.NET Framework 4

**DataAdapters and DataReaders (ADO.NET)**

You can use the ADO.NET **DataReader** to retrieve a read-only, forward-only stream of data from a database. Results are returned as the query executes, and are stored in the network buffer on the client until you request them using the **Read** method of the **DataReader**. Using the **DataReader** can increase application performance both by retrieving data as soon as it is available, and (by default) storing only one row at a time in memory, reducing system overhead.

A [DataAdapter](http://msdn.microsoft.com/en-us/library/system.data.common.dataadapter.aspx) is used to retrieve data from a data source and populate tables within a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx). The **DataAdapter** also resolves changes made to the **DataSet** back to the data source. The **DataAdapter** uses the **Connection** object of the .NET Framework data provider to connect to a data source, and it uses **Command** objects to retrieve data from and resolve changes to the data source.

Each .NET Framework data provider included with the .NET Framework has a [DbDataReader](http://msdn.microsoft.com/en-us/library/system.data.common.dbdatareader.aspx) and a [DbDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.aspx) object: the .NET Framework Data Provider for OLE DB includes an [OleDbDataReader](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdatareader.aspx) and an [OleDbDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdataadapter.aspx) object, the .NET Framework Data Provider for SQL Server includes a SqlDataReade and a [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx) object, the .NET Framework Data Provider for ODBC includes an [OdbcDataReader](http://msdn.microsoft.com/en-us/library/system.data.odbc.odbcdatareader.aspx) and an [OdbcDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.odbc.odbcdataadapter.aspx) object, and the .NET Framework Data Provider for Oracle includes an [OracleDataReader](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oracledatareader.aspx) [OracleDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oracledataadapter.aspx) object.

.NET Framework 4

**Retrieving Data Using a DataReader (ADO.NET)**

Retrieving data using a **DataReader** involves creating an instance of the **Command** object and then creating a **DataReader** by calling **Command.ExecuteReader** to retrieve rows from a data source. The following example illustrates using a **DataReader** where reader represents a valid DataReader and command represents a valid Command object.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl01_code');" \o "Copy Code)

reader = command.ExecuteReader();

You use the **Read** method of the **DataReader** object to obtain a row from the results of the query. You can access each column of the returned row by passing the name or ordinal reference of the column to the **DataReader**. However, for best performance, the **DataReader** provides a series of methods that allow you to access column values in their native data types (**GetDateTime**, **GetDouble**, **GetGuid**, **GetInt32**, and so on). For a list of typed accessor methods for data provider-specific **DataReaders**, see [OleDbDataReader](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdatareader.aspx) and [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx). Using the typed accessor methods, assuming the underlying data type is known, reduces the amount of type conversion required when retrieving the column value.

|  |
| --- |
| **NoteNote** |
| The Windows Server 2003 release of the .NET Framework includes an additional property for the **DataReader**, **HasRows**, which enables you to determine if the **DataReader** has returned any results before reading from it. |

The following code example iterates through a **DataReader** object, and returns two columns from each row.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl05_code');" \o "Copy Code)

Private Sub HasRows(ByVal connection As SqlConnection)

Using connection

Dim command As SqlCommand = New SqlCommand( \_

"SELECT CategoryID, CategoryName FROM Categories;", \_

connection)

connection.Open()

Dim reader As SqlDataReader = command.ExecuteReader()

If reader.HasRows Then

Do While reader.Read()

Console.WriteLine(reader.GetInt32(0) \_

& vbTab & reader.GetString(1))

Loop

Else

Console.WriteLine("No rows found.")

End If

reader.Close()

End Using

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl06_code');" \o "Copy Code)

static void HasRows(SqlConnection connection)

{

using (connection)

{

SqlCommand command = new SqlCommand(

"SELECT CategoryID, CategoryName FROM Categories;",

connection);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

if (reader.HasRows)

{

while (reader.Read())

{

Console.WriteLine("{0}\t{1}", reader.GetInt32(0),

reader.GetString(1));

}

}

else

{

Console.WriteLine("No rows found.");

}

reader.Close();

}

}

The **DataReader** provides an unbuffered stream of data that allows procedural logic to efficiently process results from a data source sequentially. The **DataReader** is a good choice when retrieving large amounts of data because the data is not cached in memory.

