

Exception Error Handling Implementation in MySQL/VB.NET Windows Database Applications

Written by
Dr. Ernest Bonat, Ph.D.
Visual WWW, Inc.



Visual WWW, Inc.

'Created by Developers for Developers'

www.evisualwww.com | info@evisualwww.com

Copyright © 2000 - 2007 Visual WWW, Inc. All right reserved

Table of Content

Table of Content.....	2
Introduction	2
Required Software	2
Why is Exception Error Handling Required?.....	2
Structured Exception Error Handling	3
Using Statement.....	6
Writing Exception Errors to a Log File	7
MySQL Data Load with Exception Error Log File	10
MySQL Data Insert with Exception Error Log File	12
MySQL Data Update with Exception Error Log File.....	16
MySQL Data Delete with Exception Error Log File.....	20
Conclusions.....	22
Visual WWW Legal Notice	23

Introduction

Error handling implementation in Windows database applications is a must for any Application Developer today. The main idea of error handling is to avoid application crashes by finding out the occurred errors and fixing them. Different programming languages have different ways of implementing error handling. Microsoft Visual Basic .NET (VB.NET) is the most common programming language for developing windows database applications today. VB.NET codes are provided in many websites, books and materials without error handling implementation. In this article I would like to introduce you to the basics of error handling implementation in MySQL/VB.NET windows database applications. I'll be covering structured exception error handling, which was introduced for the first time in VB.NET 2002. I will also go over the latest technologies of disposing unmanaged resources in .NET Framework with the Using statement. To write exception errors to a log text file, a generic procedure will be developed while MySQL data is loading, inserting, updating and deleting.

Required Software

- [MySQL Database Server 5.1.30](#)
- [MySQL Connector/NET 5.2.5](#)
- [Toad for MySQL Freeware 4.1](#)
- [Microsoft Visual Basic 2008 Express Edition](#)

Why is Exception Error Handling Required?

I assume this question would be easy to answer for any Application Developer. Any developer would like to know why its program does not work properly, and how to fix it. To find out these questions the Application Developers need to write the code with the error handling. Below are the three main reasons why error handling implementation in business applications development is very important:

1. Avoid application crashes
2. Show the occurred errors, in a user-friendly way, to the users
3. Store occurred errors for future application upgrades and auditing

I always recommend my clients and students to develop business applications with error handling implementation. I also suggest that they properly store these occurred errors for future reference. In general, many Application Developers store these errors in a log text file or in a specific database error-capture table. Both approaches would be acceptable. I like to use a log text file because the error may occur with the database connection or any database problem, in this case I cannot insert the error record into the table. Let's look at the

error handling implementation in VB.NET 2002 and 2003 first so we can compare them with the latest 2008 version.

Structured Exception Error Handling

Structured exception handling implementation appears in Microsoft technologies for the first time in VS.NET 2002. This error handling was implemented by using Try...Catch...Finally statement in .NET Framework. The Try...Catch...Finally statement guarantees an easy way to handle all possible errors (exception errors) that may occur while the application is running. It contains three main blocks:

Try – contains the real programming code to be controlled for any possible error.

Catch – produces the error occurred during applications execution from Try block.

Finally – always executes last, regardless of whether the code in the Catch block has also executed. In general, this block is used for cleanup resources, like closing files and releasing created custom and unmanaged resources (objects).

Below, in Listing 1, is the standard code of the exception error structure using MySQLException class:

```
' Declaring and initializing objects
Try
' Programming lines of code
Catch exError As MySqlException
' Error handling code
Finally
' Cleanup custom and unmanaged resources if necessary
End Try
```

Listing 1: Structured exception error handling code using MySQLException class

So far, the main problems I found were in the Try and Finally blocks. Most Application Developers understand the purpose of the Try block very well. It's very clear that the Try block must include the entire real programming code of the application. If any code is written outside the Try block and it generates an execute error, the application will inevitably crash. I believe no one has any doubt about this. So, why do VB.NET Application Developers write code outside this block today? I have no idea!

Let's look at the following code below (Listing 2). As you can see the connection string property (ConnectionString) of the MySQL ADO.NET connection object MySqlConnectionObject has been hardcode outside the error handling structure. This is a very bad programming practice because the connection string should not be hardcoded and set outside the error handling structure. One more thing, the ADO.NET connection object has been closed inside the Try block. This line of code should be done in the Finally block as required by definition (Listing 3). Don't dispose the connection object at the end of the Try block because if for any reason the application crashes before, the connection object will not be destroyed. It'll be floating on the server waiting for the Garbage Collector (GC) process.

```
Dim MySqlConnectionObject = New MySqlConnection
Try
MySqlConnectionObject.ConnectionString = "server=???;user id=???;password=???;database=???"
MySqlConnectionObject.Open()
' More programming code...
MySqlConnectionObject.Close()
Catch exError As MySqlException
MsgBox("An Error Occurred. " & exError.Number & " - " & exError.Message)
Finally
' Cleanup custom and unmanaged resources
```

End Try

Listing 2: Open and close the MySQL ADO.NET connection object inside the Try block

```
Dim MySqlConnectionObject = New MySqlConnection
MySqlConnectionObject.ConnectionString = "server=???;user id=???;password=???;database=???"
Try
    MySqlConnectionObject.Open()
    ' More programming code...
Catch exError As MySqlException
    MsgBox("An Error Occurred. " & exError.Number & " - " & exError.Message)
Finally
    MySqlConnectionObject.Close()
    ' Cleanup custom and unmanaged resources
End Try
```

