Volume

3

CREATIVE DATA TECHNOLOGIES, INC.

DATALAYER.NET™

Reference Guide

Table of Contents

Chapter 1 – DataConnection Reference

1.1 DataConnection Properties

First, a note about this document's structure: This document is divided into three chapters. Chapter 1 covers the Properties and Methods of the DataConnection class, and Chapter 2 covers the Properties and Methods of the DataHandler class. Furthermore, each chapter is broken down into two separate alphabetical sections. The first section lists all the **Properties**, and the second section lists all the **Methods**.

Definition: The difference between a **Property** and a **Method** is that a Property contains a value that you can set, whereas a Method is a function or subroutine that you call to carry out a task or return some value. Methods can not be assigned values (used on the left side of a statement with an equals sign), whereas properties can be assigned values.

This section provides an alphabetical reference guide for all of the DataConnection class properties. For each property, the data type, purpose, possible values, and default value are covered.

++

Property: AutoConvertNulls

Data Type: Boolean

Purpose: Use the AutoConvertNulls flag to control whether null values are automatically converted to the MinValue constants for you when accessing data.

Default Value: True (DataLayer.NET will normally convert the Null values to the MinValue constants for you unless you change this property)

++

Property: CommandTimeout

Data Type: Integer

Purpose: Use the CommandTimeout property to set the number of seconds that the DataLayer.NET library should wait for any query or command to execute (i.e. for the query's first row to be returned, or for the SQL command to finish running).

Default Value: 120 (Seconds)

++

Property: ConnectionState <Read-Only>

Data Type: System.Data.ConnectionState

Possible Values: ConnectionState.Broken
ConnectionState.Closed
ConnectionState.Connecting
ConnectionState.Executing
ConnectionState.Fetching
ConnectionState.Open

Purpose: Use the ConnectionState property to determine the current state of the DataConnection object.

Default Value: N/A

++

Property: **ConnectionString**

Data Type: **String**

Purpose: **The ConnectionString property is used to provide information about how the DataLayer.NET should connect to your database when you invoke the Connect() method. The format and content of Connection Strings are out of the scope of this document, but if you need help formulating your connection string, there is a great reference on the web at the following site: http://www.connectionstrings.com**

Default Value: **N/A (you must provide the ConnectionString to connect to the database)**

Sample Code:

```
    ' Create the DataLayer Connection object set for SQL Server mode...
    mSQLConn = New DataConnection(DataLayer_ConnectionType.SQLServer)
    mSQLConn.ConnectionString = "user id=guest;password=guestpass;" & _
                                "initial catalog=Northwind;data source=localhost"

    Try
      mSQLConn.Connect()
    Catch ex As Exception
      MsgBox(ex.Message, MsgBoxStyle.Critical, "Problem:")
      Exit Sub
    End Try
```

++

Property: **ConnectionType**

Data Type: **CDT.DATALAYER.DataLayer_ConnectionType**

Possible Values: **DataLayer_ConnectionType.Not_Set**
DataLayer_ConnectionType.ODBC
DataLayer_ConnectionType.OLEDB
DataLayer_ConnectionType.SQLServer

Purpose: **Use to configure the DataConnection object to connect to the desired type of database indicated.**

Default Value: **DataLayer_ConnectionType.Not_Set (must be set to a valid value before invoking the Connect() method).**

Notes: **The ConnectionType property can be passed as a constructor argument for the DataConnection class, as in the following sample:**

```
Dim objSQLConn As DataConnection
objSQLConn = New DataConnection(DataLayer_ConnectionType.SQLServer)
```

Alternatively, it can be set directly as a property at any time prior to calling the Connect() method:

```
    Dim objSQLConn As DataConnection = New DataConnection()
    objSQLConn.ConnectionType = DataLayer_ConnectionType.SQLServer
    mSQLConn.ConnectionString = "user id=guest;password=guestpass;" & _
                                "initial catalog=Northwind;data source=localhost"

    Try
      mSQLConn.Connect()
    Catch ex As Exception
      MsgBox(ex.Message, MsgBoxStyle.Critical, "Problem:")
      Exit Sub
    End Try
```