 Closing the DataReader

You should always call the **Close** method when you have finished using the **DataReader** object.

If your **Command** contains output parameters or return values, they will not be available until the **DataReader** is closed.

Note that while a **DataReader** is open, the **Connection** is in use exclusively by that **DataReader**. You cannot execute any commands for the **Connection**, including creating another **DataReader**, until the original **DataReader** is closed.

|  |
| --- |
| **NoteNote** |
| Do not call **Close** or **Dispose** on a **Connection**, a **DataReader**, or any other managed object in the **Finalize** method of your class. In a finalizer, only release unmanaged resources that your class owns directly. If your class does not own any unmanaged resources, do not include a **Finalize** method in your class definition. For more information, see [Garbage Collection](http://msdn.microsoft.com/en-us/library/0xy59wtx.aspx). |

 Retrieving Multiple Result Sets using NextResult

If multiple result sets are returned, the **DataReader** provides the **NextResult** method to iterate through the result sets in order. The following example shows the [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) processing the results of two SELECT statements using the [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executereader.aspx) method.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl51_ctl00_ctl02_code');" \o "Copy Code)

Private Sub RetrieveMultipleResults(ByVal connection As SqlConnection)

Using connection

Dim command As SqlCommand = New SqlCommand( \_

"SELECT CategoryID, CategoryName FROM Categories;" & \_

"SELECT EmployeeID, LastName FROM Employees", connection)

connection.Open()

Dim reader As SqlDataReader = command.ExecuteReader()

Do While reader.HasRows

Console.WriteLine(vbTab & reader.GetName(0) \_

& vbTab & reader.GetName(1))

Do While reader.Read()

Console.WriteLine(vbTab & reader.GetInt32(0) \_

& vbTab & reader.GetString(1))

Loop

reader.NextResult()

Loop

End Using

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl51_ctl00_ctl03_code');" \o "Copy Code)

static void RetrieveMultipleResults(SqlConnection connection)

{

using (connection)

{

SqlCommand command = new SqlCommand(

"SELECT CategoryID, CategoryName FROM dbo.Categories;" +

"SELECT EmployeeID, LastName FROM dbo.Employees",

connection);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.HasRows)

{

Console.WriteLine("\t{0}\t{1}", reader.GetName(0),

reader.GetName(1));

while (reader.Read())

{

Console.WriteLine("\t{0}\t{1}", reader.GetInt32(0),

reader.GetString(1));

}

reader.NextResult();

}

}

}

 Getting Schema Information from the DataReader

While a **DataReader** is open, you can retrieve schema information about the current result set using the **GetSchemaTable** method. **GetSchemaTable** returns a [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx) object populated with rows and columns that contain the schema information for the current result set. The **DataTable** contains one row for each column of the result set. Each column of the schema table row maps to a property of the column returned in the result set, where the **ColumnName** is the name of the property and the value of the column is the value of the property. The following code example writes out the schema information for **DataReader**.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl01_code');" \o "Copy Code)

Private Sub GetSchemaInfo(ByVal connection As SqlConnection)

Using connection

Dim command As SqlCommand = New SqlCommand( \_

"SELECT CategoryID, CategoryName FROM Categories;", \_

connection)

connection.Open()

Dim reader As SqlDataReader = command.ExecuteReader()

Dim schemaTable As DataTable = reader.GetSchemaTable()

Dim row As DataRow

Dim column As DataColumn

For Each row In schemaTable.Rows

For Each column In schemaTable.Columns

Console.WriteLine(String.Format("{0} = {1}", \_

column.ColumnName, row(column)))

Next

Console.WriteLine()

Next

reader.Close()

