Local Type Inference, Anonymous Types, and var

by Bill Wagner

Of all the features in C# 3.0, local type inference is generating the most questions and misunderstanding. You know, ‘var’. The fact is local type inference is not as scary as it seems. In fact, you’re not losing strong typing. It’s a simple time saver that is actually necessary to support anonymous types.

Let’s begin with anonymous types. Anonymous types are simple tuples, a class containing some set of fields and read/write properties. Essentially, the compiler creates a type that mirrors what you would have written on your own. You’re simply saved the extra typing. Take this query from one of the sample queries included in the LINQ CTP:

```
string[] words = { "aPPLE", "BlUeBeRrY", "cHeRry" };
 var upperLowerWords =
   from w in words
   select new {Upper = w.ToUpper(),
   Lower = w.ToLower()};
```

The compiler creates an anonymous type to represent each of the objects returned from the query. The new type’s definition is similar to this:

```
public class Anonymous1
{
   private string upper;
   public string Upper
   {
      get { return upper; }
      set { upper = value; }
   }

   private string lower;
   public string Lower
   {
      get { return lower; }
      set { lower = value; }
   }
}
```

I am taking quite a few liberties with the exact names of the generated class and the fields. Remember that when you create a type with multiple members, the compiler creates the new class for you. The local variable upperLowerWords is a sequence of these Anonymous1 objects. The addition of anonymous types has saved you from the drudgery of creating this type by hand. The only problem is that you don't know the name of the compiler-generated type.

This is where local type inference, and the 'var' keyword comes in. The actual type generated by the compiler gets a compiler specified name. It's probably something like <Projection>f__c1. You can't predict the class name, I only point it out here to show you what the compiler creates. Therefore, the only way to declare variables using these anonymous types is to use var. Var was added to the language so that you can use anonymous types.

Now that you see why var exists, let's look at what it actually does, and more importantly, what it doesn't do. Var does not create an untyped object; nor is it a synonym for System.Object. Rather, it's a shorthand variable declaration that means "this variable's type is the compile time type of the right hand side of this assignment"'

Therefore, these two declarations are equivalent:

```
var i = 5;
int i  = 5;
```

Both declare a variable 'i' that is an integer. To prove that, try this and verify that it does not compile:

```
var i = 5;
i = "this is a string"; // Compiler error! Can't convert string to int
```

After its initial assignment, the type of 'i' is fixed. (Note that any variable declared using the var keyword must be assigned when it is declared. The following is illegal:

```
var unknown; // Not Assigned, Not Legal!
```

In fact, there are many restrictions on the var keyword. Variables defined using 'var' must be local variables declared inside a method block (or inside a property getter or setter). You can't declare member variables with var; you can't declare a method return value as var.

Now that you know what var is and isn't let's briefly discuss some of the current thinking behind when you should use var as a variable declaration. You must use var when you are declaring a variable that is based on an anonymous type. You don't have any choice there. I also find that I use the var declaration for many results from queries. As I create new queries, I test them step by step, and I'll declare each intermediate result using 'var'. Then, as I refactor the code to its final form, I'll occasionally replace var with the actual type. In general, I'll replace var with an explicit type only if it increases the readability of the code. Often times, it doesn't.   Consider this sample (also from the CTP samples):

```
string[] words = { "cherry", "apple", "blueberry" };

var sortedWords =
   from w in words
   orderby w
   select w;
```

It’s fairly obvious that sortedWords is some sequence of strings. (It’s actually a System.Query.OrderedSequence<string, string>). In my opinion, OrderedSequence<string,string> doesn’t add any new information for me when I’m trying to read this code. In fact, I think it’s clearer with the var keyword.   If you’re really opposed for ‘var’, you might consider replacing var with IEnumerable<string>, but that would depend on the usage.

Var is really a rather simple concept: it’s used for local type inference: local variables whose value is assigned as part of the declaration of the variable. Nothing weakly typed, nothing magical. It’s a combination of a necessity for using anonymous types, and a convenience for complicated variable type definitions. The more you see it used, the more it becomes part of your regular vocabulary, and the clearer it becomes.