independent components](#).

To get rid of this warning, remove the attribute or add the assembly level attribute.

## Example

The following example generates CS3021:

```
C#  
  
// CS3021.cs  
using System;  
// Uncomment the following line to declare the assembly CLS Compliant,  
// and avoid the warning without removing the attribute on the class.  
//[assembly: CLSCompliant(true)]  
  
// Remove the next line to avoid the warning.  
[CLSCompliant(false)]           // CS3021  
public class C  
{  
    public static void Main()  
    {  
    }  
}
```

## See also

- [Language Independence and Language-Independent Components](#)



# Compiler Warning (level 3) CS0067

Article • 01/23/2023 • 2 minutes to read

The event 'event' is never used

An [event](#) was declared but never used in the class in which it was declared.

The following sample generates CS0067:

```
C#  
  
// CS0067.cs  
// compile with: /W:3  
using System;  
delegate void MyDelegate();  
  
class MyClass  
{  
    public event MyDelegate evt; // CS0067  
    // uncomment TestMethod to resolve this CS0067  
/*  
    private void TestMethod()  
    {  
        if (evt != null)  
            evt();  
    }  
*/  
    public static void Main()  
    {  
    }  
}
```

If the event is unused intentionally, for example, when it's part of an interface implementation, then you can avoid emitting an unnecessary field as follows:

```
C#  
  
using System;  
  
public interface IThing  
{  
    event Action? E;  
}  
  
public class Thing : IThing  
{  
    // no CS0067 though the event is left unused  
    public event Action? E { add { } remove { } }  
}
```

# Compiler Warning (level 3) CS0105

Article • 09/15/2021 • 2 minutes to read

The using directive for 'namespace' appeared previously in this namespace

A [namespace](#), which should only be declared once, was declared more than once; remove all duplicate namespace declarations.

The following sample generates CS0105:

C#

```
// CS0105.cs
// compile with: /W:3
using System;
using System; // CS0105

public class a
{
    public static void Main()
    {
    }
}
```

# Compiler Warning (level 3) CS0168

Article • 09/15/2021 • 2 minutes to read

The variable 'var' is declared but never used

The compiler issues a level-three warning when you declare a variable, but do not use it.

The following sample generates a CS0168 warning:

C#

```
// CS0168.cs
// compile with: /W:3
public class clx
{
    public int i;
}

public class clz
{
    public static void Main() {
        clx a; // CS0168, the variable 'a' is declared but never used
        // try the following line instead
        // clx a = new clx(); // this does not generate a warning because
the clx constructor must execute.
    }
}
```

# Compiler Warning (level 3) CS0169

Article • 11/06/2021 • 2 minutes to read

The private field 'class member' is never used

A private variable was declared but never referenced. A common way to generate this warning is when you declare a private member of a class and do not use it.

The following sample generates CS0169:

C#

```
// compile with: /W:3
using System;
public class ClassX
{
    int i;    // CS0169, i is not used anywhere
    // Remove the above variable declaration or uncomment TestMethod to clear
    warning CS0169
    /*
    public void TestMethod()
    {
        i = 5;
        System.Console.WriteLine(i);
    }
    */
}

public static void Main()
{
}
```

# Compiler Warning (level 3) CS0219

Article • 09/15/2021 • 2 minutes to read

The variable 'variable' is assigned but its value is never used

The compiler issues a level-three warning, when you declare and assign a variable, but do not use it.

## ⓘ Note

The compiler generates this warning only when the variable value is a compile-time constant. Assigning a non-constant expression or method result to a local variable makes it easier to observe those expressions in the debugger. It also makes the result reachable, preventing garbage collection while that variable is reachable.

The following sample shows the cases when and when not the warning is generated:

C#

```
// CS0219.cs
// compile with: /W:3
public class MyClass
{
    public static void Main()
    {
        var interpolated = "Interpolated";
        var a = 0; // CS0219
        int b = GetZero(); // Doesn't generate a warning.
        var c = "Regular string"; // CS0219
        var d = $"Constant interpolated string"; // Doesn't generate a
warning.
        var e = $"{interpolated} string"; // Doesn't generate a warning.
    }

    private static int GetZero()
    {
        return 0;
    }
}
```

# Compiler Warning (level 3) CS0282

Article • 11/04/2022 • 2 minutes to read

There is no defined ordering between fields in multiple declarations of partial class or struct 'type'. To specify an ordering, all instance fields must be in the same declaration.

To resolve this error, put all member variables in a single partial class definition.

A common way to get this error is by having a partial `struct` defined in more than one place, with some of the member variables in one definition, and some in another.

The following code generates CS0282.

## Example 1

This code contains one description of a `struct`. Compile these two modules together in a single step by means of the command:

```
csc /target:library cs0282_1.cs cs0282_2.cs
```

```
C#  
  
partial struct A  
{  
    int i;  
}
```

## Example 2

This code contains a conflicting description of the same `struct`.

```
C#  
  
partial struct A  
{  
    int j;  
}
```

### ⓘ Note

If the struct layout does not matter, decorating the struct with `[StructLayout(LayoutKind.Auto)]` will express it, and suppress the warning



# Compiler Warning (level 3) CS0414

Article • 09/15/2021 • 2 minutes to read

The private field 'field' is assigned but its value is never used

This warning can occur in several scenarios in which the compiler can verify that a variable is never referenced:

- A private field is assigned a constant value but is never subsequently read. The unnecessary assignment could effect performance. Consider removing the field.
- A private or internal static field is assigned a constant value only in the initializer. Consider changing the field to a const.
- A private or internal field is assigned constant values and only used in blocks that are excluded by #ifdef directives. Consider putting the field inside the #ifdef block.
- A private or internal field is assigned constant values in multiple locations but not otherwise accessed. If you do not need the field, consider removing it. Otherwise, use it in some appropriate way.

In other situations, or where the suggested workaround is not acceptable, use #pragma 0414.

The following sample shows one way in which CS0414 will be generated:

```
C#  
  
// CS0414  
// compile with: /W3  
class C  
{  
    private int i = 1; // CS0414  
  
    public static void Main()  
    { }  
}
```

## ⓘ Note

If the variable `i` is declared as `protected` or `public`, no error will be generated because the compiler cannot know whether a derived class might use it or some other client code might instantiate the class and reference the variable

## See also

- [C# Compiler Errors](#)
- [C# Compiler Options](#)

# Compiler Warning (level 3) CS0419

Article • 09/15/2021 • 2 minutes to read

Ambiguous reference in cref attribute: 'Method Name1'. Assuming 'Method Name2', but could have also matched other overloads including 'Method Name3'.

In an XML documentation comment in the code, a reference could not be resolved. This could occur if the method is overloaded, or if two different identifiers with the same name are found. To resolve the warning, use a qualified name to disambiguate the reference, or include the specific overload in parentheses.

The following sample generates CS0419.

```
C#  
  
// cs0419.cs  
// compile with: /doc:x.xml /W:3  
interface I  
{  
    /// text for F(void)  
    void F();  
    /// text for F(int)  
    void F(int i);  
}  
/// text for class MyClass  
public class MyClass  
{  
    /// <see cref="I.F"/>  
    public static void MyMethod(int i)  
    {  
    }  
/* Try this instead:  
    /// <see cref="I.F(int)">  
    public static void MyMethod(int i)  
    {  
    }  
*/  
    /// text for Main  
    public static void Main ()  
    {  
    }  
}
```

# Compiler Warning (level 3) CS0642

Article • 05/13/2022 • 2 minutes to read

Possible mistaken empty statement

A semicolon after a conditional statement may cause your code to execute differently than intended.