End Using

End Sub

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl52_ctl00_ctl02_code');" \o "Copy Code)

static void GetSchemaInfo(SqlConnection connection)

{

using (connection)

{

SqlCommand command = new SqlCommand(

"SELECT CategoryID, CategoryName FROM Categories;",

connection);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

DataTable schemaTable = reader.GetSchemaTable();

foreach (DataRow row in schemaTable.Rows)

{

foreach (DataColumn column in schemaTable.Columns)

{

Console.WriteLine(String.Format("{0} = {1}",

column.ColumnName, row[column]));

}

}

}

}

 Working with OLE DB Chapters

Hierarchical rowsets, or chapters (OLE DB type **DBTYPE\_HCHAPTER**, ADO type **adChapter**) can be retrieved using the [OleDbDataReader](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdatareader.aspx). When a query that includes a chapter is returned as a **DataReader**, the chapter is returned as a column in that **DataReader** and is exposed as a **DataReader** object.

The ADO.NET **DataSet** can also be used to represent hierarchical rowsets using parent-child relationships between tables. For more information, see [DataSets, DataTables, and DataViews (ADO.NET)](http://msdn.microsoft.com/en-us/library/ss7fbaez.aspx).

The following code example uses the MSDataShape Provider to generate a chapter column of orders for each customer in a list of customers.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl02_code');" \o "Copy Code)

Using connection As OleDbConnection = New OleDbConnection( \_

"Provider=MSDataShape;Data Provider=SQLOLEDB;" & \_

"Data Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind")

Dim custCMD As OleDbCommand = New OleDbCommand( \_

"SHAPE {SELECT CustomerID, CompanyName FROM Customers} " & \_

"APPEND ({SELECT CustomerID, OrderID FROM Orders} AS CustomerOrders " & \_

"RELATE CustomerID TO CustomerID)", connection)

connection.Open()

Dim custReader As OleDbDataReader = custCMD.ExecuteReader()

Dim orderReader As OleDbDataReader

Do While custReader.Read()

Console.WriteLine("Orders for " & custReader.GetString(1))

' custReader.GetString(1) = CompanyName

orderReader = custReader.GetValue(2)

' custReader.GetValue(2) = Orders chapter as DataReader

Do While orderReader.Read()

Console.WriteLine(vbTab & orderReader.GetInt32(1))

' orderReader.GetInt32(1) = OrderID

Loop

orderReader.Close()

Loop

' Make sure to always close readers and connections.

custReader.Close()

End Using

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl03_code');" \o "Copy Code)

Using (OleDbConnection connection = new OleDbConnection(

"Provider=MSDataShape;Data Provider=SQLOLEDB;" +

"Data Source=localhost;Integrated Security=SSPI;Initial Catalog=northwind"));

{

OleDbCommand custCMD = new OleDbCommand(

"SHAPE {SELECT CustomerID, CompanyName FROM Customers} " +

"APPEND ({SELECT CustomerID, OrderID FROM Orders} AS CustomerOrders " +

"RELATE CustomerID TO CustomerID)", connection);

connection.Open();

OleDbDataReader custReader = custCMD.ExecuteReader();

OleDbDataReader orderReader;

while (custReader.Read())

{

Console.WriteLine("Orders for " + custReader.GetString(1));

// custReader.GetString(1) = CompanyName

orderReader = (OleDbDataReader)custReader.GetValue(2);

// custReader.GetValue(2) = Orders chapter as DataReader

while (orderReader.Read())

Console.WriteLine("\t" + orderReader.GetInt32(1));

// orderReader.GetInt32(1) = OrderID

orderReader.Close();

}

// Make sure to always close readers and connections.

custReader.Close();

}

 Returning Results with Oracle REF CURSORs

The .NET Framework Data Provider for Oracle supports the use of Oracle REF CURSORs to return a query result. An Oracle REF CURSOR is returned as an [OracleDataReader](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oracledatareader.aspx).

