**C# Concepts – II**

SOLID Principle with C# Example

This also reminds me of the famous swiss knife. If one of them needs to be changed the whole set needs to be disturbed. No offense I am great fan of swiss knifes.



But if we can have each of those items separated its simple, easy to maintain and one change does not affect the other. The same principle also applies to classes and objects in software architecture.



So SRP says that a class should have only one responsibility and not multiple.So if we apply SRP we can move that logging activity to some other class who will only look after logging activities.

### 1. About SOLID

SOLID is basically 5 principles, which will help to create a good software architecture. You can see that all design patterns are based on these principles. SOLID is basically an acronym of the following:

* **S**is single responsibility principle (SRP)
* **O**stands for open closed principle (OCP)
* **L** Liskov substitution principle (LSP)
* **I** interface segregation principle (ISP)
* **D** Dependency injection principle (DIP)

I believe that with pictures, with examples, an article will be more approachable and understandable.

#### 1.1 Single responsibility principle (SRP)

A class should take one responsibility and there should be one reason to change that class. Now what does that mean? I want to share one picture to give a clear idea about this.

Now see this tool is a combination of so many different tools like knife, nail cutter, screw driver, etc. So will you want to buy this tool? I don’t think so. Because there is a problem with this tool, if you want to add any other tool to it, then you need to change the base and that is not good. This is a bad architecture to introduce into any system. It will be better if nail cutter can only be used to cut the nail or knife can only be used to cut vegetables.

Now I want to give one C# example on this principle:

Hide   Copy Code

namespace SRP

{

public class Employee

{

public int Employee\_Id { get; set; }

public string Employee\_Name { get; set; }

/// *<summary>*

/// *This method used to insert into employee table*

/// *</summary>*

/// *<param name="em">Employee object</param>*

/// *<returns>Successfully inserted or not</returns>*

public bool InsertIntoEmployeeTable(Employee em)

{

*// Insert into employee table.*

return true;

}

/// *<summary>*

/// *Method to generate report*

/// *</summary>*

/// *<param name="em"></param>*

public void GenerateReport(Employee em)

{

*// Report generation with employee data using crystal report.*

}

}

}

‘Employee’ class is taking 2 responsibilities, one is to take responsibility of employee database operation and another one is to generate employee report. Employee class should not take the report generation responsibility because suppose some days after your customer asked you to give a facility to generate the report in Excel or any other reporting format, then this class will need to be changed and that is not good.

So according to SRP, one class should take one responsibility so we should write one different class for report generation, so that any change in report generation should not affect the ‘Employee’ class.

Hide   Copy Code

public class ReportGeneration

{

/// *<summary>*

/// *Method to generate report*

/// *</summary>*

/// *<param name="em"></param>*

public void GenerateReport(Employee em)

{

*// Report reneration with employee data.*

}

}

#### 2.2 Open closed principle (OCP)

Now take the same ‘ReportGeneration’ class as an example of this principle. Can you guess what is the problem with the below class!!

Hide   Copy Code

public class ReportGeneration

{

/// *<summary>*

/// *Report type*

/// *</summary>*

public string ReportType { get; set; }

/// *<summary>*

/// *Method to generate report*

/// *</summary>*

/// *<param name="em"></param>*

public void GenerateReport(Employee em)

{

if (ReportType == "CRS")

{

*// Report generation with employee data in Crystal Report.*

}

if (ReportType == "PDF")

{

*// Report generation with employee data in PDF.*

}

}

}

Brilliant!! Yes you are right, too much ‘If’ clauses are there and if we want to introduce another new report type like ‘Excel’, then you need to write another ‘if’. This class should be open for extension but closed for modification. But how to do that!!

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

public class IReportGeneration

{

/// *<summary>*

/// *Method to generate report*

/// *</summary>*

/// *<param name="em"></param>*

public virtual void GenerateReport(Employee em)

{

*// From base*

}

}

/// *<summary>*

/// *Class to generate Crystal report*

/// *</summary>*

public class CrystalReportGeneraion : IReportGeneration

{

public override void GenerateReport(Employee em)

{

*// Generate crystal report.*

}

}

/// *<summary>*

/// *Class to generate PDF report*

/// *</summary>*

public class PDFReportGeneraion : IReportGeneration

{

public override void GenerateReport(Employee em)

{

*// Generate PDF report.*

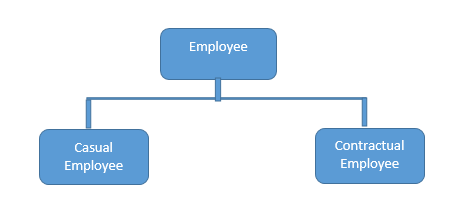
}

}

So if you want to introduce a new report type, then just inherit from IReportGeneration. So IReportGeneration is open for extension but closed for modification.

#### 2.3 Liskov substitution principle (LSP)

This principle is simple but very important to understand. Child class should not break parent class’s type definition and behavior. Now what is the meaning of this!! Ok let me take the same employee example to make you understand this principle. Check the below picture. Employee is a parent class and Casual and Contractual employee are the child classes, inhering from employee class.



Now see the below code:

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

public abstract class Employee

{

public virtual string GetProjectDetails(int employeeId)

{

return "Base Project";

}

public virtual string GetEmployeeDetails(int employeeId)

{

return "Base Employee";

}

}

public class CasualEmployee : Employee

{

public override string GetProjectDetails(int employeeId)

{

return "Child Project";

}

*// May be for contractual employee we do not need to store the details into database.*

public override string GetEmployeeDetails(int employeeId)

{

return "Child Employee";

}

}

public class ContractualEmployee : Employee

{

public override string GetProjectDetails(int employeeId)

{

return "Child Project";

}

*// May be for contractual employee we do not need to store the details into database.*

public override string GetEmployeeDetails(int employeeId)

{

throw new NotImplementedException();

}

}

Up to this is fine right? Now, check the below code and it will violate the LSP principle.

Hide   Copy Code

List<Employee> employeeList = new List<Employee>();

employeeList.Add(new ContractualEmployee());

employeeList.Add(new CasualEmployee());

foreach (Employee e in employeeList)

{

e.GetEmployeeDetails(1245);

}

Now I guess you got the problem. Yes right, for contractual employee, you will get not implemented exception and that is violating LSP. Then what is the solution? Break the whole thing in 2 different interfaces, 1. IProject 2. IEmployee and implement according to employee type.

Hide   Copy Code

public interface IEmployee

{

string GetEmployeeDetails(int employeeId);

}

public interface IProject

{

string GetProjectDetails(int employeeId);

}

Now, contractual employee will implement IEmployee not IProject. This will maintain this principle.

#### 2.4 Interface segregation principle (ISP)

This principle states that any client should not be forced to use an interface which is irrelevant to it. Now what does this mean, suppose there is one database for storing data of all types of employees (i.e. Permanent, non-permanent), now what will be the best approach for our interface?

Hide   Copy Code

public interface IEmployee

{

bool AddEmployeeDetails();

}

And all types of employee class will inherit this interface for saving data. This is fine right? Now suppose that company one day told to you that they want to read only data of permanent employees. What you will do, just add one method to this interface?

