

Define and Store MySQL Connection String in VB.NET 2005

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



Visual WWW, Inc.

'Create by Developers for Developers'

www.evisualwww.com | info@evisualwww.com

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

Table of Content

Introduction	2
Required Software	2
MySQL Connection String Definition	2
Don't Hardcode Your MySQL Connection String	3
Passing MySQL Connection String as a Constructor of a Class Object	4
Open and Close MySQL ADO.NET Connection Object	5
MySQL Data Load using VB.NET 2003 and 2005	6
Conclusion	13
Visual WWW Legal Notice	13

Introduction

In Windows and Internet web business applications development the connection to the MySQL database server is critical and requires high-level of security. In ADO.NET database technology the connection is defined in the Connection String property of the connection object. Defining and storing the MySQL Connection String properly is an important application setting task for developers today. In this article I'll show you how to setup and secure MySQL Connection String for Windows application development using VB.NET 2005. Also the difference of programming code between VB.NET 2003 and 2005 will be provided for data loading using MySQL Connector/NET 5.0.3 database driver.

Required Software

- [MySQL Database Server 5.0.27](#)
- [MySQL Connector Net 5.0.3](#)
- [Toad for MySQL Freeware 2.0.3](#)
- [Microsoft Visual Basic 2005 Express Edition](#)

MySQL Connection String Definition

The Connection String is a property of the ADO.NET connection object MySqlConnection. It gets or sets the string used to connect to a MySQL server database. The Connection String contains a series of parameters (keywords) separated by semicolons. The order of these parameters is not mandatory and they are not case sensitive. These parameters have specific names defined in the .NET provider such as, Server, Database, User Id (Uid), Password (Pwd), Connect Timeout, Persist Security Info, etc. The .NET provider for these parameters determines the default values and description too. If a parameter is not included in the Connection String its default value is assumed. The table definition for these parameters (name, default and description) is defined in the MySQL help file MySql.Data.chm located in C:\Program Files\MySQL\MySQL Connector Net 5.0.3\Documentation for MySQL Connector Net 5.0.3 version.

Listing 1 shows a simple example of MySQL Connection String:

```
Dim MySqlConnectionString As String
MySQLConnectionString = "Server=myServer;" & _
    "Database=myDatabase;" & _
    "Uid=myUserID;" & _
    "Pwd=myPassword;" & _
    "Connect Timeout=30;"
```

Listing 1: MySQL Connection String example

For example, the above code does not include Persist Security Info parameter, but by default its value is false. The same idea is applied to all parameters that have been not included in this example.

After the Connection String is defined properly, it can be used by the ADO.NET connection object MySqlConnection as a constructor or assigned property. In Listing 2 the Connection String MySqlConnectionString has been used as a constructor. The Open() method of the ADO.NET connection object establishes a new open connection from the connection pool if one is not available. Otherwise, it will use an already created one.

```
Dim MySQLConnectionString As String
Dim MyADOConnection As MySqlConnection

MySQLConnectionString = "Server=myServer;" & _
    "Database=myDatabase;" & _
    "Uid=myUserID;" & _
    "Pwd=myPassword;" & _
    "Connect Timeout=30;"
MyADOConnection = New MySqlConnection(MySQLConnectionString)
MyADOConnection.Open()
```

Listing 2: Connection String passed as a constructor of the ADO.NET connection object

Listing 3 shows how the Connection String is used as a property of the ADO.NET connection object MySqlConnection.

```
Dim MySQLConnectionString As String
Dim MyADOConnection As MySqlConnection

MySQLConnectionString = "Server=myServer;" & _
    "Database=myDatabase;" & _
    "Uid=myUserID;" & _
    "Pwd=myPassword;" & _
    "Connect Timeout=30;"
MyADOConnection = New MySqlConnection()
MyADOConnection.ConnectionString = MySQLConnectionString
MyADOConnection.Open()
```

Listing 3: Connection String used as a property of the ADO.NET connection object

Don't Hardcode Your MySQL Connection String

Hardcode (embed) the Connection String in your application's code is not secure and is hard to maintain for futures upgrades. Let's look at these two main issues:

1. A malicious Connection String injection attack can occur when a dynamic string concatenation has been hardcode in your program. At least, as you know, Server, Database, User Id and Password are available at this point (Listing 1). Based on that, the injection attacker can completely destroy the database (tables and records deletion) or receive important secure information about the business (bank accounts, employee's info, etc.)
2. If any of the Connection String parameters change (for example, Server or Database name) for any reason the entire application needs to be compiling and redeploying again to all client machines.

