

Last week I started the [first in a series of blog posts](#) I'll be making that cover some of the new VB and C# language features that are coming as part of the Visual Studio and .NET Framework "Orcas" release later this year.

My last blog post covered the new [Automatic Properties, Object Initializer and Collection Initializer features](#). If you haven't read my previous post yet, please read it [here](#). Today's blog post covers a much more significant new feature that is available with both VB and C#: *Extension Methods*.

## What are Extension Methods?

Extension methods allow developers to add new methods to the public contract of an existing CLR type, without having to sub-class it or recompile the original type. Extension Methods help blend the flexibility of "duck typing" support popular within dynamic languages today with the performance and compile-time validation of strongly-typed languages.

Extension Methods enable a variety of useful scenarios, and help make possible the really powerful LINQ query framework that is being introduced with .NET as part of the "Orcas" release.

## Simple Extension Method Example:

Ever wanted to check to see whether a string variable is a valid email address? Today you'd probably implement this by calling a separate class (probably with a static method) to check to see whether the string is valid. For example, something like:

```
string email = Request.QueryString["email"];

if ( EmailValidator.IsValid(email) ) {

}
```

Using the new "extension method" language feature in C# and VB, I can instead add a useful "IsValidEmailAddress()" method onto the string class itself, which returns whether the string instance is a valid string or not. I can then re-write my code to be cleaner and more descriptive like so:

```
string email = Request.QueryString["email"];

if ( email.IsValidEmailAddress() ) {

}
```

How did we add this new IsValidEmailAddress() method to the existing string type? We did it by defining a static class with a static method containing our "IsValidEmailAddress" extension method like below:

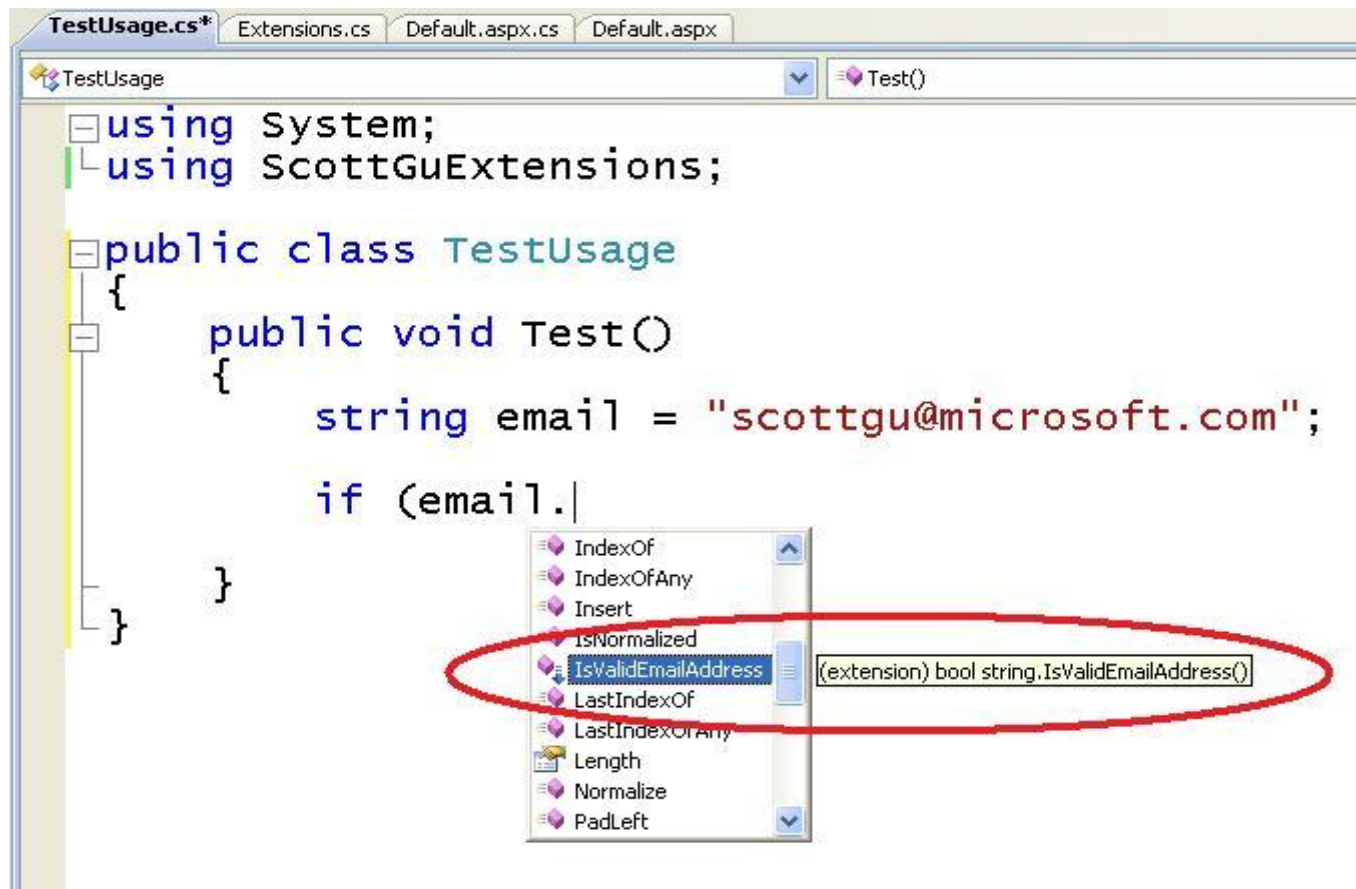
```
public static class ScottGuExtensions
{
    public static bool IsValidEmailAddress(this string s)
    {
        Regex regex = new Regex(@"^[w-\.\.]+@([\w-]+\.)+[\w-]{2,4}$");
        return regex.IsMatch(s);
    }
}
```

Note how the static method above has a "this" keyword before the first parameter argument of type string. This tells the compiler that this particular Extension Method should be added to objects of type "string". Within the IsValidEmailAddress() method implementation I can then access all of the public properties/methods/events of the actual string instance that the method is being called on, and return true/false depending on whether it is a valid email or not.