Hide   Copy Code

public interface IEmployeeDatabase

{

bool AddEmployeeDetails();

bool ShowEmployeeDetails(int employeeId);

}

But now we are breaking something. We are forcing non-permanent employee class to show their details from database. So, the solution is to give this responsibility to another interface.

Hide   Copy Code

public interface IAddOperation

{

bool AddEmployeeDetails();

}

public interface IGetOperation

{

bool ShowEmployeeDetails(int employeeId);

}

And non-permanent employee will implement only IAddOperation and permanent employee will implement both the interface.

#### 2.5 Dependency inversion principle (DIP)

This principle tells you not to write any tightly coupled code because that is a nightmare to maintain when the application is growing bigger and bigger. If a class depends on another class, then we need to change one class if something changes in that dependent class. We should always try to write loosely coupled class.

Suppose there is one notification system after saving some details into database.

Hide   Copy Code

public class Email

{

public void SendEmail()

{

*// code to send mail*

}

}

public class Notification

{

private Email \_email;

public Notification()

{

\_email = new Email();

}

public void PromotionalNotification()

{

\_email.SendEmail();

}

}

Now Notification class totally depends on Email class, because it only sends one type of notification. If we want to introduce any other like SMS then? We need to change the notification system also. And this is called tightly coupled. What can we do to make it loosely coupled? Ok, check the following implementation.

Hide   Shrink https://www.codeproject.com/images/arrow-up-16.png   Copy Code

public interface IMessenger

{

void SendMessage();

}

public class Email : IMessenger

{

public void SendMessage()

{

*// code to send email*

}

}

public class SMS : IMessenger

{

public void SendMessage()

{

*// code to send SMS*

}

}

public class Notification

{

private IMessenger \_iMessenger;

public Notification()

{

\_ iMessenger = new Email();

}

public void DoNotify()

{

\_ iMessenger.SendMessage();

}

}

Still Notification class depends on Email class. Now, we can use dependency injection so that we can make it loosely coupled. There are 3 types to DI, Constructor injection, Property injection and method injection.

##### Constructor Injection

Hide   Copy Code

public class Notification

{

private IMessenger \_iMessenger;

public Notification(Imessenger pMessenger)

{

\_ iMessenger = pMessenger;

}

public void DoNotify()

{

\_ iMessenger.SendMessage();

}

}

##### Property Injection

Hide   Copy Code

public class Notification

{

private IMessenger \_iMessenger;

public Notification()

{

}

public IMessenger MessageService

{

private get;

set

{

\_ iMessenger = value;

}

}

public void DoNotify()

{

\_ iMessenger.SendMessage();

}

}

##### Method Injection

Hide   Copy Code

public class Notification

{

public void DoNotify(IMessenger pMessenger)

{

pMessenger.SendMessage();

}

}

So, SOLID principle will help us to write loosely coupled code which is highly maintainable and less error prone.

So SRP says that a class should have only one responsibility and not multiple.So if we apply SRP we can move that logging activity to some other class who will only look after logging activities.

Hide   Copy Code

class FileLogger

{

public void Handle(string error)

{

System.IO.File.WriteAllText(@"c:\Error.txt", error);

}

}

Now customer class can happily delegate the logging activity to the “FileLogger” class and he can concentrate on customer related activities.

Hide   Copy Code

class Customer

{

private FileLogger obj = new FileLogger();

publicvirtual void Add()

{

try

{

*// Database code goes here*

}

catch (Exception ex)

{

obj.Handle(ex.ToString());

}

}

}

Now architecture thought process is an evolution. For some people who are seniors looking at above SRP example can contradict that even the try catch should not be handled by the customer class because that is not his work.

Yes, we can create a global error handler must be in theGlobal.asax file , assuming you are using ASP.NET and handle the errors in those section and make the customer class completely free.

So I will leave how far you can go and make this solution better but for now I want to keep this simple and let your thoughts have the freedom to take it to a great level.

Below is a great comment which talks about how we can take this SRP example to the next level.  
<http://www.codeproject.com/Articles/703634/SOLID-architecture-principles-using-simple-Csharp?msg=4729987#xx4729987xx>

## Understanding “O” - Open closed principle

Let’s continue with our same customer class example. I have added a simple customer type property to the class. This property decided if this is a “Gold” ora “Silver” customer.

Depending on the same it calculates discount. Have a look at the “getDiscount” function which returns discount accordingly. 1 for Gold customer and 2 for Silver customer.

Guess, what’s the problem with the below code. Hahaha, looks like this article will make you a GUESS champion;-) .

Ok, also let me add a HINT, look at the “IF” condition in the “getDiscount” function.

Hide   Copy Code

class Customer

{

private int \_CustType;

public int CustType

{

get { return \_CustType; }

set { \_CustType = value; }

}

public double getDiscount(double TotalSales)

{

if (\_CustType == 1)

{

return TotalSales - 100;

}

else

{

return TotalSales - 50;

}

}

}

The problem is if we add a new customer type we need to go and add one more “IF” condition in the “getDiscount” function, in other words we need to change the customer class.

If we are changing the customer class again and again, we need to ensure that the previous conditions with new one’s are tested again , existing client’s which are referencing this class are working properly as before.

In other words we are “MODIFYING” the current customer code for every change and every time we modify we need to ensure that all the previous functionalities and connected client are working as before.

How about rather than “MODIFYING” we go for “EXTENSION”. In other words every time a new customer type needs to be added we create a new class as shown in the below. So whatever is the current code they are untouched and we just need to test and check the new classes.

Hide   Copy Code

class Customer

{

public virtual double getDiscount(double TotalSales)

{

return TotalSales;

}

}

class SilverCustomer : Customer

{

public override double getDiscount(double TotalSales)

{

return base.getDiscount(TotalSales) - 50;

}

}

Hide   Copy Code

class goldCustomer : SilverCustomer

{

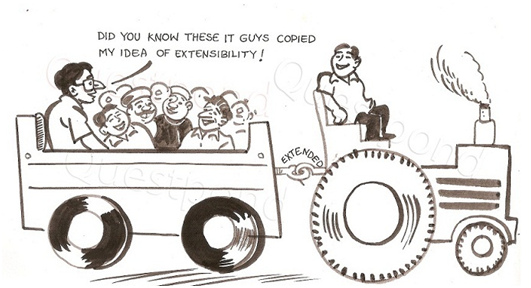
public override double getDiscount(double TotalSales)

{

return base.getDiscount(TotalSales) - 100;

}

}



Putting in simple words the “Customer” class is now closed for any new modification but it’s open for extensions when new customer types are added to the project.

## Understanding “L”- LSP (Liskov substitution principle)

Let’s continue with the same customer. Let’s say our system wants to calculate discounts for Enquiries. Now Enquiries are not actual customer’s they are just leads. Because they are just leads we do not want to save them to database for now.

So we create a new class called as Enquiry which inherits from the “Customer” class. We provide some discounts to the enquiry so that they can be converted to actual customers and we override the “Add’ method with an exception so that no one can add an Enquiry to the database.