++

Property: **InTransaction** **<Read-Only>**

Data Type: **Boolean**

Purpose: **Use the InTransaction property to determine whether the database connection is currently in the middle of an open transaction. This happens if you call the BeginTransaction() Method, but have not yet issued a CommitTransaction() or RollbackTransaction() call.**

Default Value: **N/A**

++

Property: NullStringValue

Data Type: String

Purpose: This is the string value that you would like NULL string values read from the database to be converted to. For 99.99% of all programs, leaving this property set to an empty string (the default value) will result in the desired behavior. See the User's Manual, Section 2.4 for more details.

Default Value: "" (empty string)

++

Property: ODBC_Flavor

Data Type: CDT.DATALAYER.DataLayer_ODBCFlavor

Possible Values: DataLayer_ODBCFlavor.Not_Set
DataLayer_ODBCFlavor.DB2_ODBC
DataLayer_ODBCFlavor.Generic_ODBC

Purpose: Use this property if you are connecting to a DB2 database, and need to perform updates of TIMESTAMP columns. In this situation, the TIMESTAMP values need to be sent back to DB2 in a very specific 26 character format, or you will receive format errors (SQL Error -181) from DB2 upon performing the update. Otherwise, if you are not using DB2, or if you are not needing to perform updates when using DB2, just leave this property to the default value.

Default Value: DataLayer_ODBCFlavor.Not_Set

++

Property: ODBCConnectionHandle

Data Type: System.Data.Odbc.OdbcConnection

Purpose: Provided just in case you need low level access to the actual ADO.NET ODBC Connection handle. This should rarely, if ever, be needed by your program.

Default Value: N/A

++

Property: odbcTrans

Data Type: System.Data.Odbc.OdbcTransaction

Purpose: Provides you with low level access to the ODBC transaction object, if needed. This property should rarely, if ever, be needed by your program.

Default Value: N/A

++

Property: OLEDBConnectionHandle

Data Type: System.Data.OleDb.OleDbConnection

Purpose: Provided just in case you need low level access to the actual ADO.NET OLE-DB Connection handle. This should rarely, if ever, be needed by your program.

Default Value: N/A

++

Property: oleDbTrans

Data Type: System.Data.OleDb.OleDbTransaction

Purpose: Provides you with low level access to the OLE-DB transaction object, if needed. This property should rarely, if ever, be needed by your program.

Default Value: N/A

++

Property: Parameters

Data Type: ArrayList of CmdParameter

Purpose: Use the Parameters ArrayList to add parameters to any SQL Query or Command that you are executing through the DataConnection class. For a more complete discussion of using Parameters, see the Users' Guide, section 3.6.

Default Value: Empty ArrayList (new ArrayList containing zero array elements)

Sample Code:

```
Dim strSQL As String = “SELECT id_employee, nme_first, nme_last from EMPLOYEE “ & _
                       “ WHERE nme_last LIKE @LastName”
Dim objEmployees as New EMPLOYEE(gSQLConn, strSQL)
Dim strValue As String = Trim(txtSearchBox.Text) & “%”
objEmployees.Parameters.Add(New CmdParameter(“@LastName”,DBType.String,strValue))
Try
   ObjEmployees.GetAllRows()
Catch
  MsgBox(“Error fetching employees: “ & ex.Message)
End Try
```

++

Property: **SqlConnectionHandle**

Data Type: **System.Data.Sql.SqlConnection**

Purpose: **Provided just in case you need low level access to the actual ADO.NET SQL Server Connection handle. This should be rarely, if ever, be needed by your program.**

Default Value: **N/A**

++

Property: sqlTrans

Data Type: System.Data.Sql.SqlTransaction

Purpose: Provides you with low level access to the SQL Server transaction object, if needed. This property should rarely, if ever, be needed by your program.

Default Value: N/A

++

1.2 DataConnection Methods

This section provides an alphabetical reference guide for all of the DataConnection class methods. For each method, the arguments required (if any), the return type, and the purpose for the method are covered.