You can use **NoWarn** compiler option or `#pragmas warning` to disable this warning; see [NoWarn \(C# Compiler Options\)](#) or [#pragma warning](#) for more information.

The following sample generates CS0642:

C#

```
// CS0642.cs
// compile with: /W:3
class MyClass
{
    public static void Main()
    {
        int i;

        for (i = 0; i < 10; i += 1);    // CS0642 semicolon intentional?
        {
            System.Console.WriteLine (i);
        }
    }
}
```

# Compiler Warning (level 3) CS0659

Article • 09/15/2021 • 2 minutes to read

'class' overrides Object.Equals(object o) but does not override Object.GetHashCode()

The compiler detected an override of the [Object.Equals](#) method but no override of the [Object.GetHashCode](#) method. An override of [Equals](#) implies that you also want to override [GetHashCode](#).

The following code generates CS0659:

C#

```
// CS0659.cs
// compile with: /W:3 /target:library
class Test
{
    public override bool Equals(object o) { return true; } // CS0659
}

// OK
class Test2
{
    public override bool Equals(object o) { return true; }
    public override int GetHashCode() { return 0; }
}
```

## See also

- [Object.Equals](#)
- [Object.GetHashCode](#)
- [Equality Operators](#)
- [Implementing the Equals Method](#)



# Compiler Warning (level 3) CS0660

Article • 09/15/2021 • 2 minutes to read

'class' defines operator == or operator != but does not override Object.Equals(object o)

The compiler detected the user-defined equality or inequality operator, but no override for the [Object.Equals](#) method. A user-defined equality or inequality operator implies that you also want to override the [Equals](#) method. For more information, see [How to define value equality for a type](#).

The following sample generates CS0660:

```
C#  
  
// CS0660.cs  
// compile with: /W:3 /warnaserror  
class Test // CS0660  
{  
    public static bool operator == (object o, Test t)  
    {  
        return true;  
    }  
  
    // uncomment the Equals function to resolve  
    // public override bool Equals(object o)  
    // {  
    //     return true;  
    // }  
  
    public override int GetHashCode()  
    {  
        return 0;  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 3) CS0661

Article • 09/15/2021 • 2 minutes to read

'class' defines operator == or operator != but does not override Object.GetHashCode()

The compiler detected the user-defined equality or inequality operator, but no override for the **GetHashCode** function. A user-defined equality or inequality operator implies that you also want to override the **GetHashCode** function.

The following sample generates CS0661:

```
C#  
  
// CS0661.cs  
// compile with: /W:3  
class Test // CS0661  
{  
    public static bool operator == (object o, Test t)  
    {  
        return true;  
    }  
  
    public static bool operator != (object o, Test t)  
    {  
        return true;  
    }  
  
    public override bool Equals(object o)  
    {  
        return true;  
    }  
  
    // uncomment the GetHashCode function to resolve  
    // public override int GetHashCode()  
    // {  
    //     return 0;  
    // }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 3) CS0665

Article • 09/15/2021 • 2 minutes to read

Assignment in conditional expression is always constant; did you mean to use == instead of = ?

A conditional expression used the [= operator](#) and not the [== operator](#).

The following sample generates CS0665:

C#

```
// CS0665.cs
// compile with: /W:3
class Test
{
    public static void Main()
    {
        bool i = false;

        if (i = true)    // CS0665
            // try the following line instead
            // if (i == true)
        {
        }

        System.Console.WriteLine(i);
    }
}
```

# Compiler Warning (level 3) CS0675

Article • 10/27/2021 • 2 minutes to read

Bitwise-or operator used on a sign-extended operand; consider casting to a smaller unsigned type first

The compiler implicitly widened and sign-extended a variable, and then used the resulting value in a bitwise OR operation. This can result in unexpected behavior.

The following sample generates CS0675:

C#

```
// CS0675.cs
// compile with: /W:3
using System;

public class sign
{
    public static void Main()
    {
        int hi = 1;
        int lo = -1;
        long value = (((long)hi) << 32) | lo;                // CS0675, value
contains -1 (0xffffffff_ffffffff)
        // try the following line instead
        // long value = (((long)hi) << 32) | ((uint)lo); // correct, value
contains 8589934591 (0x00000001_ffffffff)
    }
}
```

# Compiler Warning (level 3) CS0693

Article • 09/15/2021 • 2 minutes to read

Type parameter 'type parameter' has the same name as the type parameter from outer type 'type'

This error occurs when you have a generic member such as a method inside a generic class. Since the method's type parameter is not necessarily the same as the class's type parameter, you cannot give them both the same name. For more information, see [Generic Methods](#).

To avoid this situation, use a different name for one of the type parameters.

## Example

The following sample generates CS0693.

```
C#  
  
// CS0693.cs  
// compile with: /W:3 /target:library  
class Outer<T>  
{  
    class Inner<T> {} // CS0693  
    // try the following line instead  
    // class Inner<U> {}  
}
```

# Compiler Warning (level 3) CS1700

Article • 09/15/2021 • 2 minutes to read

Assembly reference Assembly Name is invalid and cannot be resolved

This warning indicates that an attribute, such as [InternalsVisibleToAttribute](#), was not specified correctly.

For more information, see [Friend Assemblies](#).

## Example

The following sample generates CS1700.

C#

```
// CS1700.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("app2, Retargetable=f")]    // CS1700
[assembly:InternalsVisibleTo("app2")]    // OK
```

# Compiler Warning (level 3) CS1702

Article • 09/15/2021 • 2 minutes to read

Assuming assembly reference "Assembly Name #1" matches "Assembly Name #2", you may need to supply runtime policy

The two assembly references have differing build and/or revision numbers, so will not automatically unify. You may need to supply run-time policy to force unification by using directives in the application .config file.

# Compiler Warning (level 3) CS1717

Article • 09/15/2021 • 2 minutes to read

Assignment made to same variable; did you mean to assign something else?

This warning occurs when you assign a variable to itself, such as `a = a`.

Several common mistakes can generate this warning:

- Writing `a = a` as the condition of an `if` statement, such as `if (a = a)`. You probably meant to say `if (a == a)`, which is always true, so you could write this more concisely as `if (true)`.
- Mistyping. You probably meant to say `a = b`.
- In a constructor where the parameter has the same name as the field, not using the `this` keyword: you probably meant to say `this.a = a`.

## Example

The following sample generates CS1717.

```
C#  
  
// CS1717.cs  
// compile with: /W:3  
public class Test  
{  
    public static void Main()  
    {  
        int x = 0;  
        x = x;    // CS1717  
    }  
}
```

# Compiler Warning (level 3) CS1718

Article • 09/15/2021 • 2 minutes to read

Comparison made to same variable; did you mean to compare something else?

If you meant to compare to something else, then correct the statement.

But another possibility is that you were testing for true or false, and were doing so by statements such as `if (a == a) (true)` or `if (a < a) (false)`. It is better to use `if (true)` or `if (false)`, for the following two reasons:

- It is simpler: it is always clearer to simply say what you mean.
- It helps avoid confusion: a new feature of C# 2.0 is nullable value types, which are analogous to the value `null` in T-SQL, the programming language used by SQL Server. Developers familiar with T-SQL might be concerned about the effect of nullable value types on expressions such as `if (a == a)`, because of the use of ternary logic in T-SQL. If you use `true` or `false`, you avoid this possible confusion.