Hide   Copy Code

class Enquiry : Customer

{

public override double getDiscount(double TotalSales)

{

return base.getDiscount(TotalSales) - 5;

}

public override void Add()

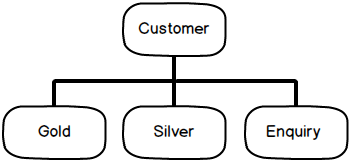
{

throw new Exception("Not allowed");

}

}

If you visualize the current customer inheritance hierarchy it looks something as shown below. In other word “Customer” is the parent class with “Gold” , “Silver” and “Enquiry” as child classes.



So as per polymorphism rule my parent “Customer” class object can point to any of it child class objects i.e. “Gold”, “Silver” or “Enquiry” during runtime without any issues.

So for instance in the below code you can see I have created a list collection of “Customer” and thanks to polymorphism I can add “Silver” , “Gold” and “Enquiry” customer to the “Customer” collection without any issues.

Thanks to polymorphism I can also browse the “Customer” list using the parent customer object and invoke the “Add” method as shown in the below code.

Now again let me tickle your brains, there is a slight problem here, THINK, THINK THINK.

HINT: -Watch when the Enquiry object is browsed and invoked in the “FOR EACH” loop.

Hide   Copy Code

List<Customer> Customers = new List<Customer>();

Customers.Add(new SilverCustomer());

Customers.Add(new goldCustomer());

Customers.Add(new Enquiry());

foreach (Customer o in Customers)

{

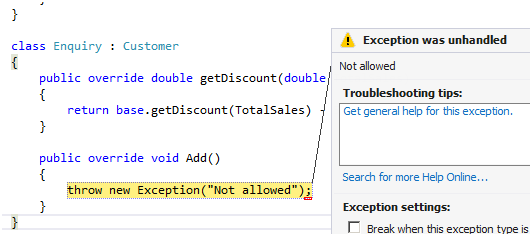
o.Add();

}

}

As per the inheritance hierarchy the “Customer” object can point to any one of its child objects and we do not expect any unusual behavior.

But when “Add” method of the “Enquiry” object is invoked it leads to below error because our “Equiry” object does save enquiries to database as they are not actual customers.



Now read the below paragraph properly to understand the problem. If you do not understand the below paragraph read it twice..

In other words the “Enquiry” has discount calculation , it looks like a “Customer” but **IT IS NOT A CUSTOMER.** So the parent cannot replace the child object seamlessly. In other words “Customer” is not the actual parent for the “Enquiry”class. “Enquiry” is a different entity altogether.

So LISKOV principle says the parent should easily replace the child object. So to implement LISKOV we need to create two interfaces one is for discount and other for database as shown below.

Hide   Copy Code

interface IDiscount

{

double getDiscount(double TotalSales);

}

interface IDatabase

{

void Add();

}

Now the “Enquiry” class will only implement “IDiscount” as he not interested in the “Add” method.

Hide   Copy Code

class Enquiry : IDiscount

{

public double getDiscount(double TotalSales)

{

return TotalSales - 5;

}

}

While the “Customer” class will implement both “IDiscount” as well as “IDatabase” as it also wants to persist the customer to the database.

Hide   Copy Code

class Customer : IDiscount, IDatabase

{

private MyException obj = new MyException();

public virtual void Add()

{

try

{

*// Database code goes here*

}

catch (Exception ex)

{

obj.Handle(ex.Message.ToString());

}

}

public virtual double getDiscount(double TotalSales)

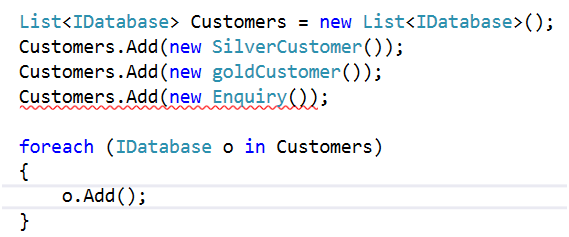
{

return TotalSales;

}

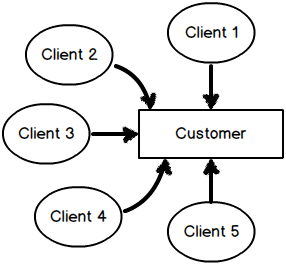
}

Now there is no confusion, we can create a list of “Idatabase” interface and add the relevant classes to it. In case we make a mistake of adding “Enquiry” class to the list compiler would complain as shown in the below code snippet.



## Understanding “I” - ISP (Interface Segregation principle)

Now assume that our customer class has become a SUPER HIT component and it’s consumed across 1000 clients and they are very happy using the customer class.



Now let’s say some new clients come up with a demand saying that we also want a method which will help us to “Read” customer data. So developers who are highly enthusiastic would like to change the “IDatabase” interfaceas shown below.

But by doing so we have done something terrible, can you guess ?

HINT: - Think about the effect of this change on the above image.

Hide   Copy Code

interface IDatabase

{

void Add(); *// old client are happy with these.*

voidRead(); *// Added for new clients.*

}

If you visualize the new requirement which has come up, you have two kinds of client’s: -

* Who want’s just use “Add” method.
* The other who wants to use “Add” + “Read”.

Now by changing the current interface you are doing an awful thing, disturbing the 1000 satisfied current client’s , even when they are not interested in the “Read” method. You are forcing them to use the “Read” method.

So a better approach would be to keep existing clients in their own sweet world and the serve the new client’s separately.

So the better solution would be to create a new interface rather than updating the current interface. So we can keep the current interface “IDatabase” as it is and add a new interface “IDatabaseV1” with the “Read” method the “V1” stands for version 1.

Hide   Copy Code

interface IDatabaseV1 : IDatabase *// Gets the Add method*

{

Void Read();

}

You can now create fresh classes which implement “Read” method and satisfy demands of your new clients and your old clients stay untouched and happy with the old interface which does not have “Read” method.

Hide   Copy Code

class CustomerwithRead : IDatabase, IDatabaseV1

{

public void Add()

{

Customer obj = new Customer();

Obj.Add();

}

Public void Read()

{

*// Implements logic for read*

}

}

So the old clients will continue using the “IDatabase” interface while new client can use “IDatabaseV1” interface.

Hide   Copy Code

IDatabase i = new Customer(); *// 1000 happy old clients not touched*

i.Add();

IDatabaseV1 iv1 = new CustomerWithread(); *// new clients*

Iv1.Read();

## Understanding “D”- Dependency inversion principle

In our customer class if you remember we had created a logger class to satisfy SRP. Down the line let’s say new Logger flavor classes are created.

Hide   Copy Code

class Customer

{

private FileLogger obj = new FileLogger();

public virtual void Add()

{

try

{

*// Database code goes here*

}

catch (Exception ex)

{

obj.Handle(ex.ToString());

}

}

}

Just to control things we create a common interface and using this common interface new logger flavors will be created.

Hide   Copy Code

interface ILogger

{

void Handle(string error);

}

Below are three logger flavors and more can be added down the line.

Hide   Copy Code

class FileLogger : ILogger

{

public void Handle(string error)

{

System.IO.File.WriteAllText(@"c:\Error.txt", error);

}

}

Hide   Copy Code

class EverViewerLogger : ILogger

{

public void Handle(string error)

{

*// log errors to event viewer*

}

}

Hide   Copy Code

class EmailLogger : ILogger

{

public void Handle(string error)

{

*// send errors in email*

}

}

Now depending on configuration settings different logger classes will used at given moment. So to achieve the same we have kept a simple IF condition which decides which logger class to be used, see the below code.

