.NET Framework Class Library

**IEnumerable Interface**

Exposes the enumerator, which supports a simple iteration over a non-generic collection.

**Namespace:**  [System.Collections](http://msdn.microsoft.com/en-us/library/system.collections.aspx)  
**Assembly:**  mscorlib (in mscorlib.dll)

http://i.msdn.microsoft.com/Global/Images/clear.gif Syntax

Visual Basic

<ComVisibleAttribute(True)> \_

<GuidAttribute("496B0ABE-CDEE-11d3-88E8-00902754C43A")> \_

Public Interface IEnumerable

C#

[ComVisibleAttribute(true)]

[GuidAttribute("496B0ABE-CDEE-11d3-88E8-00902754C43A")]

public interface IEnumerable

Visual C++

[ComVisibleAttribute(true)]

[GuidAttribute(L"496B0ABE-CDEE-11d3-88E8-00902754C43A")]

public interface class IEnumerable

F#

[<ComVisibleAttribute(true)>]

[<GuidAttribute("496B0ABE-CDEE-11d3-88E8-00902754C43A")>]

type IEnumerable = interface end

The **IEnumerable** type exposes the following members.

http://i.msdn.microsoft.com/Global/Images/clear.gif Methods

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public methodSupported by the XNA FrameworkSupported by Portable Class Library | [GetEnumerator](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.getenumerator.aspx) | Returns an enumerator that iterates through a collection. |

[Top](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx#mainBody)

http://i.msdn.microsoft.com/Global/Images/clear.gif Extension Methods

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public Extension Method | [AsParallel](http://msdn.microsoft.com/en-us/library/dd413237.aspx) | Enables parallelization of a query. (Defined by [ParallelEnumerable](http://msdn.microsoft.com/en-us/library/system.linq.parallelenumerable.aspx).) |
| Public Extension MethodSupported by Portable Class Library | [AsQueryable](http://msdn.microsoft.com/en-us/library/bb353734.aspx) | Converts an **IEnumerable** to an [IQueryable](http://msdn.microsoft.com/en-us/library/system.linq.iqueryable.aspx). (Defined by [Queryable](http://msdn.microsoft.com/en-us/library/system.linq.queryable.aspx).) |
| Public Extension MethodSupported by the XNA FrameworkSupported by Portable Class Library | [Cast<(Of <(TResult>)>)](http://msdn.microsoft.com/en-us/library/bb341406.aspx) | Converts the elements of an **IEnumerable** to the specified type. (Defined by [Enumerable](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.aspx).) |
| Public Extension MethodSupported by the XNA FrameworkSupported by Portable Class Library | [OfType<(Of <(TResult>)>)](http://msdn.microsoft.com/en-us/library/bb360913.aspx) | Filters the elements of an **IEnumerable** based on a specified type. (Defined by [Enumerable](http://msdn.microsoft.com/en-us/library/system.linq.enumerable.aspx).) |

[Top](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx#mainBody)

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

For the generic version of this interface see [System.Collections.Generic..::.IEnumerable<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx).

**Notes to Implementers**

**IEnumerable** must be implemented to support the **foreach** semantics of Microsoft Visual Basic. COM classes that allow enumerators also implement this interface.

http://i.msdn.microsoft.com/Global/Images/clear.gif Examples

The following code example demonstrates the implementation of the **IEnumerable** and [IEnumerator](http://msdn.microsoft.com/en-us/library/system.collections.ienumerator.aspx) interfaces for a custom collection. In this example, members of these interfaces are not explicitly called, but they are implemented to support the use of **foreach** (**For Each** in Visual Basic) to iterate through the collection.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl60_ctl00_ctl01_code');" \o "Copy Code)

Imports System

Imports System.Collections

Public Class Person

Public Sub New(ByVal fName As String, ByVal lName As String)

Me.firstName = fName

Me.lastName = lName

End Sub

Public firstName As String

Public lastName As String

End Class

Public Class People

Implements IEnumerable

Private \_people() As Person

Public Sub New(ByVal pArray() As Person)

\_people = New Person(pArray.Length - 1) {}

Dim i As Integer

For i = 0 To pArray.Length - 1

\_people(i) = pArray(i)

Next i

End Sub

Public Function GetEnumerator() As IEnumerator \_

Implements IEnumerable.GetEnumerator

Return New PeopleEnum(\_people)

End Function

End Class

Public Class PeopleEnum

Implements IEnumerator

Public \_people() As Person

' Enumerators are positioned before the first element

' until the first MoveNext() call.

Dim position As Integer = -1

Public Sub New(ByVal list() As Person)

\_people = list

End Sub

Public Function MoveNext() As Boolean Implements IEnumerator.MoveNext

position = position + 1

Return (position < \_people.Length)

End Function

Public Sub Reset() Implements IEnumerator.Reset

position = -1

End Sub

Public ReadOnly Property Current() As Object Implements IEnumerator.Current

Get

Try

Return \_people(position)

Catch ex As IndexOutOfRangeException

Throw New InvalidOperationException()

End Try

End Get

End Property

End Class

Class App

Shared Sub Main()

Dim peopleArray() As Person = { \_

New Person("John", "Smith"), \_

New Person("Jim", "Johnson"), \_

New Person("Sue", "Rabon")}

Dim peopleList As New People(peopleArray)

Dim p As Person

For Each p In peopleList

Console.WriteLine(p.firstName + " " + p.lastName)

Next

End Sub

End Class

' This code produces output similar to the following:

'

' John Smith

' Jim Johnson

' Sue Rabon

C#

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl60_ctl00_ctl02_code');" \o "Copy Code)

using System;

using System.Collections;

public class Person

{

public Person(string fName, string lName)

{

this.firstName = fName;

this.lastName = lName;

}

public string firstName;

public string lastName;

}

public class People : IEnumerable

{

private Person[] \_people;

public People(Person[] pArray)

{

\_people = new Person[pArray.Length];

for (int i = 0; i < pArray.Length; i++)

{

\_people[i] = pArray[i];

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return (IEnumerator) GetEnumerator();

}

public PeopleEnum GetEnumerator()

{

return new PeopleEnum(\_people);

}

}

public class PeopleEnum : IEnumerator

{

public Person[] \_people;

// Enumerators are positioned before the first element

// until the first MoveNext() call.

int position = -1;

public PeopleEnum(Person[] list)

{

\_people = list;

}

public bool MoveNext()

{

position++;

return (position < \_people.Length);

}

public void Reset()

{

position = -1;

}

object IEnumerator.Current

{

get

{

return Current;

}

}

public Person Current

{

get

{

try

{

return \_people[position];

}

catch (IndexOutOfRangeException)

{

throw new InvalidOperationException();

}

}

}

}

class App

{

static void Main()

{

Person[] peopleArray = new Person[3]

{

new Person("John", "Smith"),

new Person("Jim", "Johnson"),

new Person("Sue", "Rabon"),

};

People peopleList = new People(peopleArray);

foreach (Person p in peopleList)

Console.WriteLine(p.firstName + " " + p.lastName);

}

}

/\* This code produces output similar to the following:

\*

\* John Smith

\* Jim Johnson

\* Sue Rabon

\*

\*/