## Example

The following code generates warning CS1718.

```
C#  
  
// CS1718.cs  
using System;  
public class Tester  
{  
    public static void Main()  
    {  
        int i = 0;  
        if (i == i) { // CS1718.cs  
        //if (true)  
            i++;  
        }  
    }  
}
```

# Compiler Warning (level 4) CS0028

Article • 09/15/2021 • 2 minutes to read

'function declaration' has the wrong signature to be an entry point

The method declaration for `Main` was invalid: it was declared with an invalid signature.

`Main` must be declared as static and it must return either `int` or `void`. For more information, see [Main\(\) and Command-Line Arguments](#).

The following sample generates CS0028:

C#

```
// CS0028.cs
// compile with: /W:4 /warnaserror
public class a
{
    public static double Main(int i)    // CS0028
    // Try the following line instead:
    // public static void Main()
    {
    }
}
```

# Compiler Warning (level 4) CS0078

Article • 09/15/2021 • 2 minutes to read

The 'l' suffix is easily confused with the digit '1' -- use 'L' for clarity

The compiler warns when it detects a long literal using a lowercase l instead of an uppercase L.

The following sample generates CS0078:

C#

```
class Program
{
    public static void TestL(long i)
    {
    }

    public static void Main()
    {
        TestL(25l);    // CS0078
        // try the following line instead
        // TestL(25L);
    }
}
```

# Compiler Warning (level 4) CS0109

Article • 09/15/2021 • 2 minutes to read

The member 'member' does not hide an inherited member. The new keyword is not required

A class declaration included the [new](#) keyword even though the declaration does not override an existing declaration in a base class. You can delete the [new](#) keyword.

The following sample generates CS0109:

```
C#  
  
// CS0109.cs  
// compile with: /W:4  
namespace x  
{  
    public class a  
    {  
        public int i;  
    }  
  
    public class b : a  
    {  
        public new int i;  
        public new int j; // CS0109  
        public static void Main()  
        {  
        }  
    }  
}
```

# Compiler Warning (level 4) CS0402

Article • 09/15/2021 • 2 minutes to read

'identifier' : an entry point cannot be generic or in a generic type

The entry point was found in a generic type. To remove this warning, implement Main in a non-generic class or struct.

C#

```
// CS0402.cs
// compile with: /W:4
class C<T>
{
    public static void Main() // CS0402
    {

    }
}

class CMain
{
    public static void Main() {}
}
```

# Compiler Warning (level 4) CS0422

Article • 09/15/2021 • 2 minutes to read

The `/incremental` option is no longer supported

Incremental compilation (`/incr` or `/incremental`) is not supported in C# 2.0.

 **Note**

This compiler warning is no longer used in Roslyn.

# Compiler Warning (level 4) CS0429

Article • 09/15/2021 • 2 minutes to read

Unreachable expression code detected

This error occurs whenever part of an expression in your code is unreachable. In the following example, the condition `false && myTest()` meets this criteria because the `myTest()` method will never get evaluated due to the fact that the left side of the `&&` operation is always false. As soon as the `&&` operator evaluates the `false` statement as false, it stops the evaluation, and will never evaluate the right side.

## Example

The following code generates CS0429.

```
C#  
  
// CS0429.cs  
public class cs0429  
{  
    public static void Main()  
    {  
        if (false && myTest()) // CS0429  
        // Try the following line instead:  
        // if (true && myTest())  
        {  
        }  
        else  
        {  
            int i = 0;  
            i++;  
        }  
    }  
  
    static bool myTest() { return true; }  
}
```

# Compiler Warning (level 4) CS0628

Article • 09/15/2021 • 2 minutes to read

'member' : new protected member declared in sealed class

A [sealed](#) class cannot introduce a [protected](#) member because no other class will be able to inherit from the [sealed](#) class and use the [protected](#) member.

The following sample generates CS0628:

C#

```
// CS0628.cs
// compile with: /W:4
sealed class C
{
    protected int i;    // CS0628
}

class MyClass
{
    public static void Main()
    {
    }
}
```

# Compiler Warning (level 4) CS0649

Article • 09/15/2021 • 2 minutes to read

Field 'field' is never assigned to, and will always have its default value 'value'

The compiler detected an uninitialized private or internal field declaration that is never assigned a value.

The following sample generates CS0649:

C#

```
// CS0649.cs
// compile with: /W:4
using System.Collections;

class MyClass
{
    Hashtable table; // CS0649
    // You may have intended to initialize the variable to null
    // Hashtable table = null;

    // Or you may have meant to create an object here
    // Hashtable table = new Hashtable();

    public void Func(object o, string p)
    {
        // Or here
        // table = new Hashtable();
        table[p] = o;
    }

    public static void Main()
    {
    }
}
```

# Compiler Warning (level 4) CS1573

Article • 09/15/2021 • 2 minutes to read

Parameter 'parameter' has no matching param tag in the XML comment for 'parameter'  
(but other parameters do)

When using the [DocumentationFile](#) compiler option, a comment was specified for some but not all parameters in a method. You may have forgotten to enter a comment for these parameters.

The following sample generates CS1573:

```
C#  
  
// CS1573.cs  
// compile with: /W:4  
public class MyClass  
{  
    /// <param name='Int1'>Used to indicate status.</param>  
    // enter a comment for Char1?  
    public static void MyMethod(int Int1, char Char1)  
    {  
    }  
  
    public static void Main()  
    {  
    }  
}
```

# Compiler Warning (level 4) CS1591

Article • 12/14/2022 • 2 minutes to read

Missing XML comment for publicly visible type or member 'Type\_or\_Member'

The [DocumentationFile](#) compiler option was specified, but one or more constructs did not have comments.

The following sample generates CS1591:

C#

```
// CS1591.cs
// compile with: /W:4 /doc:x.xml

/// text
public class Test
{
    // /// text
    public static void Main()    // Try the following instead: remove "///"
from previous line (`///` about the comments)
                                //or add #pragma warning disable 1591
                                //a good practice is add the following
comments:
                                // /// <summary>
                                // ///
                                // /// </summary>
                                // this can be easily achieve at Visual
Studio in a simple way adding triple forward slashes (`///`)
    {
    }
}
```

# Compiler Warning (level 4) CS1610

Article • 09/15/2021 • 2 minutes to read

Unable to delete temporary file 'file' used for default Win32 resource -- resource

When using the [Win32Resource](#) compiler option and when your %TEMP% directory does not have DELETE permission, this warning indicates that the compiler could not delete a temporary file that it created.

Make sure that you have read/write/delete permissions for the %TEMP% directory.

If necessary, you can manually delete these files and there will be no harm to C# or any of your programs.

# Compiler Warning (level 4) CS1712

Article • 09/15/2021 • 2 minutes to read

Type parameter 'type parameter' has no matching typeparam tag in the XML comment on 'type' (but other type parameters do)

The documentation of a generic type is missing a **typeparam** tag. For more information, see [<typeparam>](#).

## Example

The following code generates warning CS1712. To resolve this error, add a **typeparam** tag for the type parameter S.

