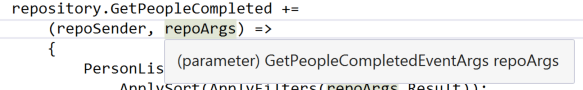
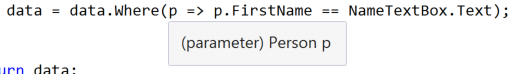
Tuesday, October 14, 2014

**"dynamic" Type Inference with Lambda Expressions**

I had a great time at the Silicon Valley Code Camp this weekend (more on that to come). I get a lot of good questions when I give presentations, and I don't always have the answers up front. This is because I have a bit of a slow thinking process. If the question is about something I've done before, then I've got a good answer for it. But if it's something I've never encountered, then I'm reluctant to give an answer. Just such a situation came up this weekend.  
  
**Lambda Expressions and Parameter Type Inference**  
In my presentation [Learn to Love Lambdas](http://www.jeremybytes.com/Demos.aspx#LLL), I talk about parameter type inference (and also in video: [Anatomy of a Lambda Expression](https://www.youtube.com/watch?v=WJItr-ecdCE)). If the compiler can figure out what the parameter types for the lambda expression are, then we don't have to put them in.  
  
Here's an example where we hook up a lambda expression to an event handler:

[](https://1.bp.blogspot.com/-DY2WACsCGHU/VD1dS9Mj0nI/AAAAAAAADA8/JXdzMyLr8io/s1600/TypeInference.png)

When we hook up an event handler, the method \*must\* match the method signature, otherwise, we have invalid code. Since the compiler knows that there is only one valid method signature to hook up to this event handler, it knows the two parameter types are "object" and "GetPeopleCompletedEventArgs". And we can see that "repoArgs" is, in fact, of that particular type even though we don't specify it in our code.  
  
This also happens quite a bit in LINQ. Here's an example of using the "Where" method to filter a collection of "Person" objects:

[](https://2.bp.blogspot.com/-3tUM3LSGHXM/VD1dTanMInI/AAAAAAAADBA/nXqi5tWUvSc/s1600/TypeInference2.png)

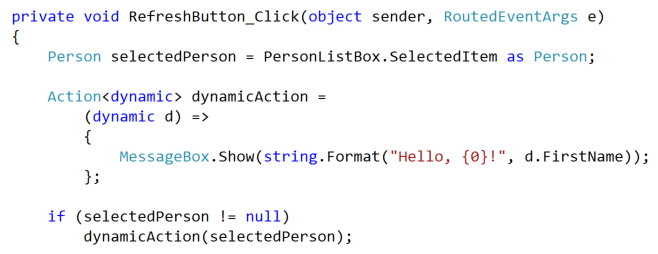
Again, we see that the parameter is a "Person" since that's the only method signature that we are allowed to use here.  
  
**What About "dynamic" Types?**  
So as I was showing how awesome parameter type inference is, I got a question:

**Q: Does parameter type inference work with "dynamic" types?**

And I'll admit that I froze up for a bit. It's something that I never thought about. So, I was honest and said that I'm not sure how it would work since I'd never tried it before. And I said that it would be a blog article some time this week (hey, here it is!).  
  
And with a little more thought, the answer was really obvious:

**A: Of course parameter type inference will work with "dynamic".**

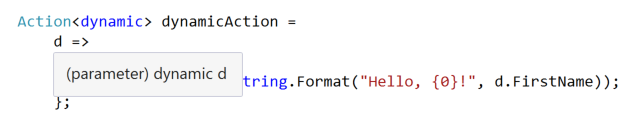
"dynamic" is a type; it just happens to be a type with some special compile-time and run-time behavior. But it is a type nonetheless. So it will work with parameter type inference just like other types. We're still matching method signatures; the only difference is that we use "dynamic" for a parameter type instead of "string" or something else.  
  
