**Exception Handling Guidelines**

Contents

[1 Document Control 3](#_Toc90472803)

[1.1 Change Record 3](#_Toc90472804)

[1.2 Reviewer 3](#_Toc90472805)

[1.3 Approver 3](#_Toc90472806)

[2 Document Purpose 3](#_Toc90472807)

[3 Best Practises for Exceptions 3](#_Toc90472808)

[3.1 Use try/catch/finally blocks to recover from errors or release resources 3](#_Toc90472809)

[3.2 Handle common conditions without throwing exceptions 4](#_Toc90472810)

[3.3 Throw exceptions instead of returning error code 4](#_Toc90472811)

[3.4 Use the predefined .Net exception types 4](#_Toc90472812)

[3.5 Include three constructors in custom exception classes 4](#_Toc90472813)

[3.6 Use grammatically correct error messages 5](#_Toc90472814)

[3.7 Include a localized string message in every exception 5](#_Toc90472815)

[3.8 In custom exceptions, provide additional properties as needed 5](#_Toc90472816)

[3.9 Use exception builder methods 6](#_Toc90472817)

[3.10 Restore state when methods don’t complete due to exceptions 6](#_Toc90472818)

[4 Using global exception handler 7](#_Toc90472819)

[5 Using custom middle ware global exception handler 8](#_Toc90472820)

[6 Using exception filters 8](#_Toc90472821)

[7 Appendix 9](#_Toc90472822)

# Document Control

## Change Record

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Change reference** |
| 12/15/2021 | Prajeesh T S | 1.1 | Initial version |

## Reviewer

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Approval/Review Date** |
| **Anoop Jose** | Staff Software Architect |  |

## Approver

|  |  |  |
| --- | --- | --- |
| **Name** | **Role** | **Approval/Review Date** |
| **Anoop Jose** | Staff Software Architect |  |

# Document Purpose

This document provides details regarding the guidelines for exception handling.

# Best Practises for Exceptions

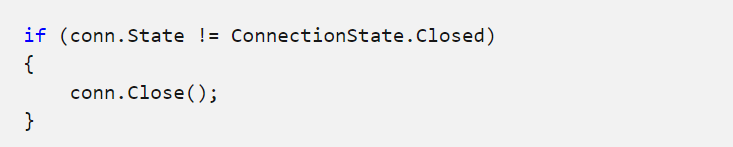
## Use try/catch/finally blocks to recover from errors or release resources

Use try/catch blocks around code that can potentially generate an exception ***and*** your code can recover from that exception. In catch blocks, always order exceptions from the most derived to the least derived. All exceptions derive from Exception Collection.

Clean up resources allocated with either using statements, or finally blocks. Prefer using statements to automatically clean up resources when exceptions are thrown. Use finally blocks to clean up resources that don't implement IDisposable.

## Handle common conditions without throwing exceptions

For conditions that are likely to occur but might trigger an exception, consider handling them in a way that will avoid the exception. For example, if you try to close a connection that is already closed, you'll get an InvalidOperationException. You can avoid that by using an if statement to check the connection state before trying to close it.



## Throw exceptions instead of returning error code

## Use the predefined .Net exception types

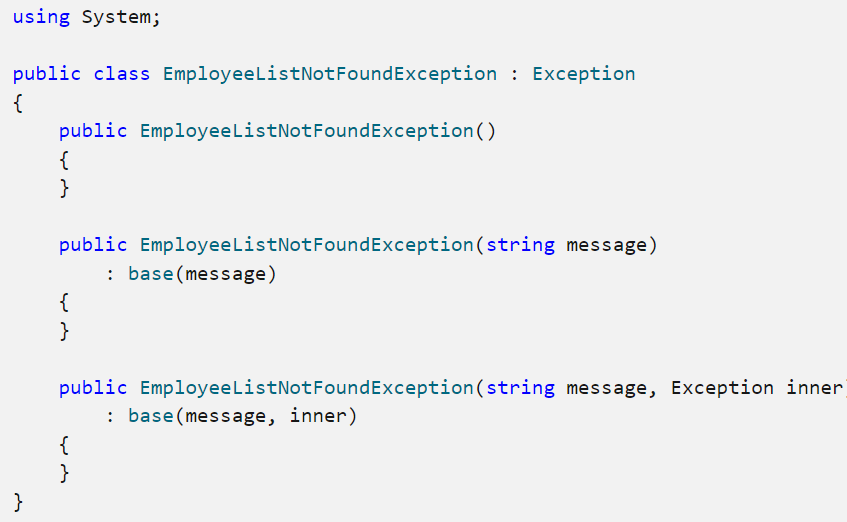
Introduce a new exception class only when a predefined one doesn't apply. For example:

* Throw an InvalidOperationException exception if a property set or method call is not appropriate given the object's current state.
* Throw an ArgumentException exception or one of the predefined classes that derive from ArgumentException if invalid parameters are passed.

## Include three constructors in custom exception classes

Use at least the three common constructors when creating your own exception classes: the parameter less constructor, a constructor that takes a string message, and a constructor that takes a string message and an inner exception.

* Exception(), which uses default values.
* Exception(String), which accepts a string message.
* Exception(String, Exception), which accepts a string message and an inner exception.



## Use grammatically correct error messages

Write clear sentences and include ending punctuation. Each sentence in the string assigned to the Exception.Message property should end in a period. For example, "The log table has overflowed." would be an appropriate message string.

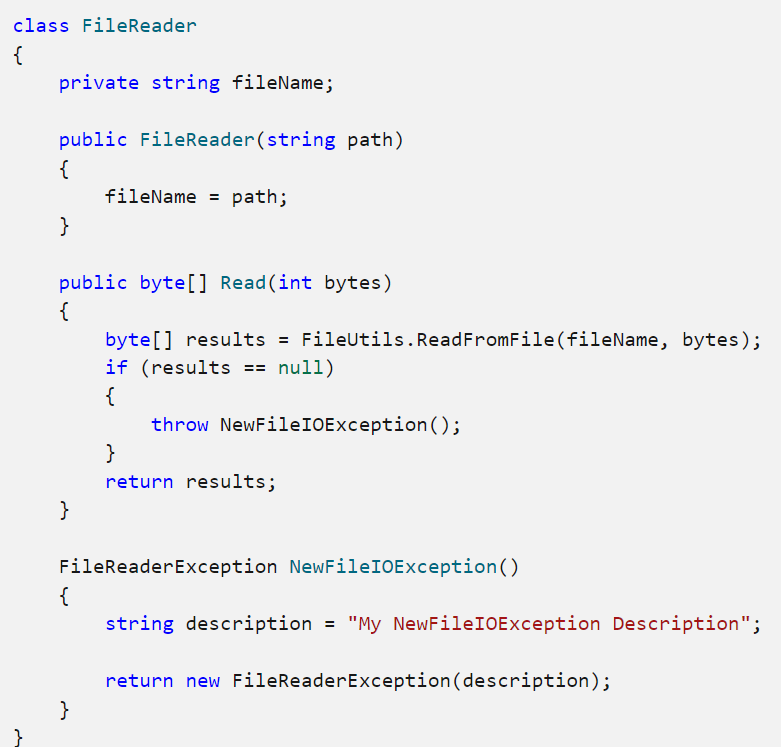
## Include a localized string message in every exception

## In custom exceptions, provide additional properties as needed

Provide additional properties for an exception (in addition to the custom message string) only when there's a programmatic scenario where the additional information is useful. For example, the FileNotFoundException provides the FileName property.

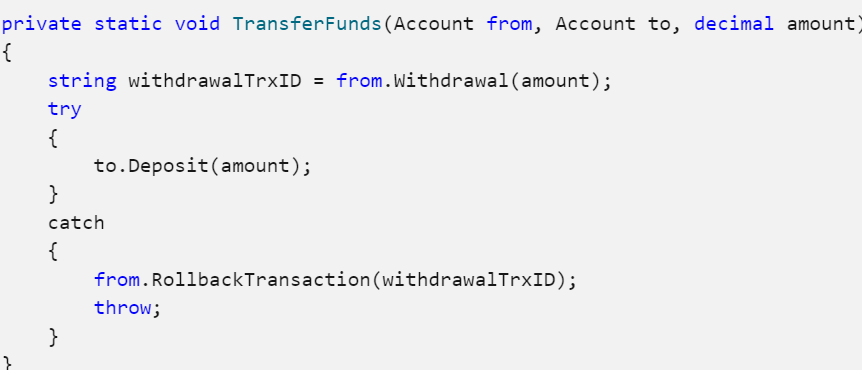
## Use exception builder methods

It is common for a class to throw the same exception from different places in its implementation. To avoid excessive code, use helper methods that create the exception and return it.