To add this specific Extension Method implementation to string instances within my code, I simply use a standard "using" statement to import the namespace containing the extension method implementation:

using ScottGuExtensions;

The compiler will then correctly resolve the IsValidEmailAddress() method on any string. C# and VB in the [public "Orcas" March CTP](#) now provide full intellisense support for extension methods within the Visual Studio code-editor. So when I hit the "." keyword on a string variable, my extension methods will now show up in the intellisense drop-downlist:



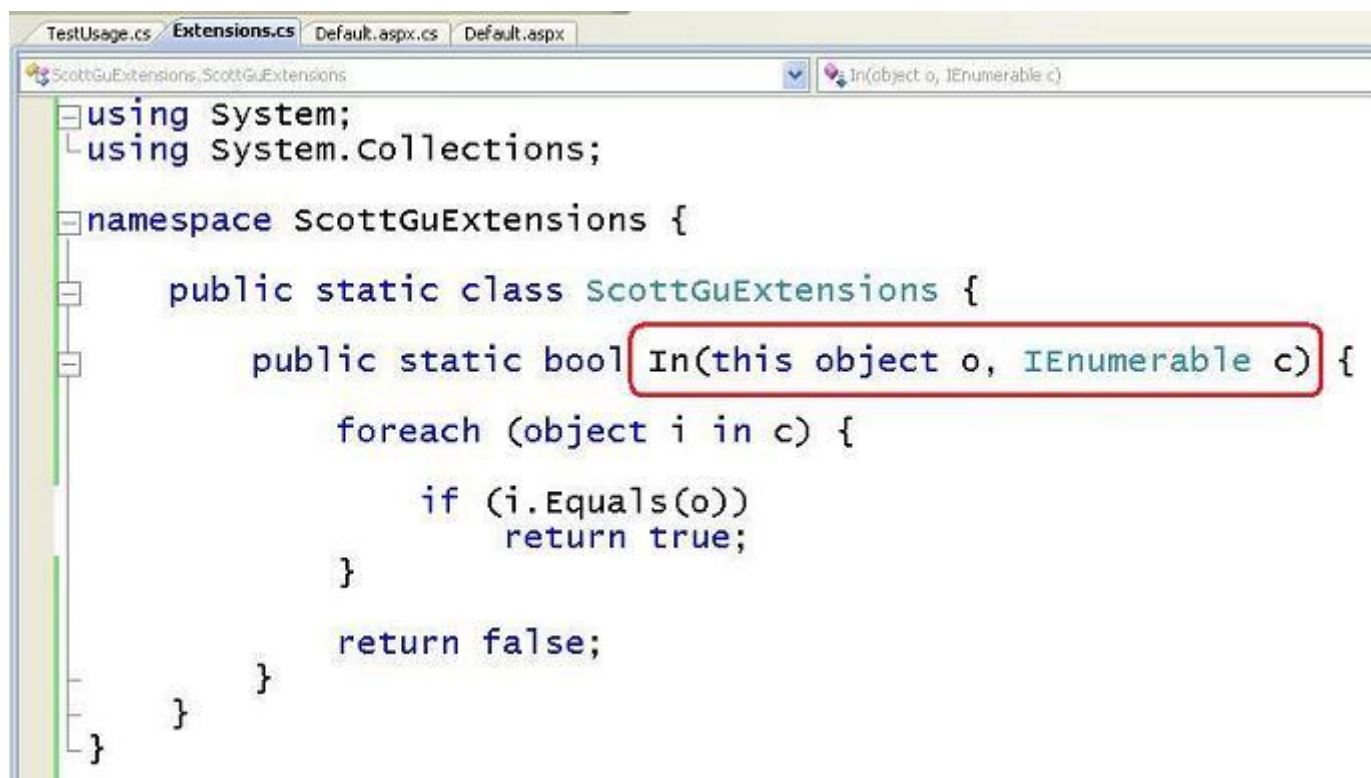
The VB and C# compilers also naturally give you compile-time checking of all Extension Method usage - meaning you'll get a compile-time error if you mis-type or mis-use one.

[Credit: Thanks to David Hayden for first coming up with the IsValidEmailAddress scenario I used above in a prior blog post of his from last year.]

## Extension Methods Scenarios Continued...

Leveraging the new extension method feature to add methods to individual types opens up a number of useful extensibility scenarios for developers. What makes Extension Methods really powerful, though, is their ability to be applied not just to individual types - but also to any parent base class or interface within the .NET Framework. This enables developers to build a variety of rich, composable, framework extensions that can be used across the .NET Framework.