Listing 3: Close the MySQL ADO.NET connection object inside the Finally block

A simple approach could be to write two generic functions for opening and closing the MySQL ADO.NET connection object inside the Try block as shown in Listing 4. We can see that the connection string was passed by value to the generic function MySQLOpenConnection() and it was set to the connection object inside the Try block. For more info about MySQL ADO Connection String please read my paper "[Define and Store MySQL ADO Connection String in VB.NET 2005](#)" and download the source code from my website [Visual WWW Downloads](#). Just to mention the connection object has been closed properly in the Try block in generic function MySQLCloseConnection(). In real business application development its bad programming practices to keep open a MySQL ADO.NET connection object all the times. We don't want many connection objects open at the same time in a production server. It's better to open the connection object, do the required database transactions (load, insert, update and delete) and then close it properly. I think, in general, these two functions could be applied in any MySQL/VB.NET Windows database applications development.

```
Public Sub MySQLOpenConnection(ByVal pConnectionString As String, _
                               ByRef pErrorMessageString As String)
    Try
        MySqlConnectionObject.ConnectionString = pConnectionString
        MySqlConnectionObject.Open()
    Catch exError As MySqlException
        pErrorMessageString = exError.Number & " - " & exError.Message
    End Try
End Sub

Public Sub MySQLCloseConnection(ByRef pErrorMessageString As String)
    Try
        If Not MySqlConnectionObject Is Nothing Then
            If MySqlConnectionObject.State = ConnectionState.Open Then
                MySqlConnectionObject.Close()
                MySqlConnectionObject.Dispose()
            End If
        End If
    Catch exError As MySqlException
        pErrorMessageString = exError.Number & " - " & exError.Message
    End Try
End Sub
```

Listing 4: Generic functions to open and close MySQL ADO.NET connection object

Let's look at our VB.NET 2008 example project. To do a better implementation of error handling and code organization let's create a class object ExceptionClass shown in Listing 5. As you can see, I always use in my

MySQL/VB.NET papers, a disposal class object. It was created to properly release the ExceptionClass recourse by implementing the interface IDisposable. The class object ExceptionClass is designed inside the namespace WritingExceptionLibrary and two libraries have been imported: the MySQL Connector/NET data library MySqlConnection for database connection and transactions; and the input/output system library IO for reading and writing to a text file.

```
Imports MySql.Data.MySqlClient
Imports System.IO

Namespace WritingExceptionLibrary
  Public Class ExceptionClass
    Inherits ObjectDisposeClass
    Private MySqlConnectionObject As New MySqlConnection
    Private PositionStreamWriter As StreamWriter
    ' Developed custom properties, methods and events...
  End Class

  Public Class ObjectDisposeClass
    Implements IDisposable
    Private disposedValue As Boolean = False
    Public Sub Dispose() Implements IDisposable.Dispose
      Dispose(True)
      GC.SuppressFinalize(Me)
    End Sub
    Protected Overridable Sub Dispose(ByVal disposing As Boolean)
      If Not Me.disposedValue Then
        If disposing Then
          ' TODO: free unmanaged resources when explicitly called
        End If
        ' TODO: free shared unmanaged resources
      End If
      Me.disposedValue = True
    End Sub
  End Class
End Namespace
```

Listing 5: Class object ExceptionClass design code

In Listing 6 we see the code of the test form object WritingExceptionErrorsForm. This form includes two imported libraries, the MySQL Connector/NET data library MySqlConnection and the namespace project WritingExceptionLibrary. A new instant ExceptionClassObject of the class object ExceptionClass is declared and initialized. The connection string MySqlConnectionString and the name of the exception log file ExceptionErrorFileString have been defined and stored in the application configuration settings file as shown in Table 1. The path of the exception log file has been defined in the same folder of the application executable file as Application.StartupPath().

```
Imports MySql.Data.MySqlClient
Imports WritingExceptionErrors.WritingExceptionLibrary

Public Class WritingExceptionErrorsForm
  Private ExceptionClassObject As New ExceptionClass
  Private MySqlConnectionString As String = My.Settings.MySqlConnectionString
  Private ExceptionErrorFileString As String = Application.StartupPath() & "\" & My.Settings.ExceptionErrorFile
  Private WriteErrorMessageString As String
  Private ErrorMessageString As String

  Private Sub WritingExceptionErrorsForm_FormClosed(ByVal sender As Object, _
```

```

        ByVal e As
System.Windows.Forms.FormClosedEventArgs) _
        Handles Me.FormClosed

    If Not ExceptionClassObject Is Nothing Then
        ExceptionClassObject.Dispose()
        ExceptionClassObject = Nothing
    End If
    Dispose()
End Sub

Private Sub ExitButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles ExitButton.Click

    Close()
End Sub

End Class
    
```

Listing 6: Form object WritingExceptionErrorsForm code

Name	Type	Scope	Value
MySQLConnectionString	(ConnectionString)	Application	Server=???;Database=???;Uid=???;Pwd=???;
ExceptionErrorFile	String	Application	ExceptionErrorFile.log

Table 1: Application configuration settings for MySQL connection string and exception error log file path

After we designed our VB.NET example project structure and before continuing to write more error handling codes, let's look at the .NET Using statement.

Using Statement

In VB.NET 2005, Microsoft introduced for the first time the Using statement to dispose unmanaged resources like file handle, COM wrapper, ADO.NET connection object, etc. Managed resources are disposed of by the .NET Framework GC without any extra coding on your part. Here is the main code structure of the Using statement (Listing 7):

```

Using (Resource list | Resource expression)
    ' Programming lines of code
End Using
    
```

Listing 7: Using statement code structure

Where:

Resource list - required if you do not supply resource expression. List one or more system resources that this Using block controls

Resource expression - required if you do not supply resource list. Reference variable or expression referring to a system resource to be controlled by this Using block statements

End Using - required. It terminates the definition of the Using block and disposes of all the resources that it controls

The program can handle an exception error that might occur within the Using statement by adding a complete Try...Catch statement as following in Listing 8:

```

Try
    Using (Resource list | Resource expression)
    
```

```
' Programming lines of code
End Using
Catch exError As MySqlException
' Error handling code
End Try
```

Listing 8: Using statement inside the Try...Catch exception handling structure

As you can see the Finally block is not required at this point because the End Using statement takes care of destroying unmanaged resources. This will definitely save a lot of codes and reduce the application development time. The above VB.NET programming structure should be the standard for any windows database applications development today, especially for managing MySQL ADO.NET objects. If we apply this idea to the same MySQL ADO.NET connection object MySqlConnectionObject shown in Listing 3, the general error handling code structure will look like (Listing 9):

```
Try
Using MySqlConnectionObject As New MySqlConnection(My.Settings.MySQLConnectionString)
    MySqlConnectionObject.Open()
    ' More programming code...
End Using
Catch exError As MySqlException
    MsgBox(exError.Number & " - " & exError.Message)
End Try
```

Listing 9: General Using statement for MySQL ADO.NET connection object

At this point, as I explained before, the connection object does not need to be disposed and destroyed. The Using statement simplifies and makes the VB.NET code easier to read. Please use this code structure every day at work in production environment. I use it and recommend my clients, developer friends and students to use it too. This is a very useful VB.NET defined and organized programming code!