QUIZ time, what is the problem here.

HINT: - Watch the CATCH block code.

Hide   Copy Code

class Customer : IDiscount, IDatabase

{

private IException obj;

public virtual void Add(int Exhandle)

{

try

{

*// Database code goes here*

}

catch (Exception ex)

{

if (Exhandle == 1)

{

obj = new MyException();

}

else

{

obj = new EmailException();

}

obj.Handle(ex.Message.ToString());

}

}

The above code is again violating SRP but this time the aspect is different ,its about deciding which objects should be created. Now it’s not the work of “Customer” object to decide which instances to be created , he should be concentrating only on Customer class related functionalities.

If you watch closely the biggest problem is the “NEW” keyword. He is taking extra responsibilities of which object needs to be created.

So if we INVERT / DELEGATE this responsibility to someone else rather the customer class doing it that would really solve the problem to a certain extent.



So here’s the modified code with INVERSION implemented. We have opened the constructor mouth and we expect someone else to pass the object rather than the customer class doing it. So now it’s the responsibility of the client who is consuming the customer object to decide which Logger class to inject.

Hide   Copy Code

class Customer : IDiscount, IDatabase

{

private Ilogger obj;

public Customer(ILogger i)

{

obj = i;

}

}

So now the client will inject the Logger object and the customer object is now free from those IF condition which decide which logger class to inject. This is the Last principle in SOLID Dependency Inversion principle.

Customer class has delegated the dependent object creation to client consuming it thus making the customer class concentrate on his work.

Hide   Copy Code

IDatabase i = new Customer(new EmailLogger());

## Revising SOLID principles

S stands for SRP (Single responsibility principle):- A class should take care of only one responsibility.

O stands for OCP (Open closed principle):- Extension should be preferred over modification.

L stands for LSP (Liskov substitution principle):- A parent class object should be able to refer child objects seamlessly during runtime polymorphism.

I stands for ISP (Interface segregation principle):- Client should not be forced to use a interface if it does not need it.

D stands for DIP (Dependency inversion principle) :- High level modules should not depend on low level modules but should depend on abstraction.

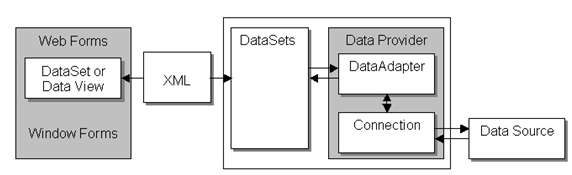
If you are already done with this article the next logical steps would be going through GOF Design Patterns, here’s [an article](http://www.codeproject.com/Articles/28309/Design-pattern-FAQ-Part-1-Training) for the same, Hope you enjoy it.

Once you understand SOLID the next step is to learn design patterns. Below is a nice video which teaches you Design Pattern with a project in 8 hours.

<https://www.codeproject.com/Articles/703634/SOLID-architecture-principles-using-simple-Csharp>

<https://www.codeproject.com/Tips/1033646/SOLID-Principle-with-Csharp-Example>

# Understanding ADO.NET Components

**ADO.NET Components**  
The ADO.NET is designed to work with multiple kinds of data sources in same fashion. You can categorize ADO.NET components in three categories: disconnected, common or shared and the .NET data providers. The disconnected components build the basic ADO.NET architecture. You can use these components (or classes) with or without data providers. For example, you can use a DataTable object with or without providers and shared or common components are the base classes for data providers. Shared or common components are the base classes for data providers and shared by all data providers. The data provider components are specifically designed to work with different kinds of data sources. For example, ODBC data providers work with ODBC data sources and OleDb data providers work with OLE-DB data sources.  
  
Figure represents the ADO.NET components model and how they work together:   
   
**Figure:**  The ADO.NET components model  
  
A data provider is a set of components, such as Connection, Command, DataAdapter and DataReader. The Connection is the first component that talks to a data source. The Connection object establishes a connection to a data source and works as a connection reference in Command and DataAdapter objects. A Command object executes a SQL query and stored procedures to read, add, update, and delete data of a data source via a DataAdapter. A DataAdapter is a bridge between a dataset and the connection. It uses Command Object to execute SQL queries and stored procedures.  
  
All data providers share the ADO.NET common components. These components represent the data. Some of the common components are DataSet, DataView, and DataViewManager. The DataSet uses XML to store and transfer data between the applications and the data provider. A DataSet is a set of DataTable objects. A DataTable represents a database table. The DataView and DataViewManager objects provide single or multiple views of a dataset. You can attach a DataView or a DataViewManager directly to dataâ€“bound controls such as a DataGrid or DataList. Other common components are DataTable, DataRow, DataColumn and so on. Now, I'll break down the ADO.NET model to show how it works.  
  
**Conclusion**  
Hope this article would have helped you in understanding ADO.NET components. See my other articles on the website on ADO.NET.

<http://www.c-sharpcorner.com/uploadfile/mahesh/understanding-ado-net-components/>

# ADO.NET Architecture

Data processing has traditionally relied primarily on a connection-based, two-tier model. As data processing increasingly uses multi-tier architectures, programmers are switching to a disconnected approach to provide better scalability for their applications.

## ADO.NET Components

The two main components of ADO.NET 3.0 for accessing and manipulating data are the .NET Framework data providers and the [DataSet](https://msdn.microsoft.com/en-us/library/system.data.dataset(v=vs.110).aspx).

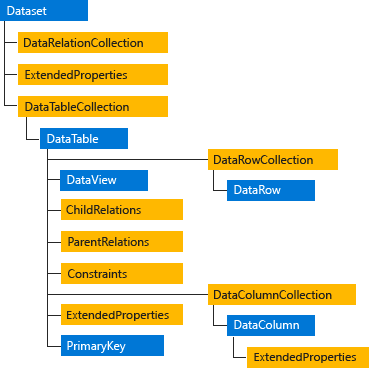
### .NET Framework Data Providers

The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data. The Connection object provides connectivity to a data source. The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information. The DataReader provides a high-performance stream of data from the data source. Finally, the DataAdapter provides the bridge between the DataSet object and the data source. The DataAdapter uses Command objects to execute SQL commands at the data source to both load the DataSet with data and reconcile changes that were made to the data in the DataSet back to the data source. For more information, see [.NET Framework Data Providers](https://msdn.microsoft.com/en-us/library/a6cd7c08(v=vs.110).aspx) and [Retrieving and Modifying Data in ADO.NET](https://msdn.microsoft.com/en-us/library/ms254937(v=vs.110).aspx).

### The DataSet

The ADO.NET DataSet is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application. The DataSet contains a collection of one or more [DataTable](https://msdn.microsoft.com/en-us/library/system.data.datatable(v=vs.110).aspx) objects consisting of rows and columns of data, and also primary key, foreign key, constraint, and relation information about the data in the DataTable objects. For more information, see [DataSets, DataTables, and DataViews](https://msdn.microsoft.com/en-us/library/ss7fbaez(v=vs.110).aspx).

The following diagram illustrates the relationship between a .NET Framework data provider and a DataSet.