You can retrieve an **OracleDataReader** object, that represents an Oracle REF CURSOR using the [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oraclecommand.executereader.aspx) method, and you can also specify an [OracleCommand](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oraclecommand.aspx) that returns one or more Oracle REF CURSORs as the **SelectCommand** for an [OracleDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oracledataadapter.aspx) used to fill a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx).

To access a REF CURSOR returned from an Oracle data source, create an **OracleCommand** for your query and add an output parameter that references the REF CURSOR to the **Parameters** collection of your **OracleCommand**. The name of the parameter must match the name of the REF CURSOR parameter in your query. Set the type of the parameter to **OracleType.Cursor**. The **ExecuteReader** method of your **OracleCommand** will return an **OracleDataReader** for the REF CURSOR.

If your **OracleCommand** returns multiple REF CURSORS, add multiple output parameters. You can access the different REF CURSORs by calling the **OracleCommand.ExecuteReader** method. The call to **ExecuteReader** returns an **OracleDataReader** referencing the first REF CURSOR. You can then call the **OracleDataReader.NextResult** method to access subsequent REF CURSORs. Although the parameters in your **OracleCommand.Parameters** collection match the REF CURSOR output parameters by name, the **OracleDataReader** accesses them in the order that they were added to the **Parameters** collection.

For example, consider the following Oracle package and package body.

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl05_code');" \o "Copy Code)

CREATE OR REPLACE PACKAGE CURSPKG AS

TYPE T\_CURSOR IS REF CURSOR;

PROCEDURE OPEN\_TWO\_CURSORS (EMPCURSOR OUT T\_CURSOR,

DEPTCURSOR OUT T\_CURSOR);

END CURSPKG;

CREATE OR REPLACE PACKAGE BODY CURSPKG AS

PROCEDURE OPEN\_TWO\_CURSORS (EMPCURSOR OUT T\_CURSOR,

DEPTCURSOR OUT T\_CURSOR)

IS

BEGIN

OPEN EMPCURSOR FOR SELECT \* FROM DEMO.EMPLOYEE;

OPEN DEPTCURSOR FOR SELECT \* FROM DEMO.DEPARTMENT;

END OPEN\_TWO\_CURSORS;

END CURSPKG;

The following code creates an **OracleCommand** that returns the REF CURSORs from the previous Oracle package by adding two parameters of type **OracleType.Cursor** to the **Parameters** collection.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl06_code');" \o "Copy Code)

Dim cursCmd As OracleCommand = New OracleCommand("CURSPKG.OPEN\_TWO\_CURSORS", oraConn)

cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output

cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl07_code');" \o "Copy Code)

OracleCommand cursCmd = new OracleCommand("CURSPKG.OPEN\_TWO\_CURSORS", oraConn);

cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output;

cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output;

The following code returns the results of the previous command using the **Read** and **NextResult** methods of the **OracleDataReader**. The REF CURSOR parameters are returned in order.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl08_code');" \o "Copy Code)

oraConn.Open()

Dim cursCmd As OracleCommand = New OracleCommand("CURSPKG.OPEN\_TWO\_CURSORS", oraConn)

cursCmd.CommandType = CommandType.StoredProcedure

cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output

cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output

Dim reader As OracleDataReader = cursCmd.ExecuteReader()

Console.WriteLine(vbCrLf & "Emp ID" & vbTab & "Name")

Do While reader.Read()

Console.WriteLine("{0}" & vbTab & "{1}, {2}", reader.GetOracleNumber(0), reader.GetString(1), reader.GetString(2))

Loop

reader.NextResult()

Console.WriteLine(vbCrLf & "Dept ID" & vbTab & "Name")

Do While reader.Read()

Console.WriteLine("{0}" & vbTab & "{1}", reader.GetOracleNumber(0), reader.GetString(1))

Loop

' Make sure to always close readers and connections.

reader.Close()

oraConn.Close()

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl09_code');" \o "Copy Code)

oraConn.Open();