Method: **BeginTransaction**

Arguments: **None**

Returns: **Nothing**

Purpose: **Begins a logical unit of work, which is a series of operations being carried out on the database that either will be completed entirely, or completely rolled back.**

Sample Code:

```
Try
  gSQLConn.BeginTransaction()
  …(SQL operation 1)
  …(SQL operation 2)
  …(etc, etc.)
  gSQLConn.CommitTransaction()
Catch
  gSQLConn.RollbackTransaction()
  MsgBox("Error encountered, transaction rolled back: " & ex.Message)
End Try
```

++

Method: **CommitTransaction**

Arguments: **None**

Returns: **Nothing**

Purpose: **Closes a logical unit of work, which is a series of operations being carried out on the database that either will be completed entirely, or completely rolled back.**

Sample Code:

```
Try
  gSQLConn.BeginTransaction()
  …(SQL operation 1)
  …(SQL operation 2)
  …(etc, etc.)
  gSQLConn.CommitTransaction()
Catch
  gSQLConn.RollbackTransaction()
  MsgBox("Error encountered, transaction rolled back: " & ex.Message)
End Try
```

++

Method: **Connect**

Arguments: **Optional: cString As String – The connection string.**

Returns: **Nothing**

Purpose: **Connects to the database. Will throw an exception if an error occurs while trying to connect to the database.**

Sample Code:

```
gSQLConn = New DataConnection(DataLayer_ConnectionType.SQLServer)
gSQLConn.ConnectionString = "user id=guest;password=guestpass;" & _
                            "initial catalog=Northwind;data source=localhost"
Try
  gSQLConn.Connect()
Catch ex As Exception
  MsgBox(ex.Message, MsgBoxStyle.Critical, "Problem:")
  Exit Sub
End Try
```

++

Method: **Disconnect**

Arguments: **None**

Returns: **Nothing**

Purpose: **Disconnects from the database.**

Sample Code:

```
    gSQLConn.Disconnect()
```

++

Method: **GetDateTimeSQLResult**

Arguments: **SQL As String – The SQL query that will return one row containing one datetime column.**

Returns: **DateTime : The DateTime response from the database, or DateTime.MinValue if a NULL response was received from the database.**

Purpose: **Obtains a single DateTime result column from the database.**

Sample Code:

```
Dim dtmMaxActivityDate As Date
Dim SQL As String

SQL = “SELECT MAX(dtm_activity) FROM ACTIVITY”
Try
    dtmMaxActivityDate = gSQLConn.GetDateTimeSQLResult(SQL)
Catch
    MsgBox(“Error getting MAX Activity Date/Time: “ & ex.Message)
    Exit Sub
End Try
```

++

Method: GetDecimalSQLResult

Arguments: SQL As String – The SQL query that will return one row containing one decimal column.

Returns: Decimal : The Decimal response from the database, or Decimal.MinValue if a NULL response was received from the database.

Purpose: Obtains a single Decimal result column from the database.

Sample Code:

```
Dim decMaxSalary As Decimal
Dim SQL As String

SQL = “SELECT MAX(amt_salary) FROM EMPLOYEE”
Try
    decMaxSalary = gSQLConn.GetDecimalSQLResult(SQL)
Catch
    MsgBox(“Error getting MAX Salary: “ & ex.Message)
    Exit Sub
End Try
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Method: GetDoubleSQLResult

Arguments: SQL As String – The SQL query that will return one row containing one double column.

Returns: Double : The Double response from the database, or Double.MinValue if a NULL response was received from the database.

Purpose: Obtains a single Double result column from the database.

Sample Code:

```
Dim dblReading As Double
Dim SQL As String
Dim intSomeKey As Integer = 41  ‘ Randomly selected record ID for this example.

SQL = “SELECT qty_reading FROM MEASUREMENT WHERE id_key = @Key”
gSQLConn.Parameters.Add(New CmdParameter(“@Key”,DBType.Int32,intSomeKey))
Try
    dblReading = gSQLConn.GetDoubleSQLResult(SQL)
Catch
    MsgBox(“Error getting measurement reading: “ & ex.Message)
    Exit Sub
End Try
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Method: GetIntegerSQLResult

Arguments: SQL As String – The SQL query that will return one row containing one integer column.