  
ADO.NET architecture

### Choosing a DataReader or a DataSet

When you decide whether your application should use a DataReader (see [Retrieving Data Using a DataReader](https://msdn.microsoft.com/en-us/library/haa3afyz(v=vs.110).aspx)) or a DataSet (see [DataSets, DataTables, and DataViews](https://msdn.microsoft.com/en-us/library/ss7fbaez(v=vs.110).aspx)), consider the type of functionality that your application requires. Use a DataSet to do the following:

* Cache data locally in your application so that you can manipulate it. If you only need to read the results of a query, the DataReader is the better choice.
* Remote data between tiers or from an XML Web service.
* Interact with data dynamically such as binding to a Windows Forms control or combining and relating data from multiple sources.
* Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.

If you do not require the functionality provided by the DataSet, you can improve the performance of your application by using the DataReader to return your data in a forward-only, read-only manner. Although the DataAdapter uses the DataReader to fill the contents of a DataSet (see [Populating a DataSet from a DataAdapter](https://msdn.microsoft.com/en-us/library/bh8kx08z(v=vs.110).aspx)), by using the DataReader, you can boost performance because you will save memory that would be consumed by the DataSet, and avoid the processing that is required to create and fill the contents of the DataSet.

## LINQ to DataSet

LINQ to DataSet provides query capabilities and compile-time type checking over data cached in a DataSet object. It allows you to write queries in one of the .NET Framework development language, such as C# or Visual Basic. For more information, see [LINQ to DataSet](https://msdn.microsoft.com/en-us/library/bb386977(v=vs.110).aspx).

## LINQ to SQL

LINQ to SQL supports queries against an object model that is mapped to the data structures of a relational database without using an intermediate conceptual model. Each table is represented by a separate class, tightly coupling the object model to the relational database schema. LINQ to SQL translates language-integrated queries in the object model into Transact-SQL and sends them to the database for execution. When the database returns the results, LINQ to SQL translates the results back into objects. For more information, see [LINQ to SQL](https://msdn.microsoft.com/en-us/library/bb386976(v=vs.110).aspx).

## ADO.NET Entity Framework

The ADO.NET Entity Framework is designed to enable developers to create data access applications by programming against a conceptual application model instead of programming directly against a relational storage schema. The goal is to decrease the amount of code and maintenance required for data-oriented applications. For more information, see [ADO.NET Entity Framework](https://msdn.microsoft.com/en-us/library/bb399572(v=vs.110).aspx).

## WCF Data Services

WCF Data Services is used to deploy data services on the Web or an intranet. The data is structured as entities and relationships according to the specifications of the Entity Data Model. Data deployed on this model is addressable by standard HTTP protocol. For more information, see [WCF Data Services 4.5](https://msdn.microsoft.com/en-us/library/cc668792(v=vs.110).aspx).

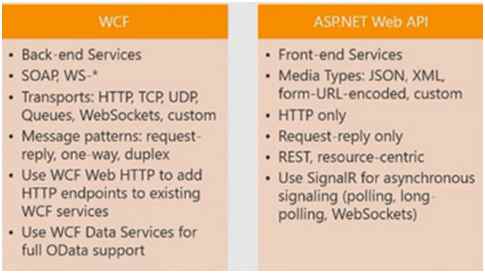
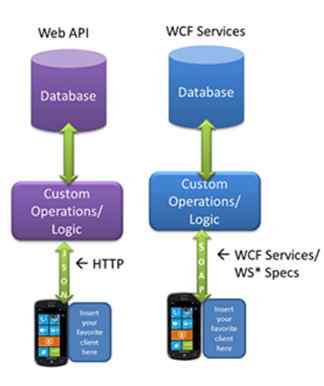
## XML and ADO.NET

ADO.NET leverages the power of XML to provide disconnected access to data. ADO.NET was designed hand-in-hand with the XML classes in the .NET Framework; both are components of a single architecture.

ADO.NET and the XML classes in the .NET Framework converge in the DataSet object. The DataSet can be populated with data from an XML source, whether it is a file or an XML stream. The DataSet can be written as World-Wide Web Consortium (W3C) compliant XML that includes its schema as XML schema definition language (XSD) schema, regardless of the source of the data in the DataSet. Because of the native serialization format of the DataSet is XML, it is an excellent medium for moving data between tiers, making the DataSet an optimal choice for remoting data and schema context to and from an XML Web service. For more information, see [XML Documents and Data](https://msdn.microsoft.com/en-us/library/2bcctyt8(v=vs.110).aspx).

<https://msdn.microsoft.com/en-us/library/27y4ybxw(v=vs.110).aspx>

# Who Is The Winner - Web API or WCF

At the end you will know the result for who is the winner and who is the loser. So, let's starts the race.  
  
As you know their are lots of technologies available that allow you to create services which can be consumed by different client, the client can be web application, window application, mobile application, etc. Services can be http enabled or support multiple protocols.   
  
As you know today we will discuss only about the WCF and Web API.   
  
**WCF** stands for Windows Communication Foundation, created by Microsoft with .NET Framework 3.0. It provides us a platform where we can build secure, reliable service that can integrate across the platforms and interoperate smoothly.  
  
**Web API**is also a framework that is used to make HTTP services. As you know, now a days we are using mobiles, tablets, apps and different types of services, so Web API is a simple and reliable platform to create HTTP enabled services that can reach wide range of clients. It is used to create complete REST services.  
  
  
  
So, let's start the race between Web API and WCF and see at the last that who is the winner.  
  
**SOAP**  
**Simple Object Access Protocol [SOAP]** is a protocol invented by Microsoft that is used to create a structured format data or message that can transfer over the internet. SOAP protocol is the main foundation of WCF and uses XML format. Web Service creates WSDL [Web Services Description Language] file which provides the information about the service data, so WSDL file is used by the client to understand the methods available on the service.  
  
**REST**  
REST stands for **Representational State Transfer**. It is not a protocol like SOAP whether it is architecture design pattern that is used for building and creating Web Services.  
  
Web API used this pattern to build web services. Basically Web API is a resource driven architecture that exposes endpoint to the client. Web API is fully REST supported. It uses GET, PUT, POST, DELETE verbs to communicate with the clients.   
  
**Interoperability**  
RESTful services like Web API are simple and light weight and easy to access by any client which supports HTTP calls. It is not like to SOAP where client need to understand WSDL to implement the service.   
  
WSDL uses some extra configuration and require some additional logic that might not be available on many devices such as phones, smart TVs, Tablets, etc.  
  
As WCF supports many protocol so it can consumed by lots of client that are using different protocol such as TCP, MSMQ, etc. not only HTTP protocol to access service.   
  
  
  
**Configuration Overhead**  
Client needs to make more configurations as compared to Web API to access WCF Service. WCF uses more configurations such as endpoints, behaviors, etc to expose the service. But Web API has more standard way to customize the service which helps it to reach more clients.   
  
**Speed**  
As RESTful services use HTTP protocol to access client and HTTP request/response are smaller than a SOAP request, so Web API is the best choice if you are going to create a service for low bandwidth devices or mobile devices to access client. HTTP request/response is also more readable as compared to SOAP because it contains header, body, etc. which makes it complex.  
  
**Security**  
WCF service provides us high level security framework which provide enterprise level security. It uses WS-I standard to provide secure service.  
  
