

3 Exception (Fehlerbehandlung)

Mit der Microsoft .NET Common Language Runtime wird nun vieles einfacher in Sachen Error-Handling. Auch wenn noch vorhanden, hat die betagte Syntax *On Error Goto* ausgedient. Hinzu kommt, dass das Microsoft .NET Framework endlich - abgesehen von sprachtypischen Syntax-Konventionen - ein einheitliches Fehlermanagement in allen .Net Sprachen bietet, nämlich die Exceptions. Exceptions, was soviel wie Ausnahmen bedeutet, können von einer Komponente ausgelöst werden und im aufrufenden Programm problemlos abgefangen werden. Das .NET Framework stellt standardmäßig eine Vielzahl von möglichen Exceptions zur Verfügung. Ausgehend vom Grundausnahmefehler, der einfachen Exception, leiten sich für die häufigsten Fehler spezifische Exceptions ab. Auch das verschachteln von Exceptions ist möglich.

```
Try
    Try
        Catch ex As Exception
            End Try
    Catch ex As Exception
    End Try
```

In den nun folgenden Beispielen zeige ich Ihnen wie einfach das Exception Handling in VB.NET zu implementieren ist. Durch die einheitliche Verankerung im .NET Framework müssen Sie, wie schon erwähnt, im Vergleich zu anderen .Net Sprachen keine Abstriche machen. Sie können wie in z.B. in C# Exceptions abfangen, selbst werfen, oder eine selbst kreierte Fehlermeldung ausgeben. Zu erwähnen ist noch das When Statment, das eine Exception unter einer bestimmten Bedingung auslöst.

3.1 Arten von Exception

Das .NET Framework stellt standardmäßig eine Vielzahl von möglichen Exceptions zur Verfügung. Ausgehend vom Grundausnahmefehler, der einfachen *Exception*, leiten sich für die häufigsten Fehler spezifische Exceptions ab.

Hier eine kleine Auswahl der häufigsten Ausnahmen:

Klasse	Bedeutung
AccessException	Fehler beim Zugriff auf eine Eigenschaft oder Methode
ArgumentException	ein Argument ist ungültig
ArgumentNullException	eine Methode erhält Nothing als Argument – sie akzeptiert es nicht
ArgumentOutOfRangeException	das übergebene Argument liegt außerhalb des Gültigkeitsbereichs
ArithmeticException	Unter- oder Überlauf
ArrayTypeMismatchException	in einem Datenfeld wurde ein falscher Wert gespeichert
DividedByZeroException	Division durch 0
FormatException	Typen unverträglich
IndexOutOfRangeException	Index außerhalb des gültigen Bereichs (beispielsweise bei Arrays)
InvalidCastException	beim Umwandeln von einem Datentyp in einen anderen ist ein Fehler aufgetreten
NullReferenceException	ein Objekt referenziert auf Nothing
OutOfMemoryException	kein Arbeitsspeicher mehr zur Verfügung
StackOverflowException	Überlauf

3.2 Rezepte für Exception

VB.NET ist in manchen Situationen (z.B. cast Operationen; Typkonvertierungen) um einiges gutmütiger als zum Beispiel C# - Datentypen werden je nach Gebrauch teilweise auch implizit konvertiert. C# würde da schon längst eine Exception und Sie zu einer expliziten Umcastung (Typenkonvertierung) auffordern.

Deshalb wollen wir uns in diesem Kapitel mit Fehlern beschäftigen, die VB.NET Ihnen schnell krumm nehmen kann.

Als Beispiel nehmen wir an: deklarieren Sie ein Array mit n Elementen. Aus irgendeinem Grund passiert es aber, dass unsere *for* Schleife n+1 Element auslesen will. Dies ist nun auch für VB.NET ein guter Grund eine Exception auszugeben. Der Programmcode zu diesem Beispiel ist folgender:

Vereinfachtes Beispiel ohne Fehlerbehandlung

```
Dim aTest() As Integer = {0, 1, 2, 3}
Dim nVar As Integer

    For nVar = 0 To 5 Step 1
        Response.Write(aTest(nVar))
    Next
```

Bei Aufruf dieses Sourcecodes erhalten wir eine *IndexOutOfRangeException* Exception.

Nun gibt es mehrere Möglichkeiten um eine geworfene Exception zu behandeln bzw. anzufangen.

Die Try-Catch Methode

Diese ist die einfachste Möglichkeit um einer abrupten Beendigung der Sourcecodeausführung zu entgehen. Der *Try ... End Try* Block enthält nicht nur den Sourcecode, der möglicherweise zu einer Exception führen könnte, sondern auch die Fehlerbehandlung mittels *Catch* Statement.

Der vorige Sourcecode bekommt nun eine wesentliche Verbesserung

```
Dim arTest() As Integer = {0, 1, 2, 3}
Dim nVar As Integer

    Try
        For nVar = 0 To 5 Step 1
            Response.Write(aTest(nVar))
        Next
    Catch ex As Exception
        Console.WriteLine(ex.Message)
    End Try
```

Der Sourcecode wird bis zum Auftreten der Exception ausgeführt. Dies erkennt man daran, dass der Inhalt des Arrays *aTest* bis zum letztgültigen Index ausgegeben wird.