C#

```
// CS1712.cs
// compile with: /doc:cs1712.xml
using System;
class Test
{
    public static void Main() {}
    /// <param name="j"> This is the j parameter.</param>
    /// <typeparam name="T"> This is the T type parameter.</typeparam>
    public void bar<T,S>(int j) {} // CS1712
}
```

# Resolve warnings related to language features and versions

Article • 01/31/2023 • 5 minutes to read

This article covers the following compiler warnings:

- CS8022, CS8023, CS8024, CS8025, CS8026, CS8059, CS8107, CS8302, CS8320, CS8370, CS8400, CS8773, CS8936, CS9058 - *Feature is not available. Use newer language version.*
- CS8192 - *Provided language version is unsupported or invalid*
- CS8303 - *Specified language version cannot have leading zeroes*
- CS8304 - *Compiler version is less than language version*
- CS1738 - *Named argument specifications must appear after all fixed arguments have been specified.*
- CS8306 - *Tuple element name is inferred.*
- CS8314 - *An expression of type cannot be handled by a pattern of type*
- CS8371 - *Field-targeted attributes on auto-properties are not supported in language version*
- CS8401 - *To use @\$ instead of \$@ for an interpolated verbatim string, use newer language version.*
- CS8511 - *An expression of type cannot be handled by a pattern of type.*
- CS8627 - *A nullable type parameter must be known to be a value type or non-nullable reference type*
- CS8630 - *Invalid nullable options. Use newer language version*
- CS8652 - *The modifier is not valid for this item.*
- CS8704 - *Type does not implement interface member. It cannot implicitly implement a non-public member.*
- CS8706 - *Type cannot implement interface member because a feature is not available in this version.*
- CS8904 - *Invalid variance: The type parameter must be valid.*
- CS8912 - *Inheriting from a record with a sealed 'Object.ToString' is not supported.*
- CS8957 - *Conditional expression is not valid in language version because a common type was not found between types.*
- CS8967 - *Newlines inside a non-verbatim interpolated string are not supported in C#*
- CS9014 - *Error: Use of possibly unassigned property. Upgrade to auto-default the property.*
- CS9015 - *Error: Use of possibly unassigned field. Upgrade to auto-default the field.*

- CS9016 - *Warning: Use of possibly unassigned property. Upgrade to auto-default the property.*
- CS9017 - *Warning: Use of possibly unassigned field. Upgrade to auto-default the field.*

In addition, the following errors and warnings relate to struct initialization changes in recent versions:

- CS0171, CS8881: *Backing field for automatically implemented property 'name' must be fully assigned before control is returned to the caller.*
- CS0188, CS8885: *The 'this' object cannot be used before all of its fields are assigned to*
- CS0843, CS8880: *Backing field for automatically implemented property 'name' must be fully assigned before control is returned to the caller*

The cause behind all these errors and warnings is that the compiler installed supports a newer version of C# than the version your project has selected. The C# compiler can conform to any previous version. You can validate syntax against an earlier version of C#, or because your project must support older libraries or runtimes.

There are two possible causes and three ways to address these errors and warnings.

## Update your target framework

The compiler determines a default based on these rules:

Target	Version	C# language version default
.NET	7.x	C# 11
.NET	6.x	C# 10
.NET	5.x	C# 9.0
.NET Core	3.x	C# 8.0
.NET Core	2.x	C# 7.3
.NET Standard	2.1	C# 8.0
.NET Standard	2.0	C# 7.3
.NET Standard	1.x	C# 7.3
.NET Framework	all	C# 7.3

If your selected framework doesn't match the language version required, you can upgrade the target framework.

## Select the matching language version

You may have an older target framework selected in your project file. If you remove the `LangVersion` element from your project file, the compiler will use the default value listed in the preceding section. The following table shows all current C# language versions. You can also specify a specific language version to enable newer features.

Value	Meaning
<code>preview</code>	The compiler accepts all valid language syntax from the latest preview version.
<code>latest</code>	The compiler accepts syntax from the latest released version of the compiler (including minor version).
<code>latestMajor</code> or <code>default</code>	The compiler accepts syntax from the latest released major version of the compiler.
<code>11.0</code>	The compiler accepts only syntax that is included in C# 11 or lower.
<code>10.0</code>	The compiler accepts only syntax that is included in C# 10 or lower.
<code>9.0</code>	The compiler accepts only syntax that is included in C# 9 or lower.
<code>8.0</code>	The compiler accepts only syntax that is included in C# 8.0 or lower.
<code>7.3</code>	The compiler accepts only syntax that is included in C# 7.3 or lower.
<code>7.2</code>	The compiler accepts only syntax that is included in C# 7.2 or lower.
<code>7.1</code>	The compiler accepts only syntax that is included in C# 7.1 or lower.
<code>7</code>	The compiler accepts only syntax that is included in C# 7.0 or lower.
<code>6</code>	The compiler accepts only syntax that is included in C# 6.0 or lower.
<code>5</code>	The compiler accepts only syntax that is included in C# 5.0 or lower.
<code>4</code>	The compiler accepts only syntax that is included in C# 4.0 or lower.
<code>3</code>	The compiler accepts only syntax that is included in C# 3.0 or lower.
<code>ISO-2</code> or <code>2</code>	The compiler accepts only syntax that is included in ISO/IEC 23270:2006 C# (2.0).
<code>ISO-1</code> or <code>1</code>	The compiler accepts only syntax that is included in ISO/IEC 23270:2003 C# (1.0/1.2).

You can learn more about the language versions supported for each framework version in the article on [Configure language version](#) in the language reference section.

## Avoid the updated feature

If you must support older libraries or runtimes, you may need to avoid using newer features.

## Breaking changes on struct initialization

All these errors and warnings help ensure that `struct` types are properly initialized before their fields are accessed. In earlier versions of C#, you must explicitly assign all fields in a struct in any constructor. The parameterless constructor initializes all fields to their default value. In later versions, all constructors initialize all fields. Either the field is explicitly set, set in a field initializer, or set to its default value.

- **CS0171, CS8881:** *Backing field for automatically implemented property 'name' must be fully assigned before control is returned to the caller.*
- **CS0188, CS8885:** *The 'this' object cannot be used before all of its fields are assigned to*
- **CS0843, CS8880:** *Backing field for automatically implemented property 'name' must be fully assigned before control is returned to the caller*

You can address this error by upgrading your language version to C# 11, when all fields are initialized by every `struct` constructor. If that's not a possible option, you must explicitly call the default constructor, as shown in the following example:

```
C#  
  
struct S  
{  
    public int AIProp { get; set; }  
    public S(int i){} //CS0843  
    // Try the following lines instead.  
    // public S(int i) : this()  
    // {  
    //     AIProp = i;  
    // }  
}  
  
class Test  
{  
    static int Main()  
    {  
        return 1;  
    }  
}
```

```
    }  
}
```

# Resolve nullable warnings

Article • 03/14/2023 • 17 minutes to read

This article covers the following compiler warnings:

- [CS8597](#) - *Thrown value may be null.*
- [CS8600](#) - *Converting null literal or possible null value to non-nullable type.*
- [CS8601](#) - *Possible null reference assignment.*
- [CS8602](#) - *Dereference of a possibly null reference.*
- [CS8603](#) - *Possible null reference return.*
- [CS8604](#) - *Possible null reference argument for parameter.*
- [CS8605](#) - *Unboxing a possibly null value.*
- [CS8607](#) - *A possible null value may not be used for a type marked with [NotNull] or [DisallowNull]*
- [CS8608](#) - *Nullability of reference types in type doesn't match overridden member.*
- [CS8609](#) - *Nullability of reference types in return type doesn't match overridden member.*
- [CS8610](#) - *Nullability of reference types in type parameter doesn't match overridden member.*
- [CS8611](#) - *Nullability of reference types in type parameter doesn't match partial method declaration.*
- [CS8612](#) - *Nullability of reference types in type doesn't match implicitly implemented member.*
- [CS8613](#) - *Nullability of reference types in return type doesn't match implicitly implemented member.*
- [CS8614](#) - *Nullability of reference types in type of parameter doesn't match implicitly implemented member.*
- [CS8615](#) - *Nullability of reference types in type doesn't match implemented member.*
- [CS8616](#) - *Nullability of reference types in return type doesn't match implemented member.*
- [CS8617](#) - *Nullability of reference types in type of parameter doesn't match implemented member.*
- [CS8618](#) - *Non-nullable variable must contain a non-null value when exiting constructor. Consider declaring it as nullable.*
- [CS8619](#) - *Nullability of reference types in value doesn't match target type.*
- [CS8620](#) - *Argument cannot be used for parameter due to differences in the nullability of reference types.*
- [CS8621](#) - *Nullability of reference types in return type doesn't match the target delegate (possibly because of nullability attributes).*