But Web API uses web standard security such as basic authentication, token authentication and for more complex such as OAuth; Web API provides more flexibility. Then use SOAP when you are going to authenticate with external services such as google, facebook, etc.  
  
  
  
**Winner?**  
So, finally to see the above point, we cannot decide which is better than the other. But I want to focus on the two points. Firstly, if you are going to create a service which would be used on different platforms, then go with WCF. Secondly, if you are creating internet service which is going to use external resource, then go with Web API.

<http://www.c-sharpcorner.com/UploadFile/8a67c0/who-is-winner-web-api-or-wcf/>

# Difference between Stored Procedure and Function in SQL Server

1. Function must return a value but in Stored Procedure it is optional( Procedure can return zero or n values).
2. Functions can have only input parameters for it whereas Procedures can have input/output parameters .
3. Functions can be called from Procedure whereas Procedures cannot be called from Function.

## Advance Difference

1. Procedure allows SELECT as well as DML(INSERT/UPDATE/DELETE) statement in it whereas Function allows only SELECT statement in it.
2. Procedures can not be utilized in a SELECT statement whereas Function can be embedded in a SELECT statement.
3. Stored Procedures cannot be used in the SQL statements anywhere in the WHERE/HAVING/SELECT section whereas Function can be.
4. Functions that return tables can be treated as another rowset. This can be used in JOINs with other tables.
5. Inline Function can be though of as views that take parameters and can be used in JOINs and other Rowset operations.
6. Exception can be handled by try-catch block in a Procedure whereas try-catch block cannot be used in a Function.
7. We can go for Transaction Management in Procedure whereas we can't go in Function.

<http://www.dotnettricks.com/learn/sqlserver/difference-between-stored-procedure-and-function-in-sql-server>

There are some similarities and differences between an interface and an abstract class that I have arranged in a table for easier comparison:

# Abstract Class versus Interface

| **Feature** | **Interface** | **Abstract class** |
| --- | --- | --- |
| Multiple inheritance | A class may inherit several interfaces. | A class may inherit only one abstract class. |
| Default implementation | An interface cannot provide any code, just the signature. | An abstract class can provide complete, default code and/or just the details that have to be overridden. |
| Access Modfiers | An interface cannot have access modifiers for the subs, functions, properties etc everything is assumed as public | An abstract class can contain access modifiers for the subs, functions, properties |
| Core VS Peripheral | Interfaces are used to define the peripheral abilities of a class. In other words both Human and Vehicle can inherit from a IMovable interface. | An abstract class defines the core identity of a class and there it is used for objects of the same type. |
| Homogeneity | If various implementations only share method signatures then it is better to use Interfaces. | If various implementations are of the same kind and use common behaviour or status then abstract class is better to use. |
| Speed | Requires more time to find the actual method in the corresponding classes. | Fast |
| Adding functionality (Versioning) | If we add a new method to an Interface then we have to track down all the implementations of the interface and define implementation for the new method. | If we add a new method to an abstract class then we have the option of providing default implementation and therefore all the existing code might work properly. |
| Fields and Constants | No fields can be defined in interfaces | An abstract class can have fields and constrants defined |

<https://www.codeproject.com/Articles/11155/Abstract-Class-versus-Interface>

# Indexes in MS SQL Server

## Introduction

I bought a book from crossword; he packed the book and added two bookmarks into my pack. A thought came to my mind. Why do I need a bookmark? I can easily memorize the page number and the next time resume from the same page when I resume reading, or read them all over to reach to the point where I stopped reading. But not all have a blessed memory; moreover, there are better things to remember, my grandpa would rather bookmark and rely on it to help him resume reading. It’s a kind of simple index, isn’t it?

This article focuses on how MS SQL Server uses indexes to read and write data. Data is arranged by SQL Server in the form of extents and pages. Each extent is of size 64 KB, having 8 pages of 8KB sizes. An extent may have data from multiple or same table, but each page holds data from a single table only. Logically, data is stored in record sets in the table. We have fields (columns) identifying the type of data contained in each of the record sets. A table is nothing but a collection of record sets; by default, rows are stored in the form of heaps unless a clustered index has been defined on the table, in which case, record sets are sorted and stored on the clustered index. The heaps structure is a simple arrangement where the inserted record is stored in the next available space on the table page.

Heaps seem a great option when the motive is simply storing data, but when data retrieval steps in, this option back fires. An index acts as a fire fighter in this scenario. Indexes are arranged in the form of a B-Tree where the leaf node holds the data or a pointer to the data. Since the stored data is in a sorted order, indexes precisely know which record is sitting where. Hence an index optimizes and enhances the data retrieval immensely.

But everything comes at a cost; the price we pay for having an index on the table is, each time there is an Insert/Update/Delete, SQL Server updates the active indexes on the table where these DML are operated. Hence simply creating indexes madly for the sake of better data retrieval will not serve the purpose. If there are 20 indexes on a table, each time a DML is done on the table, all these 20 indexes shall be updated so that they can uniquely figure out the location of the record. Let’s dive deep into the indexes.

**Setup**: All code has been tested on MS SQL Server 2008 R2.

## Clustered Index (CI)

A clustered index is something that reorganizes the way records in the table are physically stored. Therefore a table can have only one clustered index. The leaf nodes of a clustered index contain the data pages, by which I mean the key-value pair in the clustered index has the index key and the actual data value. Also remember, a clustered index will be created on a table by default the moment a primary key is created on the table. A clustered index is something like your train ticket B4/24, you know that you need to board coach B4 and sit on seat number 24. So this index physically leads you to your actual seat.

We will follow this up closely with an example:

Hide   Copy Code

USE TestDB

GO

CREATE TABLE Sales(

ID INT IDENTITY(1,1)

,ProductCode VARCHAR(20)

,Price FLOAT(53)

,DateTransaction DATETIME);

I have created a table Sales, and then created a Stored Procedure to insert 2,00,000 records into the Sales table. This sizable chunk of data will help us to notice the differences very clearly.

Hide   Copy Code

CREATE PROCEDURE InsertIntoSales

AS

SET NOCOUNT ON

BEGIN

DECLARE @PC VARCHAR(20)='A12CB'

DECLARE @Price INT = 50

DECLARE @COUNT INT = 0

WHILE @COUNT<200000

BEGIN

SET @PC=@PC+CAST(@COUNT AS VARCHAR(20))

SET @Price=@Price+@COUNT

INSERT INTO Sales VALUES (@PC,@Price,GETDATE())

SET @PC='A12CB'

SET @Price=50

SET @COUNT+=1

END

END

EXEC InsertIntoSales

Now we have created the table and inserted 2,00,000 records into it, but there is no index defined on any column.

Press Control+M. This will “Include the Actual Execution Plan” in the results. Let's run the below query.

Hide   Copy Code

SET STATISTICS IO ON

SELECT \* FROM Sales WHERE ID=189923

ID ProductCode Price DateTransaction

*----------- ---------------- ---------- -----------------------*

189923 A12CB189922 189972 2011-03-21 12:07:48.310

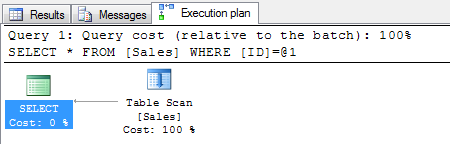
(1 row(s) affected)

Table 'Sales'. Scan count 1, logical reads 1129, physical reads 0,

read-ahead reads 0, lob logical reads 0,

lob physical reads 0, lob read-ahead reads 0.

