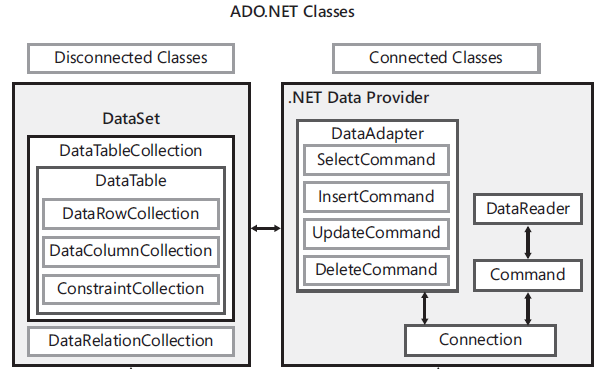
The disconnected: DataSet and DataTable



The disconnected data access are implemented in the **System.Data.dll** assembly from the .NET Framework. In System.Data namespace.

A *DataTable* object represents tabular data as an in-memory, tabular cache of rows, columns, and constraints. You typically use the *DataTable* class to perform any disconnected data access

DataTable cars = new DataTable ("Cars");

//Add the DataColumn using all properties

DataColumn vin = **new** DataColumn("Vin");

vin.DataType = typeof(string);

vin.MaxLength = 23;

vin.Unique = true;

vin.AllowDBNull = false;

vin.Caption = "VIN";

cars.Columns.Add(vin);

//Add the DataColumn using defaults

DataColumn make = new DataColumn("Make");

make.MaxLength = 35;

make.AllowDBNull = false;

cars.Columns.Add(make);

DataColumn year = new DataColumn("Year", typeof(int));

year.AllowDBNull = false;

cars.Columns.Add(year);

//Derived column using expression

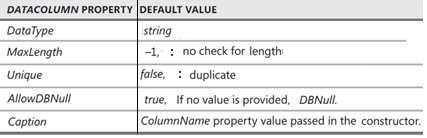
DataColumn yearMake = new DataColumn("Year and Make");

yearMake.DataType = typeof(string);

yearMake.MaxLength = 70;

yearMake.**Expression** = "Year + ' ' + Make";

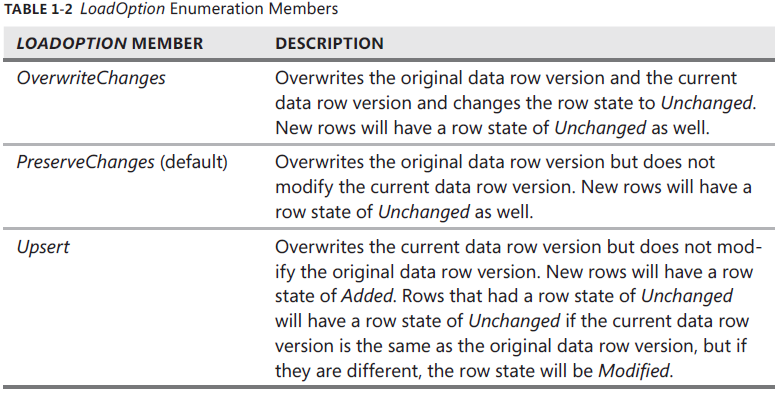
cars.Columns.Add(yearMake);



//Set the Primary Key

cars.PrimaryKey = new DataColumn[] {vin};

If you want to import ***DataRow*** objects that have been modified, you can use the ***ImportDataRow***method, which will preserve the original state and all other settings. The ***DataTable***class also provides several overloaded ***Load*** methods, which can be used to update existing *DataRow* objects or load new ***DataRow***objects. The data table requires the ***PrimaryKey***property to be set so the ***DataTable***object can locate the rows to be updated. If you need to generate a data row, you can use the ***LoadDataRow*** method, which accepts an array of objects, and a ***LoadOption***enumeration value. The possible values for the ***LoadOption*** enumeration are shown in Table 1-2.



|  |  |  |
| --- | --- | --- |
| First create DataRow | Simply add the values | Load datarow, replacing existing values |
| DataRow newCar = cars.NewRow();  newCar ["Vin"] = "123456789ABCD";  newCar ["Make"] = "Ford";  newCar ["Year"] = 2002;  cars.Rows.Add(newCar); | cars.Rows.Add("987654321XYZ",  "Buick",  2001); | cars.LoadDataRow(new object[]  {  "987654321XYZ",  "Jeep",  2002  }, LoadOption.OverwriteChanges); |

To set up an auto-increment column,

* Set *AutoIncrement* property of your data column to *true*.
* Set *AutoIncrementSeed* to the value of the first number you want
* Set *AutoIncrementStep* to the value you want to increment by each time a new row is added.

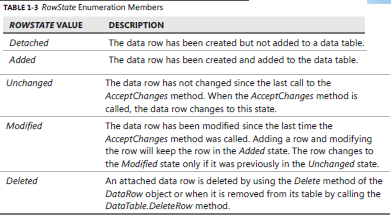
These auto-increment values are never sent to the database because the auto-increment column in the database table will provide a value when the new row is added. After each new row is added, the back-end database table **generates a new auto-increment number**, and then your application will query the database to get the newly created number.

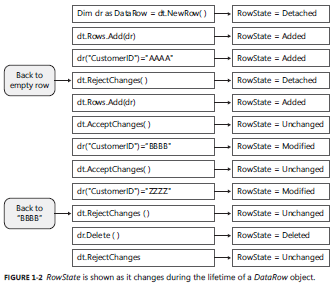
So what happens if you add new rows in your application to generate auto-increment values of 1 to 100 and then send these rows back to the database table, and the table already has 10 rows? When the first row is sent from your application, it has an auto-increment value of 1. The new auto-increment number created in the database will be 11. Your application queries for the 11 and tries to change the 1 to an 11 but throws an exception because 11 is already in your data table.

To solve this problem, set ***AutoIncrementSeed* to -1 and set *AutoIncrementStep* to -1**. This will cause negative numbers to be generated; they won’t conflict with the values coming from the database because the database doesn’t generate negative numbers.

***Viewing the State of the DataRow Object by Using DataRowState***

DataRow goes through a series of states that can be viewed and filtered at any time. You can retrieve the current state of a data row from its RowState property, which returns a value from the DataRowState enumeration. The DataRowState values are described in Table 1-3.





private void CreateSchema()

{

ds = new DataSet("VehiclesRepairs");

var vehicles = ds.Tables.Add("Vehicles");

vehicles.Columns.Add("VIN", typeof(string));

vehicles.Columns.Add("Make", typeof(string));

vehicles.Columns.Add("Year", typeof(int));

vehicles.**PrimaryKey** = new DataColumn[] { vehicles.Columns["VIN"] };

var repairs = ds.Tables.Add("Repairs");

var pk = repairs.Columns.Add("ID", typeof(int));

pk.AutoIncrement = true;

pk.AutoIncrementSeed = -1;

pk.AutoIncrementStep = -1;

repairs.Columns.Add("VIN", typeof(string));

repairs.Columns.Add("Description", typeof(string));

repairs.Columns.Add("Cost", typeof(decimal));

repairs.**PrimaryKey** = new DataColumn[] { repairs.Columns["ID"] };

ds.Relations.Add(

"vehicles\_repairs",

vehicles.Columns["VIN"],

repairs.Columns["VIN"]);

MessageBox.Show("Schema created!");

}