- **CS8622** - Nullability of reference types in type of parameter doesn't match the target delegate (possibly because of nullability attributes).
- **CS8624** - Argument cannot be used as an output due to differences in the nullability of reference types.
- **CS8625** - Cannot convert null literal to non-nullable reference type.
- **CS8629** - Nullable value type may be null.
- **CS8631** - The type cannot be used as type parameter in the generic type or method. Nullability of type argument doesn't match constraint type.
- **CS8633** - Nullability in constraints for type parameter of method doesn't match the constraints for type parameter of interface method. Consider using an explicit interface implementation instead.
- **CS8634** - The type cannot be used as type parameter in the generic type or method. Nullability of type argument doesn't match 'class' constraint.
- **CS8643** - Nullability of reference types in explicit interface specifier doesn't match interface implemented by the type.
- **CS8644** - Type does not implement interface member. Nullability of reference types in interface implemented by the base type doesn't match.
- **CS8645** - Member is already listed in the interface list on type with different nullability of reference types.
- **CS8655** - The switch expression does not handle some null inputs (it is not exhaustive).
- **CS8667** - Partial method declarations have inconsistent nullability in constraints for type parameter.
- **CS8670** - Object or collection initializer implicitly dereferences possibly null member.
- **CS8714** - The type cannot be used as type parameter in the generic type or method. Nullability of type argument doesn't match 'notnull' constraint.
- **CS8762** - Parameter must have a non-null value when exiting.
- **CS8763** - A method marked `[DoesNotReturn]` should not return.
- **CS8764** - Nullability of return type doesn't match overridden member (possibly because of nullability attributes).
- **CS8765** - Nullability of type of parameter doesn't match overridden member (possibly because of nullability attributes).
- **CS8766** - Nullability of reference types in return type of doesn't match implicitly implemented member (possibly because of nullability attributes).
- **CS8767** - Nullability of reference types in type of parameter of doesn't match implicitly implemented member (possibly because of nullability attributes).
- **CS8768** - Nullability of reference types in return type doesn't match implemented member (possibly because of nullability attributes).
- **CS8769** - Nullability of reference types in type of parameter doesn't match implemented member (possibly because of nullability attributes).

- [CS8770](#) - Method lacks `[DoesNotReturn]` annotation to match implemented or overridden member.
- [CS8774](#) - Member must have a non-null value when exiting.
- [CS8776](#) - Member cannot be used in this attribute.
- [CS8775](#) - Member must have a non-null value when exiting.
- [CS8777](#) - Parameter must have a non-null value when exiting.
- [CS8819](#) - Nullability of reference types in return type doesn't match partial method declaration.
- [CS8824](#) - Parameter must have a non-null value when exiting because parameter is non-null.
- [CS8825](#) - Return value must be non-null because parameter is non-null.
- [CS8847](#) - The switch expression does not handle some null inputs (it is not exhaustive). However, a pattern with a 'when' clause might successfully match this value.

The purpose of nullable warnings is to minimize the chance that your application throws a [System.NullReferenceException](#) when run. To achieve this goal, the compiler uses static analysis and issues warnings when your code has constructs that may lead to null reference exceptions. You provide the compiler with information for its static analysis by applying type annotations and attributes. These annotations and attributes describe the nullability of arguments, parameters, and members of your types. In this article, you'll learn different techniques to address the nullable warnings the compiler generates from its static analysis. The techniques described here are for general C# code. Learn to work with nullable reference types and Entity Framework core in [Working with nullable reference types](#).

You'll address almost all warnings using one of four techniques:

- Adding necessary null checks.
- Adding `?!` nullable annotations.
- Adding attributes that describe null semantics.
- Initializing variables correctly.

## Possible dereference of null

This set of warnings alerts you that you're dereferencing a variable whose *null-state* is *maybe-null*. These warnings are:

- [CS8602](#) - Dereference of a possibly null reference.
- [CS8670](#) - Object or collection initializer implicitly dereferences possibly null member.

The following code demonstrates one example of each of the preceding warnings:

C#

```
class Container
{
    public List<string>? States { get; set; }

internal void PossibleDereferenceNullExamples(string? message)
{
    Console.WriteLine(message.Length); // CS8602

    var c = new Container { States = { "Red", "Yellow", "Green" } }; // CS8670
}
```

In the example above, the warning is because the `Container`, `c`, may have a null value for the `States` property. Assigning new states to a collection that might be null causes the warning.

To remove these warnings, you need to add code to change that variable's *null-state* to *not-null* before dereferencing it. The collection initializer warning may be harder to spot. The compiler detects that the collection *maybe-null* when the initializer adds elements to it.

In many instances, you can fix these warnings by checking that a variable isn't null before dereferencing it. For example, the above example could be rewritten as:

C#

```
void WriteMessageLength(string? message)
{
    if (message is not null)
    {
        Console.WriteLine(message.Length);
    }
}
```

Other instances when you get these warnings may be false positive. You may have a private utility method that tests for null. The compiler doesn't know that the method provides a null check. Consider the following example that uses a private utility method, `IsNotNull`:

C#

```
public void WriteMessage(string? message)
{
    if (IsNotNull(message))
```

```
        Console.WriteLine(message.Length);  
    }
```

The compiler warns that you may be dereferencing null when you write the property `message.Length` because its static analysis determines that `message` may be `null`. You may know that `IsNotNull` provides a null check, and when it returns `true`, the *null-state* of `message` should be *not-null*. You must tell the compiler those facts. One way is to use the null forgiving operator, `!`. You can change the `WriteLine` statement to match the following code:

C#

```
Console.WriteLine(message!.Length);
```

The null forgiving operator makes the expression *not-null* even if it was *maybe-null* without the `!` applied. In this example, a better solution is to add an attribute to the signature of `IsNotNull`:

C#

```
private static bool IsNotNull([NotNullWhen(true)] object? obj) => obj != null;
```

The [System.Diagnostics.CodeAnalysis.NotNullWhenAttribute](#) informs the compiler that the argument used for the `obj` parameter is *not-null* when the method returns `true`. When the method returns `false`, the argument has the same *null-state* it had before the method was called.

### 💡 Tip

There's a rich set of attributes you can use to describe how your methods and properties affect *null-state*. You can learn about them in the language reference article on [Nullable static analysis attributes](#).

Fixing a warning for dereferencing a *maybe-null* variable involves one of three techniques:

- Add a missing null check.
- Add null analysis attributes on APIs to affect the compiler's *null-state* static analysis. These attributes inform the compiler when a return value or argument should be *maybe-null* or *not-null* after calling the method.

- Apply the null forgiving operator `!` to the expression to force the state to *not-null*.