Writing Exception Errors to a Log File

Now what we know how to catch the exception errors, we need to learn how to store them properly. One of my favorite approaches is to store these exception errors in a log text file. I found this implementation very easy to code and use. The question is what parameters of these exception errors need to be stored. They include the following five:

1. Date/Time – when the exception error occurred
2. Source – in which application object the exception error occurred (control, form, web page, etc.)
3. Location – in which application event or procedure the exception error occurred
4. Error Number – VB.NET compile error number
5. Error Description – VB.NET compile error description

I really believe that these five parameters are good enough to locate and fix any exception errors. Some Application Developers miss the application object (source) and application event or procedure (location) where exception errors occurred. This may increase the error search considerably. I have developed many complex Windows and Internet web applications for the last eighteen years and found very useful information to identify the application source and location of the exception errors. Let's look at the exception error capture procedure developed in our class object ExceptionClass. Listing 10 shows the procedure WriteExceptionErrorToFile() to write to the log text file defined in by value parameter pFileNamePathString. This parameter contains both the path and the name log file. As I explained before, the path is defined by the application executable file Application.StartupPath() (Listing 6) and the name is stored in the application configuration settings file as ExceptionErrorFile.log (Table 1). A file stream object ObjectFileStream was created with append mode FileMode.Append and write access FileAccess.Write. The stream write object ObjectStreamWriter writes the

exception errors to the log file using the WriteLine() method. After that this object needs to be destroyed by using the Flush() and Close() methods. The file stream object ObjectFileStream should be closed on time also.

```
Public Sub WriteExceptionErrorToFile(ByVal pFileNamePathString As String, _
                                     ByVal pSourceObjectNameString As String, _
                                     ByVal pProcedureNameString As String, _
                                     ByVal pWriteErrorMessageString As String, _
                                     ByRef pErrorMessageString As String)
    Dim ExceptionMessageString As String
    Try
        Dim ObjectFileStream As New FileStream(pFileNamePathString, _
                                               FileMode.Append, _
                                               FileAccess.Write, _
                                               FileShare.None)

        Dim ObjectStreamWriter As New StreamWriter(ObjectFileStream)
        ExceptionMessageString = "Date: [" & Now().ToString & "]" - " & _
                                "Source: [" & pSourceObjectNameString & "]" - " & _
                                "Procedure: [" & pProcedureNameString & "]" - " & _
                                "Error Message: [" & pWriteErrorMessageString & "]"
        ObjectStreamWriter.WriteLine(ExceptionMessageString)
        If Not ObjectStreamWriter Is Nothing Then
            ObjectStreamWriter.Flush()
            ObjectStreamWriter.Close()
        End If
        If Not ObjectFileStream Is Nothing Then
            ObjectFileStream.Close()
        End If
    Catch exError As Exception
        pErrorMessageString = "Unable to write to Application Log File. Contact your Application Administrator." &
            exError.Message
    End Try
End Sub
```

Listing 10: Procedure WriteExceptionErrorToFile() code to write exception errors to a log text file

To provide some code examples, I have decided to use the procedure WriteExceptionErrorToFile() with MySQL data load, insert, update and delete. First of all, let's look at the error occurred when the MySQL ADO.NET connection object fails because of changing the name of the database. How do we call the procedure WriteExceptionErrorToFile() from our form object WritingExceptionErrorsForm (Listing 6)? The procedure ConnectionButton1_Click() shown in Listing 11 provides the code with a simple call using our class object as ExceptionClassObject.WriteLine(ExceptionMessageString). If an error occurred during MySQL server connection, the procedure MySQLOpenConnection() will return an error message ErrorMessageString. A message will alert the user with the error occurred and it'll be stored in the log file. As you can see in procedure WriteExceptionErrorToFile() the application error source is defined as the name of the form object Me.Name and the error location is the procedure ConnectionButton1_Click(). This will allow Application Developers to find the errors and fix them quickly.

```
Private Sub ConnectionButton1_Click(ByVal sender As System.Object, _
                                     ByVal e As System.EventArgs) _
    Handles ConnectionButton1.Click
    Call ExceptionClassObject.MySQLOpenConnection(MySQLConnectionString, _
                                                  ErrorMessageString)

    ' More code...
    If Not ErrorMessageString Is Nothing Then
        MessageBox.Show("An Application Error Occurred. " & _
                        ErrorMessageString, _
                        "Exception Error Handling", _
```

```
                MessageBoxButtons.OK, _  
                MessageBoxIcon.Warning)  
    Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _  
                Me.Name, _  
                "ConnectionButton1_Click", _  
                ErrorMessageString, _  
                WriteErrorMessageString)  
  
End If  
    Call ExceptionClassObject.MySQLCloseConnection(ErrorMessageString)  
End Sub
```

Listing 11: Procedure ConnectionButton1_Click() with error log file implementation WriteExceptionErrorToFile()

Let's look at the real error when we change the name of the database. If we change the name of the database from ??? to ???_none, we get an exception error stored in the file ExceptionErrorFile.log (Listing 12). The real database name has been omitted for security reasons. As you can see from Listing 12 the MySQL Connector/NET error-description message is '1049 - Unknown database '???_none', and the application source and location is WritingExceptionErrorsForm and ConnectionButton1_Click(). I really do believe that with this line of information in the log file, any developer can go open the form WritingExceptionErrorsForm, position the click event of the button ConnectionButton1 and find the occurred error in MySQL open connection procedure MySQLOpenConnection() – it's just that simple!

```
Date: [11/13/2008 1:36:40 PM] - Source: [WritingExceptionErrorsForm] - Procedure: [ConnectionButton1_Click]  
- Error Message: [1049 - Unknown database '???_none'].
```