Because of these two reasons hardcode the Connection String value in your application's code is a bad programming practice. I can see many computer programming books and papers today with this important security and maintenance problem. So, a good Application Developer should never hardcode the Connection String in their database business applications. In fact, you should not hardcode any application setting in your code.

One way to take care of this problem today is storing your Connection String in the application configuration file. This configuration file can be deployed together with the application and properly secured. VS.NET 2005 IDE made it easy and sufficient to store application settings in the configuration file. To do this, right click in the solution project name and select Properties. Then click on the Setting tab and type in the information below (Table 1). Make sure to write your Server, Database, Uid and Pwd parameters.

Name	Type	Scope	Value
MySQLConnectionString	(Connection string)	Application	Server=xxx;Database=xxx;Uid=xxx;Pwd=xxx;

Table 1: Application setting tab table

In Windows applications, the configuration file app.config uses the section <connectionStrings> to store the Connection String value as shown below (Listing 4):

```
<connectionStrings>  
  <add name="MySQLConnectionString.My.MySettings.MySQLConnectionString"  
    connectionString="Server=xxx;Database=xxx;Uid=xxx;Pwd=xxx;" />  
</connectionStrings>
```

Listing 4: Connection String section of the application configuration file app.config

In VB.NET 2005 the Connection String value can be retrieved easily by using My application object as the following (Listing 5):

```
Dim mMySQLConnectionString As String = My.Settings.MySQLConnectionString
```

Listing 5: Retrieving Connection String value

In this case, as you can see in Listing 6 below, the Connection String has not been hardcode at all. Because of that any possible injection attacks will never be able to get it. One more thing, if for any reason any of the Connection String parameters change, the update should be done easily in the application configuration file app.config. At this point application redeployment is not necessary.

```
Dim mMySQLConnectionString As String = My.Settings.MySQLConnectionString  
Dim MyADOConnection As New MySqlConnection  
MyADOConnection.ConnectionString = MySQLConnectionString  
MyADOConnection.Open()
```

Listing 6: Open MySQL connection object with the stored Connection String

Passing MySQL Connection String as a Constructor of a Class Object

After getting the Connection String value at the Form level we need to pass it to the class object. Because we need to declare and create an instant of our class, a simple and easy way to pass the Connection String value is overloading the constructor (Sub New() procedure). Passing the Connection String as a constructor and creating a ReadOnly property will make it available for the entire class (Listing 7).

```
Private mMySQLConnectionString As String
```

```
Public Sub New(ByVal pMySQLConnectionString As String)
    mMySQLConnectionString = pMySQLConnectionString
End Sub

Public ReadOnly Property GetMySQLConnectionString() As String
    Get
        Return (mMySQLConnectionString)
    End Get
End Property
```

Listing 7: Pass and get the Connection String using a constructor and read only property procedures

Open and Close MySQL ADO.NET Connection Object

Generic function procedures used to open and close MySQL ADO.NET connection object are shown in Listing 8 and 9 respectively. Open and close ADO.NET connection object to MySQL server properly its important task for any Application Developer. To open the connection it requires three simple steps: create a new instant of the connection object MySqlConnection, pass the Connection String as a constructor or set the ConnectionString property to a value and use the Open() method (Listing 8). To close the connection we need to use the IsNothing() function and check for the connection state (Listing 9). Because an ADO.NET connection object in .NET Framework represents an unmanaged resource, it needs to be destroyed from memory by the program. The Close() method will deallocate the resource by the referenced connection object and setting it to Nothing will enable the Garbage Collection (GC). As you can see the exception error message from MySqlException class is returned back to the calling procedure. These two procedures should be calling from a class and possible from a database their implementation.

```
Private Function MySQLADODConnectionOpen(ByRef pErrorMessageString As String) As Boolean
    Try
        mMySqlConnection = New MySqlConnection
        mMySqlConnection.ConnectionString = mMySQLConnectionString
        mMySqlConnection.Open()
        Return (True)
    Catch exError As MySqlException
        pErrorMessageString = exError.Message
        Return (False)
    End Try
End Function
```

