.NET Framework 4

**Exception Handling Fundamentals**

The common language runtime supports an exception handling model based on the concepts of exception objects and protected blocks of code. The runtime creates an object to represent an exception when it occurs. You can also create your own exception classes by deriving classes from the appropriate base exception.

All languages that use the runtime handle exceptions in a similar manner. Each language uses a form of try/catch/finally structured exception handling. This section provides several examples of basic exception handling.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifIn This Section

[How to: Use the Try/Catch Block to Catch Exceptions](http://msdn.microsoft.com/en-us/library/xtd0s8kd.aspx)

Describes how to use the try/catch block to handle exceptions.

[How to: Use Specific Exceptions in a Catch Block](http://msdn.microsoft.com/en-us/library/3tca6706.aspx)

Describes how to catch specific exceptions.

[How to: Explicitly Throw Exceptions](http://msdn.microsoft.com/en-us/library/xhcbs8fz.aspx)

Describes how to throw exceptions and how to catch exceptions and then throw them again.

[How to: Create User-Defined Exceptions](http://msdn.microsoft.com/en-us/library/87cdya3t.aspx)

Describes how to create your own exception classes.

[Using User-Filtered Handlers](http://msdn.microsoft.com/en-us/library/4dy8x9k9.aspx)

Describes how to set up filtered exceptions.

[How to: Use a Finally Block](http://msdn.microsoft.com/en-us/library/ke0zf0f5.aspx)

Explains how to use the finally statement in an exception block.

.NET Framework 4

**How to: Use the Try/Catch Block to Catch Exceptions**

Place the sections of code that might throw exceptions in a try block and place code that handles exceptions in a catch block. The catch block is a series of statements beginning with the keyword **catch**, followed by an exception type and an action to be taken.

|  |
| --- |
| **Description: NoteNote** |
| Almost any line of code can cause an exception, particularly exceptions that are thrown by the common language runtime itself, such as [OutOfMemoryException](http://msdn.microsoft.com/en-us/library/system.outofmemoryexception.aspx) and [StackOverflowException](http://msdn.microsoft.com/en-us/library/system.stackoverflowexception.aspx). Most applications do not have to deal with these exceptions, but you should be aware of this possibility when writing libraries to be used by others. For suggestions on when to set code in a try block, see [Best Practices for Handling Exceptions](http://msdn.microsoft.com/en-us/library/seyhszts.aspx). |

The following code example uses a try/catch block to catch a possible exception. The Main method contains a try block with a **StreamReader** statement that opens a data file called data.txt and writes a string from the file. Following the try block is a catch block that catches any exception that results from the try block.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl00_code');" \o "Copy Code)

Imports System

Imports System.IO

Public Class ProcessFile

Public Shared Sub Main()

Try

Dim sr As StreamReader = File.OpenText("data.txt")

Console.WriteLine("The first line of this file is {0}", sr.ReadLine())

sr.Close()

Catch e As Exception

Console.WriteLine("An error occurred: '{0}'", e)

End Try

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl01_code');" \o "Copy Code)

using System;

using System.IO;

public class ProcessFile

{

public static void Main()

{

try

{

StreamReader sr = File.OpenText("data.txt");

Console.WriteLine("The first line of this file is {0}", sr.ReadLine());

sr.Close();

}

catch (Exception e)

{

Console.WriteLine("An error occurred: '{0}'", e);

}

}

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl02_code');" \o "Copy Code)

using namespace System;

using namespace System::IO;

public ref class ProcessFile

{

public:

static void Main()

{

try

{

StreamReader^ sr = File::OpenText("data.txt");

Console::WriteLine("The first line of this file is {0}", sr->ReadLine());

sr->Close();

}

catch (Exception^ e)

{

Console::WriteLine("An error occurred: '{0}'", e);

}

}

};

int main()

{

ProcessFile::Main();

}

This example illustrates a basic catch statement that will catch any exception. In general, it is good programming practice to catch a specific type of exception rather than use the basic catch statement. For more information about catching specific exceptions, see [Using Specific Exceptions in a Catch Block](http://msdn.microsoft.com/en-us/library/3tca6706.aspx).

