[IDataErrorInfo](http://msdn.microsoft.com/en-us/library/system.componentmodel.idataerrorinfo%28v=vs.100%29.aspx)

|  |
| --- |
| I've always implemented the [IDataErrorInfo](http://msdn.microsoft.com/en-us/library/system.componentmodel.idataerrorinfo%28v=vs.100%29.aspx) interface without actually wondering what this line means and how it works.  string IDataErrorInfo.this[string propertyName]  {  get { return this.GetValidationError(propertyName); }  }  How does .this[string propertyName] work, and when/how does this property get called |
| |  |  |  |  | | --- | --- | --- | --- | | |  |  | | --- | --- | | 3 |  | | I *think* this is [explicit interface implementation](http://msdn.microsoft.com/en-us/library/vstudio/ms173157.aspx) of an [indexer](http://msdn.microsoft.com/en-us/library/vstudio/6x16t2tx.aspx) and it would be called whenever you have an explicitly typed IDataErrorInfo object where you write: string myPropertyError = myDataErrorInfo["SomePropertyName"];– [Chris Sinclair](http://stackoverflow.com/users/1269654/chris-sinclair)[Jun 21 '13 at 13:28](http://stackoverflow.com/questions/17236475/how-does-idataerrorinfo-thisstring-propertyname-work-in-c#comment24974633_17236475) | | |  |  | | --- | --- | |  |  | | Yup, it's just explicit interface implementation. Are you aware of that in general? (If not, just look it up.)– [Jon Skeet](http://stackoverflow.com/users/22656/jon-skeet)[Jun 21 '13 at 13:30](http://stackoverflow.com/questions/17236475/how-does-idataerrorinfo-thisstring-propertyname-work-in-c#comment24974672_17236475) | | |  |  | | --- | --- | |  |  | | @JonSkeet I have a rough idea of what it means to implement an interface explicitly (I [asked](http://programmers.stackexchange.com/q/136319/1130) on Programmers.SE about that a while back), but the .this[string propertyName] was the main part I didn't quite understand.– [Rachel](http://stackoverflow.com/users/302677/rachel)[Jun 21 '13 at 13:32](http://stackoverflow.com/questions/17236475/how-does-idataerrorinfo-thisstring-propertyname-work-in-c#comment24974775_17236475) | | add comment |

**Answers**

|  |
| --- |
| this[key] is in fact an indexer, and is somewhat of a cross between a property and a method. It acts like a property since you can bind to it, but as opposed to regular properties, it receives a parameter.  Behind the scenes it's implemented as a method - get\_Item(key), and if you'd want to access it via reflection you'd need to use Item for a name. For example:  typeof(MyClass).GetProperty("Item");  This is also important to know when implementing INotifyPropertyChanged, in which case, "Item[]" or Binding.IndexerName should be used as a property name in order to update the UI. |

**NB: Explicit Interface Implementation**

If a [class](http://msdn.microsoft.com/en-us/library/vstudio/0b0thckt.aspx) implements two interfaces that contain a member with the same signature, then implementing that member on the class will cause both interfaces to use that member as their implementation. In the following example, all the calls to Paint invoke the same method.

class Test

{

static void Main()

{

SampleClass sc = new SampleClass();

IControl ctrl = (IControl)sc;

ISurface srfc = (ISurface)sc;

// The following lines all call the same method.

sc.Paint();

ctrl.Paint();

srfc.Paint();

}

}

interface IControl

{

void Paint();

}

interface ISurface

{

void Paint();

}

class SampleClass : IControl, ISurface

{

// Both ISurface.Paint and IControl.Paint call this method.

public void Paint()

{

Console.WriteLine("Paint method in SampleClass");

}

}

// Output:

// Paint method in SampleClass

// Paint method in SampleClass

// Paint method in SampleClass

If the two [interface](http://msdn.microsoft.com/en-us/library/vstudio/87d83y5b.aspx) members do not perform the same function, however, this can lead to an incorrect implementation of one or both of the interfaces. It is possible to implement an interface member explicitly—creating a class member that is only called through the interface, and is specific to that interface. This is accomplished by naming the class member with the name of the interface and a period. For example:

public class SampleClass : IControl, ISurface

{

void IControl.Paint()

{

System.Console.WriteLine("IControl.Paint");

}

void ISurface.Paint()

{

System.Console.WriteLine("ISurface.Paint");

}

}

The class member IControl.Paint is only available through the IControl interface, and ISurface.Paint is only available through ISurface. Both method implementations are separate, and neither is available directly on the class. For example:

// Call the Paint methods from Main.

SampleClass obj = new SampleClass();

//obj.Paint(); // Compiler error.

IControl c = (IControl)obj;

c.Paint(); // Calls IControl.Paint on SampleClass.

ISurface s = (ISurface)obj;

s.Paint(); // Calls ISurface.Paint on SampleClass.

// Output:

// IControl.Paint

// ISurface.Paint