## Possible null assigned to a nonnullable reference

This set of warnings alerts you that you're assigning a variable whose type is nonnullable to an expression whose *null-state* is *maybe-null*. These warnings are:

- **CS8597** - *Thrown value may be null.*
- **CS8600** - *Converting null literal or possible null value to non-nullable type.*
- **CS8601** - *Possible null reference assignment.*
- **CS8603** - *Possible null reference return.*
- **CS8604** - *Possible null reference argument for parameter.*
- **CS8605** - *Unboxing a possibly null value.*
- **CS8625** - *Cannot convert null literal to non-nullable reference type.*
- **CS8629** - *Nullable value type may be null.*

The compiler emits these warnings when you attempt to assign an expression that is *maybe-null* to a variable that is nonnullable. For example:

```
C#  
  
string? TryGetMessage(int id) => "";  
  
string msg = TryGetMessage(42); // Possible null assignment.
```

The different warnings indicate provide details about the code, such as assignment, unboxing assignment, return statements, arguments to methods, and throw expressions.

You can take one of three actions to address these warnings. One is to add the `?` annotation to make the variable a nullable reference type. That change may cause other warnings. Changing a variable from a non-nullable reference to a nullable reference changes its default *null-state* from *not-null* to *maybe-null*. The compiler's static analysis may find instances where you dereference a variable that is *maybe-null*.

The other actions instruct the compiler that the right-hand-side of the assignment is *not-null*. The expression on the right-hand-side could be null-checked before assignment, as shown in the following example:

```
C#  
  
string notNullMsg = TryGetMessage(42) ?? "Unknown message id: 42";
```

The previous examples demonstrate assignment of the return value of a method. You may annotate the method (or property) to indicate when a method returns a not-null value. The [System.Diagnostics.CodeAnalysis.NotNullIfNotNullAttribute](#) often specifies that a return value is *not-null* when an input argument is *not-null*. Another alternative is to add the null forgiving operator, `!` to the right-hand side:

```
C#
```

```
string msg = TryGetMessage(42)!;
```

Fixing a warning for assigning a *maybe-null* expression to a *not-null* variable involves one of four techniques:

- Change the left side of the assignment to a nullable type. This action may introduce new warnings when you dereference that variable.
- Provide a null-check before the assignment.
- Annotate the API that produces the right-hand side of the assignment.
- Add the null forgiving operator to the right-hand side of the assignment.

## Nonnullable reference not initialized

This set of warnings alerts you that you're assigning a variable whose type is non-nullable to an expression whose *null-state* is *maybe-null*. These warnings are:

- **CS8618** - *Non-nullable variable must contain a non-null value when exiting constructor. Consider declaring it as nullable.*
- **CS8762** - *Parameter must have a non-null value when exiting.*

Consider the following class as an example:

```
C#
```

```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

Neither `FirstName` nor `LastName` are guaranteed initialized. If this code is new, consider changing the public interface. The above example could be updated as follows:

```
C#
```

```
public class Person
{
    public Person(string first, string last)
    {
        FirstName = first;
        LastName = last;
    }

    public string FirstName { get; set; }
    public string LastName { get; set; }
}
```

If you require creating a `Person` object before setting the name, you can initialize the properties using a default non-null value:

C#

```
public class Person
{
    public string FirstName { get; set; } = string.Empty;
    public string LastName { get; set; } = string.Empty;
}
```

Another alternative may be to change those members to nullable reference types. The `Person` class could be defined as follows if `null` should be allowed for the name:

C#

```
public class Person
{
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
}
```

Existing code may require other changes to inform the compiler about the null semantics for those members. You may have created multiple constructors, and your class may have a private helper method that initializes one or more members. You can move the initialization code into a single constructor and ensure all constructors call the one with the common initialization code. Or, you can use the [System.Diagnostics.CodeAnalysis.MemberNotNullAttribute](#) and [System.Diagnostics.CodeAnalysis.MemberNotNullWhenAttribute](#) attributes. These attributes inform the compiler that a member is *not-null* after the method has been called. The following code shows an example of each. The `Person` class uses a common constructor called by all other constructors. The `Student` class has a helper method annotated with the [System.Diagnostics.CodeAnalysis.MemberNotNullAttribute](#) attribute:

C#

```
using System.Diagnostics.CodeAnalysis;

public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public Person(string firstName, string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }

    public Person() : this("John", "Doe") { }
}

public class Student : Person
{
    public string Major { get; set; }

    public Student(string firstName, string lastName, string major)
        : base(firstName, lastName)
    {
        SetMajor(major);
    }

    public Student(string firstName, string lastName) :
        base(firstName, lastName)
    {
        SetMajor();
    }

    public Student()
    {
        SetMajor();
    }

    [MemberNotNull(nameof(Major))]
    private void SetMajor(string? major = default)
    {
        Major = major ?? "Undeclared";
    }
}
```

Finally, you can use the null forgiving operator to indicate that a member is initialized in other code. For another example, consider the following classes representing an Entity Framework Core model:

C#

```

public class TodoItem
{
    public long Id { get; set; }
    public string? Name { get; set; }
    public bool IsComplete { get; set; }
}

public class TodoContext : DbContext
{
    public TodoContext(DbContextOptions<TodoContext> options)
        : base(options)
    {
    }

    public DbSet<TodoItem> TodoItems { get; set; } = null!;
}

```

The `DbSet` property is initialized to `null!`. That tells the compiler that the property is set to a *not-null* value. In fact, the base `DbContext` performs the initialization of the set. The compiler's static analysis doesn't pick that up. For more information on working with nullable reference types and Entity Framework Core, see the article on [Working with Nullable Reference Types in EF Core](#).

Fixing a warning for not initializing a nonnullable member involves one of four techniques:

- Change the constructors or field initializers to ensure all nonnullable members are initialized.
- Change one or more members to be nullable types.
- Annotate any helper methods to indicate which members are assigned.
- Add an initializer to `null!` to indicate that the member is initialized in other code.

## Mismatch in nullability declaration

Many warnings indicate nullability mismatches between signatures for methods, delegates, or type parameters.

- **CS8608** - *Nullability of reference types in type doesn't match overridden member.*
- **CS8609** - *Nullability of reference types in return type doesn't match overridden member.*
- **CS8610** - *Nullability of reference types in type parameter doesn't match overridden member.*
- **CS8611** - *Nullability of reference types in type parameter doesn't match partial method declaration.*

- CS8612 - Nullability of reference types in type doesn't match implicitly implemented member.
- CS8613 - Nullability of reference types in return type doesn't match implicitly implemented member.
- CS8614 - Nullability of reference types in type of parameter doesn't match implicitly implemented member.
- CS8615 - Nullability of reference types in type doesn't match implemented member.
- CS8616 - Nullability of reference types in return type doesn't match implemented member.
- CS8617 - Nullability of reference types in type of parameter doesn't match implemented member.
- CS8619 - Nullability of reference types in value doesn't match target type.
- CS8620 - Argument cannot be used for parameter due to differences in the nullability of reference types.
- CS8621 - Nullability of reference types in return type doesn't match the target delegate (possibly because of nullability attributes).
- CS8622 - Nullability of reference types in type of parameter doesn't match the target delegate (possibly because of nullability attributes).
- CS8624 - Argument cannot be used as an output due to differences in the nullability of reference types.
- CS8631 - The type cannot be used as type parameter in the generic type or method. Nullability of type argument doesn't match constraint type.
- CS8633 - Nullability in constraints for type parameter of method doesn't match the constraints for type parameter of interface method. Consider using an explicit interface implementation instead.
- CS8634 - The type cannot be used as type parameter in the generic type or method. Nullability of type argument doesn't match 'class' constraint.
- CS8643 - Nullability of reference types in explicit interface specifier doesn't match interface implemented by the type.
- CS8644 - Type does not implement interface member. Nullability of reference types in interface implemented by the base type doesn't match.
- CS8645 - Member is already listed in the interface list on type with different nullability of reference types.
- CS8667 - Partial method declarations have inconsistent nullability in constraints for type parameter.
- CS8714 - The type cannot be used as type parameter in the generic type or method. Nullability of type argument doesn't match 'notnull' constraint.
- CS8764 - Nullability of return type doesn't match overridden member (possibly because of nullability attributes).

- CS8765 - Nullability of type of parameter doesn't match overridden member (possibly because of nullability attributes).
- CS8766 - Nullability of reference types in return type of doesn't match implicitly implemented member (possibly because of nullability attributes).
- CS8767 - Nullability of reference types in type of parameter of doesn't match implicitly implemented member (possibly because of nullability attributes).
- CS8768 - Nullability of reference types in return type doesn't match implemented member (possibly because of nullability attributes).
- CS8769 - Nullability of reference types in type of parameter doesn't match implemented member (possibly because of nullability attributes).
- CS8819 - Nullability of reference types in return type doesn't match partial method declaration.

The following code demonstrates CS8764:

C#