For example, consider a scenario where I want an easy, descriptive, way to check whether an object is already included within a collection or array of objects. I could define a simple .In(collection) extension method that I want to add to all objects within .NET to enable this. I could implement this "In()" extension method within C# like so:



```
TestUsage.cs Extensions.cs Default.aspx.cs Default.aspx
ScottGuExtensions, ScottGuExtensions
In(object o, IEnumerable c)

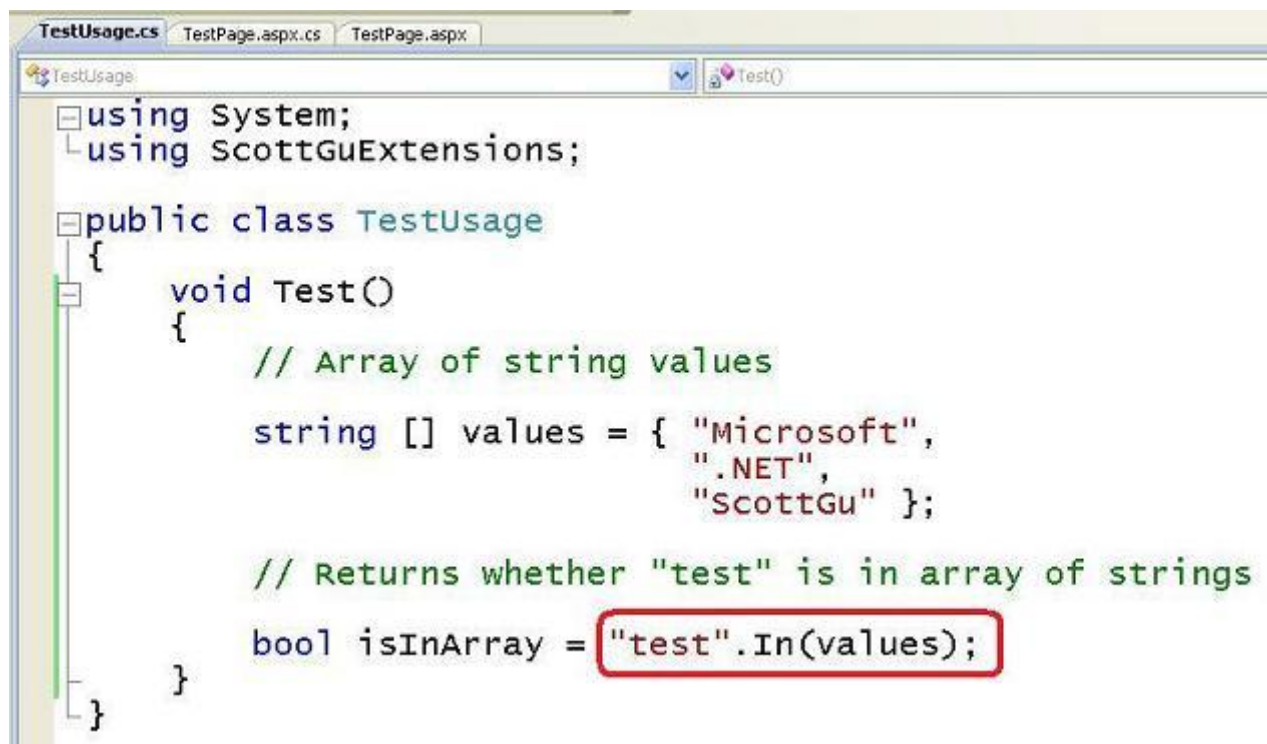
using System;
using System.Collections;

namespace ScottGuExtensions {
    public static class ScottGuExtensions {
        public static bool In(this object o, IEnumerable c) {
            foreach (object i in c) {
                if (i.Equals(o))
                    return true;
            }
            return false;
        }
    }
}
```

Note above how I've declared the first parameter to the extension method to be "this object o". This indicates that this extension method should be applied to all types that derive from the base System.Object base type - which means I can now use it on every object in .NET.

The "In" method implementation above allows me to check to see whether a specific object is included within an IEnumerable sequence passed as an argument to the method. Because all .NET collections and arrays implement the IEnumerable interface, I now have a useful and descriptive method for checking whether *any* .NET object belongs to *any* .NET collection or array.

I could then use this "In()" extension method to see whether a particular string is within an array of strings:



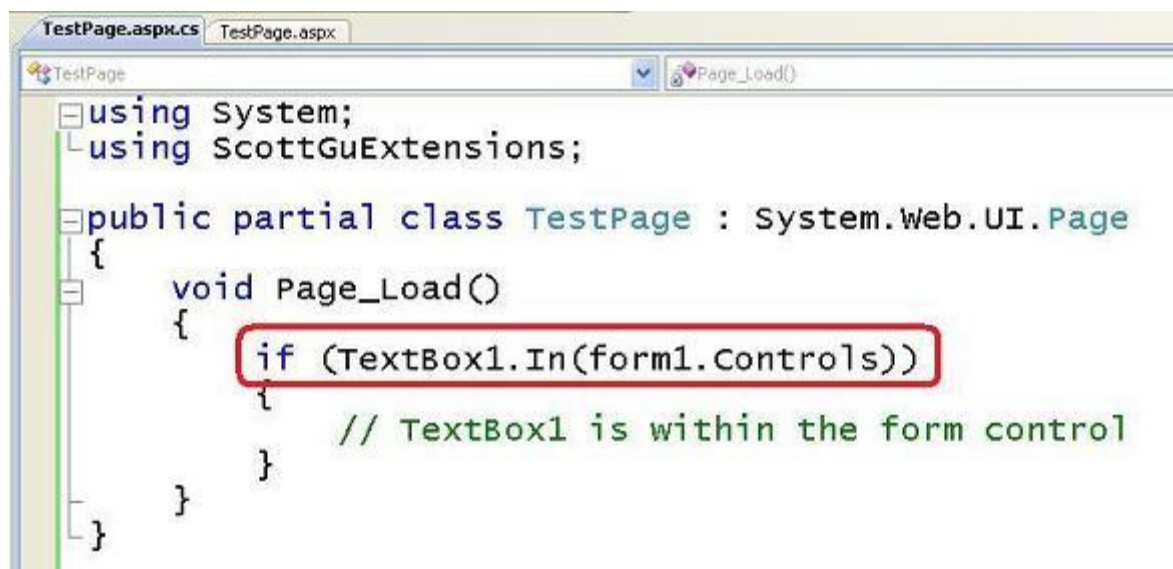
```
TestUsage.cs TestPage.aspx.cs TestPage.aspx
TestUsage
Test()

using System;
using ScottGuExtensions;

public class TestUsage
{
    void Test()
    {
        // Array of string values
        string [] values = { "Microsoft",
                            ".NET",
                            "ScottGu" };

        // Returns whether "test" is in array of strings
        bool isInArray = "test".In(values);
    }
}
```

I could use it to check to see whether a particular ASP.NET control is within a container control collection:



```
using System;
using ScottGuExtensions;

public partial class TestPage : System.Web.UI.Page
{
    void Page_Load()
    {
        if (TextBox1.In(form1.Controls))
        {
            // TextBox1 is within the form control
        }
    }
}
```

I could even use it with scalar datatypes like integers:

```
//
// Array of integer values
int [] values = { 0, 4, 5, 9, 42 };

//
// sample int variable
int testValue = 45;

//
// Check whether int variable in array
bool isInArray = testValue.In(values);

//
// Check whether scalar value in array
bool isInArray2 = 42.In(values);
```

Note above how you can even use extension methods on base datatype values (like the integer value 42). Because the CLR supports automatic boxing/unboxing of value-classes, extensions methods can be applied on numeric and other scalar datatypes directly.

As you can probably begin to see from the samples above, extension methods enable some really rich and descriptive extensibility scenarios. When applied against common base classes and interfaces across .NET, they enable some really nice domain specific framework and composition scenarios.