.NET Framework 4

**How to: Use Specific Exceptions in a Catch Block**

When an exception occurs, it is passed up the stack and each catch block is given the opportunity to handle it. The order of catch statements is important. Put catch blocks targeted to specific exceptions before a general exception catch block or the compiler might issue an error. The proper catch block is determined by matching the type of the exception to the name of the exception specified in the catch block. If there is no specific catch block, the exception is caught by a general catch block, if one exists.

The following code example uses a try/catch block to catch an [InvalidCastException](http://msdn.microsoft.com/en-us/library/system.invalidcastexception.aspx). The sample creates a class called Employee with a single property, employee level (Emlevel). A method, PromoteEmployee, takes an object and increments the employee level. An **InvalidCastException** occurs when a [DateTime](http://msdn.microsoft.com/en-us/library/system.datetime.aspx) instance is passed to the PromoteEmployee method.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl17_ctl00_ctl00_code');" \o "Copy Code)

Imports System

Public Class Employee

'Create employee level property.

Public Property Emlevel As Integer

Get

Return emlevelValue

End Get

Set

emlevelValue = Value

End Set

End Property

Private emlevelValue As Integer = 0

End Class

Public Class Ex13

Public Shared Sub PromoteEmployee(emp As Object)

'Cast object to Employee.

Dim e As Employee = CType(emp, Employee)

' Increment employee level.

e.Emlevel = e.Emlevel + 1

End Sub

Public Shared Sub Main()

Try

Dim o As Object = New Employee()

Dim newyears As New DateTime(2001, 1, 1)

'Promote the new employee.

PromoteEmployee(o)

'Promote DateTime; results in InvalidCastException as newyears is not an employee instance.

PromoteEmployee(newyears)

Catch e As InvalidCastException

Console.WriteLine("Error passing data to PromoteEmployee method. " + e.Message)

End Try

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl17_ctl00_ctl01_code');" \o "Copy Code)

using System;

public class Employee

{

//Create employee level property.

public int Emlevel

{

get

{

return(emlevel);

}

set

{

emlevel = value;

}

}

private int emlevel = 0;

}

public class Ex13

{

public static void PromoteEmployee(Object emp)

{

//Cast object to Employee.

Employee e = (Employee) emp;

// Increment employee level.

e.Emlevel = e.Emlevel + 1;

}

public static void Main()

{

try

{

Object o = new Employee();

DateTime newyears = new DateTime(2001, 1, 1);

//Promote the new employee.

PromoteEmployee(o);

//Promote DateTime; results in InvalidCastException as newyears is not an employee instance.

PromoteEmployee(newyears);

}

catch (InvalidCastException e)

{

Console.WriteLine("Error passing data to PromoteEmployee method. " + e.Message);

}

}

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl17_ctl00_ctl02_code');" \o "Copy Code)

using namespace System;

public ref class Employee

{

public:

Employee()

{

emlevel = 0;

}

//Create employee level property.

property int Emlevel

{

int get()

{

return emlevel;

}

void set(int value)

{

emlevel = value;

}

}

private:

int emlevel;

};

public ref class Ex13

{

public:

static void PromoteEmployee(Object^ emp)

{

//Cast object to Employee.

Employee^ e = (Employee^) emp;

// Increment employee level.

e->Emlevel++;

}

static void Main()

{

try

{

Object^ o = gcnew Employee();

DateTime^ newyears = gcnew DateTime(2001, 1, 1);

//Promote the new employee.

PromoteEmployee(o);

//Promote DateTime; results in InvalidCastException as newyears is not an employee instance.

PromoteEmployee(newyears);

}

catch (InvalidCastException^ e)

{

Console::WriteLine("Error passing data to PromoteEmployee method. " + e->Message);

}

}

};

int main()

{

Ex13::Main();

}

The common language runtime catches exceptions that are not caught by a catch block. Depending on how the runtime is configured, either a debug dialog box appears, or the program stops executing and a dialog box with exception information appears. For information about debugging, see [Debugging and Profiling Applications](http://msdn.microsoft.com/en-us/library/7fe0dd2y.aspx).