Listing 8: MySQL open ADO.NET connection object function

```
Public Function MySQLADODConnectionClose(ByRef pErrorMessageString As String) As Boolean
    Try
        If Not IsNothing(mMySqlConnection) Then
            If mMySqlConnection.State = ConnectionState.Open Then
                mMySqlConnection.Close()
            End If
            mMySqlConnection = Nothing
        End If
        Return (True)
    Catch exError As MySqlException
        pErrorMessageString = exError.Message
        Return (False)
    End Try
End Function
```

Listing 9: MySQL close ADO.NET connection object function

MySQL Data Load using VB.NET 2003 and 2005

Let's look at two examples of data load using VB.NET 2003 and 2005 with stored Connection String in application configuration file (Listing 4). Before we start looking at the VB.NET code the MySQL server needs to have two objects, a USA state table (`state`) with data (Listing 10) and a stored procedure (`usp_state_select_name`) to select and order by state name (Listing 11). As you can see the USA state table includes state name, state abbreviation, postal (zip) code and state capital). Both SQL scripts were developed using [Toad for MySQL 2.0.3](#) freeware version. [Quest Software, Inc.](#) is the creator of Toad Database Management Tool for MySQL, IBM DB2, Oracle and MS SQL servers. Feel free to use any tools you like to create these two objects on your MySQL server. But make sure that you use MySQL 5.0 server or highest.

```
DROP TABLE IF EXISTS `states`;
CREATE TABLE `states` (
  `statename` varchar(20) NOT NULL,
  `abbrev` varchar(10) NOT NULL,
  `postal` char(2) NOT NULL,
  `capital` varchar(20) NOT NULL,
  PRIMARY KEY (`postal`),
  KEY `statename` (`statename`),
  KEY `capital` (`capital`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Alaska','Alaska','AK','Juneau');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Alabama','Ala.','AL','Montgomery');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Arkansas','Ark.','AR','Little Rock');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Arizona','Ariz.','AZ','Phoenix');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('California','Calif.','CA','Sacramento');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Colorado','Colo.','CO','Denver');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Connecticut','Conn.','CT','Hartford');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Dist. of Columbia','D.C.','DC','Washington');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Delaware','Del.','DE','Dover');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Florida','Fla.','FL','Tallahassee');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Georgia','Ga.','GA','Atlanta');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Guam','Guam','GU','Agaña');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Hawaii','Hawaii','HI','Honolulu');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Iowa','Iowa','IA','Des Moines');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Idaho','Idaho','ID','Boise');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
```

```
('Illinois','Ill.','IL','Springfield');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Indiana','Ind.','IN','Indianapolis');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Kansas','Kans.','KS','Topeka');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Kentucky','Ky.','KY','Frankfort');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Louisiana','La.','LA','Baton Rouge');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Massachusetts','Mass.','MA','Boston');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Maryland','Md.','MD','Annapolis');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Maine','Maine','ME','Augusta');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Michigan','Mich.','MI','Lansing');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Minnesota','Minn.','MN','St Paul');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Missouri','Mo.','MO','Jefferson City');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Mississippi','Miss.','MS','Jackson');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Montana','Mont.','MT','Helena');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('North Carolina','N.C.','NC','Raleigh Durham');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('North Dakota','N.D.','ND','Bismarck');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Nebraska','Nebr.','NE','Lincoln');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('New Hampshire','N.H.','NH','Concord');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('New Jersey','N.J.','NJ','Trenton');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('New Mexico','N.M.','NM','Santa Fe');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Nevada','Nev.','NV','Carson City');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('New York','N.Y.','NY','Albany');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Ohio','Ohio','OH','Columbus');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Oklahoma','Okla.','OK','Oklahoma City');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Oregon','Ore.','OR','Salem');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Pennsylvania','Pa.','PA','Harrisburg');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Puerto Rico','P.R.','PR','San Juan');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('Rhode Island','R.I.','RI','Providence');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('South Carolina','S.C.','SC','Columbia');
insert into `states` (`statename`,`abbrev`,`postal`,`capital`) values
('South Dakota','S.D.','SD','Pierre');
```

```
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Tennessee','Tenn.','TN','Nashville');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Texas','Tex.','TX','Austin');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Utah','Utah','UT','Salt Lake City');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Virginia','Va.','VA','Richmond');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Virgin Islands','V.I.','VI','Charlotte Amalie');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Vermont','Vt.','VT','Montpelier');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Washington','Wash.','WA','Olympia');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Wisconsin','Wis.','WI','Madison');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('West Virginia','W.Va.','WV','Charleston');
insert into `states` (`statename`, `abbrev`, `postal`, `capital`) values
('Wyoming','Wyo.','WY','Cheyenne');
```