## **Built-in System.Linq Extension Methods**

One of the built-in extension method libraries that we are shipping within .NET in the "Orcas" timeframe are a set of very powerful query extension method implementations that enable developers to easily query data. These extension method implementations live under the new "System.Linq" namespace, and define standard query operator extension methods that can be used by any .NET developer to easily query XML, Relational Databases, .NET objects that implement IEnumerable, and/or any other type of data structure.

A few of the advantages of using the extension method extensibility model for this query support include:

- 1) It enables a common query programming model and syntax that can be used across all types of data (databases, XML files, in-memory objects, web-services, etc).



2) It is composable and allows developers to easily add new methods/operators into the query syntax. For example: we could use our custom "In()" method together with the standard "Where()" method defined by LINQ as part of a single query. Our custom In() method will look just as natural as the "standard" methods supplied under the System.Linq namespace.

3) It is extensible and allows any type of data provider to be used with it. For example: an existing ORM engine like NHibernate or LLBLGen could implement the LINQ standard query operators to enable LINQ queries against their existing ORM implementation and mapping engines. This will enable developers to learn a common way to query data, and then apply the same skills against a wide variety of rich data store implementations.

I'll be walking through LINQ much more over the next few weeks, but wanted to leave you with a few samples that show how to use a few of the built-in LINQ query extension methods with different types of data:

## **Scenario 1: Using LINQ Extension Methods Against In-Memory .NET Objects**

Assume we have defined a class to represent a "Person" like so:

```
public class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public int Age { get; set; }
}
```

I could then use the [new object initializer and collection initializer features](#) to create and populate a collection of "people" like so:

```
// Create a list of people

List<Person> people = new List<Person> {
    new Person { FirstName="Scott", LastName="Guthrie", Age=32 },
    new Person { FirstName="Bill", LastName="Gates", Age=50 },
    new Person { FirstName="Susanne", LastName="Guthrie", Age=32 }
};
```

I could then use the standard "Where()" extension method provided by System.Linq to retrieve a sequence of those "Person" objects within this collection whose FirstName starts with the letter "S" like so:

```
// Retrieve people whose first name starts with "S"

IEnumerable<Person> queryResult;

queryResult = people.Where(p => p.FirstName.StartsWith("S"));
```

The new p => syntax above is an example of a "Lambda expression", which is a more concise evolution of C# 2.0's anonymous method support, and enables us to easily express a query filter with an argument (in this case we are indicating that we only want to return a sequence of those Person objects where the firstname property starts with the letter "S"). The above query will then return 2 objects as part of the sequence (for Scott and Susanne).

I could also write code that takes advantage of the new "Average" and "Max" extension methods provided by System.Linq to determine the average age of the people in my collection, as well as the age of the oldest person like so:

```
// Retrieve average age of people in List
double averageAge = people.Average(p => p.Age);

// Retrieve max age within the list
int maxAge = people.Max(p => p.Age);
```

## Scenario 2: Using LINQ Extension Methods Against an XML File

It is probably rare that you manually create a collection of hard-coded data in-memory. More likely you'll retrieve the data either from an XML file, a database, or a web-service.

Let's assume we have an XML file on disk that contains the data below:



I could obviously use the existing System.Xml APIs today to either load this XML file into a DOM and access it, or use a low-level XmlReader API to manually parse it myself. Alternatively, with "Orcas" I can now use the System.Xml.Linq implementation that supports the standard LINQ extension methods (aka "XLINQ") to more elegantly parse and process the XML.

The below code-sample shows how to use LINQ to retrieve all of the <person> XML Elements that have a <person> sub-node whose inner value starts with the letter "S":

```
XDocument people = XDocument.Load("test.xml");

IEnumerable<XElement> queryResult;

queryResult = people.Descendants("person")
    .where(p=>p.Element("firstname").Value.StartsWith("S"));
```

Note that it uses the exact same Where() extension method as with the in-memory object sample. Right now it is returning a sequence of "XElement" elements, which is an un-typed XML node element. I could alternatively re-write the query to "shape" the data that is returned instead by using LINQ's Select() extension method and provide a Lambda expression that uses the new object initializer syntax to populate the same "Person" class that we used with our first in-memory collection example:

```

XDocument people = XDocument.Load("test.xml");

IEnumerable<Person> queryResult;

queryResult = people.Descendants("person")
    .Where(p => p.Element("firstname").Value.StartsWith("S"))
    .Select( p=>new Person {
        FirstName = p.Element("firstname").Value,
        LastName = p.Element("lastname").Value,
        Age = Convert.ToInt32(p.Attribute("age").Value)
    }
);

```

The above code does all the work necessary to open, parse and filter the XML in the "test.xml" file, and return back a strongly-typed sequence of Person objects. No mapping or persistence file is necessary to map the values - instead I am expressing the shaping from XML->objects directly within the LINQ query above.

I could also use the same Average() and Max() LINQ extension methods as before to calculate the average age of <person> elements within the XML file, as well as the maximum age like so:

```

// Retrieve average age of people in XML file

double averageAge = people.Descendants("person")
    .Average(p=>Convert.ToInt32(p.Attribute("age").Value));

// Retrieve max age within the list

int maxAge = people.Descendants("person")
    .Max(p=>Convert.ToInt32(p.Attribute("age").Value));

```

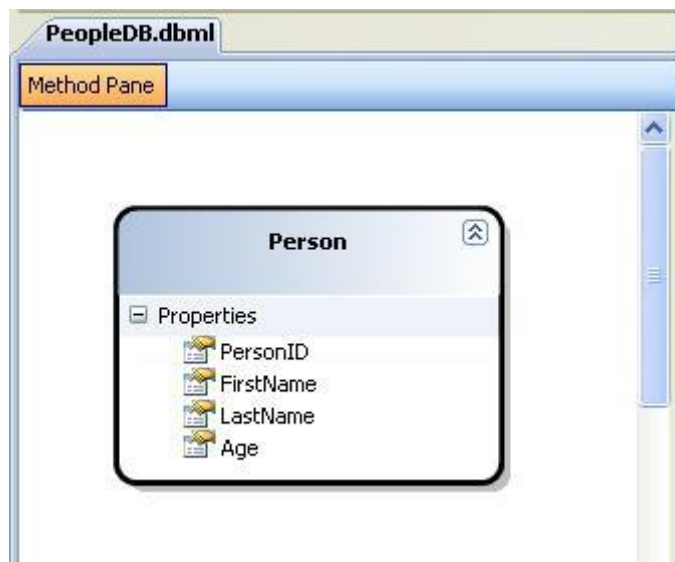
I do not have to manually parse the XML file. Not only will XLINQ handle that for me, but it will parse the file using a low-level XMLReader and not have to create a DOM in order to evaluate the LINQ expression. This means that it is lightening fast and doesn't allocate much memory.

