

```

Imports System
Imports System.IO
Imports System.Text
Imports System.Data
Imports System.Data.SqlClient

Namespace SqlDataProvider

    ''' <summary>
    ''' This class provides a fast and universal method for accessing SQL
    Server database.This class cannot be inherited.
    ''' </summary>
    Public NotInheritable Class SqlDatabase

#Region " Local Property Declarations "

        Dim _connectionString As String

#End Region

#Region " Constructor "

        ''' <summary>
        ''' Initializes a new instance of the ADO.SqlDatabase class.
        ''' </summary>
        ''' <param name="connectionString">The connection used to open the
SQL Server database.</param>
        Public Sub New(ByVal connectionString As String)
            _connectionString = connectionString
        End Sub

#End Region

#Region " Public Properties "

        ''' <summary>
        ''' Gets or sets the string used to open a SQL Server database.
        ''' </summary>
        ''' <returns>The connection string that includes the source database
name, and other parameters needed to establish the initial
connection.</returns>
        Public Property ConnectionString() As String
            Get
                Return _connectionString
            End Get
            Set(ByVal value As String)
                _connectionString = value
            End Set
        End Property

#End Region

#Region " Private Methods "

        Private Sub AssignParameters(ByVal cmd As SqlCommand, ByVal
cmdParameters() As SqlParameter)
            If (cmdParameters Is Nothing) Then Exit Sub

```

```

        For Each p As SqlParameter In cmd.Parameters
            cmd.Parameters.Add(p)
        Next
    End Sub

    Private Sub AssignParameters(ByVal cmd As SqlCommand, ByVal
parameterValues() As Object)
        If Not (cmd.Parameters.Count - 1 = parameterValues.Length) Then
            Throw New ApplicationException("Stored procedure's parameters and parameter
values does not match.")
        Dim i As Integer
        For Each param As SqlParameter In cmd.Parameters
            If Not (param.Direction = ParameterDirection.Output) AndAlso
Not (param.Direction = ParameterDirection.ReturnValue) Then
                param.Value = parameterValues(i)
                i += 1
            End If
        Next
    End Sub

#End Region

#Region " ExecuteNonQuery "

    ''' <summary>
    ''' Executes a Transact-SQL statement against the connection and
returns the number of rows affected.
    ''' </summary>
    ''' <param name="cmd">The Transact-SQL statement or stored procedure
to execute at the data source.</param>
    ''' <param name="cmdType">A value indicating how the
System.Data.SqlClient.SqlCommand.CommandText property is to be
interpreted.</param>
    ''' <param name="parameters">The parameters of the Transact-SQL
statement or stored procedure.</param>
    ''' <returns>The number of rows affected.</returns>
    Public Function ExecuteNonQuery(ByVal cmd As String, ByVal cmdType As
CommandType, Optional ByVal parameters() As SqlParameter = Nothing) As
Integer
        Dim connection As SqlConnection = Nothing
        Dim transaction As SqlTransaction = Nothing
        Dim command As SqlCommand = Nothing
        Dim res As Integer = -1
        Try
            connection = New SqlConnection(_connectionString)
            command = New SqlCommand(cmd, connection)
            command.CommandType = cmdType
            Me.AssignParameters(command, parameters)
            connection.Open()
            transaction = connection.BeginTransaction()
            command.Transaction = transaction
            res = command.ExecuteNonQuery()
            transaction.Commit()
        Catch ex As Exception
            If Not (transaction Is Nothing) Then
                transaction.Rollback()
            End If
        End Try
    End Function

