.NET 2.0 Features

Venkat Subramaniam

venkats@agiledeveloper.com

http://www.agiledeveloper.com/download.aspx

Code examples from this presentation may be downloaded from the above URL

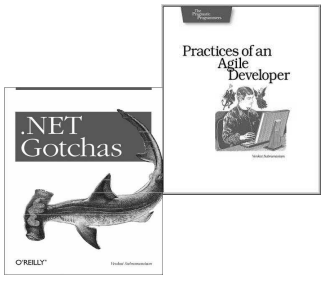
Abstract

Abstract The next version of .NET (Whidbey) has some exciting language features. What are these features and how are they useful? How can you put them to use on your projects? Are there things to avoid? This session will present the new features in .NET 2.0 with examples. We will take a closer look at features like Generics, partial classes, anonymous methods, nullable classes, static classes, property enhancements, iterators, assembly aliasing and rafactoring.

About the Speaker *Dr. Venkat Subramaniam*, founder of Agile Developer, Inc., has trained and mentored thousands of software developers in the US, Canada and Europe. He has significant experience in architecture, design, and development of software applications. Venkat helps his clients effectively apply and succeed with agile practices on their software projects, and speaks frequently at conferences.

He is also an adjunct faculty at the University of Houston (where he received the 2004 CS department teaching excellence award) and teaches the professional software developer series at Rice University School of continuing studies.

Venkat has been a frequent speaker at No Fluff Just Stuff Software Symposium since Summer 2002.



.NET 2.0 Features

- **Whidbey**
- Generics
- Partial classes
- Anonymous methods
- Nullable classes
- Static classes
- Property enhancements
- Iterators
- Assembly aliasing
- Refactoring
- Conclusion

Whidbey

- As of this writing we are working with a product in Beta
- First significant upgrade to .NET
- What ever presented here is subject to change
- It will change

.NET 2.0 Features

- .NET 2.0 Whidbey
- **Generics**
- Partial classes
- Anonymous methods
- Nullable classes
- Static classes
- Property enhancements
- Iterators
- Assembly aliasing
- Refactoring
- Conclusion

Generics

- Remember the good old Templates in C++?
- Java went the route of using Object as generic type
  - Problem is when you pull some thing out of a collection, how do you call methods on it?
  - Only after casting it to the correct type right
  - Much worst if you are dealing with value types
    - These have to be boxed and unboxed
- Having collections that are type safe will eliminate this issue
  - Back to what C++ originally provided ☺

Need for Generics

- This is highly debatable
- First question is do we really need a type safe language
  - What about dynamically typed languages
- If we used dynamically typed languages, then we do not really care about generics!
- But then, we are talking about C# here
- So, how do we solve the issues with such a strongly typed language

Usage

- .NET 2.0 provides System.Collections.Generics namespace
- Various container classes have been parametrized
- Simply create an instance of a Generic Collection class and use it like you would any other class
  - List<int> listOfInt = new List<int>(); …

CLR Support for Generics

- The Generics "map" to MSIL differently based on value types or reference types
- For each value type ***usage***, one class is rolled out at the MSIL level (at runtime)
  - For the following two List classes are created, one of type int, the other of type double
    - l1 = List<int>();
    - l2 = new List<int>();
    - l3 = new List<double>();
- For all reference type usage, only one class is rolled out at the MSIL level
  - For the following only one List class that takes an Object type is created
    - l1 = new List<Car>();
    - l2 = new List<Dog>();

This is what the documentation says!
But that does not appear true
The behavior for reference type appears same as value type

Unbounded Type Parameters

- If you write a Generic like MyClass<T>, there is no restriction on what type argument may be used to instantiate
- Some restrictions apply in this case
  - Can't use == and != operators
  - Can't convert to and from System.Object or any interface
  - Compare with null, but always returns false if type is value type
- One option is for you to constrain the type your generic will accept

Constraints on Generics

- What if the author of a Generic wants to put some restrictions on its usage
- This is done using the where clause
- For instance, you may restrict that a collection you write may accept only objects of classes derived from a Base class

Types of Constraints

- where T: struct
  - type must be a value type
- where T: class
  - type must be a reference type
- where T: new()
  - type must have public parameter less constructor (specified as last constraint)
- where T: *base_class_name*
  - type must be of or derives from base_class_name
- where T: interface_name
  - type must be or must implement the said interface

The constraints themselves may be generic as well

Advantage of Constraints

- Constrains allow you to invoke methods or access properties of a class more freely
- Without constraints, you can only write code for the least common denominator