### Scenario 3: Using LINQ Extension Methods Against a Database

Let's assume we have a SQL database that contains a table called "People" that has the following database schema:

dbo.People: Ta...ost.Northwind)		
Column Name	Data Type	Allow Nulls
PersonID	int	<input type="checkbox"/>
▶ FirstName	nvarchar(50)	<input type="checkbox"/>
LastName	nvarchar(50)	<input type="checkbox"/>
Age	int	<input type="checkbox"/>
		<input type="checkbox"/>

I could use the new LINQ to SQL WYSIWYG ORM designer within Visual Studio to quickly create a "Person" class that maps to the database:



I can then use the same LINQ Where() extension method I used previously with objects and XML to retrieve a sequence of strongly-typed "Person" objects from the database whose first name starts with the letter "S":

```
// DataContext for Database
PeopleDBDataContext peopleDb = new PeopleDBDataContext();

// Retrieve all people whose first name starts with "s"
IEnumerable<Person> queryResults;

queryResults = peopleDb.Persons.Where(p=>p.FirstName.StartsWith("S"));
```

Note how the query syntax is the same as with objects and XML.

I could then use the same LINQ Average() and Max() extension methods as before to retrieve the average and maximum age values from the database like so:

```
// Retrieve average age of people in DB
int averageAge = peopleDb.Persons.Average(p=>p.Age);

// Retrieve maximum age
int maxAge = peopleDb.Persons.Max(p=>p.Age);
```

You don't need to write any SQL code yourself to have the above code snippets work. The LINQ to SQL object relational mapper provided with "Orcas" will handle retrieving, tracking and updating objects that map to your database schema and/or SPROC's. You can simply use any LINQ extension method to filter and shape the results, and LINQ to SQL will execute the SQL code necessary to retrieve the data (note: the Average and Max extension methods above obviously don't return all the rows from the table - they instead use TSQL aggregate functions to compute the values in the database and just return a scalar result).

[Please watch this video](#) I did in January to see how LINQ to SQL dramatically improves data productivity in "Orcas". In the video you can also see the new LINQ to SQL WYSIWYG ORM designer in action, as well as see full intellisense provided in the code-editor when writing LINQ code against the data model.

## Summary



Hopefully the above post gives you a basic understanding of how extension methods work, and some of the cool extensibility approaches you will be able to take with them. As with any extensibility mechanism, I'd really caution about not going overboard creating new extension methods to begin with. Just because you have a shiny new hammer doesn't mean that everything in the world has suddenly become a nail!

To get started trying out extension methods, I'd recommend first exploring the standard query operators provided within the System.Linq namespace in "Orcas". These enable rich query support against any array, collection, XML stream, or relational database, and can dramatically improve your productivity when working with data. I think you'll find they'll significantly reduce the amount of code you write within your applications, and allow you to write really clean and descriptive syntax. They'll also enable you to get automatic intellisense and compile-time checking of query logic within your code.

In the next few weeks I'll continue this series on new language features in "Orcas" and explore Anonymous Types and Type Inference, as well as talk more about Lambdas and other cool features. I'll also obviously be talking a lot more about LINQ.

**[April 21st Update:** I recently posted the next topic in this Orcas language series - covering Query Syntax -- [here](#).]

Hope this helps,

Scott

## 63 Comments

- I think Extension Methods in Orcas are great. I've played a bit the new features and I'm very happy with it. There are so much possibilities now.

I'm abled to fill the gap of the missing:

string.TrimOrNullEmpty - Method

I've never understood why there was no overload for string.IsNullOrEmpty to also do a trim().

Thanks!

[Jens Hofmann](#) - [Tuesday, March 13, 2007 10:02:43 AM](#)

- The "In" is definitely very useful...I've been looking for something like this a long time back...cool!

[Rachit](#) - [Tuesday, March 13, 2007 11:12:58 AM](#)

- Hi Scott,

well - this is cool - somehow...

Extension methods may be funky and geeky (and even very useful) - but they will make understanding code (let alone code reviews) \*much\* harder in the future...

What is your opinion on that? How do you handle this fact (especially the code review part) internally...?

thanks  
dominick

[dominick](#) - [Tuesday, March 13, 2007 11:16:47 AM](#)

- Hi Scott,

How does the performance of these in-built linq tools measure up? e.g how does loading and parsing an xml file using XLinQ compare with using System.Xml?

Extension Methods look like a nice idea but I can see them being a bit confusing to new programmers.

**Chris Moseley** - [Tuesday, March 13, 2007 12:23:26 PM](#)

- If anybody is interested in the VB syntax, feel free to visit my blog :-)

**Daniel Moth** - [Tuesday, March 13, 2007 12:25:30 PM](#)

- Scott,

Are these Orcas features, or new features in C# 3.0? This is causing some confusion IMO.

For instance, will I be able to use extension methods in C# 2.0 using Orcas? I'm guessing the answer is "no". But will I be able to target .NET 2.0 using Orcas? The last I heard, the answer was "yes".

So, if I cannot use extension methods in C# 2.0 using Orcas, then it isn't really an Orcas feature is it?

SM

**Sahil Malik** - [Tuesday, March 13, 2007 12:52:12 PM](#)

- You write "I simply use a standard 'using' statement to import the namespace containing the extension method implementation", but it seems as if you're importing the class instead of the namespace... Typo or coincidence?

**nstlgc** - [Tuesday, March 13, 2007 2:16:52 PM](#)

- Why you killing the language?

**Angry c# guy** - [Tuesday, March 13, 2007 2:19:34 PM](#)

- Hey Scott!

I learned a couple of weeks ago that you changed your position within Microsoft and have now "inherited" more responsibilities.

Congratulations!