```

```

        Throw New SqlDatabaseException(ex.Message, ex.InnerException)
    Finally
        If Not (connection Is Nothing) AndAlso (connection.State =
ConnectionState.Open) Then connection.Close()
        If Not (command Is Nothing) Then command.Dispose()
        If Not (transaction Is Nothing) Then transaction.Dispose()
    End Try
    Return res
End Function

''' <summary>
''' Executes a Transact-SQL statement against the connection and
returns the number of rows affected.
''' </summary>
''' <param name="spname">The stored procedure to execute at the data
source.</param>
''' <param name="returnValue">The returned value from stored
procedure.</param>
''' <param name="parameterValues">The parameter values of the stored
procedure.</param>
''' <returns>The number of rows affected.</returns>
Public Function ExecuteNonQuery(ByVal spname As String, ByRef
returnValue As Integer, ByVal ParamArray parameterValues() As Object) As
Integer
    Dim connection As SqlConnection = Nothing
    Dim transaction As SqlTransaction = Nothing
    Dim command As SqlCommand = Nothing
    Dim res As Integer = -1
    Try
        connection = New SqlConnection(_connectionString)
        command = New SqlCommand(spname, connection)
        command.CommandType = CommandType.StoredProcedure
        connection.Open()
        SqlCommandBuilder.DeriveParameters(command)
        Me.AssignParameters(command, parameterValues)
        transaction = connection.BeginTransaction()
        command.Transaction = transaction
        res = command.ExecuteNonQuery()
        returnValue = command.Parameters(0).Value
        transaction.Commit()
    Catch ex As Exception
        If Not (transaction Is Nothing) Then
            transaction.Rollback()
        End If
        Throw New SqlDatabaseException(ex.Message, ex.InnerException)
    Finally
        If Not (connection Is Nothing) AndAlso (connection.State =
ConnectionState.Open) Then connection.Close()
        If Not (command Is Nothing) Then command.Dispose()
        If Not (transaction Is Nothing) Then transaction.Dispose()
    End Try
    Return res
End Function

#End Region

#Region " ExecuteScalar "

```

```

    ''' <summary>
    ''' Executes the query, and returns the first column of the first row
in the result set returned by the query. Additional columns or rows are
ignored.
    ''' </summary>
    ''' <param name="cmd">The Transact-SQL statement or stored procedure
to execute at the data source.</param>
    ''' <param name="cmdType">A value indicating how the
System.Data.SqlClient.SqlCommand.CommandText property is to be
interpreted.</param>
    ''' <param name="parameters">The parameters of the Transact-SQL
statement or stored procedure.</param>
    ''' <returns>The first column of the first row in the result set, or
a null reference if the result set is empty.</returns>
    Public Function ExecuteScalar(ByVal cmd As String, ByVal cmdType As
CommandType, Optional ByVal parameters() As SqlParameter = Nothing) As Object
        Dim connection As SqlConnection = Nothing
        Dim transaction As SqlTransaction = Nothing
        Dim command As SqlCommand = Nothing
        Dim res As Object = Nothing
        Try
            connection = New SqlConnection(_connectionString)
            command = New SqlCommand(cmd, connection)
            command.CommandType = cmdType
            Me.AssignParameters(command, parameters)
            connection.Open()
            transaction = connection.BeginTransaction()
            command.Transaction = transaction
            res = command.ExecuteScalar()
            transaction.Commit()
        Catch ex As Exception
            If Not (transaction Is Nothing) Then
                transaction.Rollback()
            End If
            Throw New SqlDatabaseException(ex.Message, ex.InnerException)
        Finally
            If Not (connection Is Nothing) AndAlso (connection.State =
ConnectionState.Open) Then connection.Close()
            If Not (command Is Nothing) Then command.Dispose()
            If Not (transaction Is Nothing) Then transaction.Dispose()
        End Try
        Return res
    End Function

    ''' <summary>
    ''' Executes the query, and returns the first column of the first row
in the result set returned by the query. Additional columns or rows are
ignored.
    ''' </summary>
    ''' <param name="spname">The stored procedure to execute at the data
source.</param>
    ''' <param name="returnValue">The returned value from stored
procedure.</param>
    ''' <param name="parameterValues">The parameter values of the stored
procedure.</param>
    ''' <returns>The first column of the first row in the result set, or

