



Hanselminutes

Hanselminutes is a weekly audio talk show with noted web developer and technologist Scott Hanselman and hosted by Carl Franklin. Scott discusses utilities and tools, gives practical how-to advice, and discusses ASP.NET or Windows issues and workarounds.

Text transcript of show #75

July 30, 2007

John Lam on IronRuby

Scott sits down with John Lam at OSCON the day that IronRuby Pre-Alpha 1 was released and talks about the announcement to host on RubyForge.

(Transcription services provided by [PWOP Productions](#))



Our Sponsors

 **telerik**
deliver more than expected
<http://www.telerik.com>

 **nsoftware**
<http://www.nsoftware.com>

 **NET DEVELOPER'S JOURNAL**
<http://dotnet.sys-con.com>





Lawrence Ryan: From hanselminutes.com, it's Hanselminutes, a weekly discussion with web developer and technologist, Scott Hanselman, hosted by Carl Franklin. This is Lawrence Ryan, announcing show #75, recorded Tuesday, July 24, 2007. Support for Hanselminutes is provided by Telerik RadControls, the most comprehensive suite of components for Windows Forms and ASP.NET web applications, online at www.telerik.com, and by .NET Developers Journal, the world's leading .NET developer magazine, online at www.sys-con.com. In this episode, Scott talks with John Lam at the OSCON conference about the IronRuby Pre-Alpha 1 release.

Scott Hanselman: Hi, this is Scott Hanselman and this is another episode of Hanselminutes and here at the O'Reilly Open Source Conference, OSCON, with John Lam. Did you announce today that you're doing -- today is Monday by the way.

John Lam: Yeah, I announced at 2:00 a.m. today on Monday.

Scott Hanselman: What did you announce?

John Lam: We shipped our first source distribution of the IronRuby implementation from Microsoft.

Scott Hanselman: So, you shipped IronRuby. This means you shipped your first drop. I saw that it was labeled Pre-Alpha 1.

John Lam: Yeah. Yeah, yeah, yeah.

Scott Hanselman: And I downloaded it off of your website.

John Lam: Uh-huh.

Scott Hanselman: But what's the plan for when that's going to be hosted?

John Lam: So, down the road, what we want to do is we want this thing to be hosted up on RubyForge, which is the community site for Ruby software development and right now we had to just ship it out as a ZIP file on my blog simply because that was the quickest, easiest, lowest friction way to get the code out there to people.

Scott Hanselman: And the plan is to get it up on RubyForge and I understand that you're going to be taking contributions.

John Lam: Yeah. Absolutely. So, that's probably the biggest announcement there is not only are we going to be up on RubyForge by the end of August, but we are going to be accepting community contributions back into a subset of the IronRuby language project, so specifically that subset is in the libraries. What we're doing initially is we're going to reserve the compiler pieces. So, we're going to keep that. We're not going to accept contributions back into the compiler pieces for a very, very good reason and that reason is today, IronRuby is built on top of another piece of software that my team creates, which is called the Dynamic Language Runtime. So, the DLR is not, in *Microsoftian terms* fully baked. So, in essence, we don't have public interfaces for this thing yet and code is still kind of migrating back and forth between what is in the DLR and what is in the compilers. So, until we can firm up those interfaces and until we can get to essentially a 1.0 quality release with publicly defined APIs for the DLR, we can't accept contributions back because potentially external code could wind up, in Microsoft terms, tainting the DLR which will ship with the CLR and all that kind of stuff, so we simply can't have that stuff happen, but once we get these public interfaces inside of the DLR itself, then we will fully open up the IronRuby compiler all for community contributions.

Scott Hanselman: Now, I spent the better part of today, this is Monday and I think this podcast is being listened to on Friday, with my intern, with my high school interns and we took the IronRuby stuff and we built up using Scott Guthrie's Hello World example, a WPF example, that we wrote it in Notepad and it calls a C# client that is a REST frontend, pulls out some C# type like an array of accounts and we go and pull some information down from a financial services website. We put that account balances information into a WPF app. So, we had C# in there from a library that we wrote, we had some WPF in .NET 3.0, all talking together and as I said on my blog post, it was very, very Pre-Alpha and I was accepting that, but what I thought was really interesting was the kinds of stuff that we bumped into, stuff that this wasn't done yet, gave me a lot of insight into the kind of work that you guys need to do. For example, I got back an array of accounts. I put it in a variable called A and I said A[0] to pull an account out and I got, what?

John Lam: You probably got nothing. You probably got a missing method exception at that one time.