Returns: Integer : The Integer response from the database, or Integer.MinValue if a NULL response was received from the database.

Purpose: Obtains a single Integer result column from the database.

Sample Code:

```
Dim intMaxSequence As Integer
Dim SQL As String

SQL = “SELECT MAX(num_seq_activity) FROM ACTIVITY”
Try
    intMaxSequence = gSQLConn.GetIntegerSQLResult(SQL)
Catch
    MsgBox(“Error getting MAX Sequence Number: “ & ex.Message)
    Exit Sub
```

End Try
++

Method: GetStringSQLResult

Arguments: SQL As String – The SQL query that will return one row containing one String column.

Returns: String : The String response from the database, or an empty string (or the current value of NullStringValue) if a NULL response was received from the database.

Purpose: Obtains a single String result column from the database.

Sample Code:

```
Dim strCompanyName As String
Dim SQL As String
Dim intCompanyID As Integer = 243  ‘ Randomly selected Company ID for this example.

SQL = “SELECT nme_company FROM COMPANY WHERE id_company = @CompanyID”
gSQLConn.Parameters.Add(New CmdParameter(“@CompanyID”,DBType.Int32,intCompanyID))
Try
    strCompanyName = gSQLConn.GetStringSQLResult(SQL)
Catch
    MsgBox(“Error getting Company Name: “ & ex.Message)
    Exit Sub
End Try
```

++

Method: **RollbackTransaction**

Arguments: **None**

Returns: **Nothing**

Purpose: **Cancels a logical unit of work, which is a series of operations being carried out on the database that either will be completed entirely, or completely rolled back.**

Sample Code:

```
Try
  gSQLConn.BeginTransaction()
  …(SQL operation 1)
  …(SQL operation 2)
  …(etc, etc.)
  gSQLConn.CommitTransaction()
Catch
  gSQLConn.RollbackTransaction()
  MsgBox("Error encountered, transaction rolled back: " & ex.Message)
End Try
```

++

Method: **RunSQLStatement**

Arguments: **SQL As String – The SQL Statement you would like to execute.**

Returns: **Nothing**

Purpose: **To directly submit a particular SQL statement to the database. The SQL should not return any rows.**

Sample Code:

```
Dim strSQL As String
strSQL = "select * into #TempProducts FROM Products"
Try
   gSQLConn.RunSQLStatement(strSQL)
Catch
   MsgBox("Error creating the temp products table: " & ex.Message
    Exit Sub
End Try
```

++

Chapter 2

Chapter 2 – DataHandler Reference

2.1 DataHandler Properties

This section provides an alphabetical reference guide for all of the DataHandler class properties. For each property, the data type, purpose, possible values, and default value is covered.

Property: **CurrentPage** **<Read-Only>**

Data Type: **Integer**

Purpose: **When using the page retrieval methods of the DataHandler class (i.e. – GetPage(n), GetFirstPage(), GetLastPage(), GetNextPage(), GetPrevPage()), the CurrentPage property will tell you the current page that is currently in the DataSet buffer.**

Default Value: **N/A**

++

Property: **DataConn**

Data Type: **CDT.DATALAYER.DataConnection**

Purpose: **Provides a handle to the current DataConnection.**

Default Value: **N/A**

Notes: **In some situations, such as 3-tier programming, you may need to re-assign the active DataConnection object at runtime if you are passing some DataHandler objects through the layers of your application.**

++

Property: DataSet

Data Type: System.Data.DataSet

Purpose: Provides a handle to DataSet buffer for the DataHandler object.

Default Value: N/A

Notes: Sometimes you need to have a direct handle to the DataSet buffer object, such as when you are programming a DataGrid control to display / edit the records on the screen.

++

Property: PageSize

Data Type: Integer

Purpose: When using the DataHandler object in paging mode (using routines such as GetPage(n), GetFirstPage, GetLastPage, GetNextPage, GetPrevPage), this property controls the number of records that are fetched into the DataSet buffer per page.