OracleCommand cursCmd = new OracleCommand("CURSPKG.OPEN\_TWO\_CURSORS", oraConn);

cursCmd.CommandType = CommandType.StoredProcedure;

cursCmd.Parameters.Add("EMPCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output;

cursCmd.Parameters.Add("DEPTCURSOR", OracleType.Cursor).Direction = ParameterDirection.Output;

OracleDataReader reader = cursCmd.ExecuteReader();

Console.WriteLine("\nEmp ID\tName");

while (reader.Read())

Console.WriteLine("{0}\t{1}, {2}", reader.GetOracleNumber(0), reader.GetString(1), reader.GetString(2));

reader.NextResult();

Console.WriteLine("\nDept ID\tName");

while (reader.Read())

Console.WriteLine("{0}\t{1}", reader.GetOracleNumber(0), reader.GetString(1));

// Make sure to always close readers and connections.

reader.Close();

oraConn.Close();

The following example uses the previous command to populate a **DataSet** with the results of the Oracle package.

|  |
| --- |
| **NoteNote** |
| To avoid an **OverflowException**, we recommend that you also handle any conversion from the Oracle NUMBER type to a valid .NET Framework type before storing the value in a **DataRow**. You can use the **FillError** event to determine if an **OverflowException** has occurred. For more information on the **FillError** event, see [Handling DataAdapter Events (ADO.NET)](http://msdn.microsoft.com/en-us/library/6d1wk41s.aspx). |

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl12_code');" \o "Copy Code)

Dim ds As DataSet = New DataSet()

Dim adapter As OracleDataAdapter = New OracleDataAdapter(cursCmd)

adapter.TableMappings.Add("Table", "Employees")

adapter.TableMappings.Add("Table1", "Departments")

adapter.Fill(ds)

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl54_ctl00_ctl13_code');" \o "Copy Code)

DataSet ds = new DataSet();

OracleDataAdapter adapter = new OracleDataAdapter(cursCmd);

adapter.TableMappings.Add("Table", "Employees");

adapter.TableMappings.Add("Table1", "Departments");

adapter.Fill(ds);

.NET Framework 4

**Populating a DataSet from a DataAdapter (ADO.NET)**

The ADO.NET [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is a memory-resident representation of data that provides a consistent relational programming model independent of the data source. The **DataSet** represents a complete set of data that includes tables, constraints, and relationships among the tables. Because the **DataSet** is independent of the data source, a **DataSet** can include data local to the application, and data from multiple data sources. Interaction with existing data sources is controlled through the **DataAdapter**.

The **SelectCommand** property of the **DataAdapter** is a **Command** object that retrieves data from the data source. The **InsertCommand**, **UpdateCommand**, and **DeleteCommand** properties of the **DataAdapter** are **Command** objects that manage updates to the data in the data source according to modifications made to the data in the **DataSet**. These properties are covered in more detail in [Updating Data Sources with DataAdapters (ADO.NET)](http://msdn.microsoft.com/en-us/library/33y2221y.aspx).

The **Fill** method of the **DataAdapter** is used to populate a **DataSet** with the results of the **SelectCommand** of the **DataAdapter**. **Fill** takes as its arguments a **DataSet** to be populated, and a **DataTable** object, or the name of the **DataTable** to be filled with the rows returned from the **SelectCommand**.

|  |
| --- |
| **NoteNote** |
| Using the **DataAdapter** to retrieve all of a table takes time, especially if there are many rows in the table. This is because accessing the database, locating and processing the data, and then transferring the data to the client is time-consuming. Pulling all of the table to the client also locks all of the rows on the server. To improve performance, you can use the **WHERE** clause to greatly reduce the number of rows returned to the client. You can also reduce the amount of data returned to the client by only explicitly listing required columns in the **SELECT** statement. Another good workaround is to retrieve the rows in batches (such as several hundred rows at a time) and only retrieve the next batch when the client is finished with the current batch. |