Scott Hanselman: Exactly. I got a missing method exception []].

John Lam: So, the important thing that's interesting is you probably could have locked up to that class that represented the .NET type and added the square bracket method yourself.

Scott Hanselman: And that's interesting. I had that thought myself because I ended up just calling GetValue because GetValue was on System.Array and I said, "I wonder if I could use the method missing features to go and built this thing up and make it look the way I wanted it to look."

John Lam: Yeah. We fully support method missing in the set of bits of that we shipped today. In the future, we're going to do a bunch of automatic stuff. So, a lot of these things I've already done in a former life when I built Ruby CLR. So, in Ruby CLR, when I detect that you are marshalling a type from the CLR that implements IEnumerable Interface, I do all of the right things. I provided each implementation for you. I auto mix in the IEnumerable module as well or the IEnumerable mixin, so then you can use all of the standard Ruby idioms for iterating over some collection.

Scott Hanselman: Yeah and that was the thing I thought was really interesting was that it didn't feel very Rubyesque because each time I wanted to use a Rubyism, it wasn't there yet and I started looking at the source and that was what was so much fun was to figure out, "Oh, I see where John and team are gonna have to get in there and hook this up." I brought in a Single, a System.Single. I typed it as a float, it came in as a System.Single, I wanted to do a two-string, it looks like it wasn't there. So, then I ended up switching it to a Double and two-string was there. Then I went and looked at some of the built-ins that you had in the Ruby stack. You've got like *fixnum*.

John Lam: Uh-huh.

Scott Hanselman: Can you talk about that? I thought *fixnum* was an extension of an integer? Is that a derivation?

John Lam: No. It's not an extension. It is a type substitution. So, in essence, what we're doing is we're saying that, "Okay, anytime where you see an int, instead please treat this as the Ruby type *fixnum*." So, all of our method dispatch code essentially where you can essentially dispatch against an integer, it

goes, "Oh, okay. I know exactly where we need to route this method is to the *fixnum* type." So, what you'll see if you look in the implementation, there's a file called Initializer.Generated.cs, which is a giant like 100 K file right now in the sources. If you look inside there, that's essentially the file where we mark these methods which are effectively extension methods, especially using attributes. When you're looking through the source code, I'm sure you saw lots and lots of attributes sitting on top of methods. So, what we do is we've got a little tool that will go through the methods, read the metadata, the attributes, and then we will Codegen out a bunch of *goop* to essentially say, "Okay, this is the name of the class. It has this set of methods on it and it really represents this .NET type in that case."

Scott Hanselman: So, forgive my ignorance if I use the wrong words, but is type substitution the equivalent of what I would have thought of as marshalling? Are we moving from one world to another?

John Lam: Well, no, because an integer is an integer. So, essentially, what we're saying is that any time the Ruby implementation -- because remember the Ruby implementation runs on top of the CLR, so it is using things like *ints*, and *doubles*.

Scott Hanselman: We're not moving in and out of universe. It's like we do with COM Interop.

John Lam: No, no, and quite unlike a Ruby CLR where, yeah, I had to marshal types across C boundary because across, there are two completely different worlds. One is a C-based interpreter and the other thing is the CLR. So, everything is just in the CLR where it essentially gives you a different view over the types that you currently have today in the CLR

Scott Hanselman: So, if I newAP a string in the Ruby.NET or in the IronRuby world and I pass that string into my C# library, what happen there? I mean the string is still sitting somewhere on the heap.

John Lam: One thing to really remember is that what's different between Ruby and most other implementations is that the Ruby strings are mutable strings. CLR strings are immutable strings. So, we have to convert when we cross the "boundary."

Scott Hanselman: Because of the way strings work.



John Lam: Exactly and if you take a closer look at the way we've implemented the extension methods and the various types on our libraries, what I generally do is we wind up having an overload of them that takes a regular CLR string and then also another overload that takes immutable string. So, the design of the library is such that we want to stay as strongly typed as possible wherever we can and in cases where the type being or the primary type being passed in where we can't determine that, right then we'll accept objects and we'll do conversions and stuff. So, there are all sorts of patterns in the Ruby library, for example, where you can pass some random objects and what the library will do is attempt to call two int or actually attempt to see whether or not there's a two int method implemented on that object. If it finds that, then it does a different thing where it essentially treats that as an integer.

Scott Hanselman: So, for the purpose of somebody, some of the viewers may not necessarily know the difference or remember the difference between immutable string and an immutable string, so if you could just speak on that for just a moment as sort of reminder.