```
public class B
{
    public virtual string GetMessage(string id) => string.Empty;
}
public class D : B
{
    public override string? GetMessage(string? id) => default;
}
```

The preceding example shows a `virtual` method in a base class and an `override` with different nullability. The base class returns a non-nullable string, but the derived class returns a nullable string. If the `string` and `string?` are reversed, it would be allowed because the derived class is more restrictive. Similarly, parameter declarations should match. Parameters in the `override` method can allow null even when the base class doesn't.

Other situations can generate these warnings. You may have a mismatch in an interface method declaration and the implementation of that method. Or a delegate type and the expression for that delegate may differ. A type parameter and the type argument may differ in nullability.

To fix these warnings, update the appropriate declaration.

## Code doesn't match attribute declaration

The preceding sections have discussed how you can use [Attributes for nullable static analysis](#) to inform the compiler about the null semantics of your code. The compiler warns you if the code doesn't adhere to the promises of that attribute:

- **CS8607** - *A possible null value may not be used for a type marked with `[NotNull]` or `[DisallowNull]`*
- **CS8763** - *A method marked `[DoesNotReturn]` should not return.*
- **CS8770** - *Method lacks `[DoesNotReturn]` annotation to match implemented or overridden member.*
- **CS8774** - *Member must have a non-null value when exiting.*
- **CS8775** - *Member must have a non-null value when exiting.*
- **CS8776** - *Member cannot be used in this attribute.*
- **CS8777** - *Parameter must have a non-null value when exiting.*
- **CS8824** - *Parameter must have a non-null value when exiting because parameter is non-null.*
- **CS8825** - *Return value must be non-null because parameter is non-null.*

Consider the following method:

C#

```
public bool TryGetMessage(int id, [NotNullWhen(true)] out string? message)
{
    message = null;
    return true;
}
```

The compiler produces a warning because the `message` parameter is assigned `null` and the method returns `true`. The `NotNullWhen` attribute indicates that shouldn't happen.

To address these warnings, update your code so it matches the expectations of the attributes you've applied. You may change the attributes, or the algorithm.

## Exhaustive switch expression

Switch expressions must be *exhaustive*, meaning that all input values must be handled. Even for non-nullable reference types, the `null` value must be accounted for. The compiler issues warnings when the null value isn't handled:

- **CS8655** - *The switch expression does not handle some null inputs (it is not exhaustive).*

- CS8847 - *The switch expression does not handle some null inputs (it is not exhaustive). However, a pattern with a 'when' clause might successfully match this value.*

The following example code demonstrates this condition:

C#

```
int AsScale(string status) =>
    status switch
    {
        "Red" => 0,
        "Yellow" => 5,
        "Green" => 10,
        {} => -1
    };
}
```

The input expression is a `string`, not a `string?`. The compiler still generates this warning. The `{ }` pattern handles all non-null values, but doesn't match `null`. To address these errors, you can either add an explicit `null` case, or replace the `{ }` with the `_` (discard) pattern. The discard pattern matches null as well as any other value.

# Pattern matching warnings

Article • 11/15/2022 • 2 minutes to read

There are several pattern matching warnings. Learn how to address these warnings.

- **CS8509** - *The switch expression does not handle all possible values of its input type (it is not exhaustive). For example, the pattern '...' is not covered.*

The following code snippets generate CS8509:

```
C#  
  
// CS8509.cs  
enum EnumValues  
{  
    Value1,  
    Value2,  
    Value3  
}  
  
void Method(EnumValues enumValues)  
{  
    var result = enumValues switch  
    {  
        EnumValues.Value1 => 1,  
        EnumValues.Value2 => 2,  
    };  
}
```

To address this warning, add switch arms that cover all possible input values. For example:

```
C#  
  
enum EnumValues  
{  
    Value1,  
    Value2,  
    Value3  
}  
  
void Method(EnumValues enumValues)  
{  
    var result = enumValues switch  
    {  
        EnumValues.Value1 => 1,  
        EnumValues.Value2 => 2,  
        EnumValues.Value3 => 3,  
        _ => throw new ArgumentException("Input isn't a valid enum value",  
    });  
}
```

```
nameof(enumValues)),  
    };  
}
```

The `_` pattern matches all remaining values. It's often used to match invalid values, as shown in the preceding example.

For more information, see [Switch](#).

# C# Warning waves

Article • 09/29/2022 • 7 minutes to read

New warnings and errors may be introduced in each release of the C# compiler. When new warnings could be reported on existing code, those warnings are introduced under an opt-in system referred to as a *warning wave*. The opt-in system means that you shouldn't see new warnings on existing code without taking action to enable them.

Warning waves are enabled using the [AnalysisLevel](#) element in your project file. When `<TreatWarningsAsErrors>true</TreatWarningsAsErrors>` is specified, enabled warning wave warnings generate errors. Warning wave 5 diagnostics were added in C# 9. Warning wave 6 diagnostics were added in C# 10. Warning wave 7 diagnostics were added in C# 11.

## CS8981 - The type name only contains lower-cased ascii characters.

*Warning wave 7*

Any new keywords added for C# will be all lower-case ASCII characters. This warning ensures that none of your types conflict with future keywords. The following code produces CS8981:

```
C#  
  
public class lowercasename  
{  
}
```

You can address this warning by renaming the type to include at least one non-lower case ASCII character, such as an upper case character, a digit, or an underscore.

## CS8826 - Partial method declarations have signature differences.

*Warning wave 6*

This warning corrects some inconsistencies in reporting differences between partial method signatures. The compiler always reported an error when the partial method signatures created different CLR signatures. Now, the compiler reports CS8826 when the signatures are syntactically different C#. Consider the following partial class:

C#

```
public partial class PartialType
{
    public partial void M1(int x);

    public partial T M2<T>(string s) where T : struct;

    public partial void M3(string s);

    public partial void M4(object o);
    public partial void M5(dynamic o);
    public partial void M6(string? s);
}
```

The following partial class implementation generates several examples of CS8626:

C#

```
public partial class PartialType
{
    // Different parameter names:
    public partial void M1(int y) { }