```

```

a null reference if the result set is empty.</returns>
    Public Function ExecuteScalar(ByVal spname As String, ByRef
returnValue As Integer, ByVal ParamArray parameterValues() As Object) As
Object
    Dim connection As SqlConnection = Nothing
    Dim transaction As SqlTransaction = Nothing
    Dim command As SqlCommand = Nothing
    Dim res As Object = Nothing
    Try
        connection = New SqlConnection(_connectionString)
        command = New SqlCommand(spname, connection)
        command.CommandType = CommandType.StoredProcedure
        connection.Open()
        SqlCommandBuilder.DeriveParameters(command)
        Me.AssignParameters(command, parameterValues)
        transaction = connection.BeginTransaction()
        command.Transaction = transaction
        res = command.ExecuteScalar()
        returnValue = command.Parameters(0).Value
        transaction.Commit()
    Catch ex As Exception
        If Not (transaction Is Nothing) Then
            transaction.Rollback()
        End If
        Throw New SqlDatabaseException(ex.Message, ex.InnerException)
    Finally
        If Not (connection Is Nothing) AndAlso (connection.State =
ConnectionState.Open) Then connection.Close()
        If Not (command Is Nothing) Then command.Dispose()
        If Not (transaction Is Nothing) Then transaction.Dispose()
    End Try
    Return res
End Function

#End Region

#Region " ExecuteReader "

    ''' <summary>
    ''' Sends the System.Data.SqlClient.SqlCommand.CommandText to the
System.Data.SqlClient.SqlCommand.Connection, and builds a
System.Data.SqlClient.SqlDataReader using one of the
System.Data.CommandBehavior values.
    ''' </summary>
    ''' <param name="cmd">The Transact-SQL statement or stored procedure
to execute at the data source.</param>
    ''' <param name="cmdType">A value indicating how the
System.Data.SqlClient.SqlCommand.CommandText property is to be
interpreted.</param>
    ''' <param name="parameters">The parameters of the Transact-SQL
statement or stored procedure.</param>
    ''' <returns>A System.Data.SqlClient.SqlDataReader object.</returns>
    Public Function ExecuteReader(ByVal cmd As String, ByVal cmdType As
CommandType, Optional ByVal parameters() As SqlParameter = Nothing) As
IDataReader
        Dim connection As SqlConnection = Nothing
        Dim command As SqlCommand = Nothing

```

```

Dim res As SqlDataReader = Nothing
Try
    connection = New SqlConnection(_connectionString)
    command = New SqlCommand(cmd, connection)
    command.CommandType = cmdType
    Me.AssignParameters(command, parameters)
    connection.Open()
    res = command.ExecuteReader(CommandBehavior.CloseConnection)
Catch ex As Exception
    Throw New SqlDatabaseException(ex.Message, ex.InnerException)
End Try
Return CType(res, IDataReader)
End Function

''' <summary>
''' Sends the System.Data.SqlClient.SqlCommand.CommandText to the
System.Data.SqlClient.SqlCommand.Connection, and builds a
System.Data.SqlClient.SqlDataReader using one of the
System.Data.CommandBehavior values.
''' </summary>
''' <param name="spname">The stored procedure to execute at the data
source.</param>
''' <param name="returnValue">The returned value from stored
procedure.</param>
''' <param name="parameterValues">The parameter values of the stored
procedure.</param>
''' <returns>A System.Data.SqlClient.SqlDataReader object.</returns>
Public Function ExecuteReader(ByVal spname As String, ByRef
returnValue As Integer, ByVal ParamArray parameterValues() As Object) As
IDataReader
    Dim connection As SqlConnection = Nothing
    Dim command As SqlCommand = Nothing
    Dim res As SqlDataReader = Nothing
    Try
        connection = New SqlConnection(ConnectionString)
        command = New SqlCommand(spname, connection)
        command.CommandType = CommandType.StoredProcedure
        connection.Open()
        SqlCommandBuilder.DeriveParameters(command)
        Me.AssignParameters(command, parameterValues)
        res = command.ExecuteReader(CommandBehavior.CloseConnection)
        returnValue = command.Parameters(0).Value
    Catch ex As Exception
        Throw New SqlDatabaseException(ex.Message, ex.InnerException)
    End Try
    Return CType(res, IDataReader)
End Function

#End Region

#Region " FillDataset "

''' <summary>
''' Adds or refreshes rows in the System.Data.DataSet to match those
in the data source using the System.Data.DataSet name, and creates a
System.Data.DataTable named "Table."
''' </summary>