Listing 12: Exception error occurred when the name of the MySQL database changes

If we use the Using statement in procedure ConnectionButton1_Click(), the code will look similar to the code shown in Listing 13. As you can see the MySQL close connection procedure MySQLCloseConnection() is not required at this point because the Using statement will take care of destroying properly the connection object mMySqlConnection.

```
Private Sub ConnectionButton2_Click(ByVal sender As System.Object, _  
                ByVal e As System.EventArgs) _  
                Handles ConnectionButton2.Click  
  
    Try  
        Using mMySqlConnection As New MySqlConnection(MySQLConnectionString)  
            mMySqlConnection.Open()  
            ' More programming code...  
        End Using  
    Catch exError As MySqlException  
        ErrorMessageString = exError.Number & " - " & exError.Message  
        MessageBox.Show("An Application Error Occurred. " & _  
                ErrorMessageString, _  
                "Exception Error Handling", _  
                MessageBoxButtons.OK, _  
                MessageBoxIcon.Warning)  
        Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _  
                Me.Name, _  
                "ConnectionButton2_Click", _  
                ErrorMessageString, _  
                WriteErrorMessageString)  
  
    End Try  
End Sub
```

Listing 13: Procedure ConnectionButton2_Click() with Using statement implementation

MySQL Data Load with Exception Error Log File

I have received many questions around the Open Source world about the best approach for MySQL data load in Windows database applications. My answer to this question is always the same. Today, the best solution for this implementation is to use the combination of MySQL data reader object `MySqlDataReader` and database stored queries (stored procedures, functions and triggers). I assume that practically all Application Developers know the MySQL data reader object from the Connector/.NET data library. For example, how about designing, writing and debugging MySQL stored procedures? Well, unfortunately not a lot of Application Developers know how to do so and don't care to know it either. They think that dynamic SQL:2003 embedded in VB.NET code is good enough to do anything with MySQL database tables. The reality of application development is that dynamic SQL:2003 implementation in VB.NET code is a bad programming practice. I can't image getting any Application Developer position today without knowing how to develop stored queries in Microsoft, Oracle, IBM DB2 and Open Source database technologies. In my previous paper "[MySQL Data Loading with Lookup Tables](#)" I talked about this approach very clearly. Feel free to download the document of the paper and the example source code from my website [Visual WWW Downloads](#).

Listings 14 and 15 show the MySQL table ``data`` and user stored procedure ``usp_data_select_id_name`` script definition. I developed these database objects using [Toad for MySQL Freeware 4.1](#) version from [Quest Software, Inc.](#)

```
DROP TABLE IF EXISTS `data`;  
CREATE TABLE `data` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `name` varchar(50) NOT NULL,  
  `birthdate` date NOT NULL,  
  `numberofchildren` smallint(20) DEFAULT NULL,  
  `married` tinyint(1) DEFAULT '0',  
  `computerpc` tinyint(1) DEFAULT '0',  
  `computerlaptop` tinyint(1) DEFAULT '0',  
  `salary` double(10,2) DEFAULT NULL,  
  `comment` varchar(300) DEFAULT NULL,  
  PRIMARY KEY (`id`),  
  KEY `ix1_data` (`name`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Listing 14: Table ``data`` script definition

```
DROP PROCEDURE IF EXISTS `usp_data_select_id_name`;  
CREATE PROCEDURE `usp_data_select_id_name`()  
BEGIN  
  SELECT `data`.`id`, `data`.`name`  
  FROM `data`  
  ORDER BY `data`.`name`;  
END;
```

Listing 15: User stored procedure ``usp_data_select_id_name`` script definition

To show an example of VB.NET 2008 code for MySQL data load I used a standard `ComboBox` control (Listing 16). From this listing you can quickly see that all the MySQL .NET objects like connection `MySqlConnection`, command `MySqlCommand` and data reader `MySqlDataReader` have been declared and initialized by the `Using` statement. For this reason, as I explained earlier the `Finally` block is not required in the code. One thing to consider is that a general object variable `AnyDataValue` has been declared to get any data type by the `GetString()` method of the data reader `mMySqlDataReader`. In the `Catch` block the procedure `WriteExceptionErrorToFile()` has been called to store the exception errors in the log file if needed. The application errors source is the name of the form object `Me.Name` and the error location is the click event of the button `DataLoadComboBoxButton`.

```
Private Sub DataLoadComboBoxButton_Click(ByVal sender As System.Object, _
                                         ByVal e As System.EventArgs) _
                                         Handles DataLoadComboBoxButton.Click
Dim IdInt32 As Int32, NameString As String, AnyDataValue As Object
Try
    Using mMySqlConnection As New MySqlConnection(MySqlConnectionStrings)
        mMySqlConnection.Open()
        Using mMySqlCommand As New MySqlCommand
            With mMySqlCommand
                .Connection = mMySqlConnection
                .CommandType = CommandType.StoredProcedure
                .CommandText = "usp_data_select_id_name"
            End With
            Using mMySqlDataReader As MySqlDataReader = _
                mMySqlCommand.ExecuteReader(CommandBehavior.SingleResult)
                With DataComboBox
                    .Items.Clear()
                    If mMySqlDataReader.HasRows Then
                        .BeginUpdate()
                        Do While mMySqlDataReader.Read()
                            AnyDataValue = mMySqlDataReader.GetString(0)
                            If Not IsDBNull(AnyDataValue) Then
                                IdInt32 = Convert.ToInt32(AnyDataValue)
                            Else
                                IdInt32 = 0
                            End If
                            AnyDataValue = mMySqlDataReader.GetString(1)
                            If Not IsDBNull(AnyDataValue) Then
                                NameString = Convert.ToString(AnyDataValue)
                            Else
                                NameString = String.Empty
                            End If
                            .Items.Add(IdInt32 & " - " & NameString)
                        Loop
                        .EndUpdate()
                        .SelectedIndex = 0
                    End If
                End With
            End Using
        End Using
    End Using
Catch exError As MySqlException
    ErrorMessageString = exError.Number & " - " & exError.Message
    MessageBox.Show("An Application Error Occurred. " & _
                    ErrorMessageString, _
                    "Exception Error Handling", _
                    MessageBoxButtons.OK, _
                    MessageBoxIcon.Warning)
    Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _
                                                         Me.Name, _
                                                         "DataLoadComboBoxButton_Click", _
                                                         ErrorMessageString, _
                                                         WriteErrorMessageString)
End Try
End Sub
```