    // Different type parameter names:
    public partial TResult M2<TResult>(string s) where TResult : struct =>
default;

    // Relaxed nullability
    public partial void M3(string? s) { }

    // Mixing object and dynamic
    public partial void M4(dynamic o) { }

    // Mixing object and dynamic
    public partial void M5(object o) { }

    // Note: This generates CS8611 (nullability mismatch) not CS8826
    public partial void M6(string s) { }
}
```

### ⓘ Note

If the implementation of a method uses a non-nullable reference type when the other declaration accepts nullable reference types, CS8611 is generated instead of CS8826.

To fix any instance of these warnings, ensure the two signatures match.

## CS7023 - A static type is used in an 'is' or 'as' expression.

*Warning wave 5*

The `is` and `as` expressions always return `false` for a static type because you can't create instances of a static type. The following code produces CS7023:

```
C#  
  
static class StaticClass  
{  
    public static void Thing() {}  
}  
  
void M(object o)  
{  
    // warning: cannot use a static type in 'is' or 'as'  
    if (o is StaticClass)  
    {  
        Console.WriteLine("Can't happen");  
    }  
    else  
    {  
        Console.WriteLine("o is not an instance of a static class");  
    }  
}
```

The compiler reports this warning because the type test can never succeed. To correct this warning, remove the test and remove any code executed only if the test succeeded. In the preceding example, the `else` clause is always executed. The method body could be replaced by that single line:

```
C#  
  
Console.WriteLine("o is not an instance of a static class");
```

## CS8073 - The result of the expression is always 'false' (or 'true').

*Warning wave 5*

The `==` and `!=` operators always return `false` (or `true`) when comparing an instance of a `struct` type to `null`. The following code demonstrates this warning. Assume `S` is a `struct` that has defined `operator ==` and `operator !=`:

C#

```
class Program
{
    public static void M(S s)
    {
        if (s == null) { } // CS8073: The result of the expression is always
        'false'
        if (s != null) { } // CS8073: The result of the expression is always
        'true'
    }
}

struct S
{
    public static bool operator ==(S s1, S s2) => s1.Equals(s2);
    public static bool operator !=(S s1, S s2) => !s1.Equals(s2);
    public override bool Equals(object? other)
    {
        // Implementation elided
        return false;
    }
    public override int GetHashCode() => 0;

    // Other details elided...
}
```

To fix this error, remove the null check and code that would execute if the object is `null`.

## CS8848 - Operator 'from' can't be used here due to precedence. Use parentheses to disambiguate.

*Warning wave 5*

The following examples demonstrate this warning. The expression binds incorrectly because of the precedence of the operators.

C#

```
bool b = true;
var source = new Src();
b = true;
```

```
source = new Src();
var a = b && from c in source select c;
Console.WriteLine(a);

var indexes = new Src2();
int[] array = { 1, 2, 3, 4, 5, 6, 7 };
var range = array[0..from c in indexes select c];
```

To fix this error, put parentheses around the query expression:

C#

```
bool b = true;
var source = new Src();
b = true;
source = new Src();
var a = b && (from c in source select c);
Console.WriteLine(a);

var indexes = new Src2();
int[] array = { 1, 2, 3, 4, 5, 6, 7 };
var range = array[0..(from c in indexes select c)];
```

## Members must be fully assigned, use of unassigned variable (CS8880, CS8881, CS8882, CS8883, CS8884, CS8885, CS8886, CS8887)

*Warning wave 5*

Several warnings improve the definite assignment analysis for `struct` types declared in imported assemblies. All these new warnings are generated when a struct in an imported assembly includes an inaccessible field (usually a `private` field) of a reference type, as shown in the following example:

C#

```
public struct Struct
{
    private string data = String.Empty;
    public Struct() { }
}
```

The following examples show the warnings generated from the improved definite assignment analysis:

- CS8880: Auto-implemented property 'Property' must be fully assigned before control is returned to the caller.
- CS8881: Field 'field' must be fully assigned before control is returned to the caller.
- CS8882: The out parameter 'parameter' must be assigned to before control leaves the current method.
- CS8883: Use of possibly unassigned auto-implemented property 'Property'.
- CS8884: Use of possibly unassigned field 'Field'
- CS8885: The 'this' object can't be used before all its fields have been assigned.
- CS8886: Use of unassigned output parameter 'parameterName'.
- CS8887: Use of unassigned local variable 'variableName'

C#

```
public struct DefiniteAssignmentWarnings
{
    // CS8880
    public Struct Property { get; }

    // CS8881
    private Struct field;

    // CS8882
    public void Method(out Struct s)
    {

    }

    public DefiniteAssignmentWarnings(int dummy)
    {
        // CS8883
        Struct v2 = Property;
        // CS8884
        Struct v3 = field;
        // CS8885:
        DefiniteAssignmentWarnings p2 = this;
    }

    public static void Method2(out Struct s1)
    {
        // CS8886
        var s2 = s1;
        s1 = default;
    }

    public static void UseLocalStruct()
    {
        Struct r1;
        var r2 = r1;
    }
}
```

You can fix any of these warnings by initializing or assigning the imported struct to its default value:

```
C#  
  
public struct DefiniteAssignmentNoWarnings  
{  
    // CS8880  
    public Struct Property { get; } = default;  
    // CS8881  
    private Struct field = default;  
  
    // CS8882  
    public void Method(out Struct s)  
    {  
        s = default;  
    }  
  
    public DefiniteAssignmentNoWarnings(int dummy)  
    {  
        // CS8883  
        Struct v2 = Property;  
        // CS8884  
        Struct v3 = field;  
        // CS8885:  
        DefiniteAssignmentNoWarnings p2 = this;  
    }  
  
    public static void Method2(out Struct s1)  
    {  
        // CS8886  
        s1 = default;  
        var s2 = s1;  
    }  
  
    public static void UseLocalStruct()  
    {  
        Struct r1 = default;  
        var r2 = r1;  
    }  
}
```

**CS8892 - Method will not be used as an entry point because a synchronous entry point 'method' was found.**

*Warning wave 5*

This warning is generated on all async entry point candidates when you have multiple valid entry points, including one or more synchronous entry point.

The following example generates CS8892:

```
C#  
  
public static void Main()  
{  
    RunProgram();  
}  
  
// CS8892  
public static async Task Main(string[] args)  
{  
    await RunProgramAsync();  
}
```

#### ⓘ Note

The compiler always uses the synchronous entry point. In case there are multiple synchronous entry points, you get a compiler error.

To fix this warning, remove or rename the asynchronous entry point.

## CS8897 - Static types can't be used as parameters

*Warning wave 5*

Members of an interface can't declare parameters whose type is a static class. The following code demonstrates both CS8897 and CS8898:

```
C#  
  
public static class Utilities  
{  
    // elided  
}  
  
public interface IUtility  
{  
    // CS8897  
    public void SetUtility(Utilities u);  
  
    // CS8898
```

```
    public Utilities GetUtility();
}
```

To fix this warning, change the parameter type or remove the method.

## CS8898 - static types can't be used as return types

*Warning wave 5*

Members of an interface can't declare a return type that is a static class. The following code demonstrates both CS8897 and CS8898:

```
C#  
  
public static class Utilities  
{  
    // elided  
}  
  
public interface IUtility  
{  
    // CS8897  
    public void SetUtility(Utilities u);  
  
    // CS8898  
    public Utilities GetUtility();  
}
```

To fix this warning, change the return type or remove the method.