Default Value: 0 (this must be set to a non-zero value by you before calling the paging routines mentioned above)

++

Property: **Parameters**

Data Type: **ArrayList of CmdParameter objects**

Purpose: **Provides the interface to using Parameterized SQL in your application. For more complete coverage of using Parameterized SQL, refer to section 3.6 in the User's Manual.**

Default Value: **Empty ArrayList (zero elements)**

Notes: **Every time a query runs, it initializes the Parameters array back to an empty ArrayList.**

Sample Code:

```
Dim strSQL As String = "SELECT id_employee, nme_first, nme_last from EMPLOYEE " & _
                       " WHERE nme_last LIKE @LastName"
Dim objEmployees as New EMPLOYEE(gSQLConn, strSQL)
Dim strValue As String = Trim(txtSearchBox.Text) & "%"
objEmployees.Parameters.Add(New CmdParameter("@LastName",DBType.String,strValue))
Try
   ObjEmployees.GetAllRows()
Catch
  MsgBox("Error fetching employees: " & ex.Message)
End Try
```

++

Property: **SQLQuery**

Data Type: **String**

Purpose: **This lets you directly specify the SQL query that will run when you execute your query using the GetAllRows() command (or the paging routines).**

Default Value: **Empty String “”**

Notes: **You can also pass in the SQL Query as a constructor argument, if desired.**

Sample Code:

```
Dim strSQL As String = “SELECT id_employee, nme_first, nme_last from EMPLOYEE “
Dim objEmployees as New EMPLOYEE(gSQLConn)
objEmployees.SQLQuery = strSQL
Try
   ObjEmployees.GetAllRows()
Catch
  MsgBox(“Error fetching employees: “ & ex.Message)
End Try
```

++

Property: **UpdatesPending** **<Read-Only>**

Data Type: **Boolean**

Purpose: **Use this property to tell whether any changes have been made to the DataSet buffer’s row data.**

Default Value: **False**

Notes: **Your application does not really need to use this property. You can blindly call the Update() method, even if no changes have been made, and no updates are being sent to the database (no error will occur).**

++

Property: **UpdateTable**

Data Type: **String**

Purpose: **This property is usually specified in the New() constructor method of the DataHandler class files that are generated for you by the Code Generator. This tells the DataLayer.NET library which table to update when executing the Update routine.**

Default Value: **N/A**

Notes: **Your application normally will not need to manually set this property. It is set for you by the code generator in your DataHandler class files.**

++

2.2 DataHandler Methods

This section provides an alphabetical reference guide for all of the DataHandler class methods. For each method, the arguments required (if any), the return type, and the purpose for the method is covered.

Method: **AddRow**

Arguments: **None**

Returns: **Integer : The new row's index number.**

Purpose: **Adds a new row to the DataSet buffer.**

Sample Code:

```
Dim intNewRow As Integer

intNewRow = objEmployee.AddRow()
objEmployee.NME_FIRST(intNewRow) = "BOB"
objEmployee.NME_LAST(intNewRow) = "BARKER"
```

++

Method: **DeleteRow**

Arguments: **RowNumber As Integer – The index of the row you want to delete.**

Returns: **Nothing**

Purpose: **Deletes a row from the DataSet buffer.**

Sample Code:

```
Dim intRow As Integer = 71     ' Row number 71 chosen for this example.

objEmployee.DeleteRow(intRow)
```

++

Method: **GetAllRows**

Arguments: **None**

Returns: **Integer : The number of rows retrieved.**

Purpose: **Retrieves all of the rows for the Query into the DataSet buffer.**

Sample Code:

```
Dim strSQL As String = “SELECT id_employee, nme_first, nme_last from EMPLOYEE “
 Dim objEmployees as New EMPLOYEE(gSQLConn)
objEmployees.SQLQuery = strSQL
Try
    ObjEmployees.GetAllRows()
Catch
   MsgBox(“Error fetching employees: “ & ex.Message)
End Try
```

++

Method: **GetBaseSQL**

Arguments: **None**

Returns: **String : The SELECT Statement containing the base SQL Query for the DataHandler object.**