```
.CommandType = CommandType.Text
.CommandText = SQLInsertString
.ExecuteNonQuery()
End With
End Using
End Using
Catch exError As MySqlException
ErrorMessageString = exError.Number & " - " & exError.Message
MessageBox.Show("An Application Error Occurred. " & _
    ErrorMessageString, _
    "Exception Error Handling", _
    MessageBoxButtons.OK, _
    MessageBoxIcon.Warning)
Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _
    Me.Name, _
    "DataInsertButton1_Click", _
    ErrorMessageString, _
    WriteErrorMessageString)
End Try
End Sub
```

Listing 18: MySQL data insert procedure with dynamic SQL and error log file implementation
WriteExceptionErrorToFile()

The dynamic SQL SQLInsertString shown above could be stored and pre-compiled at the MySQL database engine level. We can do that by implementing this dynamic SQL inside the user stored queries (stored procedures, functions and triggers). Listing 19 shows the script of the user stored procedure `usp_data_insert`. You can easily figure out the BEGIN-END block contains the INSERT INTO SQL statement. The VALUES of this statement are input parameters of the procedure. These parameters should have the same data type and size as the columns from the table `data`. The MySQL build-in function LAST_INSERT_ID() is required to determine the latest unique inserted `id` number. This value needs to be known by the GUI for any future transaction.

```
DROP PROCEDURE IF EXISTS `usp_data_insert`;
CREATE PROCEDURE `usp_data_insert`(
  IN par_name varchar(50),
  IN par_birthdate date,
  IN par_numberofchildren smallint(20),
  IN par_married tinyint(1),
  IN par_computerpc tinyint(1),
  IN par_computerlaptop tinyint(1),
  IN par_salary double(10,2),
  IN par_comment varchar(300),
  OUT par_last_id int(11)
)
BEGIN
  INSERT INTO `data`
  (`name`,
  `birthdate`,
  `numberofchildren`,
  `married`,
  `computerpc`,
  `computerlaptop`,
  `salary`,
  `comment`)
  VALUES
  (par_name,
```

```
par_birthdate,  
par_numberofchildren,  
par_married,  
par_computerpc,  
par_computerlaptop,  
par_salary,  
par_comment);  
SET par_last_id = LAST_INSERT_ID();  
END;
```

Listing 19: User stored procedure `usp_data_insert` script to insert a new row into the table `data`

How do we call a user stored procedure and pass parameters value using MySQL Connector/NET? I think, I got this question asked every week from the Open Source world community. I have decided to write and explain the code to everyone for inserting, updating and deleting MySQL records using user stored procedures. Well, Listing 20 shows the code that will answer your question for inserting records. First of all, every input and output parameter needs to have a defined MySQL .NET parameter object MySqlCommandParameter. As you can see before the Try block, all of them have been declared and initialized. The property CommandType of the command object has been set to CommandType.StoredProcedure and the CommandText to the name of the user stored procedure “usp_data_insert”. After that, every parameter object needs to have the following properties defined: ParameterName, Direction, MySqlDbType, Size (if necessary) and Value. I would like to mention that the output parameter par_last_id needs to be defined before the command method ExecuteNonQuery() with property Direction equal to ParameterDirection.Output. After the execution method is set, you can get the value of this parameter stored in the generic object variable AnyDataValue. With the user stored procedure `usp_data_insert`, we can get the latest unique inserted `id` number as an output parameter. To get the same number with dynamic SQL you'll need to execute one more dynamic SQL using the aggregate function MAX(`id`). This is a very good advantage of stored procedures development vs. dynamic SQL embedded in VB.NET application code. Moreover, you can move complete clients business rules inside the user stored queries - just something for you to think about!

```
Private Sub DataInsertButton2_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles DataInsertButton2.Click  
    Dim NameString, BirthDateString, CommentString As String  
    Dim BirthDateDate As Date, NoOfChildrenInt32, LastIdInt32 As Int32  
    Dim MarriedInt16, ComputerPCInt16, ComputerLaptopInt16 As Int16  
    Dim SalaryDouble As Double, AnyDataValue As Object  
    ' Sample hardcoded data  
    NameString = "James White"  
    BirthDateDate = #10/10/2008#  
    BirthDateString = BirthDateDate.ToString("yyyy-MM-dd")  
    NoOfChildrenInt32 = 2  
    MarriedInt16 = 1  
    ComputerPCInt16 = 1  
    ComputerLaptopInt16 = 1  
    SalaryDouble = 65000.0  
    CommentString = "The James White record"  
  
    Dim IdMySqlParameter As New MySqlCommandParameter  
    Dim LastIdMySqlParameter As New MySqlCommandParameter  
    Dim NameMySqlParameter As New MySqlCommandParameter  
    Dim BirthDateMySqlParameter As New MySqlCommandParameter  
    Dim NumberOfChildrenMySqlParameter As New MySqlCommandParameter  
    Dim MarriedMySqlParameter As New MySqlCommandParameter  
    Dim ComputerPCMySqlParameter As New MySqlCommandParameter  
    Dim ComputerLaptopMySqlParameter As New MySqlCommandParameter  
    Dim SalaryMySqlParameter As New MySqlCommandParameter
```

