Learning VB.Net

Tutorial 20 – Try & Catch

Hello everyone… welcome to vb.net tutorials. These are going to be very basic tutorials about using the language to create simple applications, hope you enjoy it. If you have any notes about it, please send them to **notes@mka-soft.com** I will be happy to answer them. Finally if you find these tutorials are useful, it would be nice from you to send a small donation via PayPal to **donation@mka-soft.com**.

Tutorial posted on 2010-May-29.

Try & Catch

When you develop a vb.net application it is very common to get what is known as Runtime error. These are errors that might happen due to some wrong input for example, or some computational operation during program execution. To demonstrate the idea, create a simple vb.net application that read two values from the display and divide them. The code should be similar to this:

```
Dim A As Integer
Dim B As Integer
Dim C As Integer
A = TextBox1.Text
B = TextBox2.Text
C = A / B
MsgBox("The result is:" & C.ToString)
```

This code is correct, and should work fine. However if the value of B is zero (by entering a value of zero in textbox2), then you will get a runtime error. This is simply because you can not divide any number by zero (C=A/B).

This is just a simple example of the errors that you might get. VB allows you to catch such errors so that your program don't crash, and you can give a friendly message to the end user or treat the error. The way to do it is by using the try statement. It should be similar to the following:

```
Try
     The code that could cause error goes here
Catch ex As Exception
     The treatement of the error goes here
End Try
```

To use this one, you can rewrite the code as follows:

```
Dim A As Integer
Dim B As Integer
Dim C As Integer
A = TextBox1.Text
B = TextBox2.Text

Try

    C = A / B
    MsgBox("The result is:" & C.ToString)

Catch ex As Exception

    MsgBox("Error")

End Try
```

What happens here is that the statements between Try and Catch are monitored for any errors. If an error happens, then the execution will be interrupted, and a new execution starts in the Catch part. So in the example above if the division is by zero, then a friendly message is displayed telling the end user that there is some kind of problem is there. You program will not crash in this case.

Another thing is that there is an object called **ex**. This one holds details about the error. You can get some details about the error itself. For example:

```
Try

    C = A / B
    MsgBox("The result is:" & C.ToString)

Catch ex As Exception

    MsgBox(ex.Message)

End Try
```

Here the program will give the end user the detail of the error (the message property describes the error here). You can display other error details, or store them for debugging purposes. Some of these are `ex.StackTrace` which gives you the calls that caused the errors, and where the error exactly happened. So it is very useful.

The try statement can catch different set of errors, so using `ex.Message` is useful because it can tell you what kind of error you are getting and hence you can identify where the error is.

Last thing is that you can use a finally part with the Try statement

```
Try
     The code that could cause error goes here.
Catch ex As Exception
     The treatement of the error goes here.
Finally
     A number of statements that always get executed.
End Try
```

This part is always executed regardless of the state of execution errors. This part can be eliminated, and you can place your code after the try statement resulting in exactly the same effect.

So this concludes our last tutorial about vb.net. It was enjoyable doing these tutorials. If you have questions, or suggestions please send them to: **notes@mka-soft.com**.
Thanks.

mkaatr