Listing 10: State table definition and SQL insert data statements

```
DROP PROCEDURE IF EXISTS `usp_state_select_name`;
CREATE PROCEDURE `usp_state_select_name`()
BEGIN
    SELECT `statename`
    FROM `states`
    ORDER BY `statename`;
END;
```

Listing 11: User stored procedure to select and order by USA state name

To demonstrate the data load using VB.NET 2003 and 2005, I created a simple solution project with a Form shown in Figure 1. This Form loads the states name in a ListBox control and selects the first item.

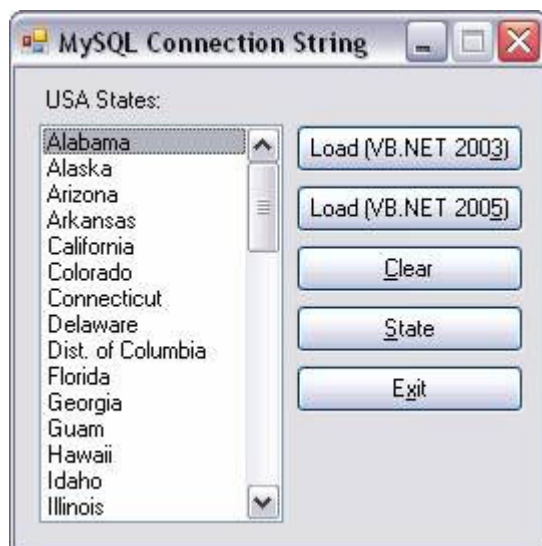


Figure 1: MySQL Connection String Form

The click event code of the Load (VB.NET 2003) button is shown in Listing 12. The code creates a new instant of a custom class MySQLConnectionStringClass and passes the Connection String mMySQLConnectionString as a constructor. The sub procedure ListBoxLoadVBNET2003() is called (Listing 13) with state ListBox control ListBoxStates and returned error message mErrorMessageString parameters. If an error occurs a message will be displayed to the end-user.

```
Private Sub LoadVBNET2003Button_Click(ByVal sender As System.Object,
                                     ByVal e As System.EventArgs)
    Handles LoadVBNET2003Button.Click
    Cursor = Cursors.WaitCursor
    Dim ConnectionStringObject As New
    MySQLConnectionStringClass(mMySQLConnectionString)
    Call ConnectionStringObject.ListBoxLoadVBNET2003(ListBoxStates, _
                                                    mErrorMessageString)

    If Not IsNothing(mErrorMessageString) Then
        Cursor = Cursors.Default
        MessageBox.Show(mErrorMessageString, _
                        "MySQL Connection String", _
                        MessageBoxButtons.OK, _
                        MessageBoxIcon.Information)
    End If
    If Not IsNothing(ConnectionStringObject) Then
        ConnectionStringObject.Dispose()
        ConnectionStringObject = Nothing
    End If
    Cursor = Cursors.Default
End Sub
```