```
Dim CommentMySQLParameter As New MySQLParameter
```

```
Try
```

```
Using mMySQLConnection As New MySQLConnection(MySQLConnectionString)
```

```
mMySQLConnection.Open()
```

```
Using mMySQLCommand As New MySQLCommand
```

```
With mMySQLCommand
```

```
.Connection = mMySQLConnection
```

```
.CommandType = CommandType.StoredProcedure
```

```
.CommandText = “usp_data_insert”
```

```
With NameMySQLParameter
```

```
.ParameterName = “par_name”
```

```
.Direction = ParameterDirection.Input
```

```
.MySqlDbType = MySqlDbType.VarChar
```

```
.Size = 50
```

```
.Value = NameString
```

```
End With
```

```
.Parameters.Add(NameMySQLParameter)
```

```
With BirthDateMySQLParameter
```

```
.ParameterName = “par_birthdate”
```

```
.Direction = ParameterDirection.Input
```

```
.MySqlDbType = MySqlDbType.Date
```

```
.Value = BirthDateString
```

```
End With
```

```
.Parameters.Add(BirthDateMySQLParameter)
```

```
With NumberOfChildrenMySQLParameter
```

```
.ParameterName = “par_numberofchildren”
```

```
.Direction = ParameterDirection.Input
```

```
.MySqlDbType = MySqlDbType.Int16
```

```
.Value = NoOfChildrenInt32
```

```
End With
```

```
.Parameters.Add(NumberOfChildrenMySQLParameter)
```

```
With MarriedMySQLParameter
```

```
.ParameterName = “par_married”
```

```
.Direction = ParameterDirection.Input
```

```
.MySqlDbType = MySqlDbType.Int16
```

```
.Value = MarriedInt16
```

```
End With
```

```
.Parameters.Add(MarriedMySQLParameter)
```

```
With ComputerPCMySQLParameter
```

```
.ParameterName = “par_computerpc”
```

```
.Direction = ParameterDirection.Input
```

```
.MySqlDbType = MySqlDbType.Int16
```

```
.Value = ComputerPCInt16
```

```
End With
```

```
.Parameters.Add(ComputerPCMySQLParameter)
```

```
With ComputerLaptopMySQLParameter
```

```
.ParameterName = “par_computerlaptop”
```

```
.Direction = ParameterDirection.Input
```

```
.MySqlDbType = MySqlDbType.Int16
```

```
.Value = ComputerLaptopInt16
```

```
End With
```

```
.Parameters.Add(ComputerLaptopMySQLParameter)
```

```
With SalaryMySQLParameter
```

```
.ParameterName = “par_salary”
```

```
.Direction = ParameterDirection.Input
```

```
.MySqlDbType = MySqlDbType.Double
```

```
.Value = SalaryDouble
End With
.Parameters.Add(SalaryMySQLParameter)
With CommentMySQLParameter
.ParameterName = "par_comment"
.Direction = ParameterDirection.Input
.MySqlDbType = MySqlDbType.VarChar
.Size = 300
.Value = CommentString
End With
.Parameters.Add(CommentMySQLParameter)
With LastIdMySQLParameter
.ParameterName = "par_last_id"
.Direction = ParameterDirection.Output
.MySqlDbType = MySqlDbType.Int32
.Value = LastIDInt32
End With
.Parameters.Add(LastIdMySQLParameter)
.ExecuteNonQuery()
AnyDataValue = .Parameters("par_last_id").Value
If Not IsDBNull(AnyDataValue) Then
    LastIDInt32 = Convert.ToInt32(AnyDataValue)
Else
    LastIDInt32 = Nothing
End If
End With
End Using
End Using
Catch exError As MySQLException
    ErrorMessageString = exError.Number & " - " & exError.Message
    MessageBox.Show("An Application Error Occurred. " & _
        ErrorMessageString, _
        "Exception Error Handling", _
        MessageBoxButtons.OK, _
        MessageBoxIcon.Warning)
    Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _
        Me.Name, _
        "DataInsertButton2_Click", _
        ErrorMessageString, _
        WriteErrorMessageString)
End Try
End Sub
```

Listing 20: MySQL data insert procedure with user stored procedure “usp_data_insert” and error log file implementation WriteExceptionErrorToFile()

The following error occurred when change the name of the user stored procedure to “usp_data_insert_nore” (Listing 21).

```
Date: [11/17/2008 11:48:54 AM] - Source: [WritingExceptionErrorsForm] - Procedure: [DataInsertButton2_Click]
- Error Message: [0 - Procedure or function 'usp_data_insert_none' cannot be found in database 'vwww'.].
```

Listing 21: Exception error occurred when the name of the user stored procedure changes

MySQL Data Update with Exception Error Log File

Same as in Listing 18, Listing 22 contains the code to update MySQL table `data` with dynamic SQL defined in variable SQLUpdateString. To position the row to be updated an `id` value IdInt32 is required.

```
Private Sub DataUpdateButton1_Click(ByVal sender As System.Object, _
                                     ByVal e As System.EventArgs) _
    Handles DataUpdateButton1.Click
    Dim SQLUpdateString, NameString, BirthDateString, CommentString As String
    Dim BirthDateDate As Date, IdInt32, NoOfChildrenInt32 As Int32
    Dim MarriedInt16, ComputerPCInt16, ComputerLaptopInt16 As Int16
    Dim SalaryDouble As Double
    ' Sample hardcoded data
    IdInt32 = 23
    NameString = "John Smith"
    BirthDateDate = #1/1/2008#
    BirthDateString = BirthDateDate.ToString("yyyy-MM-dd")
    NoOfChildrenInt32 = 3
    MarriedInt16 = 1
    ComputerPCInt16 = 0
    ComputerLaptopInt16 = 1
    SalaryDouble = 45000.0
    CommentString = "The John Smith record"
    Try
        SQLUpdateString = "UPDATE `data` SET `name` = " & NameString _
            & ", `birthdate` = " & BirthDateString _
            & ", `numberofchildren` = " & NoOfChildrenInt32 _
            & ", `married` = " & MarriedInt16 _
            & ", `computerpc` = " & ComputerPCInt16 _
            & ", `computerlaptop` = " & ComputerLaptopInt16 _
            & ", `salary` = " & SalaryDouble _
            & ", `comment` = " & CommentString & " " _
            & "WHERE `id` = " & IdInt32
        Using mMySqlConnection As New MySqlConnection(MySqlConnectionStrings)
            mMySqlConnection.Open()
            Using mMySqlCommand As New MySqlCommand
                With mMySqlCommand
                    .Connection = mMySqlConnection
                    .CommandType = CommandType.Text
                    .CommandText = SQLUpdateString
                    .ExecuteNonQuery()
                End With
            End Using
        End Using
    Catch exError As MySqlException
        ErrorMessageString = exError.Number & " - " & exError.Message
        MessageBox.Show("An Application Error Occurred. " & _
            ErrorMessageString, _
            "Exception Error Handling", _
            MessageBoxButtons.OK, _
            MessageBoxIcon.Warning)
        Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _
            Me.Name, _
            "DataUpdateButton1_Click", _
            ErrorMessageString, _
            WriteErrorMessageString)
    End Try
End Sub
```

Listing 22: MySQL data update procedure with dynamic SQL and error log file implementation
WriteExceptionErrorToFile()

Listing 23 shows the script of the user stored procedure `usp_data_update`. As we can see the input parameter par_id is required in the WHERE clause statement to position a row to be updated.