Purpose: **This method is implemented by the Code Generator program for each one of the DataHandler classes. It returns the bare SQL Statement for your DataHandler object without any WHERE clause. This method is often used to save time typing the long SQL queries when retrieving data.**

Notes: **The GetBaseSQL routine is declared as a Shared routine, so you don't actually need an instantiated DataHandler object to use this routine. This makes it possible to specify the complete SQL in the constructor of your class, as in the sample below.**

Sample Code:

```
' Get all the employee records, sorted by lastname, first…
Dim objEmployees as New EMPLOYEE(gSQLConn, EMPLOYEE.GetBaseSQL & _
                                        " ORDER BY nme_last, nme_first")
Try
   ObjEmployees.GetAllRows()
Catch
  MsgBox("Error fetching employees: " & ex.Message)
End Try
```

++

Method: **GetColumnType**

Arguments: **ColumnName As String**

Returns: **DBType : The data type of column name specified.**

Purpose: **This method obtains the data type of the column specified.**

Notes: **This routine should be rarely needed, if ever, by your applications.**

++

Method: GetDateTimeData

Arguments: RowNumber As Integer – The index of the row desired
ColumnName As String – The name of the desired

Returns: Date (DateTime): The value of the column specified for the row specified. It will be converted to the data type's MinValue if a NULL value is detected (assuming AutoConvertNulls is True).

Purpose: This method is utilized for you by the Code Generator program, and is part of the column property. You should not normally have to use this routine in your code.

Sample Code: (taken directly from a Code Generator generated class file)

```
  <Updateable()> Public Property DTM_CREATED(ByVal RowNum As Integer) As Date
    Get
      DTM_CREATED = GetDateTimeData(RowNum, "DTM_CREATED")
    End Get
    Set(ByVal Value As Date)
      SetData(RowNum, "DTM_CREATED", Value)
    End Set
  End Property
```

++

Method: GetDecimalData

Arguments: RowNumber As Integer – The index of the row desired
ColumnName As String – The name of the desired column

Returns: Decimal: The value of the column specified for the row specified. It will be converted to the data type's MinValue if a NULL value is detected (assuming AutoConvertNulls is True).

Purpose: This method is utilized for you by the Code Generator program, and is part of the column property. You should not normally have to use this routine in your code.

Sample Code: (taken directly from a Code Generator generated class file)

```
  <Updateable()> Public Property QTY_DAYS(ByVal RowNum As Integer) As Decimal
    Get
      QTY_DAYS = GetDecimalData(RowNum, "QTY_DAYS")
    End Get
    Set(ByVal Value As Decimal)
      SetData(RowNum, "QTY_DAYS", Value)
    End Set
  End Property
```

++

Method: GetDoubleData

Arguments: RowNumber As Integer – The index of the desired row
ColumnName As String – The name of the desired column

Returns: Double: The value of the column specified for the row specified. It will be converted to the data type's MinValue if a NULL value is detected (assuming AutoConvertNulls is True).

Purpose: This method is utilized for you by the Code Generator program, and is part of the column property. You should not normally have to use this routine in your code.

Sample Code: (taken directly from a Code Generator generated class file)

```
  <Updateable()> Public Property AMT_PAYMENT(ByVal RowNum As Integer) As Double
    Get
      AMT_PAYMENT = GetDoubleData(RowNum, "AMT_PAYMENT")
    End Get
    Set(ByVal Value As Double)
      SetData(RowNum, "AMT_PAYMENT", Value)
    End Set
  End Property
```

++

Method: GetFirstPage

Arguments: None

Returns: Integer: The number of rows actually fetched for this page. This could be less than the PageSize value if there are not enough rows to fill the page (partial page).

Purpose: Use this method to retrieve the first page of data into the DataSet buffer. Refer to the User's Guide, section 3.9 for full coverages of using the Paging routines.

++

Method: GetIntegerData

Arguments: RowNumber As Integer – The index of the row you want data from
ColumnName As String – The name of the column you want the value for

Returns: Integer: The value of the column specified for the row specified. It will be converted to the data type's MinValue if a NULL value is detected (assuming AutoConvertNulls is True).