```

```

        ''' <param name="cmd">The Transact-SQL statement or stored procedure
to execute at the data source.</param>
        ''' <param name="cmdType">A value indicating how the
System.Data.SqlClient.SqlCommand.CommandText property is to be
interpreted.</param>
        ''' <param name="parameters">The parameters of the Transact-SQL
statement or stored procedure.</param>
        ''' <returns>A System.Data.DataSet object.</returns>
        Public Function FillDataset(ByVal cmd As String, ByVal cmdType As
CommandType, Optional ByVal parameters() As SqlParameter = Nothing) As
DataSet
            Dim connection As SqlConnection = Nothing
            Dim command As SqlCommand = Nothing
            Dim sqlda As SqlDataAdapter = Nothing
            Dim res As New DataSet
            Try
                connection = New SqlConnection(_connectionString)
                command = New SqlCommand(cmd, connection)
                command.CommandType = cmdType
                AssignParameters(command, parameters)
                sqlda = New SqlDataAdapter(command)
                sqlda.Fill(res)
            Catch ex As Exception
                Throw New SqlDatabaseException(ex.Message, ex.InnerException)
            Finally
                If Not (connection Is Nothing) Then connection.Dispose()
                If Not (command Is Nothing) Then command.Dispose()
                If Not (sqlda Is Nothing) Then sqlda.Dispose()
            End Try
            Return res
        End Function
#End Region

#Region " ExecuteDataset "

        ''' <summary>
        ''' Calls the respective INSERT, UPDATE, or DELETE statements for
each inserted, updated, or deleted row in the System.Data.DataSet with the
specified System.Data.DataTable name.
        ''' </summary>
        ''' <param name="insertCmd">A command used to insert new records into
the data source.</param>
        ''' <param name="updateCmd">A command used to update records in the
data source.</param>
        ''' <param name="deleteCmd">A command for deleting records from the
data set.</param>
        ''' <param name="ds">The System.Data.DataSet to use to update the
data source. </param>
        ''' <param name="srcTable">The name of the source table to use for
table mapping.</param>
        ''' <returns>The number of rows successfully updated from the
System.Data.DataSet.</returns>
        Public Function ExecuteDataset(ByVal insertCmd As SqlCommand, ByVal
updateCmd As SqlCommand, ByVal deleteCmd As SqlCommand, ByVal ds As DataSet,
ByVal srcTable As String) As Integer
            Dim connection As SqlConnection = Nothing

```

```

    Dim sqlda As SqlDataAdapter = Nothing
    Dim res As Integer = 0
    Try
        connection = New SqlConnection(_connectionString)
        sqlda = New SqlDataAdapter
        If Not (insertCmd Is Nothing) Then insertCmd.Connection =
connection : sqlda.InsertCommand = insertCmd
        If Not (updateCmd Is Nothing) Then updateCmd.Connection =
connection : sqlda.UpdateCommand = updateCmd
        If Not (deleteCmd Is Nothing) Then deleteCmd.Connection =
connection : sqlda.DeleteCommand = deleteCmd
        res = sqlda.Update(ds, srcTable)
    Catch ex As Exception
        Throw New SqlDatabaseException(ex.Message, ex.InnerException)
    Finally
        If Not (connection Is Nothing) Then connection.Dispose()
        If Not (insertCmd Is Nothing) Then insertCmd.Dispose()
        If Not (updateCmd Is Nothing) Then updateCmd.Dispose()
        If Not (deleteCmd Is Nothing) Then deleteCmd.Dispose()
        If Not (sqlda Is Nothing) Then sqlda.Dispose()
    End Try
    Return res
End Function

```

#End Region

#Region " ExecuteScript "

```

    ''' <summary>
    ''' Executes a SQL query file against the connection.
    ''' </summary>
    ''' <param name="filename">SQL query file name.</param>
    ''' <param name="parameters">The parameters of the SQL query
file.</param>
    Public Sub ExecuteScript(ByVal filename As String, Optional ByVal
parameters() As SqlParameter = Nothing)
        Dim fStream As FileStream = Nothing
        Dim sReader As StreamReader = Nothing
        Dim connection As SqlConnection = Nothing
        Dim command As SqlCommand = Nothing
        Try
            fStream = New FileStream(filename, FileMode.Open,
FileAccess.Read)
            sReader = New StreamReader(fStream)
            connection = New SqlConnection(ConnectionString)
            command = connection.CreateCommand()
            connection.Open()
            While (Not sReader.EndOfStream)
                Dim sb As New StringBuilder
                While (Not sReader.EndOfStream)
                    Dim s As String = sReader.ReadLine
                    If (Not String.IsNullOrEmpty(s)) AndAlso
(s.ToUpper.Trim = "GO") Then
                        Exit While
                    End If
                    sb.AppendLine(s)
                End While
            End While

```

```
        command.CommandText = sb.ToString
        command.CommandType = CommandType.Text
        AssignParameters(command, parameters)
        command.ExecuteNonQuery()
    End While
Catch ex As Exception
    Throw New SqlDatabaseException(ex.Message, ex.InnerException)
Finally
    If (Not IsNothing(connection)) AndAlso (connection.State =
ConnectionState.Open) Then connection.Close()
    If (Not IsNothing(command)) Then command.Dispose()
    If (Not IsNothing(sReader)) Then sReader.Close()
    If (Not IsNothing(fStream)) Then fStream.Close()
End Try
End Sub

#End Region

    End Class

End Namespace
```