John Lam: Sure. When you create a CLR string, once it's been created, you cannot change the contents of the string. Every single API that you would call on a string that you would imagine is changing the string, it doesn't actually change the string. It actually creates a new string with the modification inside of that string.

Scott Hanselman: So, your implementation of a mutable string, if I remember correctly from looking at the source code, we had a StringBuilder in there.

John Lam: Yup. Exactly and that's an implementation detail. So, today, you also notice that unlike trying to do that type substitution thing that I did with *fixnum* and integer, instead we wrapped a StringBuilder and a mutable string. The reason why is that eventually StringBuilder is going to go away in our implementation.

Scott Hanselman: I see.

John Lam: The reason why is that Ruby itself treats strings as byte arrays, not char arrays, so effectively a StringBuilder is a dynamic char array.

Scott Hanselman: Oh I see.

John Lam: So, for that reason and for compatibility with Ruby, somewhere down the road we're going to swap out the implementation of StringBuilder and swap in a mutable byte array.

Scott Hanselman: As you said, that's certainly an implementation detail.

John Lam: Yup.

Scott Hanselman: I just find it to be a particularly interesting one.

John Lam: Yup.

Scott Hanselman: So, in that case, strings are really the special case as it seems with almost all languages. We keep saying strings are strings but ultimately every language has its different perspective on how a string should go out.

John Lam: Absolutely.

Scott Hanselman: Now, I noticed that a number of times I had to call two_string and two_s, are those the same methods?

John Lam: No. Two_str and two_s are two different methods. What two_str is it's another one of these Ruby signal methods effectively, so what Ruby does or what the Ruby libraries do is they walk up to that object and invoke a respond to on that, so "Respond to two_str: Yes or No." If so, it treats that thing as a string and invokes other methods. So, effectively, it treats that as a duck-type string.

Scott Hanselman: Say something about duck typing for our listeners.

John Lam: Sure. So, duck typing, you know, there's that old saying about if it walks like a duck and quacks like a duck, then it probably is a duck. So, you're essentially saying if an object responds to the methods that it would expect a string to respond to, then it must be a string. It's about as simple as that. So, back to the idea about using two_str, it's one of these marker things. Two_int is another one of these marker things inside of the Ruby libraries. So, we just simply go off and check to see if it's a string and if so, then we treat it as a string form that point always.



Scott Hanselman: So, in this process, there were a number of times where I passed an object and then had to call two string on the object, but I actually ended up having to call U and U had me call a secret method called two CLR string.

John Lam: Right.

Scott Hanselman: Is that something that I'm not going to have to think about in the future? Why do I have to do that?

John Lam: Well, right now, that was just the most obvious way for us to deal with what happens when we need to convert immutable string into a CLR string. It's essentially for us to explicitly do it. Down the road, we will, you know, do some magic to take care of that for you.

Scott Hanselman: Sure. Even though there were a couple of things like that that I bumped into, it was funny, we were looking at the code and it was mess of comments and whatnot, we ended up stripping all of the comments out and it was pretty simple. I mean, really, there was nothing to it. We spent more time putting together what WPF XML was going to look like.

John Lam: Sure.

Scott Hanselman: Doing it by code, building the object model ourselves.

John Lam: Yup.

Scott Hanselman: Now, how do you anticipate this is going to be used? I assume it will be a first among equals kind of a situation where I can do anything if I feel like doing WPF in Ruby or ASP.NET?

John Lam: You shouldn't have any inherent limitations in terms of the kinds of libraries that you should be able to consume from our implementation. I think there are a number of really interesting scenarios that you can look at, the Silverlight one being one of the obvious ones because right now, today, nobody else really has the ability to run Ruby in the browser potentially across an extremely large deployed base of browsers that support Silverlight and also cross-platform between Mac and Windows, so that's a really interesting reach opportunity for us and "WPFy" style programming, I'm not sure what the actual name of it, internally we call it

SPF, but I'm sure there's some other kind of name for what...

Scott Hanselman: Is SPF Silverlight Presentation Foundation?

John Lam: I think so, but I think its acronym status has been revoked so I don't think that actually means anything anymore.

Scott Hanselman: It's a three-letter acronym.

John Lam: Yeah. So, I still call those guys the SPF guys.

Scott Hanselman: I see.