Listing 12: Click event of the Load (VB.NET 2003) button

The custom class object ConnectionStringObject must be released from memory. To do that in VB.NET a disposable class ObjectDisposeClass must be created with IDisposable interface implementation as shown Listing 14. This code was automatically generated in VB.NET 2005. Because the MySQLConnectionStringClass class inherits the ObjectDisposeClass class, the release of the class object ConnectionStringObject is simple by using the Dispose() method ConnectionStringObject.Dispose() (Listing 12).

```
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
```

Listing 14: IDisposable interface class implementation generated by VB.NET

The Listing 13 shows the code of the ListBox data load using VB.NET 2003. This code is based on standard Try...Catch...Finally block. In the Try block the ADO.NET connection object is open using the function MySQLADODConnectionOpen() (Listing 8). A new instant of the command object mMySQLCommand is created to get the data reader object mMySQLDataReader by using the ExecuteReader() method with single result set. As you can see the user stored procedure usp_state_select_name was assigned to the CommandText property of the Command object. By looping the data reader object mMySQLDataReader it will get the values of the state name to be added to the ListBox by using the Items.Add() method. In the Finally block the data reader, command and connection objects must be disposed or closed properly. This general code structure belongs to VB.NET 2003 programming style.

```
Public Sub ListBoxLoadVBNET2003(ByVal pListBox As ListBox, _
                                ByRef pErrorMessageString As String)
    Dim StateNameString As String
    Try
        mReturnBoolean = MySQLADODConnectionOpen(pErrorMessageString)
        If Not IsNothing(pErrorMessageString) Then
            Exit Sub
        End If
        mMySQLCommand = New MySqlCommand
        With mMySQLCommand
            .Connection = mMySQLConnection
            .CommandType = CommandType.StoredProcedure
            .CommandText = "usp_state_select_name"
            mMySQLDataReader = .ExecuteReader(CommandBehavior.SingleResult)
        End With
        With pListBox
            .Items.Clear()
            If mMySQLDataReader.HasRows Then
                .BeginUpdate()
                Do While mMySQLDataReader.Read()
                    mObjectjValue = mMySQLDataReader.GetString(0)
                    If Not IsDBNull(mObjectjValue) Then
                        StateNameString = mObjectjValue.ToString()
                    Else
                        StateNameString = String.Empty
                    End If
                    .Items.Add(StateNameString)
                Loop
                .EndUpdate()
                .SelectedIndex = 0
            End If
        End With
        Catch exError As MySqlException
            pErrorMessageString = exError.Message
        Finally
            If Not IsNothing(mMySQLDataReader) Then
                mMySQLDataReader.Close()
                mMySQLCommand = Nothing
            End If
            If Not IsNothing(mMySQLCommand) Then
                mMySQLCommand.Dispose()
                mMySQLCommand = Nothing
            End If
            mReturnBoolean = MySQLADODConnectionClose(pErrorMessageString)
        End Sub
```

```
End Try
End Sub
```

Listing 13: Sub procedure ListBoxLoadVBNET2003() code

The Listing 15 shows the code of the click event of the Load (VB.NET 2005) button. This code is close to the code shown in the Load (VB.NET 2003) button. The only difference is that the sub procedure ListBoxLoadVBNET2005() gets a call at this time.

```
Private Sub LoadVBNET2005Button_Click(ByVal sender As System.Object,
                                       ByVal e As System.EventArgs)
    Handles LoadVBNET2005Button.Click
    Cursor = Cursors.WaitCursor
    Using ConnectionStringObject As New
        MySqlConnectionClass(mMySQLConnectionString)
        Call ConnectionStringObject.ListBoxLoadVBNET2005(ListBoxStates, _
                                                         mErrorMessageString)

        If Not IsNothing(mErrorMessageString) Then
            Cursor = Cursors.Default
            MessageBox.Show(mErrorMessageString, _
                            "MySQL Connection String", _
                            MessageBoxButtons.OK, _
                            MessageBoxIcon.Information)
        End If
    End Using
    Cursor = Cursors.Default
End Sub
```