Die Exception ist zwar jetzt elegant abgefangen, aber das war's auch schon. Code, der nach dem aufgetretenen Fehler folgt, wird nicht mehr ausgeführt (innerhalb *Try-End Try*). Damit zeigt sich schon die Notwendigkeit nach einem ausgereiften Prozess. Damit wären wir auch schon bei der nächsten Exception Erweiterung:

Die Try-Catch-Finally Methode

Für den Fall, das es notwendig ist das dass Programm nach der Exception weiterlaufen soll, d.h. dass etwaiger Sourcecode nach einer möglichen Exception weiter ausgeführt werden soll, bietet sich zusätzlich das *Finally* Statement an.

```
Dim aTest() As Integer = {0, 1, 2, 3}
Dim nVar As Integer

Try
    For nVar = 0 To 5 Step 1
        Response.Write(aTest(nVar))
    Next
Catch ex As Exception
    Console.Write(ex.Message)
Finally
    Console.Write(„Hier gehts in jedem Fall weiter“)
End Try
```

Der Code in *Finally* wird immer ausgeführt - ob der Code eine Exception ausgelöst hat oder nicht. Deshalb bietet sich der *Finally* Block für "Cleanup" an: um Ressourcen freizugeben

Wie in anderen .Net Sprachen (C++, C#) können Sie natürlich auch mehrere Exceptions abfangen. Schachteln Sie einfach die *Catch* Statements innerhalb des *Try ... End Try* Blockes.

Bei der Anwendung von mehreren *Catch* Statements kann in manchen Fällen schon eine Exception ausreichen, um das Ergebnis oder die weitere Abarbeitung des Sourcecodes sinnlos zu machen. Um eben den *Try - End Try* Block sofort nach der Exception zu verlassen verwenden Sie *Exit Try* nach dem *Catch* Statement.

When-Anweisung

Die Catch Anweisung kann nun noch mit When-Anweisung erweitert werden. Hier können nun bestimmte Bedingungen festgelegt werden, für die eine Fehlermeldung erzeugt werden soll.

```
Dim aTest() As Integer = {0, 1, 2, 3}
Dim nVar As Integer

Try
    For nVar = 0 To 5 Step 1
        Response.Write(aTest(nVar))
    Next

    Catch When nVar = 0
        Console.WriteLine(ex.Message)

End Try
```

Benutzerdefinierte Fehlermeldungen

Die Fehlermeldungen, die Sie nach dem Abfangen einer Exception erhalten sind nicht für jeden "lesbar". Um den Client präzise und leicht verständlich auf seinen Fehler aufmerksam zu machen, können Sie Ihre eigenen Fehlermeldungen in Ihrer Anwendung ausgeben.

```
Sub Main()
    Dim nZahl As Long
    Dim nZahl1 As Long = 100
    Dim nZahl2 As Long = 0

    Try
        ' Zum abfangen der Fehler ist die Reihenfolge der
        ' Catch-Anweisung wichtig, damit nicht ein Fehler unbehandelt bleibt
        nZahl = nZahl1 / nZahl2
        Console.WriteLine("Ergebnis:" & nZahl.ToString)

    Catch ex As ArithmeticException
        ' Standardverhalten bei einem Arithmetischen Fehler
        Console.WriteLine("Division durch Null; nZahl = " & nZahl2.ToString)

    Finally
        ' Die Finally-Anweisung wird immer ausgeführt
    End Try

End Sub
```

Throwing Exceptions

Als letztes möchte ich noch demonstrieren, dass Sie nicht nur Exceptions abfangen können, sondern auch selbst Exceptions erzeugt (throw) werden können. Das "Erzeugen" von Exceptions ist genauso einfach wie das Abfangen. Sie deklarieren die Exception und verwenden dann dort wo Sie "erzeugt" werden soll die *Throw*-Anweisung.

```
Dim nVar As Integer
Dim AEx As New ArgumentException

Try
    nVar = "10"
    if nVar > 5 Then
        Throw AEx
    Else
        Response.Write("Alles OK")
    End If
Catch e As Exception
    Response.Write(e.ToString)
End Try
```

Tips zur Verwendung von Exceptions:

Um ein sinnvolles Exception Handling zu gewährleisten, sollten Sie:

- lieber Exceptions genau spezifizieren, als nur eine allgemeine Exception werfen
- wenn ungültige Werte übergeben werden immer die *ArgumentException* werfen
- Für Exceptions, die vom Client ausgelöst werden einen verständlichen Text ausgeben auf das Werfen von Exceptions verzichten wenn der sichere und vollständige Programmablauf gefährdet ist
- auf Exceptions verzichten, wenn triviale Fehler behandelt werden sollen

3.2.1 Exception_01

Rezept	11.	Exception
Beispiel		Exception_01\

Das vorliegende Beispiel verdeutlicht die Arbeit mit Exception und Ihre Möglichkeiten.

3.2.2 TryCatchFinally

Rezept	12.	Exception
Beispiel		TryCatchFinally

3.2.3 Debug

Rezept	13.	Exception
Beispiel		DebugDemo