The **Fill** method uses the **DataReader** object implicitly to return the column names and types that are used to create the tables in the **DataSet**, and the data to populate the rows of the tables in the **DataSet**. Tables and columns are only created if they do not already exist; otherwise **Fill** uses the existing **DataSet** schema. Column types are created as .NET Framework types according to the tables in [Data Type Mappings in ADO.NET](http://msdn.microsoft.com/en-us/library/4e5xt97a.aspx). Primary keys are not created unless they exist in the data source and **DataAdapter.MissingSchemaAction** is set to **MissingSchemaAction.AddWithKey**. If **Fill** finds that a primary key exists for a table, it will overwrite data in the **DataSet** with data from the data source for rows where the primary key column values match those of the row returned from the data source. If no primary key is found, the data is appended to the tables in the **DataSet**. **Fill** uses any mappings that may exist when you populate the **DataSet** (see [DataAdapter DataTable and DataColumn Mappings (ADO.NET)](http://msdn.microsoft.com/en-us/library/ks92fwwh.aspx)).

|  |
| --- |
| **NoteNote** |
| If the **SelectCommand** returns the results of an OUTER JOIN, the **DataAdapter** does not set a **PrimaryKey** value for the resulting **DataTable**. You must define the **PrimaryKey** yourself to make sure that duplicate rows are resolved correctly. For more information, see [Defining Primary Keys (ADO.NET)](http://msdn.microsoft.com/en-us/library/z24kefs8.aspx). |

The following code example creates an instance of a [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx) that uses a [SqlConnection](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.aspx) to the Microsoft SQL Server **Northwind** database and populates a [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx) in a **DataSet** with the list of customers. The SQL statement and [SqlConnection](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.aspx) arguments passed to the [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx) constructor are used to create the [SelectCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.selectcommand.aspx) property of the [SqlDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldataadapter.aspx).

 Example

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl00_code');" \o "Copy Code)

' Assumes that connection is a valid SqlConnection object.

Dim queryString As String = \_

"SELECT CustomerID, CompanyName FROM dbo.Customers"

Dim adapter As SqlDataAdapter = New SqlDataAdapter( \_

queryString, connection)

Dim customers As DataSet = New DataSet

adapter.Fill(customers, "Customers")

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl53_ctl00_ctl01_code');" \o "Copy Code)

// Assumes that connection is a valid SqlConnection object.

string queryString =

"SELECT CustomerID, CompanyName FROM dbo.Customers";

SqlDataAdapter adapter = new SqlDataAdapter(queryString, connection);

DataSet customers = new DataSet();

adapter.Fill(customers, "Customers");

|  |
| --- |
| **NoteNote** |
| The code shown in this example does not explicitly open and close the **Connection**. The **Fill** method implicitly opens the **Connection** that the **DataAdapter** is using if it finds that the connection is not already open. If **Fill** opened the connection, it also closes the connection when **Fill** is finished. This can simplify your code when you deal with a single operation such as a **Fill** or an **Update**. However, if you are performing multiple operations that require an open connection, you can improve the performance of your application by explicitly calling the **Open** method of the **Connection**, performing the operations against the data source, and then calling the **Close** method of the **Connection**. You should try to keep connections to the data source open as briefly as possible to free resources for use by other client applications. |

 Multiple Result Sets

If the **DataAdapter** encounters multiple result sets, it creates multiple tables in the **DataSet**. The tables are given an incremental default name of Table*N*, starting with "Table" for Table0. If a table name is passed as an argument to the **Fill** method, the tables are given an incremental default name of TableName*N*, starting with "TableName" for TableName0.

 Populating a DataSet from Multiple DataAdapters

Any number of **DataAdapter**objects can be used with a **DataSet**. Each **DataAdapter** can be used to fill one or more **DataTable** objects and resolve updates back to the relevant data source. **DataRelation** and **Constraint** objects can be added to the **DataSet** locally, which enables you to relate data from dissimilar data sources. For example, a **DataSet** can contain data from a Microsoft SQL Server database, an IBM DB2 database exposed through OLE DB, and a data source that streams XML. One or more **DataAdapter** objects can handle communication to each data source.

