A DEEP DIVE INTO C# INTERFACE

An interface acts as a contract between itself and any class or struct which implements it. It means a class that implement an interface is bound to implement all its members. Interface has only member's declaration or signature and implicitly every member of an interface is public and abstract.

For example, the most common use of interfaces is, within SOA (Service Oriented Architecture). In SOA (WCF), a service is exposed through interfaces to different clients. Typically, an interface is exposed to a group of clients which needs to use common functionalities.

```
1. interface IStore
2. {
3.  void Read();
4.  void Write();
5. }
6.
7. interface ICompress
8. {
9.  void Compress();
10.   void Decompress();
11.  }
12.
13.  public class Document : IStore, ICompress
14.  {
15.   #region IStore
16.
17.   public void Read()
18.   {
19.   Console.WriteLine("Executing Document's Read Method for IStore");
20.   }
21.
22.   public void Write()
23.   {
24.   Console.WriteLine("Executing Document's Write Method for IStore");
25.   }
26.
```

```
27.   #endregion // IStore
28.
29.   #region ICompress
30.
31.   public void Compress()
32.   {
33.   Console.WriteLine("Executing   Document's   Compress   Method   for
   ICompress");
34.   }
35.   public void Decompress()
36.   {
37.   Console.WriteLine("Executing   Document's   Decompress   Method   for
   ICompress");
38.   }
39.   #endregion // ICompress
40.  }
```

Features of Interface

1. An interface doesn't provide inheritance like a class or abstract class but it only declare members which an implementing class need to be implement.
2. It cannot be instantiated but it can be referenced by the class object which implements it. Also, Interface reference works just like object reference and behave like as object.

```
1. IStore IObjStore = new Document();
2. ICompress IObjCompress = new Document();
```

3. It contains only properties, indexers, methods, delegates and events signature.
4. It cannot contains constants members, constructors, instance variables, destructors, static members or nested interfaces.
5. Members of an interface cannot have any access modifiers even public.
6. Implicitly, every member of an interface is public and abstract. Also, you are not allowed to specify the members of an interface public and abstract or virtual.
7. An interface can be inherited from one or more interfaces.
8. An interface can extend another interface.
9. A class or struct can implements more than one interfaces.
10. A class that implements an interface can mark any method of the interface as virtual and this method can be overridden by derived classes.

11. Implementing multiple interfaces by a class, sometimes result in a conflict between member signatures. You can resolve such conflicts by explicitly implementing an interface member.

```
interface IStore
{
 void Read();
 void Write();
}

interface ICompress
{
 void Compress();
  void Decompress();
 }

 public class Document : IStore, ICompress
 {
  #region IStore Explicit Implementation

  public void IStore.Read()
  {
  Console.WriteLine("Executing Document's Read Method for IStore");
  }

  public void IStore.Write()
  {
  Console.WriteLine("Executing Document's Write Method for IStore");
  }

  #endregion // IStore

  #region ICompress Implicit Implementation

  public void Compress()
  {
  Console.WriteLine("Executing Document's Compress Method for
ICompress");
```

```
34.   }
35.   public void Decompress()
36.   {
37.   Console.WriteLine("Executing   Document's   Decompress   Method   for
   ICompress");
38.   }
39.   #endregion // ICompress
40. }
```

12. It is a good practice to start all interface names with a capital "I" letter.

Common design guidelines for Interface

1. Keep your interfaces focused on the problem you are trying to solve and keep related tasks (methods) into a interface. Interfaces that have multiple unrelated tasks tend to be very difficult to implement in a class. Split up interfaces that contain unrelated functionality.
2. Make sure your interface does not contain too many methods. Since too many methods makes implementing the interface difficult as the implementing class has to implement each and every method in the interface.
3. Don't make interfaces for specific functionality. An interface should define the common functionality that can be implemented by the classes of different modules or subsystems.

When to use

1. Need to provide common functionality to unrelated classes.
2. Need to group objects based on common behaviors.
3. Need to introduce polymorphic behavior to classes since a class can implements more than one interfaces.
4. Need to provide more abstract view to a model which is unchangeable.
5. Need to create loosely coupled components, easily maintainable and pluggable components (like log4net framework for logging) because implementation of an interface is separated from itself.

Disadvantage of Interface

1. The main issue with an interface is that when you add a new members to its, then you must implement those members within all of the classes which implement that interface.
2. Interfaces are slow as these required extra in-direction to find corresponding method in in the actual class.

Source : http://www.dotnet-tricks.com/Tutorial/csharp/5T27291013-A-Deep-Dive-into-C#-Interface.html