My comment is just to say 'Thanks!' for taking time in writing about new features coming up in the .NET framework and for being very pragmatic...even past midnight! (Did I read correctly? You wrote this post at 2:27 AM?)

Take care! ;)

**Brian Di Croce** - [Tuesday, March 13, 2007 2:38:17 PM](#)

- excellent post scott! when do you think you all will release a beta 1 of orcas - soon???

**cgm** - [Tuesday, March 13, 2007 2:55:33 PM](#)

- The new extension methods sound like they could be quite useful in a number of situations.

Can an extension method be applied to a sealed class?

**Erick** - [Tuesday, March 13, 2007 3:26:41 PM](#)

- Hi Dominick,

When used correctly, I think you'll find that Extension Methods can significantly improve code comprehension within code reviews, and lead to fewer lines of code and fewer bugs.

The reason I think this is because they allow you to more clearly express your intent. For example, instead of a filter like below with imperative code:

```
List originalList = ...;
List results = new List();

foreach(Person p in originalList) {

    if (p.Age > 50) {
        results.Add(p);
    }
}
```

You could instead just write:

```
results = originalList.Where(p=>p.Age > 50).ToList();
```

If your conditional and filtering logic was complicated, I believe you'll find that the implementation using LINQ will be much easier to read and understand the intent. Usually this leads to better code and fewer bugs (whereas large unfactored imperative code blocks are usually rife with bugs).

Where you want to be careful is to make sure you don't use extension methods just for silly pet tricks. That is why I suggested in the article above that you start just by using the built-in LINQ extension methods, and not go overboard creating your own just yet until you really understand them well and how best to use/create new ones.

Hope this helps,

Scott

**ScottGu** - [Tuesday, March 13, 2007 3:45:03 PM](#)

- Personally, I don't feel the need to use extension methods and agree with dominick about code reviews.

LINQ does look powerful, however!

**Jason Kealey** - [Tuesday, March 13, 2007 3:46:05 PM](#)

- Hi Erick,

Yep - Extension Methods can be used on Sealed Classes, which is one place they can be very useful.

Hope this helps,

Scott

**ScottGu** - [Tuesday, March 13, 2007 3:46:10 PM](#)

- Hi Brian,

Thanks. :-)

I've actually had my new role now for about 18 months. It still keeps me busy though. ;-)

And sadly the post was at 2:27am. It took a few hours longer than I was planning to write.

Thanks,

Scott

**ScottGu** - [Tuesday, March 13, 2007 3:47:48 PM](#)

- Hi nstlgc,

Sorry for the confusion - it is the namespace that you want to import with the using statement. In my code example above, both the class and namespace have the same name which was confusing.

Thanks,

Scott

**ScottGu** - [Tuesday, March 13, 2007 3:49:09 PM](#)

- Hi Sahil,

Extension Methods are actually implemented entirely by the compiler - no new IL instructions within the CLR are required to support them.

This means that you can use the VS "Orcas" C# compiler and write code that uses Extension Methods, and use the multi-targetting features to run it on a vanilla .NET 2.0 box.

What is included within the "Orcas" .NET Framework are the LINQ libraries and namespace. If you want to use the LINQ extension methods that are built-in, you'll need to have "Orcas" installed.

Hope this helps,

Scott

**ScottGu** - [Tuesday, March 13, 2007 3:51:53 PM](#)



- Hi Moth,

Nice post on using VB with extension methods: <http://danielmoth.com/Blog/2007/02/extension-methods-c-30-and-vb9.html>

Thanks!

Scott

**ScottGu** - [Tuesday, March 13, 2007 3:52:22 PM](#)

- Hi Chris,

I think you'll find that XLINQ is faster at parsing XML except if you are a real expert at using the low-level XML XMLReader API (which is a fairly low-level API).

If you are getting your XML using the DOM or XQuery today, then XLINQ should be faster (since it is using the more efficient low-level System.XML APIs).

Hope this helps,

Scott

**ScottGu** - [Tuesday, March 13, 2007 3:54:40 PM](#)

- Hi Chris,

Yes - the DLINQ data model classes do support serialization across web-services.

You can also attach a standalone object to the datacontext object - which allows for update scenarios across web-services as well.

Thanks,

Scott

**ScottGu** - [Tuesday, March 13, 2007 3:55:54 PM](#)

- Hi Scott, when using extension methods what kind of access do we have to the base class? Would we have access to protected methods and data as we would if we inherited from the base class or are we restricted to public access only?

**Erick** - [Tuesday, March 13, 2007 4:15:48 PM](#)

- Thanks Scott. Another great article :-)

**West** - [Tuesday, March 13, 2007 4:35:47 PM](#)

- Hi Scott,

OK - well, if people would read your summary basically saying "don't overdo it" - things are fine...and in the context of Linq this is totally powerful...

I just have the fear that people will overdo it..time will tell..

It will be interesting to see what the "framework design guidelines" have to say in 12 months...

cheers  
dominick

**dominick** - [Tuesday, March 13, 2007 4:43:20 PM](#)

- Gosh... I'm really hesitant to form an opinion. I agree that it's very powerful, and very cool, but I too worry about code readability and how it would affect code reviews. I guess I'll have to wait and see.

**Jeff** - [Tuesday, March 13, 2007 4:51:44 PM](#)

- Holy moly, thanks for the clarification Scott. This is HUGE - why don't you point that out more clearly in your blogposts, that these things are not specific to C# 3.0 only.

BTW - another thing I feel you should add is,

DONOT MISUSE Extension methods!!

:)

**Sahil Malik** - [Tuesday, March 13, 2007 5:14:55 PM](#)

- Hey Scott,

Thanks for taking time out and posting things in such great detail.. only question I have is about the "Lambda expression", seems it always is  $P \Rightarrow$  as we are working with the person class in all the queries. Is it possible for it to be anything other than that?? if no then why is it required?

Also I am not sure if its true for others but I am finding it difficult to read the code with Linq extension methods specially, `people.descendents("person").Where(p=>p.FirstName.StartsWith("S")).Select...`

a word by word english translation will help!!

**hp** - [Tuesday, March 13, 2007 7:42:34 PM](#)

- Thanks for the post Scott. I'm a little confused, however, by your comparison of Extension Methods to Duck Typing. While Extension methods are really cool, they seem to me more comparable to a feature like Ruby Mixins rather than Duck Typing.