Generic Classes Inheritance

- A Generic class may derived from a closed-constructed Generic
  - class MyG2<T> : MyG0<int> …
- It may derive from an open-constructed Generic, provided the type is parametrized
  - class MyG2<T> : MyG1<T, int> …
  - class MyG2<T> : MyG1<T, X> … //ERROR
- A non-generic class may derive from a closed-constructed Generic but not open-constructed Generic

Generic Methods

- No need to specify type on a method of a Generic class
  - Results in warning if you do redundantly
  - like class A<T> { public void foo<T>() …
- However, what if you want to refer to make the parameters Generic? Simply declare the arguments of that type
  - class A<T> { public void foo(T obj) …
- What about parametrized methods? Sure
  - class A<T> { public void foo<X>(T o1, X o2) …

Generic Methods…

- For most part compiler figures out the type for methods from the arguments
- If it can't you have to help it when calling
- class A<T>
  { public bool check<X>(int id) …
- When calling you say
  - A<Order> obj = new A<Car>();
  - obj.check<Book>(22);

Generic Delegates

- Delegates use generic in the same way classes do
  - After all under the hood, generics are classes, isn't it?
- Where this is useful?
  - Can pass arguments that are strongly typed and do not have to cast down

Reflection and Generics

- Type object provide information about Generics
  - IsGenericParameter tells if Type is a parameter of a Generic Type
  - IsGenericTypeDefinition tells if Type is a definition of a generic Type (type from which other generic types are created – like A<int> is created from A<T>)
  - GetGenericArguments returns the parametrized types
  - HasGenericArguments tells if the type has generic type arguments (and there is a generic type)

.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- **Partial classes**
- Anonymous methods
- Nullable classes
- Static classes
- Property enhancements
- Iterators
- Assembly aliasing
- Refactoring
- Conclusion

Partial Classes

- .NET 1.0/1.1 requires that class be in one file
- You have to see every thing, like it or not
- How about separating code into multiple files?
  - You can break the class into partial classes
  - Generated code can be separate
- Restrictions:
  - All code related class must be marked partial
  - partial can appear only before class, struct, interface
  - All partial code must be in the same assembly/module
  - Generics can be partial–all parameters must be same and in same order
- Attributes on partial types are merged

Why Partial Classes?

- Multiple programmers can edit the class simultaneously
- Code generators can add code to a class without recreating a file
  - You can keep your code separate from the Visual Studio vomit ☺
- You may keep your nested test cases separate from code you are testing!
  - Occasionally useful to test private members

.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- Partial classes
- **Anonymous methods**
- Nullable classes
- Static classes
- Property enhancements
- Iterators
- Assembly aliasing
- Refactoring
- Conclusion

Anonymous Methods

```
public delegate void SomeDelegate(int a);

class Service
{
    SomeDelegate dlg = null;

    static void Main(string[] args)
    {
        Service p = new Service();

        p.dlg += new SomeDelegate(myMethod);
    }

    public static void myMethod(int val)
    {
        //...
    }
}
```

```
public delegate void SomeDelegate(int a);

class Service
{
    SomeDelegate dlg = null;

    static void Main(string[] args)
    {
        Service p = new Service();

        p.dlg += delegate(int val)
        {
            //...
        };
    }
}
```


Anonymous Methods…

- Can be placed anywhere a delegate is expected
- Reduces code size
- Though may affect readability until you get really used to it
- Scope of variables declared within is within anonymous method block
- You can't jump or goto out of it
- Captured variables are in the scope of anonymous methods–value captured at the invocation of the method
- Can't access ref or out parameters of outer scope
- No unsafe code allowed within

Quiz Time

.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- Partial classes
- Anonymous methods
- **Nullable classes**
- Static classes
- Property enhancements
- Iterators
- Assembly aliasing
- Refactoring
- Conclusion

Nullable Class

- There is a difference between 0 and does not exist
- For example,
  - String s = ""; denotes empty string
  - String s = null; denotes nonexistent String
- Now, how do you do that with value types?
- System.Nullable<T> represents a class that will provide a value for non-existent types

Nullable class

- Think of this like your good old Union
  - HasValue property tells you if value is valid
  - Value will throw exception if non-existent

```
int? x = 123;
int? y = null;

try
{
    Console.WriteLine(x.GetType().FullName);
    Console.WriteLine(x.HasValue);
    Console.WriteLine(x.Value);
    Console.WriteLine(y.HasValue);
    Console.WriteLine(y.Value);
}
catch (Exception ex)
{
    Console.WriteLine("Oops: " + ex.Message);
}
```

```
System.Nullable`1[[System.Int32, mscorlib, Version=2.0.3600.0, Culture=neutral
PublicKeyToken=b77a5c561934e089]]
True
123
False
Oops: Nullable object must have a value.
```

- Short form for Nullable<T> is "T?"
  - like int? stands for Nullable<int>

.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- Partial classes
- Anonymous methods
- Nullable classes
- **Static classes**
- Property enhancements
- Iterators
- Assembly aliasing
- Refactoring
- Conclusion

Static Classes

- What if you need only static methods in a class
- How to ensure that you only have static methods?
  - Unit testing you say ☺
- Static class
  - allows only static members
  - Can't be instantiated
  - are sealed
  - can't contain a constructor
    - (allows static constructor though)
- Can't be used as variable types, etc.

