

Re: Question about Iteration and forEach

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-06/msg01928.html>

- *From:* "Jeremy Shovan" <jeremy.shovan@xxxxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 12 Jun 2007 01:48:01 -0600
-

Never say you can't. We are programmers we can do anything!!!
Here is a sample.. It is probably not as simple as you would like. but it does what you want.
I just defined a couple of interfaces; IPeakableEnumerator<T> and IPeakEnumerable<T>. I then created a collection that derives from List<T> that implements the interfaces.
And viola. You now have a Peak() method on an enumerator that will get the next element for you without moving to the next element.

```
public class Test
{
    public static void DoTest()
    {
        MyList<string> a = new MyList<string>();
        a.Add("a");
        a.Add("b");
        a.Add("c");
        a.Add("d");
        a.Add("e");

        IPeakableEnumerator<string> en = a.GetPeakableEnumerator();
        while (en.MoveNext())
        {
            Console.WriteLine(en.Current);
            Console.WriteLine(en.Peak());
        }
    }
}
```

```
public interface IPeakableEnumerator<T> : IEnumerator<T>
{
    T Peak();
}
public interface IPeakEnumerable<T>
{
    IPeakableEnumerator<T> GetPeakableEnumerator();
}
```

Re: Question about Iteration and forEach

```
public class MyList<T> : List<T>, IPeakEnumerable<T>
{
public IPeakableEnumerator<T> GetPeakableEnumerator()
{
return new MyListEnumerator(this);
}

[Serializable, StructLayout(LayoutKind.Sequential)]
public class MyListEnumerator : IPeakableEnumerator<T>, IEnumerator<T>, IDisposable, IEnumerator
{
private MyList<T> list;
private int index;
private T current;
internal MyListEnumerator(MyList<T> list)
{
this.list = list;
this.index = 0;
this.current = default(T);
}

public void Dispose()
{
}

public bool MoveNext()
{
if (this.index < this.list.Count)
{
this.current = this.list[this.index];
this.index++;
return true;
}
this.index = this.list.Count + 1;
this.current = default(T);
return false;
}

public T Current
{
get
{
return this.current;
}
}
object IEnumerator.Current
{
get
{
if ((this.index == 0) || (this.index == (this.list.Count + 1)))
```

Re: Question about Iteration and forEach

```
{
throw new InvalidOperationException("Enumeration failed");
}
return this.Current;
}
}
void IEnumerator.Reset()
{
this.index = 0;
}

public T Peak()
{
if (this.index >= this.list.Count)
{
return default(T);
}
return this.list[index];
}
}
}
```

Jeremy Shovan
<http://www.jeremyshovan.com>

"Peter Duniho" <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx> wrote in message
<news:op.ttsigpm68jd0ej@xxxxxxxxxxxxxxxxxxxxxxxxxxxx>

On Mon, 11 Jun 2007 19:50:40 -0700, news.microsoft.com <billgower@xxxxxxxxxxxx>
wrote:

I am looping through an iteration and I would like to test the next item but
if its not the one that I want how do I put it back so that when my foreach
continues it is in the next iteration?

You can't. The "foreach" statement strictly enumerates one by one through the collection.

If you want the ability to adjust your position within the enumeration, you can usually just use
a normal "for" loop with an index to access individual items within the collection. Then you
can adjust the index as needed.

I will note that if you are truly enumerating a list, having a need to look at a specific item and
then as a result of that inspection reverse course and go back to the previous item is generally
a bad sign. It either means that you're not really enumerating the items in the list, and so
enumeration semantics aren't appropriate, or you are enumerating the list in a manner
indicative of poor design.

Re: Question about Iteration and forEach

Hopefully you're in the former case, and you really don't want a true enumeration, but you should at least consider the possibility of the latter case. :) For more specific advice, you'd have to post the code for the actual loop you're talking about.

Pete