The "var" keyword in c# 3.0 seems to be a way to achieve Duck Typing with local variables, but I have yet to see a way to achieve it outside of local variable scope. Am I missing something?

**Andy Alm** - [Tuesday, March 13, 2007 8:31:49 PM](#)

- Hi Scott,

Thanks for great article.

Is it possible to add a property to existing class? Or are we only allowed to add methods? For

example, in the Valid Email address example, it would be nice to just write `bool isValid = email.IsValidEmailAddress;`

**Amrinder Sandhu** - [Tuesday, March 13, 2007 9:09:50 PM](#)

- I'm really looking forward to this. The partial classes of .NET 2.0 was a step of the way, but this just takes it though the roof. Excellent.

**Mads Kristensen** - [Tuesday, March 13, 2007 9:14:55 PM](#)

- Scott,  
Cool stuff. But since Microsoft's own guidance is don't extend types that you don't own, perhaps samples extending string and IEnumerable aren't the best ones to put out there. Extension methods have to potential to cause serious maintenance problems if misused.

**kevindente** - [Tuesday, March 13, 2007 9:26:30 PM](#)

- Hi Kevin,

I'm not entirely sure what you mean about not extending types you don't own - since we do this quite a bit today (for example: sub-classing controls, pages, providers, etc).

In general, you do need to be careful/thoughtful about how you extend existing types. With extension methods, you can just use the public type contract for the type/interface you are working against - this should keep you safe going forward from an extensibility perspective.

Note also that extension methods don't change the original object type itself, so other compiled control libraries that work with the same type won't be affected. This also makes it far less brittle than changing the object dynamically at runtime.

Hope this helps,

Scott

**ScottGu** - [Tuesday, March 13, 2007 9:40:56 PM](#)

- Hi Amrinder,

Right now we don't support extension properties. It is something I know the language teams are considering for the future - but for right now you can only add methods.

Hope this helps,

Scott

**ScottGu** - [Tuesday, March 13, 2007 9:41:55 PM](#)

- Hi hp,

I'll do an upcoming post on Lambda expressions that help explain how to use them more.

Thanks,

Scott

**ScottGu** - [Tuesday, March 13, 2007 9:43:04 PM](#)

- Hi JFo,

In the "Orcas" release the only framework extensions that take advantage of extension methods is with LINQ.

I doubt we'd ever add extension methods to existing namespaces. Instead we'd probably add extensions to sub-namespaces (like System.Xml.Linq) so as to keep things clear.

As I mentioned in my blog post above, you want to be careful about where and how exactly you use it. It is a powerful tool, but you don't want to be careful not to use it. Likewise we'll be careful with the .NET Framework about the places we leverage it to make sure it totally makes sense and integrates cleanly.

Thanks,

Scott

**ScottGu** - [Tuesday, March 13, 2007 9:58:33 PM](#)

- Hi Scott,

In .net 2.0, I typically create a data layer which returns generic lists of objects. For example:

```
public static List GetAllWidgets(){...}
```

I call a stored procedure in that method and map the resulting dataset into the strong-typed object list.

The ORM designer interests me because if it can generate the same methods and classes it would save a lot of time.

However, as the owner of the data layer I don't want a developer to use LINQ to do things that a stored procedure would be best at.

For example:

```
public static List GetWidgetsByFactoryId(FactoryId){...}
```

...will call a stored procedure which joins two tables.

I definitely do not want the user to join "MyDatabase.Widgets" and "MyDatabase.Factories" in LINQ, as the records could be in the hundreds of thousands for each table.

So my question is: is there a way to use the ORM designer, yet still enforce good practices? Or, would I have to discourage the use of LINQ in favor of hand-written functions to make sure people do not do this.



Thanks,  
Roger

[Roger](#) - [Tuesday, March 13, 2007 11:01:53 PM](#)

- Great introduction, Scott. I have a question, how would inheritance work with extension method? For example you have extension method `In()` for class A, how can you have an enhanced version of `In()` for class B which is a subclass of A? Thanks.

[Buu Nguyen](#) - [Tuesday, March 13, 2007 11:36:44 PM](#)

- Scott,  
I'm referring to the guidance posted here:  
<http://blogs.msdn.com/vbteam/archive/2007/03/10/extension-methods-best-practices-extension-methods-part-6.aspx>

"Think twice before extending types you don't own"

[kevindente](#) - [Wednesday, March 14, 2007 2:36:22 AM](#)

- This is really very very cool.

[Vikram](#) - [Wednesday, March 14, 2007 4:37:23 AM](#)

- Is Linq to sql is sql server specific or we can use it with any database {like mysql,oracle,msaccess e.t.c.}

[Kamran Shahid](#) - [Wednesday, March 14, 2007 5:53:48 AM](#)

- Scott, would you consider extending the IntelliSense for an extension method to display where the method originates?

[michielvoo](#) - [Wednesday, March 14, 2007 9:15:16 AM](#)

- Hi Scott, this seems like a really nice feature. Will it also be possible to extend classes with new interface implementations?

So could I do this:

```
public static class ScottGuExtensions : IFormattable
{
    public static string ToString(this bool b, string format, IFormatProvider formatProvider)
    {
        return "whatever";
    }
}
```

So that `System.Boolean` will now also implement `IFormattable`?

Would be strange to have a static class implement an interface though. But then again, it's also a bit strange that a static method of a static class will become a non-static method of another class...

[Wouter](#) - [Wednesday, March 14, 2007 11:16:42 AM](#)

- Remind me why we do OO ?

**purpleblob** - [Wednesday, March 14, 2007 11:34:07 AM](#)

- Is it possible or doable to have generic extension methods? I.e.

```
static class Extension
{
    public static bool In(this T s, IEnumerable c)
    {
        foreach (T t in c)
            if (t.Equals(s))
                return true;
        return false;
    }
}
```

**Natasa Manousopoulou** - [Wednesday, March 14, 2007 12:12:04 PM](#)

- Great blog post Scott. You keep cranking them out, don't you?!

I think your example about extending string with `IsValidEmail` address feels a bit wrong, as it's not extending anything purely string related, but specifically Internet e-mail addresses, which some apps would not need. Instead, things like `IsNullOrEmpty()` or some different uses/overloads of `Join()` feel more natural to me to extend the string class with. As is your second example to check for values in a collection, using `In()`.