John Lam: So, that programming model is I think in need of having some additional support for building applications. If you look over the Silverlight stuff, it's a very low-level API essentially for doing drawing and presentation. So, given that, what we would really like to be able to do is go off and build a library and make it so much easier and much more approachable for people to go off and do that stuff and I think that Ruby would just be a wonderful language for using to experiment with building out such a framework.

Scott Hanselman: Yeah, I don't know -- of course, I don't work for Microsoft yet. I'm going to go to work for Microsoft in September.

John Lam: Congratulations on the new position.

Scott Hanselman: Thank you. Thank you very much. September 3rd. So, until then, I'm sure I can get myself into a lot more trouble, although I plan on getting into plenty of trouble once I get there. I always thought it would be a really interesting scenario to have the Rails programmers build some kind of a Silverlight support gem such that the relationship between Ruby on the server side and Ruby on the client side would become something maybe unique to that environment.

John Lam: Sure.

Scott Hanselman: That somehow they would come up with ways to serialize objects in and out perhaps JSON such that it would invent a new kind of a deployment, a new kind of programming, such that the browser and the big fat Pentium Core 2 that's not



doing any work today would actually start doing some really serious heavy lifting and it would almost have like a new tier to classic three-tier architecture.

John Lam: Yeah and I think that would be a really interesting place for people to explore with some of the options because certainly instead of using RGS to go off and generate JavaScript that goes off and runs on the client, you just emit some Ruby code and that can go off and run on the client as well and I think that having a language, that makes it easier for you to build modular software like Ruby on the client. It's going to really do a lot and spur a lot of innovation in the browser.

Scott Hanselman: Now, IronRuby is adding to the list of dynamic languages that it support. We have Python. We're going to have JavaScript. We're also going to have VBx, Dynamic Visual Basic?

John Lam: Yup, some to-be-named product that is the Visual Basic.

Scott Hanselman: Sure. So, that's at least four dynamic languages.

John Lam: Yup.

Scott Hanselman: I know that you guys had a language lab and they had the guys that do languages like Boo and I think someone was talking about doing a Squeak implementation?

John Lam: So, there was Boo. There are some guys that do ColdFusion there.

Scott Hanselman: Oh wow.

John Lam: There's, of course, Peter Fisk who on the outside is building the Smalltalk implementation as well.

Scott Hanselman: That's hot.

John Lam: That runs inside the browser.

Scott Hanselman: Vista Smalltalk?

John Lam: Vista Smalltalk, yeah.

Scott Hanselman: Oh, that's awesome.

John Lam: And other people. So, I think that there's going to be a lot of opportunities again, especially once there's more code.

Scott Hanselman: Sure.

John Lam: Because DLR still isn't fully baked. So, once we have some more code, more examples, more programming languages, and some documentation and that kind of stuff, I think it's going to make a lot easier for people to make some progress, although Peter Fisk with the current state of the stuff we just dropped a pile of source code on him and he's been making fantastic progress.

Scott Hanselman: Really? So, do you anticipate it will be people like that who will push the new updates and who are going to contribute the most source code is going to be language builders?

John Lam: Yeah, yeah. Well, there are several places. It's the classic you got to scratch an itch kind of thing in open source, so if there's something that's bothering you right now, I think that filling that gap by building some libraries or contributing the libraries and stuff like you don't need to be a compiler geek in order to go off and build some libraries and to port some existing libraries over. So, those are the kinds of things that I think are going to be opportunities for all sorts of people to contribute, not just compiler geeks.

Scott Hanselman: Well, I really appreciate you taking the time to come and talk to me today. I know that you're in between sessions right now and we're sitting in the hallway here.

John Lam: Sure.

Scott Hanselman: Thanks a lot and I encourage the listeners to go and check this out, start poking around. It is early, but it's a lot of fun to see this kind of development going on, particularly around dynamic languages. On my last show, the interns we're talking, they said, "Dynamic languages are just easier. It's just more pleasant to program in."

John Lam: Sure.

Scott Hanselman: It's very fun to see almost the arm wrestling that's going on as C# continues to evolve into a more dynamic language as VB kind of reinvents itself or reasserts itself as a dynamic language to see these other languages come in.



John Lam: Yup.

Scott Hanselman: It's almost like the promise of .NET when it first came out. It was going to be multiple language...

John Lam: The *common language runtime*.

Scott Hanselman: Yeah, the *common language runtime*. Now, 2007, that's actually happening.

John Lam: Yeah and I think that's really exciting time, you know, to be working in CLR team right now.

Scott Hanselman: Cool. Thanks a lot and this has been another episode of Hanselminutes. We'll see you next week.