

try-finally (C# Reference)

07/20/2015 • 3 minutes to read •  +8

In this article

[Example](#)

[C# language specification](#)

[See also](#)

By using a `finally` block, you can clean up any resources that are allocated in a `try` block, and you can run code even if an exception occurs in the `try` block. Typically, the statements of a `finally` block run when control leaves a `try` statement. The transfer of control can occur as a result of normal execution, of execution of a `break`, `continue`, `goto`, or `return` statement, or of propagation of an exception out of the `try` statement.

Within a handled exception, the associated `finally` block is guaranteed to be run. However, if the exception is unhandled, execution of the `finally` block is dependent on how the exception unwind operation is triggered. That, in turn, is dependent on how your computer is set up. The only cases where `finally` clauses don't run involve a program being immediately stopped. An example of this would be when [InvalidProgramException](#) gets thrown because of the IL statements being corrupt. On most operating systems, reasonable resource cleanup will take place as part of stopping and unloading the process.

Usually, when an unhandled exception ends an application, whether or not the `finally` block is run is not important. However, if you have statements in a `finally` block that must be run even in that situation, one solution is to add a `catch` block to the `try-finally` statement. Alternatively, you can catch the exception that might be thrown in the `try` block of a `try-finally` statement higher up the call stack. That is, you can catch the exception in the method that calls the method that contains the `try-finally` statement, or in the method that calls that method, or in any method in the call stack. If the exception is not caught, execution of the `finally` block depends on whether the operating system chooses to trigger an exception unwind operation.

Example

In the following example, an invalid conversion statement causes a `System.InvalidCastException` exception. The exception is unhandled.

C#

 Copy

```
public class ThrowTestA
{
    public static void Main()
    {
        int i = 123;
        string s = "Some string";
        object obj = s;

        try
        {
            // Invalid conversion; obj contains a string, not a numeric
type.
            i = (int)obj;

            // The following statement is not run.
            Console.WriteLine("WriteLine at the end of the try block.");
        }
        finally
        {
            // To run the program in Visual Studio, type CTRL+F5. Then
            // click Cancel in the error dialog.
            Console.WriteLine("\nExecution of the finally block after an
unhandled\n" +
                "error depends on how the exception unwind operation is
triggered.");
            Console.WriteLine("i = {0}", i);
        }
    }
    // Output:
    // Unhandled Exception: System.InvalidCastException: Specified cast is
not valid.
    //
    // Execution of the finally block after an unhandled
    // error depends on how the exception unwind operation is triggered.
    // i = 123
}
```

In the following example, an exception from the TryCast method is caught in a method farther up the call stack.

C#

 Copy

```
public class ThrowTestB
{
    public static void Main()
    {
        try
        {
            // TryCast produces an unhandled exception.
            TryCast();
        }
        catch (Exception ex)
        {

```

```

        // Catch the exception that is unhandled in TryCast.
        Console.WriteLine
            ("Catching the {0} exception triggers the finally block.",
            ex.GetType());

        // Restore the original unhandled exception. You might not
        // know what exception to expect, or how to handle it, so pass
        // it on.
        throw;
    }
}

static void TryCast()
{
    int i = 123;
    string s = "Some string";
    object obj = s;

    try
    {
        // Invalid conversion; obj contains a string, not a numeric
type.
        i = (int)obj;

        // The following statement is not run.
        Console.WriteLine("WriteLine at the end of the try block.");
    }
    finally
    {
        // Report that the finally block is run, and show that the value
of
        // i has not been changed.
        Console.WriteLine("\nIn the finally block in TryCast, i =
{0}.\n", i);
    }
}
// Output:
// In the finally block in TryCast, i = 123.

// Catching the System.InvalidCastException exception triggers the fi-
nally block.

// Unhandled Exception: System.InvalidCastException: Specified cast is
not valid.
}

```

For more information about `finally`, see [try-catch-finally](#).

C# also contains the [using statement](#), which provides similar functionality for [IDisposable](#) objects in a convenient syntax.



C# language specification

For more information, see [The try statement](#) section of the [C# language specification](#).

See also

- [C# Reference](#)
- [C# Programming Guide](#)
- [C# Keywords](#)
- [try, throw, and catch Statements \(C++\)](#)
- [throw](#)
- [try-catch](#)
- [How to: Explicitly Throw Exceptions](#)

Is this page helpful?

 Yes  No