.NET Framework 4

**How to: Explicitly Throw Exceptions**

You can explicitly throw an exception using the **throw** statement. You can also throw a caught exception again using the **throw** statement. It is good coding practice to add information to an exception that is re-thrown to provide more information when debugging.

The following code example uses a try/catch block to catch a possible [FileNotFoundException](http://msdn.microsoft.com/en-us/library/system.io.filenotfoundexception.aspx). Following the try block is a catch block that catches the [FileNotFoundException](http://msdn.microsoft.com/en-us/library/system.io.filenotfoundexception.aspx)and writes a message to the console if the data file is not found. The next statement is the throw statement that throws a new [FileNotFoundException](http://msdn.microsoft.com/en-us/library/system.io.filenotfoundexception.aspx) and adds text information to the exception.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl15_ctl00_ctl00_code');" \o "Copy Code)

Option Strict On

Imports System

Imports System.IO

Public Class ProcessFile

Public Shared Sub Main()

Dim fs As FileStream = Nothing

Try

'Opens a text file.

fs = New FileStream("c:\temp\data.txt", FileMode.Open)

Dim sr As New StreamReader(fs)

Dim line As String

'A value is read from the file and output to the console.

line = sr.ReadLine()

Console.WriteLine(line)

Catch e As FileNotFoundException

Console.WriteLine("[Data File Missing] {0}", e)

Throw New FileNotFoundException("[data.txt not in c:\temp directory]", e)

Finally

If fs IsNot Nothing Then fs.Close

End Try

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl15_ctl00_ctl01_code');" \o "Copy Code)

using System;

using System.IO;

public class ProcessFile

{

public static void Main()

{

FileStream fs = null;

try

{

//Opens a text tile.

fs = new FileStream(@"C:\temp\data.txt", FileMode.Open);

StreamReader sr = new StreamReader(fs);

string line;

//A value is read from the file and output to the console.

line = sr.ReadLine();

Console.WriteLine(line);

}

catch(FileNotFoundException e)

{

Console.WriteLine("[Data File Missing] {0}", e);

throw new FileNotFoundException(@"[data.txt not in c:\temp directory]",e);

}

finally

{

if (fs != null)

fs.Close();

}

}

}

.NET Framework 4

**How to: Create User-Defined Exceptions**

If you want users to be able to programmatically distinguish between some error conditions, you can create your own user-defined exceptions. The .NET Framework provides a hierarchy of exception classes ultimately derived from the base class [Exception](http://msdn.microsoft.com/en-us/library/system.exception.aspx). Each of these classes defines a specific exception, so in many cases you only have to catch the exception. You can also create your own exception classes by deriving from the [Exception](http://msdn.microsoft.com/en-us/library/system.exception.aspx) class.

When creating your own exceptions, it is good coding practice to end the class name of the user-defined exception with the word "Exception." It is also good practice to implement the three recommended common constructors, as shown in the following example.

|  |
| --- |
| **Description: NoteNote** |
| In situations where you are using remoting, you must ensure that the metadata for any user-defined exceptions is available at the server (callee) and to the client (the proxy object or caller). For example, code calling a method in a separate application domain must be able to find the assembly containing an exception thrown by a remote call. For more information, see [Best Practices for Handling Exceptions](http://msdn.microsoft.com/en-us/library/seyhszts.aspx). |

In the following example, a new exception class, EmployeeListNotFoundException, is derived from [Exception](http://msdn.microsoft.com/en-us/library/system.exception.aspx). Three constructors are defined in the class, each taking different parameters.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl00_code');" \o "Copy Code)

Imports System

Public Class EmployeeListNotFoundException

Inherits Exception

Public Sub New()

End Sub

Public Sub New(message As String)

MyBase.New(message)

End Sub

Public Sub New(message As String, inner As Exception)

MyBase.New(message, inner)

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl01_code');" \o "Copy Code)

using System;

public class EmployeeListNotFoundException: Exception

{

public EmployeeListNotFoundException()

{

}

public EmployeeListNotFoundException(string message)

: base(message)

{

}

public EmployeeListNotFoundException(string message, Exception inner)

: base(message, inner)

{

}

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl19_ctl00_ctl02_code');" \o "Copy Code)

using namespace System;

public ref class EmployeeListNotFoundException : Exception

{

public:

EmployeeListNotFoundException()

{

}

EmployeeListNotFoundException(String^ message)

: Exception(message)

{

}

EmployeeListNotFoundException(String^ message, Exception^ inner)

: Exception(message, inner)

{

}

};

.NET Framework 4

**Using User-Filtered Exception Handlers**

Currently, Visual Basic supports user-filtered exceptions. User-filtered exception handlers catch and handle exceptions based on requirements you define for the exception. These handlers use the **Catch** statement with the **When** keyword.