(1 row(s) affected)



The Execution plan tab on the results show that the record has been retrieved on a table scan and the logical reads are 1129.

Now let’s build a clustered index on the ID column of the Sales table.

Hide   Copy Code

CREATE CLUSTERED INDEX CL\_ID ON SALES(ID);

Let us press CTRL+M and rerun the same query:

Hide   Copy Code

SET STATISTICS IO ON

SELECT \* FROM Sales WHERE ID=189923

ID ProductCode Price DateTransaction

*----------- ---------------- ------------ -----------------------*

189923 A12CB189922 189972 2011-03-21 12:07:48.310

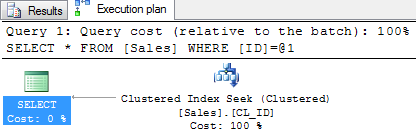
(1 row(s) affected)

Table 'Sales'. Scan count 1, logical reads 3, physical reads 0,

read-ahead reads 0, lob logical reads 0,

lob physical reads 0, lob read-ahead reads 0.

(1 row(s) affected)



The Execution plan tab on the results shows that the record has been retrieved on Index seek and the logical reads are 3. After the clustered index creation, SQL Server has been able to reduce the logical reads dramatically and the query has been optimized. Clearly the index knows where to look for the record.

## Non-Clustered Index (NCI)

A non-clustered index is a special type of index in which the logical order of the index does not match the physical stored order of the rows on disk. The leaf node of a non-clustered index does not consist of the data pages but a pointer to it. That goes to say that a non-clustered index can’t survive on its own - it needs a base to live on. A non-clustered index uses a clustered index (if defined) or the heap to build itself.

When a non-clustered index uses the heap, the leaf node (or the pointer) is a physical location of the data. When it uses a clustered index, the leaf node (or the pointer) is the clustered index key value and this key value in turn points to the actual data.

## Part I: When NCI Uses a CI

Getting back to Sales, we already have a CI (CL\_ID) on the ID column, now if we have a query something like:

Hide   Copy Code

SET STATISTICS IO ON

SELECT \* FROM Sales WHERE ProductCode like 'A12CB908%' order by Price

Press Control+M and execute the query

There are arround 111 records retrived

*-----------------------------------------------------*

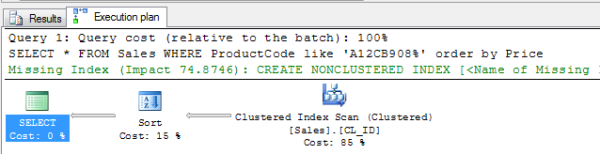
(111 row(s) affected)

Table 'Sales'. Scan count 1, logical reads 1130, physical reads 0,

read-ahead reads 0, lob logical reads 0,

lob physical reads 0, lob read-ahead reads 0.

(1 row(s) affected)



We find that the query first uses the clustered index to get 111 records and then uses a sort operation; the logical reads are as high as 1130. There is also a missing index suggestion.

Let’s consider SQL Server’s advice and create a non-clustered index (NONCI\_PC) on the ProductCode column. Since we have a CI already, this NCI would be built on the CI.

Hide   Copy Code

CREATE NONCLUSTERED INDEX NONCI\_PC ON SALES(ProductCode);

Press Control+M and rerun the same query, this time around, we can see the data fetch plan change.

Hide   Copy Code

SET STATISTICS IO ON

SELECT \* FROM Sales WHERE ProductCode like 'A12CB908%' order by Price

*-------------------------------*

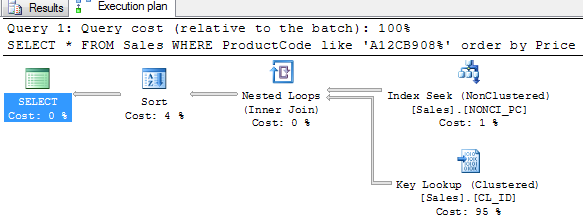
(111 row(s) affected)

Table 'Sales'. Scan count 1, logical reads 351, physical reads 0,

read-ahead reads 7, lob logical reads 0,

lob physical reads 0, lob read-ahead reads 0.

(1 row(s) affected)



The logical reads have been minimized and the revised execution plan is as in the figure. This was the example where a **non-clustered index used a clustered index**.

## Part II: When NCI Uses a Heap

When there is no clustered index built on a table and a non-clustered index is built, it uses the heap for data retrieval. The indexed column or columns are sorted along with a pointer to the physical location of the data.

The big question is, how do I know if I should create an NCI on a CI or on a heap?

The answer is in the query, if data is queried typically on one particular column, it would be beneficial to build NCI upon a CI, but when the case is something like what we have in the present Sales example where we will be building a one-column NC index on a heap, the NCI would be merely a two-column table containing the key-value pair (index key and physical location, i.e., value). This would be the best optimization in this scenario. To follow this up, let’s review the example closely.

With respect to the sales example, let’s delete the clustered index CL\_ID created on the ID column and re-evaluate.

Hide   Copy Code

DROP INDEX Sales.CL\_ID;

SET STATISTICS IO ON

SELECT \* FROM Sales WHERE ProductCode like 'A12CB908%' order by Price

*------------------------------------*

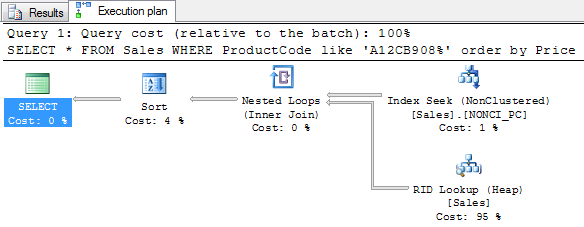
(111 row(s) affected)

Table 'Sales'. Scan count 1, logical reads 114, physical reads 0,

read-ahead reads 0, lob logical reads 0,

lob physical reads 0, lob read-ahead reads 0.

(1 row(s) affected)



The logical reads have been further optimized and the execution plan also has been revised. In this case, the query uses the non-clustered index to be run on the heap.

We have been able to create indexes so that our queries work with a minimal performance overhead. So now the next big question is, Wouldn’t it be great if there was someone to help us out in prompting on the indexes to be built based on our queries?

Yes there is, DTA reports work for helping us out with this..

<https://www.codeproject.com/Articles/190263/Indexes-in-MS-SQL-Server>

# DataRowState Enumeration

Gets the state of a [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx) object.

This enumeration has a [FlagsAttribute](https://msdn.microsoft.com/en-us/library/system.flagsattribute(v=vs.110).aspx) attribute that allows a bitwise combination of its member values.

