### **Part 42 - C# Tutorial - Custom Exceptions**

**In this part we will learn**  
**1.** When to create custom exceptions  
**2.** Creating a custom exception from the scratch  
**3.** Throwing and catching the custom exception  
  
  
**To understand custom exceptions, you should have good understanding of**  
[Part 21 - Inheritance](http://csharp-video-tutorials.blogspot.com/2012/06/part-21-c-tutorial-inheritance-in-c.html)  
[Part 40 - Exception Handling Basics](http://csharp-video-tutorials.blogspot.com/2012/06/part-40-c-tutorial-exception-handling.html)  
[Part 41 - Inner Exceptions](http://csharp-video-tutorials.blogspot.com/2012/06/part-41-c-tutorial-inner-exceptions.html)

**When do you usually go for creating your own custom exceptions?**  
If none of the already existing dotnet exceptions adequately describes the problem.   
  
  
**Consider that**  
**1.** I have an asp.net web application.  
**2.** The application should allow the user to have only one logged in session.  
**3.** If the user is already logged in, and if he opens another browser window and tries to login again, the application should throw an error stating he is already logged in another browser window.  
  
  
With in the .NET framework we donot have any exception, that adequately describes this problem. So this scenario is one of the examples where you want to create a custom exception.  
  
  
We know that an exception is a class. So to create a Custom exception,   
**1. Create a class that derives from System.Exception class. As a convention, end the class name with Exception suffix. All .NET exceptions end with,  exception suffix. If you don't do so, you won't get a compiler error, but you will be deviating from the guidelines for creating custom exceptions.**  
public class UserAlreadyLoggedInException : Exception  
{  
}  
  
  
**2. Provide a public constructor, that takes in a string parameter. This constructor simply passes the string parameter, to the base exception class constructor.**  
public UserAlreadyLoggedInException(string message)  
        : base(message)  
{  
}  
  
  
**3. Using InnerExceptions, you can also track back the original exception. If you want to provide this capability for your custom exception class, then overload the constructor as shown below. If you are new to Constructor Overloading, please watch this video.**  
public UserAlreadyLoggedInException(string message, Exception innerException)  
    : base(message, innerException)  
{  
}  
  
  
**4. If you want your Exception class object to work across application domains, then the object must be serializable. To make your exception class serializable mark it with Serializable attribute and provide a constructor that invokes the base Exception class constructor that takes in SerializationInfo and StreamingContext objects as parameters.**  
[Serializable]  
public class UserAlreadyLoggedInException : Exception  
{  
    public UserAlreadyLoggedInException(SerializationInfo info, StreamingContext context)  
        : base(info, context)  
    {  
    }  
}  
  
  
***Note:****It is also possible to provide your own custom serialization, which will discuss in a later session.*  
  
  
**Complete Example of creating a custom exception:**  
using System;  
using System.Runtime.Serialization;  
  
  
public class CustomExceptions  
{  
    public static void Main()  
    {  
        try  
        {  
            throw new UserAlreadyLoggedInException("User Already logged in");  
        }  
        catch (UserAlreadyLoggedInException ex)  
        {  
            Console.WriteLine(ex.Message);  
        }  
    }  
}  
  
  
[Serializable]  
public class UserAlreadyLoggedInException : Exception  
{  
    public UserAlreadyLoggedInException(string message)  
        : base(message)  
    {  
    }  
  
  
    public UserAlreadyLoggedInException(string message, Exception innerException)  
        : base(message, innerException)  
    {  
    }  
      
    public UserAlreadyLoggedInException(SerializationInfo info, StreamingContext context)  
        : base(info, context)  
    {  
    }  
}