throw (C# Reference)

03/02/2015 • 3 minutes to read • +8

In this article

Signals the occurrence of an exception during program execution.

Remarks

The syntax of `throw` is:

C# Copy

```
throw [e];
```

where `e` is an instance of a class derived from System.Exception. The following example uses the `throw` statement to throw an IndexOutOfRangeException if the argument passed to a method named `GetNumber` does not correspond to a valid index of an internal array.

C# Copy

```
using System;

namespace Throw2
{
public class NumberGenerator
{
   int[] numbers = { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 };

   public int GetNumber(int index)
   {
      if (index < 0 || index >= numbers.Length)
      {
         throw new IndexOutOfRangeException();
      }
      return numbers[index];
```

```
    }
}
```

Method callers then use a `try-catch` or `try-catch-finally` block to handle the thrown exception. The following example handles the exception thrown by the `GetNumber` method.

C# Copy

```
using System;

public class Example
{
   public static void Main()
   {
      var gen = new NumberGenerator();
      int index = 10;
      try
      {
          int value = gen.GetNumber(index);
          Console.WriteLine($"Retrieved {value}");
      }
      catch (IndexOutOfRangeException e)
      {
         Console.WriteLine($"{e.GetType().Name}: {index} is outside the
bounds of the array");
      }
   }
}
// The example displays the following output:
//        IndexOutOfRangeException: 10 is outside the bounds of the array
```

Re-throwing an exception

`throw` can also be used in a `catch` block to re-throw an exception handled in a `catch` block. In this case, `throw` does not take an exception operand. It is most useful when a method passes on an argument from a caller to some other library method, and the library method throws an exception that must be passed on to the caller. For example, the following example re-throws an NullReferenceException that is thrown when attempting to retrieve the first character of an uninitialized string.

C# Copy

```
using System;

namespace Throw
{
public class Sentence
{
```

```
    public Sentence(string s)
    {
        Value = s;
    }

    public string Value { get; set; }

    public char GetFirstCharacter()
    {
        try
        {
            return Value[0];
        }
        catch (NullReferenceException e)
        {
            throw;
        }
    }
}

public class Example
{
    public static void Main()
    {
        var s = new Sentence(null);
        Console.WriteLine($"The first character is {s.GetFirstCharacter()}");
    }
}
// The example displays the following output:
//    Unhandled Exception: System.NullReferenceException: Object reference
not set to an instance of an object.
//       at Sentence.GetFirstCharacter()
//       at Example.Main()
```

> **Important**
>
> You can also use the `throw e` syntax in a `catch` block to instantiate a new exception that you pass on to the caller. In this case, the stack trace of the original exception, which is available from the **StackTrace** property, is not preserved.

The `throw` expression

Starting with C# 7.0, `throw` can be used as an expression as well as a statement. This allows an exception to be thrown in contexts that were previously unsupported. These include:

- the conditional operator. The following example uses a `throw` expression to throw an ArgumentException if a method is passed an empty string array. Before C# 7.0, this logic would need to appear in an `if/else` statement.

C# Copy

```
private static void DisplayFirstNumber(string[] args)
{
   string arg = args.Length >= 1 ? args[0] :
                                throw new ArgumentException("You must
supply an argument");
   if (Int64.TryParse(arg, out var number))
      Console.WriteLine($"You entered {number:F0}");
   else
      Console.WriteLine($"{arg} is not a number.");
}
```

- the null-coalescing operator. In the following example, a `throw` expression is used with a null-coalescing operator to throw an exception if the string assigned to a `Name` property is `null`.

C# Copy

```
public string Name
{
    get => name;
    set => name = value ??
        throw new ArgumentNullException(paramName: nameof(value), mes-
sage: "Name cannot be null");
}
```

- an expression-bodied lambda or method. The following example illustrates an expression-bodied method that throws an InvalidCastException because a conversion to a DateTime value is not supported.

C# Copy

```
DateTime ToDateTime(IFormatProvider provider) =>
         throw new InvalidCastException("Conversion to a DateTime is
not supported.");
```

C# language specification

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

See also

- C# Reference
- C# Programming Guide

- try-catch
- C# Keywords
- How to: Explicitly Throw Exceptions

Is this page helpful?

Yes No