**Some Experimenting**  
So, I did some experimenting to see how the IDE behaves. I started with the code from the Learn to Love Lambdas session ([which you can download](http://www.jeremybytes.com/Demos.aspx#LLL)) -- specifically, the button click event handler in the code-behind for MainWindow.xaml:

[](https://2.bp.blogspot.com/-2-mDlV7VYLI/VD1dT0D2MWI/AAAAAAAADBM/mzNihz_hu8Y/s1600/VerboseAction.png)

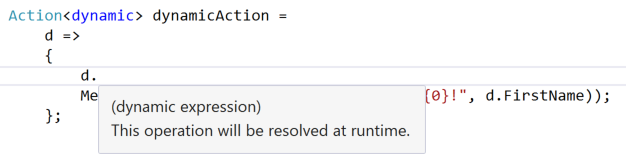
I started out a bit verbose (so that we could follow this through). The first line of the method is existing code that grabs the selected item out of a list box in the UI. This is a "Person" that has 4 properties: FirstName, LastName, StartDate, and Rating.  
  
Next, I created a delegate of type "Action<dynamic>". Action is just a built in delegate type that takes zero or more parameters and returns void. So our "dynamicAction" delegate needs to be hooked up to a method that takes a "dynamic" as a parameter and returns "void".  
  
And that's what we do with the lambda expression. I start out being explicit about the parameter type, using "dynamic d". And in the body of the method, we just say hello in a message box.  
  
Finally, we check to make sure that there was a "selectedPerson", and if so, we execute the "dynamicAction" delegate with that person as the parameter.  
  
**Parameter Type Inference**  
But we're still being explicit with the parameter type. Let's strip out the type and rely on parameter type inference (we'll also remove the parentheses since we only have 1 parameter and remove the curly braces since we only have 1 statement):

[https://3.bp.blogspot.com/-7nMQ_1PeQ0c/VD1dR8Io5aI/AAAAAAAADBg/-C8WrL_8oJs/s1600/CompactAction.png](https://3.bp.blogspot.com/-7nMQ_1PeQ0c/VD1dR8Io5aI/AAAAAAAADBg/-C8WrL_8oJs/s1600/CompactAction.png)

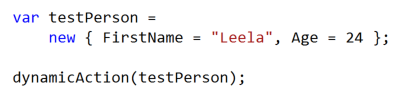
This is equivalent to the code that we have above, and it runs exactly the same. If we hover the mouse over the parameter, we can see the type:

[](https://1.bp.blogspot.com/-CllUvf4to44/VD1dScpMFpI/AAAAAAAADA4/i66v_s345wE/s1600/DynamicTypeInference.png)

It is "dynamic" just like we expect. And if we use the parameter in the body of the method, the IDE treats it like a dynamic:

[](https://4.bp.blogspot.com/-srwLxwt32Qw/VD1dRyb4jLI/AAAAAAAADA0/EC2CEkLFIHQ/s1600/DynamicCodeCompletion.png)

So, if we type "d." code completion doesn't give us any hints. That's because we can type in whatever we want when we use a "dynamic" type, and the method or property call will be resolved at run-time.  
  
**An Anonymous Type**  
To show that we can take advantage of the dynamic nature of this parameter, let's create an anonymous type and try to use it with this delegate:

[](https://3.bp.blogspot.com/-RCaVHYHlp7o/VD1dRsfxxsI/AAAAAAAADAw/5hNWMZx9SBU/s1600/AnonymousType.png)

This code creates an anonymous type that has 2 properties: FirstName and Age. When we call our "dynamicAction" delegate with this parameter, it works just fine. That's because our anonymous type does have a "FirstName" property (which is the only property that is being used in the body of the method).  
  
**Wrap Up**  
Since "dynamic" is just another type, parameter type inference in lambda expressions works just fine. It seems very obvious to me now. But it wasn't obvious when I was "on the spot". When I get questions, I never just make up an answer. Sometimes I will give an educated guess, but I will always qualify it as such. I'm much more comfortable taking a little extra time to get things right.  
  
Happy Coding!