Purpose: This method is utilized for you by the Code Generator program, and is part of the column property. You should not normally have to use this routine in your code.

Sample Code: (taken directly from a Code Generator generated class file)

```
<Updateable()> Public Property ID_TOKEN(ByVal RowNum As Integer) As Integer
  Get
    ID_TOKEN = GetIntegerData(RowNum, "ID_TOKEN")
  End Get
  Set(ByVal Value As Integer)
    SetData(RowNum, "ID_TOKEN", Value)
  End Set
End Property
```

++

Method: GetLastPage

Arguments: None

Returns: Integer: The number of rows actually fetched for this page. This could be less than the PageSize value if there are not enough rows to fill the page (partial page).

Purpose: Use this method to retrieve the last page of data into the DataSet buffer. Refer to the User's Guide, section 3.9 for full coverage of using the Paging routines.

++

Method: GetNextPage

Arguments: None

Returns: Integer: The number of rows actually fetched for this page. This could be less than the PageSize value if there are not enough rows to fill the page (partial page).

Purpose: Use this method to retrieve the next page of data into the DataSet buffer. Refer to the User's Guide, section 3.9 for full coverage of using the Paging routines.

++

Method: GetNextSequenceNumber

Arguments: None

Returns: Integer: The next available value for the column marked as the <Sequencer()> column in your DataHandler object, from inspection of all the records currently in the DataSet buffer.

Purpose: Use this method Generate the next available sequence number. Refer to the User's Guide, section 3.15 for full coverage of using this routine.

++

Method: GetPage(n)

Arguments: PageNumber As Integer – The page number you would like to retrieve.

Returns: Integer: The number of rows actually fetched for this page. This could be less than the PageSize value if there are not enough rows to fill the page (partial page).

Notes: The page numbers are 1-based. In other words, the first page number is 1, not 0.

Purpose: Use this method to retrieve a particular page of data into the DataSet buffer. Refer to the User's Guide, section 3.9 for full coverage of using the Paging routines.

++

Method: GetPrevPage

Arguments: None

Returns: Integer: The number of rows actually fetched for this page. This could be less than the PageSize value if there are not enough rows to fill the page (partial page).

Purpose: Use this method to retrieve the previous page of data into the DataSet buffer. Refer to the User's Guide, section 3.9 for full coverage of using the Paging routines.

++

Method: GetQueryRowCount

Arguments: None

Returns: Integer: The number of rows present in the database for the SQL Query.

Notes: This routine queries the database and performs a SELECT COUNT(*) for the current SQLQuery that is defined for the DataHandler object.

Purpose: Use this method to find out how many rows there are for the query you are about to fetch (i.e. – GetAllRows). This is useful in situations where you have provided a search window for the user, but you want to limit the number of rows that they can fetch. If the number of rows matching the criteria entered by the user is in excess of the desired maximum number of rows ("governor"), then you can display an error message asking them to narrow their search criteria.

++

Method: GetStringData

Arguments: RowNumber As Integer – The index of the row desired
ColumnName As String – The name of the desired column

Returns: String: The value of the column specified for the row specified. It will be converted to an empty string if a NULL value is detected (assuming AutoConvertNulls is True).

Purpose: This method is utilized for you by the Code Generator program, and is part of the column property. You should not normally have to use this routine in your code.

Sample Code: (taken directly from a Code Generator generated class file)

```
  <Updateable()> Public Property NME_FIRST(ByVal RowNum As Integer) As String
    Get
      NME_FIRST = GetStringData(RowNum, "NME_FIRST")
    End Get
    Set(ByVal Value As String)
      SetData(RowNum, "NME_FIRST", Value)
    End Set
  End Property
```

++

Method: **InitializeEmptyBuffer**

Arguments: **None**

Returns: **Nothing**

Purpose: **This method is only needed when you are inserting new records into the database, and you do not have any previous rows already fetched into the DataSet buffer. Refer to the User's Guide, Sectin 3.4 for full coverage of this routine.**