This technique is useful when a particular exception object corresponds to multiple errors. In this case, the object typically has a property that contains the specific error code associated with the error. You can use the error code property in the expression to select only the particular error you want to handle in that **Catch** clause.

The following Visual Basic example illustrates the **Catch/When** statement.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl01_code');" \o "Copy Code)

Try

'Try statements.

Catch When Err = VBErr\_ClassLoadException

'Catch statements.

End Try

The expression of the user-filtered clause is not restricted in any way. If an exception occurs during execution of the user-filtered expression, that exception is discarded and the filter expression is considered to have evaluated to false. In this case, the common language runtime continues the search for a handler for the current exception.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifCombining the Specific Exception and the User-Filtered Clauses

A catch statement can contain both the specific exception and the user-filtered clauses. The runtime tests the specific exception first. If the specific exception succeeds, the runtime executes the user filter. The generic filter can contain a reference to the variable declared in the class filter. Note that the order of the two filter clauses cannot be reversed.

The following Visual Basic example shows the specific exception ClassLoadException in the **Catch** statement as well as the user-filtered clause using the **When** keyword.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl10_ctl00_ctl00_code');" \o "Copy Code)

Try

'Try statements.

Catch cle As ClassLoadException When cle.IsRecoverable()

'Catch statements.

End Try

.NET Framework 4

**How to: Use Finally Blocks**

When an exception occurs, execution stops and control is given to the closest exception handler. This often means that lines of code you expect to always be called are not executed. Some resource cleanup, such as closing a file, must always be executed even if an exception is thrown. To accomplish this, you can use a finally block. A finally block is always executed, regardless of whether an exception is thrown.

The following code example uses a try/catch block to catch an [ArgumentOutOfRangeException](http://msdn.microsoft.com/en-us/library/system.argumentoutofrangeexception.aspx). The Main method creates two arrays and attempts to copy one to the other. The action generates an **ArgumentOutOfRangeException** and the error is written to the console. The finally block executes regardless of the outcome of the copy action.

Description: http://i.msdn.microsoft.com/Global/Images/clear.gifExample

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_ctl00_ctl00_code');" \o "Copy Code)

Imports System

Class ArgumentOutOfRangeExample

Public Shared Sub Main()

Dim array1() As Integer = {0, 0}

Dim array2() As Integer = {0, 0}

Try

Array.Copy(array1, array2 , -1)

Catch e As ArgumentOutOfRangeException

Console.WriteLine("Error: {0}", e)

Finally

Console.WriteLine("This statement is always executed.")

End Try

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_ctl00_ctl01_code');" \o "Copy Code)

using System;

class ArgumentOutOfRangeExample

{

public static void Main()

{

int[] array1 = {0, 0};

int[] array2 = {0, 0};

try

{

Array.Copy(array1, array2, -1);

}

catch (ArgumentOutOfRangeException e)

{

Console.WriteLine("Error: {0}", e);

}

finally

{

Console.WriteLine("This statement is always executed.");

}

}

}

Visual C++

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl12_ctl00_ctl02_code');" \o "Copy Code)

using namespace System;

ref class ArgumentOutOfRangeExample

{

public:

static void Main()

{

array<int>^ array1 = {0, 0};

array<int>^ array2 = {0, 0};

try

{

Array::Copy(array1, array2, -1);

}

catch (ArgumentOutOfRangeException^ e)

{

Console::WriteLine("Error: {0}", e);

}

finally

{

Console::WriteLine("This statement is always executed.");

}

}

};

int main()

{

ArgumentOutOfRangeExample::Main();

}