The type must be a reference type in order to use it as parameter 'T' in the generic type or method

195

16

I'm getting deeper into generics and now have a situation I need help with. I get a compile error on the 'Derived' class below as shown in the subject title. I see many other posts similar to this one but I'm not seeing the relationship. Can someone tell me how to resolve this?

```
using System;
using System.Collections.Generic;


namespace Example
{
    public class ViewContext
    {
        ViewContext() { }
    }

    public interface IModel
    {
    }

    public interface IView<T> where T : IModel
    {
        ViewContext ViewContext { get; set; }
    }

    public class SomeModel : IModel
    {
        public SomeModel() { }
        public int ID { get; set; }
    }

    public class Base<T> where T : IModel
    {

        public Base(IView<T> view)
        {
```

```
        public Derived(IView<SomeModel> view)
            : base(view)
        {
            SomeModel m = (SomeModel)Activator.CreateInstance(typeof(SomeModel));
            Service<SomeModel> s = new Service<SomeModel>();
            s.Work(m);
        }
    }

    public class Service<SomeModel> where SomeModel : IModel
    {
        public Service()
        {
        }

        public void Work(SomeModel m)
        {

        }
    }
}
```

c# generics

edited Nov 8 '12 at 19:57
Servy
181k 18 253 362

asked Jun 23 '11 at 8:13
ChrisS
1,170 2 10 19

I don't get any compile errors – Vince Panuccio Jun 23 '11 at 8:23

This code doesn't show that error. Compiles cleanly. – Marc Gravell ♦ Jun 23 '11 at 8:25

3 Answers

```
public class Derived<SomeModel> : Base<SomeModel> where SomeModel : class, IModel
                                                                     ^^^^^
                                                              see this bit
```

answered Jun 23 '11 at 8:27

Marc Gravell ♦
812k 205 2209 2602

11 Thank you, yes that's it. Once I added the class constraint the compile error went away. The following seems to satisfy the need for refernce type. – ChrisS Jun 23 '11 at 8:33

here's what works. public class Base<T> where T : class, IModel { public Base(IView<T> view) { } } public class Derived<SomeModel> : Base<SomeModel> where SomeModel : class, IModel { public Derived(IView<SomeModel> view) : base(view) { SomeModel m = (SomeModel)Activator.CreateInstance(typeof(SomeModel)); Service<SomeModel> s = new Service<SomeModel>(); s.Work(m); } } – ChrisS Jun 23 '11 at 8:34

Helped as well:) Thanks :) As a side note, I think we shouldn't copy the same constrait again and again if it's already applied in interface, IMO. – Celdor Nov 23 '14 at 20:09

42

You get this error if you have constrained `T` to being a `class`

answered Jun 23 '11 at 8:25

thekip
2,917 1 16 39

8

If you put constrains on a generic class or method, every other generic class or method that is using it need to have "at least" those constrains.

answered Mar 7 '13 at 14:48

Guish