Sample Code:

```
    mEmployees = New EMPLOYEES(gSQLConn, EMPLOYEES.GetBaseSQL)

    ' Initialize the DataSet buffer...
    Try
      mEmployees.InitializeEmptyBuffer()
    Catch ex As Exception
      MsgBox("Error initializeing Employees buffer: " & ex.Message)
      Exit Sub
    End Try

    ' Add the new Employee record to the DataSet buffer in memory...
    Dim intRow As Integer
    IntRow = mEmployees.AddRow()

    ' Set a few of the values for the new Employee...
    mEmployees.ID_EMPLOYEE(intRow) = 247
    mEmployees.FIRST_NAME(intRow) = "Henry"
    mEmployees.LAST_NAME(intRow) = "Longfellow"

    ' Now go ahead and INSERT the new Employee record into the database...
    Try
      mEmployees.Update()
    Catch ex As Exception
      MsgBox("Error Inserting the Employee: " & ex.Message)
      Exit Sub
    End Try
```

++

Method: **PageCount**

Arguments: **None**

Returns: **Integer – The number of pages of data in the database for the SQL Query.**

Purpose: **This method is used when you are utilizing the paging routines. It performs a database query to find out how many rows are present in the database for the SQL Query (using the GetQueryRowCount routine), and divides the number of rows by the PageSize value to return the number of pages of data present in the database. This is typically used in an ASP.NET application where you are paging through the database, and you want to display something like "Page 1 of 14" at the bottom near the paging buttons for the grid. Refer to the User's Guide, section 3.9 for full coverage of using the Paging routines.**

++

Method: **RowCount**

Arguments: **None**

Returns: **Integer – The number of rows of data presently in the DataSet buffer.**

Purpose: **This method returns the number of rows currently present in the DataSet buffer.**

Sample Code:

```
Dim strSQL As String = "SELECT id_employee, nme_first, nme_last from EMPLOYEE "
Dim objEmployees as New EMPLOYEE(gSQLConn)
objEmployees.SQLQuery = strSQL
Try
   objEmployees.GetAllRows()
Catch
  MsgBox("Error fetching employees: " & ex.Message)
End Try

MsgBox("There were " & CStr(objEmployees.RowCount()) & " employees retrieved.")
```

++

Method: SetData

Arguments: RowNumber As Integer – The desired row number
ColumnName As String – The name of the desired column
Value As Object – The new value to set

Returns: Nothing

Purpose: This method sets the value of a particular column on a given row to a new value. This routine is called by the DataHandler classes that are generated for you by the Code Generator Program. You should not have to directly call this routine in your applications.

Sample Code: (taken directly from a Code Generator generated class file)

```
  <Updateable()> Public Property NME_FIRST(ByVal RowNum As Integer) As String
    Get
      NME_FIRST = GetStringData(RowNum, "NME_FIRST")
    End Get
    Set(ByVal Value As String)
      SetData(RowNum, "NME_FIRST", Value)
    End Set
  End Property
```

++

Method: Update

Arguments: None

Returns: Nothing

Purpose: This method sends all the updates that have been made to the DataSet buffer to the database.

Sample Code:

```
   Try
      mEmployees.Update()
    Catch ex As Exception
      MsgBox("Error Updating the Employees table: " & ex.Message)
      Exit Sub
    End Try
```

++

Method: GetFirstRowForStringColumnValue

Arguments: sColName – The column name you would like to search for the value in.
sValue – The string value that you are searching for.

Returns: Integer : The row number where the value was found. Zero if not found.

++

Method: GetFirstRowForIntegerColumnValue

Arguments: sColName – The column name you would like to search for the value in.
iValue – The Integer value that you are searching for.

Returns: Integer : The row number where the value was found. Zero if not found.

++

Method: Refresh_Row_Pointers

Arguments: None

Returns: Nothing

Purpose: This method refreshes the internal list of row pointers for the DataHandler object. You normally do not have to call this unless you are letting a DataGridView control directly add and delete rows from the DataSet instead of using the DataHandler methods AddRow and DeleteRow.