I realize you're giving an illustrative example, but think your second example is more in line with your cautious advice at the end. ;-)

Keep up the good work.

**Wim Hollebrandse** - [Wednesday, March 14, 2007 9:34:57 PM](#)

- Hi Scott

Nice post... seeing as this entirely handled by the compiler, can we reflect on an extension method at runtime?

**Hainesy** - [Wednesday, March 14, 2007 9:37:08 PM](#)

- 

Hi Scott,

Aside from the intellisense indicating that it is an extension method, will the editor color code it also to differentiate it from object methods? One might get used to that method without being aware that it is one, then move to another computer and wonder why it doesn't appear in the intellisense.

Thanks,

**bonskijr** - [Thursday, March 15, 2007 5:23:47 AM](#)

- Very nice. This almost gives .NET the same capabilities as java's prototypes. Almost. If an extension name and a base class name are the same, unfortunately the way it currently stands, the base class method is executed instead of the extension, which I'd prefer if it was the other way around, but I

understand just how complicated it would be to try and retrofit that in .NET now. A nice compromise I guess, but still irks be that java's implementation is more powerful.

**Robert McKee** - [Thursday, March 15, 2007 5:56:54 AM](#)

- Is it possible to define generic extension methods? For example, could your "In" method be defined as:

```
public static bool In(this T o, IEnumerable c)
{
    foreach (T i in c)
    {
        if (i.Equals(o))
            return true;
    }

    return false;
}
```

**Richard** - [Thursday, March 15, 2007 1:25:24 PM](#)

- Nice introduction on a new concept. Also, this feature is another step towards SmallTalk'alizing C# - I guess the old folks had some good ideas that are now being more widespread and common :-)

**Dag H. Baardsen** - [Thursday, March 15, 2007 2:50:22 PM](#)

- Great article Scott. I'm really looking forward to using all of the new language features. So, when can we generally expect Orcas to be officially released? :)

**Josh Schwartzberg** - [Thursday, March 15, 2007 10:21:11 PM](#)

- Hi Josh,

Orcas will officially RTM the second half of this year. We'll should have Beta1 out in the next month or so.

thanks,

Scott

**ScottGu** - [Friday, March 16, 2007 6:19:16 AM](#)

- Hi Kamran,

We'll have a provider model that allows you to connect to multiple backend databases when it is all done. So it won't be SQL Server specific.

Hope this helps,

Scott

**ScottGu** - [Friday, March 16, 2007 6:32:21 AM](#)

- Hi Roger,

The good news is that you can map/model stored procedures using LINQ to SQL really elegantly. I think this will give you what you are after in terms of modelling your database.

I'll make sure to do some blog posts on this in the weeks ahead.

Thanks,

Scott

**ScottGu** - [Friday, March 16, 2007 6:47:07 AM](#)

- Hi Natasa,

Yes - you can definitely use generics for extension methods. You would define them exactly as you have above:

```
public static bool In(this T o, IEnumerable c)
{
    foreach (T i in c)
    {
        if (i.Equals(o))
            return true;
    }

    return false;
}
```

Hope this helps,

Scott

**ScottGu** - [Friday, March 16, 2007 6:48:48 AM](#)

- I can't wait for extension methods.

```
public static IEnumerable EachCustomAttribute(this T o, bool inherit)
where T : class
where TAttribute : System.Attribute
{
    TAttribute[] list = (TAttribute[]) o.GetType().GetCustomAttributes(typeof(TAttribute), inherit);
    foreach (TAttribute attr in list)
        yield return attr;
}
```

**Dave** - [Friday, March 16, 2007 10:07:59 PM](#)

- How come I don't see the namespace for System.Xml.Linq I am using the march ctp for 2007. Am I missing something. When I try to add a reference I don't see the dll for system.xml.linq but I do see it installed in gac at



C:\WINDOWS\assembly\GAC\_MSIL\System.Xml.Linq\2.0.0.0\_\_b77a5c561934e089\System.Xml.Linq.dll

is anyone else having problems finding system.xml.linq namespace?

**Zeeshan Hirani** - [Saturday, March 17, 2007 6:26:29 AM](#)

- Hi Zeeshan,

Make sure you add the System.Core.dll assembly reference to your project. I believe there also might be a separate assembly reference you'll want/need to add for the System.Xml.Linq assembly.

Hope this helps,

Scott

**ScottGu** - [Monday, March 19, 2007 6:15:27 AM](#)

- Hi Scott,

Thanks for the great intro to these extensions and continued LINQ discussion. I do have a question though. From an OO perspective I don't see how this feature fits in. I mean, doesn't functionality like 'IsEmailAddress' really belong in a 'format checking' context like a StringFormat type class rather than being incorporated as a method of a primitive type? I'm worried that by using these I may be potentially loosing some of the structure of my applications.

Thanks,

James

**James** - [Monday, March 19, 2007 4:19:08 PM](#)

- Scott,

I think the most obvious benefit is it will almost eliminate the use of helper classes/functions. Can properties be added to types in the same manner? Similarly, if you could implement "extension interfaces" such that you could add the methods and properties to sealed classes to make them implement a custom interface, this would be really powerful and reduce a lot of unnecessary method overloading and logic forks. Will these features also be available?

-Mark

**Mark Hildreth** - [Thursday, March 22, 2007 4:53:51 PM](#)

- Hi Mark,

The extension mechanism works for both interfaces and classes.

Right now you can only add extension methods (and not properties). I know the team is considering adding properties in the future though.

Hope this helps,

Scott

**ScottGu** - [Monday, March 26, 2007 4:33:05 AM](#)

- I know me earlier message may come across a little strongly and somewhat personal... I only mean it as positive criticism and have a lot of respect for Scott in general and recommend his blog as a great source of information on ASP.NET, ASP.NET AJAX Library, and Visual Studio.

**Vic** - [Thursday, March 29, 2007 6:00:50 PM](#)

- This is outstanding Scott! I am really looking forward to using LINQ in my current projects. Keep up the good work.

**MikeR** - [Saturday, April 7, 2007 6:22:51 PM](#)

- Extension methods are the straw that made the camel even more fun to program with.