```
DROP PROCEDURE IF EXISTS `usp_data_update`;
CREATE PROCEDURE `usp_data_update`(
    IN par_id int(11),
    IN par_name varchar(50),
    IN par_birthdate date,
    IN par_numberofchildren smallint(20),
    IN par_married tinyint(1),
    IN par_computerpc tinyint(1),
    IN par_computerlaptop tinyint(1),
    IN par_salary double(10,2),
    IN par_comment varchar(300)
)
BEGIN
    UPDATE `data`
    SET `name` = par_name,
        `birthdate` = par_birthdate,
        `numberofchildren` = par_numberofchildren,
        `married` = par_married,
        `computerpc` = par_computerpc,
        `computerlaptop` = par_computerlaptop,
        `salary` = par_salary,
        `comment` = par_comment
    WHERE `id` = par_id;
END;
```

Listing 23: User stored procedure `usp_data_update` script to update a row in table `data`

In Listing 24 we can see how to update MySQL table `data` with user stored procedure “usp_data_update”. Because an `id` value is required a parameter object IdMySQLParameter has been defined for it.

```
Private Sub DataUpdateButton2_Click(ByVal sender As System.Object, _
                                   ByVal e As System.EventArgs) _
    Handles DataUpdateButton2.Click
    Dim NameString, BirthDateString, CommentString As String
    Dim BirthDateDate As Date, NoOfChildrenInt32, IdInt32 As Int32
    Dim MarriedInt16, ComputerPCInt16, ComputerLaptopInt16 As Int16
    Dim SalaryDouble As Double
    ' Sample hardcoded data
    IdInt32 = 45
    NameString = "Ernest Bonat"
    BirthDateDate = #10/10/2008#
    BirthDateString = BirthDateDate.ToString("yyyy-MM-dd")
    NoOfChildrenInt32 = 2
    MarriedInt16 = 1
    ComputerPCInt16 = 1
    ComputerLaptopInt16 = 1
    SalaryDouble = 95000.0
    CommentString = "The James White record"

    Dim IdMySQLParameter As New MySQLParameter
    Dim NameMySQLParameter As New MySQLParameter
```

```
Dim BirthDateMySqlParameter As New MySqlParameter
Dim NumberOfChildrenMySqlParameter As New MySqlParameter
Dim MarriedMySqlParameter As New MySqlParameter
Dim ComputerPCMySqlParameter As New MySqlParameter
Dim ComputerLaptopMySqlParameter As New MySqlParameter
Dim SalaryMySqlParameter As New MySqlParameter
Dim CommentMySqlParameter As New MySqlParameter

Try
    Using mMySqlConnection As New MySqlConnection(MySQLConnectionString)
        mMySqlConnection.Open()
        Using mMySqlCommand As New MySqlCommand
            With mMySqlCommand
                .Connection = mMySqlConnection
                .CommandType = CommandType.StoredProcedure
                .CommandText = "usp_data_update"
                With IdMySqlParameter
                    .ParameterName = "par_id"
                    .Direction = ParameterDirection.Input
                    .MySqlDbType = MySqlDbType.Int32
                    .Value = IdInt32
                End With
                .Parameters.Add(IdMySqlParameter)
                With NameMySqlParameter
                    .ParameterName = "par_name"
                    .Direction = ParameterDirection.Input
                    .MySqlDbType = MySqlDbType.VarChar
                    .Size = 50
                    .Value = NameString
                End With
                .Parameters.Add(NameMySqlParameter)
                With BirthDateMySqlParameter
                    .ParameterName = "par_birthdate"
                    .Direction = ParameterDirection.Input
                    .MySqlDbType = MySqlDbType.Date
                    .Value = BirthDateString
                End With
                .Parameters.Add(BirthDateMySqlParameter)
                With NumberOfChildrenMySqlParameter
                    .ParameterName = "par_numberofchildren"
                    .Direction = ParameterDirection.Input
                    .MySqlDbType = MySqlDbType.Int16
                    .Value = NoOfChildrenInt32
                End With
                .Parameters.Add(NumberOfChildrenMySqlParameter)
                With MarriedMySqlParameter
                    .ParameterName = "par_married"
                    .Direction = ParameterDirection.Input
                    .MySqlDbType = MySqlDbType.Int16
                    .Value = MarriedInt16
                End With
                .Parameters.Add(MarriedMySqlParameter)
                With ComputerPCMySqlParameter
                    .ParameterName = "par_computerpc"
                    .Direction = ParameterDirection.Input
                    .MySqlDbType = MySqlDbType.Int16
                    .Value = ComputerPCInt16
                End With
            End With
        End Using
    End Using
End Try
```