[FlagsAttribute]

public enum DataRowState

## Members

|  |  |  |
| --- | --- | --- |
|  | **Member name** | **Description** |
|  | Added | The row has been added to a [DataRowCollection](https://msdn.microsoft.com/en-us/library/system.data.datarowcollection(v=vs.110).aspx), and [AcceptChanges](https://msdn.microsoft.com/en-us/library/system.data.datarow.acceptchanges(v=vs.110).aspx) has not been called. |
|  | Deleted | The row was deleted using the [Delete](https://msdn.microsoft.com/en-us/library/system.data.datarow.delete(v=vs.110).aspx) method of the [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx). |
|  | Detached | The row has been created but is not part of any [DataRowCollection](https://msdn.microsoft.com/en-us/library/system.data.datarowcollection(v=vs.110).aspx). A [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx) is in this state immediately after it has been created and before it is added to a collection, or if it has been removed from a collection. |
|  | Modified | The row has been modified and [AcceptChanges](https://msdn.microsoft.com/en-us/library/system.data.datarow.acceptchanges(v=vs.110).aspx) has not been called. |
|  | Unchanged | The row has not changed since [AcceptChanges](https://msdn.microsoft.com/en-us/library/system.data.datarow.acceptchanges(v=vs.110).aspx) was last called. |

## Remarks

The DataRowState enumeration is returned by the [RowState](https://msdn.microsoft.com/en-us/library/system.data.datarow.rowstate(v=vs.110).aspx) property of the [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx) class.

## Examples

The following example first creates a new [DataTable](https://msdn.microsoft.com/en-us/library/system.data.datatable(v=vs.110).aspx) with one column, then creates a single [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx). As the [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx) is created, added, modified, and deleted, its [RowState](https://msdn.microsoft.com/en-us/library/system.data.datarow.rowstate(v=vs.110).aspx) is printed.

private void DemonstrateRowState() {

//Run a function to create a DataTable with one column.

DataTable myTable = MakeTable();

DataRow myRow;

// Create a new DataRow.

myRow = myTable.NewRow();

// Detached row.

Console.WriteLine("New Row " + myRow.RowState);

myTable.Rows.Add(myRow);

// New row.

Console.WriteLine("AddRow " + myRow.RowState);

myTable.AcceptChanges();

// Unchanged row.

Console.WriteLine("AcceptChanges " + myRow.RowState);

myRow["FirstName"] = "Scott";

// Modified row.

Console.WriteLine("Modified " + myRow.RowState);

myRow.Delete();

// Deleted row.

Console.WriteLine("Deleted " + myRow.RowState);

}

private DataTable MakeTable(){

// Make a simple table with one column.

DataTable dt = new DataTable("myTable");

DataColumn dcFirstName = new DataColumn("FirstName", Type.GetType("System.String"));

dt.Columns.Add(dcFirstName);

return dt;

}

# DataRow.RowState Property

Gets the current state of the row with regard to its relationship to the [DataRowCollection](https://msdn.microsoft.com/en-us/library/system.data.datarowcollection(v=vs.110).aspx).

public DataRowState RowState { get; }

#### Property Value

Type: [System.Data.DataRowState](https://msdn.microsoft.com/en-us/library/system.data.datarowstate(v=vs.110).aspx)

One of the [DataRowState](https://msdn.microsoft.com/en-us/library/system.data.datarowstate(v=vs.110).aspx) values.

## Remarks

The value of the RowState depends on two factors: the kind of operation has been performed on the row, and whether [AcceptChanges](https://msdn.microsoft.com/en-us/library/system.data.datarow.acceptchanges(v=vs.110).aspx) has been called on the [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx).

## Examples

The following example first creates a new [DataTable](https://msdn.microsoft.com/en-us/library/system.data.datatable(v=vs.110).aspx) with one column, and then creates a single [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx). As the [DataRow](https://msdn.microsoft.com/en-us/library/system.data.datarow(v=vs.110).aspx) is created, added, modified, and deleted, its RowState is printed.

private void DemonstrateRowState()

{

// Run a function to create a DataTable with one column.

DataTable table = MakeTable();

DataRow row;

// Create a new DataRow.

row = table.NewRow();

// Detached row.

Console.WriteLine("New Row " + row.RowState);

table.Rows.Add(row);

// New row.

Console.WriteLine("AddRow " + row.RowState);

table.AcceptChanges();

// Unchanged row.

Console.WriteLine("AcceptChanges " + row.RowState);

row["FirstName"] = "Scott";

// Modified row.

Console.WriteLine("Modified " + row.RowState);

row.Delete();

// Deleted row.

Console.WriteLine("Deleted " + row.RowState);

}

private DataTable MakeTable()

{

// Make a simple table with one column.

DataTable table = new DataTable("table");

DataColumn dcFirstName = new DataColumn(

"FirstName", Type.GetType("System.String"));

table.Columns.Add(dcFirstName);

return table;

}

<https://msdn.microsoft.com/en-us/library/system.data.datarow.rowstate(v=vs.110).aspx>

# Safe Type Casting with IS and AS Operator

Type Casting is the mechanism to convert one data type to another. While type casting of one data type to another, we get exception if the previous one data type is not compatible with the new data type. To avoid this exception, we have IS and AS operator in C# for safe type casting. Let's understand how to use both of them.

## IS Operator

The IS operator checks whether the type of an given object is compatible with the new object type. It returns boolean type value : true if given object is compatible with new one, else false. In this way IS operator help you to do safe type casting.

### How to do it..

1. **Object obj = new Object(); *// Creates a new Object obj***
2. ***// checking compatibility of obj object with other type***
3. **Boolean b1 = (obj is Object); *// b1 is set to true.***
4. **Boolean b2 = (obj is Employee); *// The cast fails: no exception is thrown, but b2 is set to false.***
5. ***//we can also use it***
6. **if (obj is Employee)**
7. **{**
8. **Employee emp = (Employee) obj;**
9. ***// TO DO:***
10. **}**

#### Note

1. If the reference of the given object is null, the IS operator will return false since there is no object available to check its type.

In this way, CLR is checking the obj object type twice. First time with in the if condition and if it is true, with in the if block. Actually this way affect the performance since each and every time CLR will walk the inheritance hierarchy, checking each base type against the specified type (Employee). To avoid this we have AS operator.

## AS Operator

The AS operator also checks whether the type of an given object is compatible with the new object type. It returns non-null if given object is compatible with new one, else null. In this way AS operator help you to do safe type casting. The above code can be re-written by using AS operator in a better way.

### How to do it..

1. **Object obj = new Object(); *// Creates a new Object obj***
2. ***// checking compatibility of obj object with other type***
3. **Employee emp = obj as Employee; *// The cast fails: no exception is thrown, but emp is set to null.***
4. **if (emp != null)**
5. **{**
6. ***// TO:DO***
7. **}**

#### Note

1. If the reference of the given object is null, the AS operator will return NULL since there is no object available to check its type.
2. AS operator performs only reference conversions, nullable conversions, and boxing conversions. This operator cannot perform other conversions like as user-defined conversions.

In this way, CLR is checking the obj object type only one time. Hence AS operator provide good performance over IS operator. Now if you want to use emp object then it will throw NullReferenceException as given below.

1. **Object obj = new Object();**
2. **Employee emp = obj as Employee; *// try to Cast obj to an Employee***
3. ***// The above cast fails: no exception is thrown, but emp is set to null.***
4. **emp.ToString(); *// Accessing emp throws a NullReferenceException.***

<http://www.dotnettricks.com/learn/csharp/safe-type-casting-with-is-and-as-operator>