**Example**

The following code example populates a list of customers from the **Northwind** database on Microsoft SQL Server 2000, and a list of orders from the **Northwind** database stored in Microsoft Access 2000. The filled tables are related with a **DataRelation**, and the list of customers is then displayed with the orders for that customer. For more information about **DataRelation** objects, see [Adding DataRelations (ADO.NET)](http://msdn.microsoft.com/en-us/library/ay82azad.aspx) and [Navigating DataRelations (ADO.NET)](http://msdn.microsoft.com/en-us/library/d6s958d6.aspx).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl55_ctl00_ctl02_code');" \o "Copy Code)

' Assumes that customerConnection is a valid SqlConnection object.

' Assumes that orderConnection is a valid OleDbConnection object.

Dim custAdapter As SqlDataAdapter = New SqlDataAdapter( \_

"SELECT \* FROM dbo.Customers", customerConnection)

Dim ordAdapter As OleDbDataAdapter = New OleDbDataAdapter( \_

"SELECT \* FROM Orders", orderConnection)

Dim customerOrders As DataSet = New DataSet()

custAdapter.Fill(customerOrders, "Customers")

ordAdapter.Fill(customerOrders, "Orders")

Dim relation As DataRelation = \_

customerOrders.Relations.Add("CustOrders", \_

customerOrders.Tables("Customers").Columns("CustomerID"), \_

customerOrders.Tables("Orders").Columns("CustomerID"))

Dim pRow, cRow As DataRow

For Each pRow In customerOrders.Tables("Customers").Rows

Console.WriteLine(pRow("CustomerID").ToString())

For Each cRow In pRow.GetChildRows(relation)

Console.WriteLine(vbTab & cRow("OrderID").ToString())

Next

Next

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl55_ctl00_ctl03_code');" \o "Copy Code)

// Assumes that customerConnection is a valid SqlConnection object.

// Assumes that orderConnection is a valid OleDbConnection object.

SqlDataAdapter custAdapter = new SqlDataAdapter(

"SELECT \* FROM dbo.Customers", customerConnection);

OleDbDataAdapter ordAdapter = new OleDbDataAdapter(

"SELECT \* FROM Orders", orderConnection);

DataSet customerOrders = new DataSet();

custAdapter.Fill(customerOrders, "Customers");

ordAdapter.Fill(customerOrders, "Orders");

DataRelation relation = customerOrders.Relations.Add("CustOrders",

customerOrders.Tables["Customers"].Columns["CustomerID"],

customerOrders.Tables["Orders"].Columns["CustomerID"]);

foreach (DataRow pRow in customerOrders.Tables["Customers"].Rows)

{

Console.WriteLine(pRow["CustomerID"]);

foreach (DataRow cRow in pRow.GetChildRows(relation))

Console.WriteLine("\t" + cRow["OrderID"]);

}

 SQL Server Decimal Type

By default, the **DataSet** stores data by using .NET Framework data types. For most applications, these provide a convenient representation of data source information. However, this representation may cause a problem when the data type in the data source is a SQL Server decimal or numeric data type. The .NET Framework **decimal** data type allows a maximum of 28 significant digits, whereas the SQL Server **decimal** data type allows 38 significant digits. If the **SqlDataAdapter** determines during a **Fill** operation that the precision of a SQL Server **decimal** field is larger than 28 characters, the current row is not added to the **DataTable**. Instead the **FillError** event occurs, which enables you to determine whether a loss of precision will occur, and respond appropriately. For more information about the **FillError** event, see [Handling DataAdapter Events (ADO.NET)](http://msdn.microsoft.com/en-us/library/6d1wk41s.aspx). To get the SQL Server **decimal** value, you can also use a [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx) object and call the [GetSqlDecimal](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.getsqldecimal.aspx) method.

ADO.NET 2.0 introduced enhanced support for [System.Data.SqlTypes](http://msdn.microsoft.com/en-us/library/system.data.sqltypes.aspx) in the **DataSet**. For more information, see [SqlTypes and the DataSet (ADO.NET)](http://msdn.microsoft.com/en-us/library/ms172137.aspx).