```
End With
.Parameters.Add(ComputerPCMySQLParameter)
With ComputerLaptopMySQLParameter
.ParameterName = "par_computerlaptop"
.Direction = ParameterDirection.Input
.MySqlDbType = MySqlDbType.Int16
.Value = ComputerLaptopInt16
End With
.Parameters.Add(ComputerLaptopMySQLParameter)
With SalaryMySQLParameter
.ParameterName = "par_salary"
.Direction = ParameterDirection.Input
.MySqlDbType = MySqlDbType.Double
.Value = SalaryDouble
End With
.Parameters.Add(SalaryMySQLParameter)
With CommentMySQLParameter
.ParameterName = "par_comment"
.Direction = ParameterDirection.Input
.MySqlDbType = MySqlDbType.VarChar
.Size = 300
.Value = CommentString
End With
.Parameters.Add(CommentMySQLParameter)
.ExecuteNonQuery()
End With
End Using
End Using
Catch exError As MySqlException
ErrorMessageString = exError.Number & " - " & exError.Message
MessageBox.Show("An Application Error Occurred. " & _
    ErrorMessageString, _
    "Exception Error Handling", _
    MessageBoxButtons.OK, _
    MessageBoxIcon.Warning)
Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _
    Me.Name, _
    "DataUpdateButton2_Click", _
    ErrorMessageString, _
    WriteErrorMessageString)
End Try
End Sub
```

Listing 24: MySQL data update procedure with user stored procedure “usp_data_update” and error log file implementation WriteExceptionErrorToFile()

MySQL Data Delete with Exception Error Log File

Delete a row from MySQL table `data` is a simple task. Using the DELETE FROM statement in dynamic SQL variable SQLDeleteString with an `id` value, it allows us to delete any selected row (Listing 25).

```
Private Sub DataDeleteButton1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles DataDeleteButton1.Click

Dim SQLDeleteString As String
Dim IdInt32 As Int32
' Sample hardcoded data
```

```
IdInt32 = 11
Try
    SQLDeleteString = "DELETE FROM `data` WHERE `id` = " & IdInt32
    Using mMySQLConnection As New MySqlConnection(MySqlConnectionStrings)
        mMySQLConnection.Open()
        Using mMySQLCommand As New MySqlCommand
            With mMySQLCommand
                .Connection = mMySQLConnection
                .CommandType = CommandType.Text
                .CommandText = SQLDeleteString
                .ExecuteNonQuery()
            End With
        End Using
    End Using
Catch exError As MySqlException
    ErrorMessageString = exError.Number & " - " & exError.Message
    MessageBox.Show("An Application Error Occurred. " & _
        ErrorMessageString, _
        "Exception Error Handling", _
        MessageBoxButtons.OK, _
        MessageBoxIcon.Warning)
    Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _
        Me.Name, _
        "DataDeleteButton1_Click", _
        ErrorMessageString, _
        WriteErrorMessageString)
End Try
End Sub
```

Listing 25: MySQL data delete procedure with dynamic SQL and error log file implementation
WriteExceptionErrorToFile()

Listing 26 shows the script of the user stored procedure `usp_data_delete`. As we can see the input parameter par_id is also required for the WHERE clause statement to position a row to be deleted.

```
DROP PROCEDURE IF EXISTS `usp_data_delete`;
CREATE PROCEDURE `usp_data_delete` (
    IN par_id int(11)
)
BEGIN
    DELETE FROM `data`
    WHERE `id` = par_id;
END;
```

Listing 26: User stored procedure `usp_data_delete` script to delete a row from the table `data`

Listing 27 shows how to delete a row from MySQL table `data` with user stored procedure “usp_data_delete”. As I already explained, in this case, the only required MySQL parameter should be for column `id`.

```
Private Sub DataDeleteButton2_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles DataDeleteButton2.Click

    Dim IdInt32 As Int32
    ' Sample hardcoded data
    IdInt32 = 20
    Dim IdMySQLParameter As New MySQLParameter
    Try
```

```
Using mMySqlConnection As New MySqlConnection(MySQLConnectionString)
mMySqlConnection.Open()
Using mMySqlCommand As New MySqlCommand
    With mMySqlCommand
        .Connection = mMySqlConnection
        .CommandType = CommandType.StoredProcedure
        .CommandText = "usp_data_delete"
        With IdMySqlParameter
            .ParameterName = "par_id"
            .Direction = ParameterDirection.Input
            .MySqlDbType = MySqlDbType.Int32
            .Value = IdInt32
        End With
        .Parameters.Add(IdMySqlParameter)
        .ExecuteNonQuery()
    End With
End Using
End Using
Catch exError As MySqlException
    ErrorMessageString = exError.Number & " - " & exError.Message
    MessageBox.Show("An Application Error Occurred. " & _
        ErrorMessageString, _
        "Exception Error Handling", _
        MessageBoxButtons.OK, _
        MessageBoxIcon.Warning)
    Call ExceptionClassObject.WriteExceptionErrorToFile(ExceptionErrorFileString, _
        Me.Name, _
        "DataDeleteButton2_Click", _
        ErrorMessageString, _
        WriteErrorMessageString)
End Try
End Sub
```

Listing 27: MySQL data delete procedure with stored procedure “usp_data_delete” and error log file implementation WriteExceptionErrorToFile()

Conclusions

Possible conclusions from this paper include:

- Exception error handling implementation is always required for any MySQL/VB.NET Widows database applications
- Try block should not be used to dispose unmanaged objects. Unmanaged objects dispose implementation, which should be done in the Finally block
- Using statement simplifies the VB.NET code by disposing .NET unmanaged recourses
- The combination of Try...Catch...Finally and Using statements provide a flexible and easy way to read/write VB.NET code
- Exception errors should be store for future database application upgrades and auditing
- Use log text file provides an easy way to store occurred application exception errors

To download the source codes and a PDF format for this article go to [Visual WWW Downloads](#).

Visual WWW Legal Notice

This document contains freeware information. The information described in this document may be used or copied or transmitted for computer programming development purposes only without the written permission of Visual WWW, Inc. No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose.

The information contained in this document is subject to change without notice. Visual WWW makes no warranty of any kind with respect to this information. VISUAL WWW SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTY OF THE MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Visual WWW shall not be liable for any direct, indirect, incidental, consequential, or other damage alleged in connection with the furnishing or use of this information.

Visual WWW, Inc. and Visual WWW logo are registered trademarks of Visual WWW, Inc. Other trademarks and registered trademarks used in this document are property of their respective owners.