

# Creating and Throwing Exceptions

05/14/2021 • 3 minutes to read • 

## In this article

[Things to Avoid When Throwing Exceptions](#)

[Defining Exception Classes](#)

[C# Language Specification](#)

[See also](#)

Exceptions are used to indicate that an error has occurred while running the program. Exception objects that describe an error are created and then *thrown* with the [throw](#) keyword. The runtime then searches for the most compatible exception handler.

Programmers should throw exceptions when one or more of the following conditions are true:

- The method can't complete its defined functionality. For example, if a parameter to a method has an invalid value:

C#

 Copy

```
static void CopyObject(SampleClass original)
{
    _ = original ?? throw new ArgumentException("Parameter cannot be
null", nameof(original));
}
```

- An inappropriate call to an object is made, based on the object state. One example might be trying to write to a read-only file. In cases where an object state doesn't allow an operation, throw an instance of [InvalidOperationException](#) or an object based on a derivation of this class. The following code is an example of a method that throws an [InvalidOperationException](#) object:

C#

 Copy

```
public class ProgramLog
{
    FileStream logFile = null!;
    public void OpenLog(FileInfo fileName, FileMode mode) { }


    public void WriteLog()
    {
        if (!logFile.CanWrite)
        {
```

```

        throw new InvalidOperationException("Logfile cannot be
read-only");
    }
    // Else write data to the log and return.
}
}

```

- When an argument to a method causes an exception. In this case, the original exception should be caught and an [ArgumentException](#) instance should be created. The original exception should be passed to the constructor of the [ArgumentException](#) as the [InnerException](#) parameter:

C#	 Copy
<pre> static int GetValueFromArray(int[] array, int index) {     try     {         return array[index];     }     catch (IndexOutOfRangeException ex)     {         throw new ArgumentException("Index is out of range", nameof(index), ex);     } } </pre>	

Exceptions contain a property named [StackTrace](#). This string contains the name of the methods on the current call stack, together with the file name and line number where the exception was thrown for each method. A [StackTrace](#) object is created automatically by the common language runtime (CLR) from the point of the `throw` statement, so that exceptions must be thrown from the point where the stack trace should begin.

All exceptions contain a property named [Message](#). This string should be set to explain the reason for the exception. Information that is sensitive to security shouldn't be put in the message text. In addition to [Message](#), [ArgumentException](#) contains a property named [ParamName](#) that should be set to the name of the argument that caused the exception to be thrown. In a property setter, [ParamName](#) should be set to `value`.

Public and protected methods throw exceptions whenever they can't complete their intended functions. The exception class thrown is the most specific exception available that fits the error conditions. These exceptions should be documented as part of the class functionality, and derived classes or updates to the original class should retain the same behavior for backward compatibility.


## Things to Avoid When Throwing Exceptions

The following list identifies practices to avoid when throwing exceptions:

- Don't use exceptions to change the flow of a program as part of ordinary execution. Use exceptions to report and handle error conditions.
- Exceptions shouldn't be returned as a return value or parameter instead of being thrown.
- Don't throw [System.Exception](#), [System.SystemException](#), [System.NullReferenceException](#), or [System.IndexOutOfRangeException](#) intentionally from your own source code.
- Don't create exceptions that can be thrown in debug mode but not release mode. To identify run-time errors during the development phase, use `Debug.Assert` instead.

## Defining Exception Classes

Programs can throw a predefined exception class in the [System](#) namespace (except where previously noted), or create their own exception classes by deriving from [Exception](#). The derived classes should define at least four constructors: one parameterless constructor, one that sets the message property, and one that sets both the [Message](#) and [InnerException](#) properties. The fourth constructor is used to serialize the exception. New exception classes should be serializable. For example:

C#	 Copy
<pre>[Serializable] public class InvalidDepartmentException : Exception {     public InvalidDepartmentException() : base() { }     public InvalidDepartmentException(string message) : base(message) { }     public InvalidDepartmentException(string message, Exception inner) : base(message, inner) { }      // A constructor is needed for serialization when an     // exception propagates from a remoting server to the client.     protected InvalidDepartmentException(System.Runtime.Serialization.SerializationInfo info,     System.Runtime.Serialization.StreamingContext context) : base(info, context) { } }</pre>	

Add new properties to the exception class when the data they provide is useful to resolving the exception. If new properties are added to the derived exception class, `ToString()` should be overridden to return the added information.

# C# Language Specification

For more information, see [Exceptions](#) and [The throw statement](#) in the [C# Language Specification](#). The language specification is the definitive source for C# syntax and usage.

## See also

- [Exception Hierarchy](#)

## Is this page helpful?

 Yes  No

## Recommended content

### [How to convert a string to a number - C# Programming Guide](#)

Learn how to convert a string to a number in C# by calling the Parse, TryParse, or Convert class methods.

### [?: operator - C# reference](#)

Learn about the C# ternary conditional operator that returns the result of one of the two expressions based on a Boolean expression's result.

### [C# switch statement](#)

switch (C# reference)

### [How to concatenate multiple strings \(C# Guide\)](#)

There are multiple ways to concatenate strings in C#. Learn the options and the reasons behind different choices.

Show more 