Listing 15: Click event of the Load (VB.NET 2005) button

The sub procedure ListBoxLoadVBNET2005() shown in Listing 16 uses the Using statement. This statement was introduced in VS.NET 2005 to guarantee the disposal of unmanaged resources when the application code is finished with them. For more explanation about this statement go to MSDN Library (<http://msdn2.microsoft.com/en-us/library/htd05whh.aspx>). As you can see the Finally block is not required any more at all. At the End Using statement the data reader mMySqlDataReader, command mMySqlCommand and connection mMySqlConnection objects will be released from memory properly. Thanks to the Using statement to take care of unmanaged resources automatically for us. I did not see a lot of VB.NET developers using this new statement in their application today. I hope this article will give them some thing to look at it.

```
Public Sub ListBoxLoadVBNET2005(ByVal pListBox As ListBox, _
                                ByRef pErrorMessageString As String)
    Dim StateNameString As String
    Try
        Using mMySqlConnection As New MySqlConnection(mMySQLConnectionString)
            mMySqlConnection.Open()
            Using mMySqlCommand As New MySqlCommand
                With mMySqlCommand
                    .Connection = mMySqlConnection
                    .CommandType = CommandType.StoredProcedure
                    .CommandText = "usp_state_select_name"
                    mMySqlDataReader = .ExecuteReader(CommandBehavior.SingleResult)
                End With
                With pListBox
                    .Items.Clear()
                    If mMySqlDataReader.HasRows Then
                        .BeginUpdate()
                        Do While mMySqlDataReader.Read()

```

```
        mObjectjValue = mMySqlDataReader.GetString(0)
        If Not IsDBNull(mObjectjValue) Then
            StateNameString = mObjectjValue.ToString()
        Else
            StateNameString = String.Empty
        End If
        .Items.Add(StateNameString)
    Loop
    .EndUpdate()
    .SelectedIndex = 0
End If
End With
End Using
End Using
Catch exErr As MySqlException
    pErrorMessageString = exErr.Message
End Try
End Sub
```

Listing 16: Sub procedure ListBoxLoadVBNET2005() code

To remove all the items in the ListBox the Clear() method of the Items property must be implemented (Listing 17).

```
Private Sub ClearButton_Click(ByVal sender As System.Object,
                              ByVal e As System.EventArgs)
    Handles ClearButton.Click

    Cursor = Cursors.WaitCursor
    ListBoxStates.Items.Clear()
    Cursor = Cursors.Default
End Sub
```

Listing 17: Click event of the Clear button

The State name selected in the ListBox can be displayed by converting to string the selected item value as shown in Listing 18.

```
Private Sub StateButton_Click(ByVal sender As System.Object,
                              ByVal e As System.EventArgs)
    Handles StateButton.Click

    Dim StateNameString As String
    If ListBoxStates.Items.Count > 0 Then
        StateNameString = Convert.ToString(ListBoxStates.SelectedItem)
        MessageBox.Show(StateNameString, _
                        "MySQL Connection String", _
                        MessageBoxButtons.OK, _
                        MessageBoxIcon.Information)
    End If
End Sub
```

Listing 18: Click event of the State button

I decided to show the code of the Exit button (Listing 19) with the Close() method of the Form. Just in case, the Dispose() method of the Form was implemented in the closed event to make sure that it will be released from memory properly.

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

```
        Close()  
    End Sub  
  
    Private Sub ConnectionStringForm_FormClosed(ByVal sender As Object,  
                                                ByVal e As System.Windows.Forms.FormClosedEventArgs)  
        Handles Me.FormClosed  
        Dispose()  
    End Sub
```

Listing 19: Click and closed event of the Exit button and Form

Conclusion

Here are some conclusions that I found very necessary for you to understand. The MySQL Connection String should not be hardcode never in your application code. This will not allow injection attacks to get and/or destroy your application security information. VB.NET 2005 interface provides an easy way of storing application settings in the system configuration file. These settings can be retrieved by using My application object. VB.NET 2005 introduced a Using statement to guarantee the disposal of unmanaged resources when the application code is finished with them. This statement considerable reduces the amount of the code by eliminating the Finally block in the Try...Catch error handling statement used in VB.NET 2003.

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.