.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- Partial classes
- Anonymous methods
- Nullable classes
- Static classes
- **Property enhancements**
- Iterators
- Assembly aliasing
- Refactoring
- Conclusion

Property Enhancements

- Until now, the get and set methods of a class must have the same access privilege
- What if you want to allow get for public, but restrict set to derived classes only?
- Now, you can set them to your desired value independent of each other

Property Enhancements

- Still can't use access modifiers on interface methods and explicit interface methods
- Use only if you have both get and set, and that too only on one of them
- If you are overriding property from base, you need to match with that
- You can only restrict access given at the property declaration level

.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- Partial classes
- Anonymous methods
- Nullable classes
- Static classes
- Property enhancements
- **Iterators**
- Assembly aliasing
- Refactoring
- Conclusion

Iterator

- Foreach requires your class to implement IEnumerable interface and return an IEnumerator on call to GetEnumerator()
- What does it take to do that?
  - quite an effort
  - or .NET 2.0 make that a breeze
- Iterators take care of keeping tab of current element in collection
- yield keyword does the trick
  - It yields execution to code using foreach but only after remembering the location
  - Execution restarts from here on next access

Iterators…

- Iterators return a sequence of values
- Use the yield statement to return a value or values
- Use it on method body, operator or getter
  - Return type must be IEnumerator, IEnumerable or a generic version of these
  - parameters can't be ref or out
- No need to mess with IEnumerator/IEnumerable. Let compiler take care of that
- When using yield
  - Unsafe block not allowed
  - Can't appear in anonymous methods, in finally block, in catch blocks or try with one or more catch

.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- Partial classes
- Anonymous methods
- Nullable classes
- Static classes
- Property enhancements
- Iterators
- **Assembly aliasing**
- Refactoring
- Conclusion

External Assembly Aliasing

- Allows you to refer to different versions of a component from the same assembly
- Provide an alias for the assembly first
- Use the extern keyword to specify the assembly alias
- You can use aliases to refer to different versions of the class name

global:: namespace aliasing

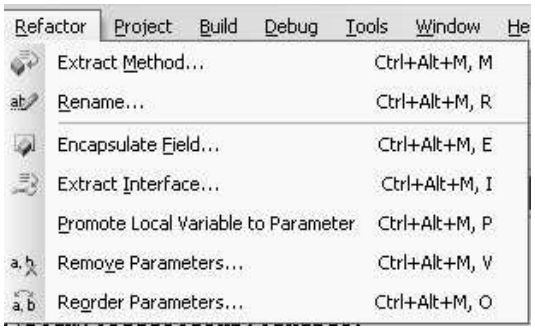
- global:: allows you to specify namespaces from the global namespace.
- May also use aliases instead of global::
- Allows you to get around types that hide namespaces in your code

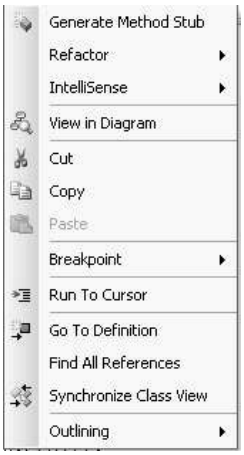
.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- Partial classes
- Anonymous methods
- Nullable classes
- Static classes
- Property enhancements
- Iterators
- Assembly aliasing
- **Refactoring**
- Conclusion

Refactoring Facilities in VS.NET 8

- The first sign of refactoring life in Studio





Popup offers to generate methods

Quiz Time

.NET 2.0 Features

- .NET 2.0 Whidbey
- Generics
- Partial classes
- Anonymous methods
- Nullable classes
- Static classes
- Property enhancements
- Iterators
- Assembly aliasing
- Refactoring
- **Conclusion**

Conclusion

- We have seen a number of enhancements
- Significant change since first release
- Some major some minor
- Look ahead to see how you can benefit from these features
- Have fun

References

1. Microsoft Developer Network (MSDN).
   http://msdn.microsoft.com

2. http://msdn.microsoft.com/vs2005

3. http://www.AgileDeveloper.com/download.aspx

Thanks!

Please fill out your evaluations!