 OLE DB Chapters

Hierarchical rowsets, or chapters (OLE DB type **DBTYPE\_HCHAPTER**, ADO type **adChapter**) can be used to fill the contents of a **DataSet**. When the [OleDbDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdataadapter.aspx) encounters a chaptered column during a **Fill** operation, a **DataTable** is created for the chaptered column, and that table is filled with the columns and rows from the chapter. The table created for the chaptered column is named by using both the parent table name and the chaptered column name in the form "*ParentTableNameChapteredColumnName*". If a table already exists in the **DataSet** that matches the name of the chaptered column, the current table is filled with the chapter data. If there is no column in an existing table that matches a column found in the chapter, a new column is added.

Before the tables in the **DataSet** are filled with the data in the chaptered columns, a relation is created between the parent and child tables of the hierarchical rowset by adding an integer column to both the parent and child table, setting the parent column to auto-increment, and creating a **DataRelation** using the added columns from both tables. The added relation is named by using the parent table and chapter column names in the form "*ParentTableNameChapterColumnName*".

Note that the related column only exists in the **DataSet**. Subsequent fills from the data source can cause new rows to be added to the tables instead of changes being merged into existing rows.

Note also that, if you use the **DataAdapter.Fill** overload that takes a **DataTable**, only that table will be filled. An auto-incrementing integer column will still be added to the table, but no child table will be created or filled, and no relation will be created.

The following example uses the MSDataShape Provider to generate a chapter column of orders for each customer in a list of customers. A **DataSet** is then filled with the data.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl57_ctl00_ctl01_code');" \o "Copy Code)

Using connection As OleDbConnection = New OleDbConnection( \_

"Provider=MSDataShape;Data Provider=SQLOLEDB;" & \_

"Data Source=(local);Integrated " & \_

"Security=SSPI;Initial Catalog=northwind")

Dim adapter As OleDbDataAdapter = New OleDbDataAdapter( \_

"SHAPE {SELECT CustomerID, CompanyName FROM Customers} " & \_

"APPEND ({SELECT CustomerID, OrderID FROM Orders} AS Orders " & \_

"RELATE CustomerID TO CustomerID)", connection)

Dim customers As DataSet = New DataSet()

adapter.Fill(customers, "Customers")

End Using

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl57_ctl00_ctl02_code');" \o "Copy Code)

using (OleDbConnection connection = new OleDbConnection("Provider=MSDataShape;Data Provider=SQLOLEDB;" +

"Data Source=(local);Integrated Security=SSPI;Initial Catalog=northwind"))

{

OleDbDataAdapter adapter = new OleDbDataAdapter("SHAPE {SELECT CustomerID, CompanyName FROM Customers} " +

"APPEND ({SELECT CustomerID, OrderID FROM Orders} AS Orders " +

"RELATE CustomerID TO CustomerID)", connection);

DataSet customers = new DataSet();

adapter.Fill(customers, "Customers");

}

When the **Fill** operation is complete, the **DataSet** contains two tables: **Customers** and **CustomersOrders**, where **CustomersOrders** represents the chaptered column. An additional column named **Orders** is added to the **Customers** table, and an additional column named CustomersOrders is added to the **CustomersOrders** table. The **Orders** column in the **Customers** table is set to auto-increment. A **DataRelation**, **CustomersOrders**, is created by using the columns that were added to the tables with **Customers** as the parent table. The following tables show some sample results.

**TableName: Customers**

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **CompanyName** | **Orders** |
| ALFKI | Alfreds Futterkiste | 0 |
| ANATR | Ana Trujillo Emparedados y helados | 1 |

**TableName: CustomersOrders**

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **OrderID** | **CustomersOrders** |
| ALFKI | 10643 | 0 |
| ALFKI | 10692 | 0 |
| ANATR | 10308 | 1 |
| ANATR | 10625 | 1 |