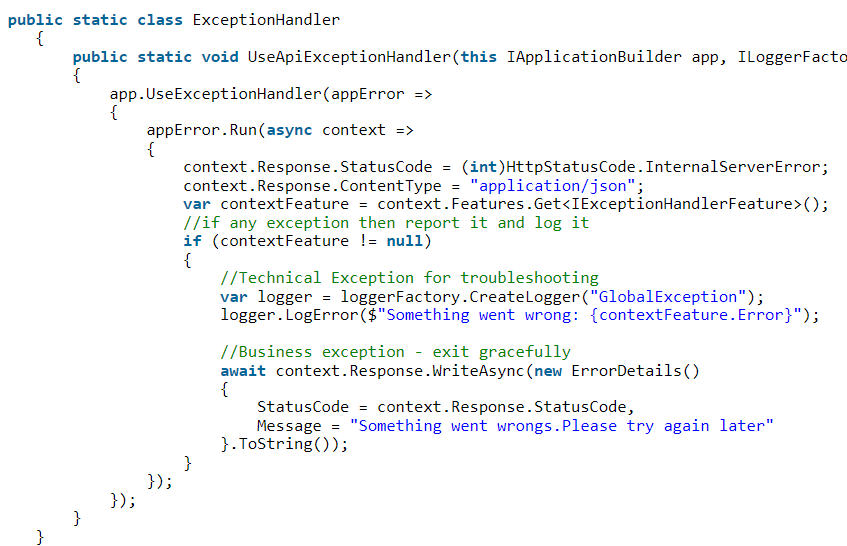
## Restore state when methods don’t complete due to exceptions

State should restore when exception occurs, the below code restoring state when exception happening in the deposit method.



# Using global exception handler

NET Core provides an interface called “IExceptionHandlerFeature*”*which provides the ability to handle exception details within an API pipeline.

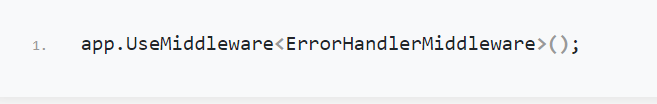


Register the global handler in start-up file



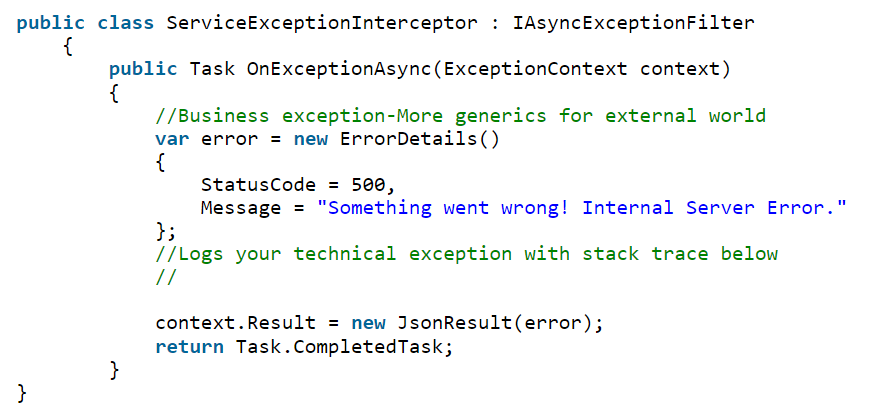
# Using custom middle ware global exception handler

.Net allow us to create custom middle ware classes to handle custom exceptions , one point to take is to register the middle ware in start-up class.



# Using exception filters

Using exception filters is effective for only API pipelines., if there is an exception in other layers of your app apart from the API layer then this approach won’t be helpful.





# Appendix

* [Best Practices for exceptions - .NET | Microsoft Docs](https://docs.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions)