



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 #167

June 25, 2009

Convention Over Configuration with Jeremy Miller

Scott's Norway interviews continue this week, this time with Jeremy Miller, author of Structure Map. Scott and Jeremy chat about fluent interfaces, Convention Over Configuration and how to best simplify your systems.

(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 #167, recorded live Friday, June 19, 2009. 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 convention over configuration with Jeremy Miller.

Scott Hanselman: Hi this is Scott Hanselman and this is another episode of Hanselminutes and I'm here in Norway still and I'm hanging out with Jeremy Miller, the Shade Tree Developer and also the author of structure map. Thanks for coming up here and talking to me today.

Jeremy Miller: Thanks for having me.

Scott Hanselman: So we had a lot of really cool talks here in Norway and you had a talk on convention over configuration and I knew we needed to talk about that so we hear convention over configuration basically everywhere that's not in the .NET space and we've only, recently started hearing about it within the context of ASP.NET MVC which in fact borrows from a lot of places that aren't in a typical .NET space. .NET developers are used to a lot of angled brackets and config files that go on for days. I know that for a while structure map had a lot of configuration involved and recently that's changed. So you're getting into this convention thing as well?

Jeremy Miller: Oh, absolutely and it's definitely a case of Ruby on Rails envy.

Scott Hanselman: Really?

Jeremy Miller: I think it just took us a couple of years to realize that we could do it ourselves. It's just that it manifests itself very differently when we do it in .NET development.

Scott Hanselman: Okay and why is that?

Jeremy Miller: Well, we're statically typed. We can't do a lot of this dynamic property generation with meta-program and that can in Ruby but we have all these other tools. We have the typing system. We can take advantage of open-generic, generic declarations. There's a lot of rich type metadata we can get at that we're already doing that's creating information about our system that we were then recreating inside of explicit configuration whether it be an angle bracket hell or just through programmatic configuration. Instead, why not just scan the information that's already in the type system and use that to wire things up.

Scott Hanselman: Okay. Now, when I think of configuration, I think of what assembly type I should load, my connection strings, maybe a directory location for something. What kinds of things that were typically configuration can be replaced with convention?

Jeremy Miller: Well, how pieces wire up. I think the easiest to example to go at is thinking about configuring object relational mapping.

Scott Hanselman: Okay.

Jeremy Miller: Inversion of control is good too. Well, let's talk about ORM mapping.

Scott Hanselman: All right.

Jeremy Miller: Three years ago, we we're using NHibernate. You might have been using the active record attributes for us we were using an XML file. We were specifying a lot of facts, A class maps to a table that probably has the same name but I still have to say class name and table name. I'm mapping columns, I'm specifying what the primary key of the object is even though you can find this information in the database but I still had to specify it in NHibernate mappings. How the identifier was created, is it controlled by the database, is it controlled by classes. That's a lot of explicit stuff that I have to do.

Scott Hanselman: So you're saying something simple like public class customer is from the customer's table that was still being repeated in some configuration.

Jeremy Miller: Yes.

Scott Hanselman: And there's also the pluralization of that?

Jeremy Miller: Yes, if you want to do the pluralization.

Scott Hanselman: I don't know if you do, is that a religious argument?

Jeremy Miller: I don't know. I don't bother with it.

Scott Hanselman: I remember seeing David Heinemeier Hansson give a talk about how awesome Ruby on Rails was and how the pluralization actually had a specific thing in case you had a table called Octopus and a collection that would be octopi and he was really proud of that.

Jeremy Miller: I think...

Scott Hanselman: Because you'll never know when you might have a table called Octopus.



Jeremy Miller: I think that's really cool but I think that's a lot like going back to the VB days when we argued about two spaces or four spaces for indentions but a consistent convention and we have these conventions in our code all the time or we just do things the default way. So, why not instead let the ORM Mapper infer from the shape of the object the way the mappings work by default and I should, at this point, I should only have to override the things that deviate from this convention.

Scott Hanselman: Okay. So only exceptional cases would get any configuration information at all?

Jeremy Miller: Yes.

Scott Hanselman: Does that mean that in a typical case, in the 80% case you wouldn't require any configuration at all?

Jeremy Miller: If you're following your standards and your conventions, yes.

Scott Hanselman: Okay. So is this something that is only in NHibernate in ORMs or is this something that -- how would I start applying these to my applications?

Jeremy Miller: Well, so the ORMs a big deal by itself inversion of control containers, we do the same thing. You mentioned MVC development...

Scott Hanselman: Right.

Jeremy Miller: MVC, out of the box, your URLs, you have the application root and then the next argument is a controller name, that's fine. The controllers by just convention or an idiom, we would name them home controller, report controller, something controller. The URL would be report home/something to the action.

Scott Hanselman: Right.

Jeremy Miller: So, a lot of us are using inversion of control tools to host the controllers. You just go to the IoC tool and say, "Give me the controller for this name." We have that name in convention. Let's just let the controller scan over the types in our assembly and know that home controller is registered as lower case home. That's a convention. It's a naming convention we can take advantage of. So as I start to add new things to the system, I can add a new controller, code it up, unit test it but I don't have to worry about telling the container or the MVC that, "Hey this new thing exists, it's already there and available and accessible through a URL."

Scott Hanselman: Okay. I'm trying to think what else we could expand this too. I mean I've got to think of other kinds of configuration information that I've

got. I know that you could probably take it to a really dramatic length like I know that I've had configuration for development for staging and production. I suppose if I want to take convention over configuration to as far as I could, I can even name my systems, my computer names a certain way.

Jeremy Miller: Well, that's supposed to be a bad practice, right? Isn't that supposed to make your system easy to hack?

Scott Hanselman: I don't know.

Jeremy Miller: Well, so maybe -- some of the other things I can think of, I know there are people that are experimenting with the creation of UIs, like as we build a lot of these CRUD screens we're using textboxes, we're using drop downs. The drop downs have to be filled from somehow. We're probably doing these things very consistently.

Scott Hanselman: True.

Jeremy Miller: Anytime I have a date field that's editable on the screen, I want it to show up with this particular date control. So, something I'm seeing a lot of people experiment with is just simply expressing, "I want the input element for this object, for this property."

Scott Hanselman: Right.

Jeremy Miller: And the conventions will take over and say, "Well look, it's a string field. I'm going to use the textbox but it's also decorated with some kind of validation attribute that says it can only be 100 long. So, I'm also going to come back and set the maximum length of the textbox and look, it says that it's a required filled. Somehow I can tie in the validation and I know it's a required filled, so maybe my designers want all required input elements to be colored pale blue or no, what if they want a red asterisk, yeah."

Scott Hanselman: Yeah, absolutely.

Jeremy Miller: That's the kind of coding that I don't really want to spend a lot of time on, I just want it to happen.

Scott Hanselman: Right.

Jeremy Miller: We can set up convention, we're really just setting up, not necessarily a convention but a policy that, this is the way it always is.

Scott Hanselman: I like that, convention is a good word but policy, I think is a really -- all we need to do is to get a team of us whether it be five of us or a hundred of us to just agree and I think that in my experience on developing for the web, CSS people



and designers are like the first people to get this convention or configuration right. Just say, "This textbox has a class of date than all of these validators happen for free and all of this look-in field happens for free and it just works that can be pushed down farther and farther and farther." Actually one of the other applications not just ASP.NET MVC but ASP.NET Dynamic Data was another app that no configuration at all. One line to say the database is over there and everything else was completely conventional if there was a Boolean type and then you use Boolean.ASCX and it's interesting because it freaks .NET developers out because since 2002 everything has been configurable, the location of the file, the name of the file, the extension, all of it has had some angle bracket somewhere buried that could be overwritten and how to tell them that well no, just by the virtue that this thing was named this way. It's just going to work.

Jeremy Miller: Well, you bring up a really good point. It is the potential problem with it is it's black magic. There's no explicit code that you can look at that's going to tell you exactly what's going to happen. So there's a couple of ways you can beat the problem and the first is kind of a team wide social contract, these are our conventions. You have to have the entire team understanding and agreeing to follow these conventions, that's step number one. Step number two is you probably need to think about some diagnostics. Take the IOC example. What's in my container? I make up these conventions, the controller configuration, I use the `Isomething`, `Isomething` is implemented by something to automatically wire something to `Isomething`.

Scott Hanselman: For those listeners that don't speak English, that whole sentence may have been a little bit confusing but you're saying that by virtue of the fact that I have an `IPerson` interface and I might have a person class, I can glean that just because I'm putting I in front of my interface.

Jeremy Miller: Yeah. So that can be a little bit scary. Now you're letting the program itself infer what it's supposed to do.

Scott Hanselman: Right.

Jeremy Miller: You're going to need to troubleshoot and understand what's my convention coming up with? You can beat that by creating some diagnostics. In our case, like the controller example, let's make sure we have an easy report that can tell us what controllers does the IOC container think it has and under what name as a beginning to troubleshooting.

Scott Hanselman: Okay. So I can checkdisk for your app then just go and say, "Let me know all the different possibilities and dump it out and look at it and say, 'Does this look correct?'"

Jeremy Miller: Exactly.

Scott Hanselman: Okay. Now, there are other things I've seen but this maybe a silly example but it actually was useful to me. I was working at a large company recently as a consultant. I can't say what it is but let's just say that they were doing cars and we needed some graphics and there's all these different car manufacturers out there and we figured out just by poking around in their website that they had like `images.foo.com\` and I could say `ford.png` and I get the Ford logo and we were doing this demo for the vice president. So, it's really just proof of concept code but we knew that putting some graphics in would kind of sex it up a little bit and they said, "Oh, put the logo for the car company here," and we say, "Oh, wait a second. We have an image server, right? Let's see what happens if we put in Ford, Chevy, Audi, Fiat and oh look, they've come up with a very basic convention." It was simple but I was worried, legitimately worried because I've had this happen before that I was going to have to go to a database somewhere and get the actual URL for this image and by virtue of the fact that this company had done this very simple and commonsense, although that's the thing about commonsense, it's not that common. We were able to in 30 seconds of extra coding, put in some graphics that made the demo just a little fraction of a percent easier. So image pass, CSS naming, scripts, all these things can make life a lot easier.

Jeremy Miller: Definitely and that's a great example of the value of this kind of thing. A lot of these things we're talking about, they're small things but they're the small things that we do over and over again by making it the less friction to make a change or an addition to the system. It makes us easier, of course, to extend the system but it also makes it easier for us to iterate to the system and take the example of a user interface. You come up with an idea of how you think it should work. You show it to the user and you find out that you were completely wrong.

Scott Hanselman: Right.

Jeremy Miller: You need to change it by having less machinery to change those images on those pages. Maybe you could iterate faster and take advantage of the feedback that you got and come out with a better product.

Scott Hanselman: Yeah, absolutely.

Hi, this is Scott Hanselman with a word from our sponsor. Do you know how to build web 2.0 AJAX application with web 1.0 components? You really can't. You want to do the next generation web applications? You'll need next generation components just like the ones our friends at Telerik have got, their RadControls for ASP AJAX. It is a



huge pack of web controls built on top of ASP.NET AJAX that will add previously impossible performance in your activity to your next project. The new controls mirror the AJAX API from ASP.NET so development is really straightforward. The client scripts are shared so loading time is not a problem. You just set a couple of properties and you'll be able to automatically bind the web services for a really efficient operation. The new RadEditor from ASP.NET AJAX Telerik loads up to four times faster than before, and the new RadGrid handles thousands of records in just milliseconds, but as always, it's best to try it for yourself. You can visit telerik.com/aspnetajax and download a trial. Thanks a lot.

What are some other examples that you gave in your talk about convention over configuration? What was the general gist of what you were trying to get across to the attendees?

Jeremy Miller: The general gist is a meme I see a lot more often now, the idea of essence versus ceremony, essence being what your business partners want you to deliver and ceremony being all the hoops, the developer-type hoops we have to jump through, type decorations, XML files, configuration, registry settings, stuff that's pure overhead. What we're trying to do is to create more essence with less ceremony. In the talk, I talked about inversional control container configuration. We talked a lot about sensible defaults.

Scott Hanselman: Sensible defaults?

Jeremy Miller: Sure. So, when I'm building a database, an object model that is persistent as a database, I can identify things in several different ways. I can create IDs. I can use GUIDs. I could use composite keys, I could use natural keys but instead, let's say that we start a sensible default that every single entity is simply identified by a numeric ID field.

Scott Hanselman: Okay.

Jeremy Miller: Or a grid.

Scott Hanselman: Sure.

Jeremy Miller: Now, I say that every single object, every domain entity is created this way. I can start to share policies about the way that it's persisted which also allows me to make some assumptions in my infrastructure, assuming that every entity can be treated the same way as far as locating it, saving it, validating it. That enables us to share and re-use a lot of infrastructure between things like CRUD screens and reporting screens and it's a great kind of opportunity to shrink down the size of your code base. Looking at it from another way, something you'll hear a lot from the Ruby guys, they talk about the idea of opinionated software and this kind of frustrates some

people but it's the idea that we can make things easier by taking away choice from the developers, that the system really expects things to be in a certain way and if you'll just do them this way, the infrastructure will let it flow easier.

Scott Hanselman: So you're going to increase productivity, if you can release a little bit of control.

Jeremy Miller: Exactly.

Scott Hanselman: Okay.

Jeremy Miller: Now the problem and the downside, I know I've heard from the MVC team the feedback on Rails is that the conventions are baked in and inflexible. I think the lesson that we've learned from it and I've seen in projects like Fluent NHibernate and FubuMVC is actually giving you the ability to create your own conventions and policies so that a team can customize the naming conventions for their database mappings or how controls are rendered on the screen, so you're not locked into whatever convention I happen to like.

Scott Hanselman: But if we take, if we take this idea forward five years, 10 years, where every piece of software is opinionated software, does that mean that there's even more syntax and more obscurity to the point where every single framework is its own little DSL that I'm going to have to understand? Because I've written Rails code and I always have found it difficult to be an expert because to be an expert is also to be an expert in trivia, maybe trivia is not the right word but in the tricks. Is that really the case in every system?

Jeremy Miller: Well, what you described is a definite downside to the technique but so taking the sample of all these DSLs, these all little languages I have to learn.

Scott Hanselman: DSLs are Domain Specific Language for the listeners.

Jeremy Miller: Yeah. I think that concern is a little bit of a red herring because every time I pick up a framework or a new tool, I have to learn a completely new kind of API which even though it's in C# or VB.NET. It may be speaking Greek, okay? The DSL idea is an attempt at creating a better API, an API that allows you to express the essence of what you are trying to do without jumping through quite so many XML angle bracket hoops to get there. Whether you succeed or not in that DSL depends on a lot of factors but that's the definite goal.

Scott Hanselman: I know that you've been experimenting a lot lately taking what would have been 10, 20, 30 lines of angle brackets inside of an XML file and turning into what they're calling a fluent interface which almost is an English sentence which



is method and objects strung together all into one long kind of line and you've done this recently, I think with both structure mapping and you've been using Fluent NHibernate, is that right?

Jeremy Miller: Both Fluent NHibernate and structure map -- maybe a quick example I can give you or I should say the purpose of what I'm trying to do is to gather things that today may be a lot of related pieces but they're spread out over the code base, big XML files. I'm trying to compress that content and the declaration of intent in a smaller area, I guess in the surface area. Take the example in WPF of creating keyboard shortcuts.

Scott Hanselman: Right.

Jeremy Miller: You have a XAML note somewhere that defines a bunch of commands and you give the command a key name, it's pointed to a class, who knows where it's at? At another place, maybe in the same XAML file or maybe not, you're creating an entirely different set of XAML nodes to define a key stroke, control one, control two calls this command by name in this thing.

Scott Hanselman: It's all kind of very normalized.

Jeremy Miller: Yeah, it's data-centric programming. We're coding directly to the low levels of WPF. Instead, coming up with a very small fluent interface where I can say these keys call this command and just pass in the command type and a generic. Behind the scenes it's going to a container and getting the object model. It's doing a lot of stuff behind the scenes. It's creating the same input binding, command object, all the stuff that was happening in the XAML before. The difference is you have a small place in the code where you can see what all of these things are and see how they connect directly to one another. Taking it farther, this small expression, I can be a keyboard shortcut, I can have it show up in a dynamically created menu and maybe even a button bar across the top that express one place in one small area of the code.

Scott Hanselman: Is this something that exists now or just a theory you've come up?

Jeremy Miller: This is in the Storyteller code base.

Scott Hanselman: Oh really? So in the Storyteller code base that I can poke around in subversion right now?

Jeremy Miller: That will be checked in by the time you dear listeners receive this podcast.

Scott Hanselman: Very cool. So that's cool because I actually did a podcast earlier with Ian Griffith about WPF and we were talking about that

stuff. So, now I've got a little thing there that I can use. Talk to me about what a fluent interface would look like. I know that we're doing this in audio so people can't see the code but this really is almost the sentence.

Jeremy Miller: Yes.

Scott Hanselman: Maybe explain like a structure map example like one line of code and walk me through what a typical fluent structure map conventional thing would look like.

Jeremy Miller: Sure.

Scott Hanselman: And what that would really expound out to if I have to do it in XML.

Jeremy Miller: So, actually, can I start from the XML first?

Scott Hanselman: Okay, whatever makes you happy.

Jeremy Miller: So, Scott inferred earlier that the XML configuration structure map early on was terrible. He said it politely.

Scott Hanselman: I said it politely.

Jeremy Miller: It was terrible.

Scott Hanselman: Okay.

Jeremy Miller: You would have to express that it's okay to pull types from this assembly. So there's a note that says "assembly this name," then I would say I have a plug-in family of classes that plug into this interface. There's another XML note. Then I might say this concrete class can be plugged into this interface.

Scott Hanselman: At this point you've already lost me.

Jeremy Miller: Yeah, yeah.

Scott Hanselman: Now we're four or five deep into an arrow-shaped XML file, right?

Jeremy Miller: Okay. Now that you're horrified, we'll move on to the FI approach. Today, to do that same kind of thing in the trunk of Storyteller, I would say...

Scott Hanselman: You said the FI approach, the fluent interface.

Jeremy Miller: I'm sorry, fluent interface. So in the fluent interface approach I would simply say if I have an IFoo interface and it has one class that I want to be plugged in it's the default IFoo, that's Foo,



classic Foo example. I would say for and use generic angle brackets.

Scott Hanselman: So the word for.

Jeremy Miller: For [IFoo ().use [Foo.

Scott Hanselman: For IFoo use Foo?

Jeremy Miller: Yes.

Scott Hanselman: And instead of spaces in your English there, you're using angle brackets and parentheses.

Jeremy Miller: Yes.

Scott Hanselman: But for all intents, it's an English sentence. It's a declaration.

Jeremy Miller: Yes.

Scott Hanselman: It's a classic, Martin Fowler designed fluent interface.

Jeremy Miller: Yes. Now do be careful. We have experimented and I was guilty with this, trying too hard to make it look like a sentence...

Scott Hanselman: Oh.

Jeremy Miller: So, it reads for requested type, IFoo the default.is.ofconcrete type Foo.

Scott Hanselman: It's interesting that you said that because talking about being polite versus being real. Some of the initial examples that I've seen in your blog when you were just kind of exploring that and others who have made similar interfaces, when I see for.is and I'm like, "Okay, come on. Is? Really? You have an Is object?"

Jeremy Miller: It's an expression interface.

Scott Hanselman: But you know what I'm saying?

Jeremy Miller: Yeah. We tried this experiment first, I would say that we have some internal DSLs that I can show to my product manager and he can actually say, "Yeah that's right, that's wrong..."

Scott Hanselman: And your non-technical, non-programmer product manager?

Jeremy Miller: Non-technical people.

Scott Hanselman: Okay.

Jeremy Miller: But we're developers, we can read code, instead I found, really adopting it from JQuery.

Scott Hanselman: Really.

Jeremy Miller: JQuery style code going for a much more terse infrastructure and that means I'm ditching the for and with and the and's and trying to make it as tight as possible.

Scott Hanselman: Right.

Jeremy Miller: So far, I think I'm liking that approach but maybe just because we use so much JQuery now.

Scott Hanselman: That is really interesting – it's a bit of a tangent but the idea that looking at JQuery and the way JQuery expresses itself, this is just somehow not really quantifiably but clearly such a comfortable list, comfortable to right thing that we would look to the JavaScript library that we used a lot and apply that to code that we're writing somewhere else in C#.

Jeremy Miller: Terse is good.

Scott Hanselman: Yeah, it's interesting because it's funny because JQuery still has that magic language, the selective language inside of its string where part of it's fluent until you get into a string and you pass stuff in. When you're creating a fluent interface for structure map you find times when you rather trying to string objects together, you'll really just end up with some tiny DSL on a string or have you not gone that far yet?

Jeremy Miller: So, that's a good, that's something to think about. I would say I have not done any kind of interpretation. There are some places I do that a lot for setting up test data.

Scott Hanselman: Yeah.

Jeremy Miller: There are other tricks besides method chaining. You could find all of this on Martin Fowler's website. There is a book, there's a section called the DSL work in progress and he talks a lot about a different design pattern for creating internal Domain Specific Languages.

Scott Hanselman: And this is the book that Martin has been writing for going on three years now and he's been doing it in progress. He's basically writing this book and putting it up his notes really.

Jeremy Miller: Yes.

Scott Hanselman: As he writes it and you can watch him write the book.

Jeremy Miller: But the point is there are other tricks, very useful trick besides method chaining. Method chaining gets kind of nasty to implement



when you get past a few things. You can use object initializers, much like the way the Ruby guys can idiomatically pass around dictionaries.

Scott Hanselman: Right.

Jeremy Miller: That's a way to kind of create more readable code.

Scott Hanselman: Do you think that -- this is a little bit of a tangent -- do you think that C#4 and the dynamic key word and the ability to not using IntelliSense and instead, figure out a method at runtime would make a full interface easier to write?

Jeremy Miller: Maybe easier to read and consume.

Scott Hanselman: Yeah, point now is you should write because you wouldn't have IntelliSense.

Jeremy Miller: But that's the point to me, this is tangent but at that point I think you just need to retry IronRuby or IronPython.

Scott Hanselman: Really? Rather than using -- C# has kind of attempted dynamism. Dynamism?

Jeremy Miller: Yes. If you're trying to create a DSL, an internal DSL, Ruby is going to be far more readable because there's less...

Scott Hanselman: That's really interesting.

Jeremy Miller: Stuff in the code.

Scott Hanselman: So as somebody who writes C# for a living, myself and yourself, it will be interesting to see if we start doing conventional reconfiguration and clone interfaces and DSLs to drive our C# applications if IronRuby in this case becomes the de facto way to do it. I know that Oren, that Ayende uses Boo as his kind of primary DSL and he's been pushing it for years. Has that caught on beyond just Oren himself?

Jeremy Miller: Maybe not so much in the .NET camp. I think it's much more common maybe in the Java camp where they've had alternative languages longer than us.

Scott Hanselman: Right.

Jeremy Miller: So, the same kind of things we see the Java guys doing have maybe the hard-core domain and anything that's performance intensive or infrastructure being in Java and the business rules and the glue code moving up into more dynamic languages like Ruby or JRuby where there's much more terse syntax and you can create more readable code. We may be doing the same kind of things.

Scott Hanselman: You and I were having an interesting discussion earlier about doing acceptance testing and you had mentioned a tool that a lot of Ruby guys use called Cucumber. We talked about fit and fitness and Uncle Bob Martin is here. We're going to figure out how to do this and a guy, one of my old bosses actually, had given me a demo of Cucumber and he showed how he could write what looked like English and that was the convention that he passed onto the suits that lived in his company and then he showed me the other side which were the tests. Then after he showed me kind of the wonderful English he'd written and that his suits worked on and the wonderful tests he'd written and I was like, "Wow, this is amazing. Wait a second. How does this fit together?" and then he like, "Well, here, let me open this text file full of regular expressions."

Jeremy Miller: Yeah.

Scott Hanselman: And then suddenly things stopped being so magical and started being kind of disgusting and he was like, "Well, yeah, we really haven't got an answer for that." This file uses regular expressions to read English and map it to these tests right here. Do conventional configuration systems necessarily have to have something disgusting in the middle or can it be very, very basic like the example I had said of if you name your GIFs this way, it will just work or if you name your interfaces that way, it will work? How often is there black magic like buried somewhere that we don't want to talk about I guess is my question.

Jeremy Miller: Well, I think it's all black magic.

Scott Hanselman: Really?

Jeremy Miller: It may be simpler...

Scott Hanselman: Not necessarily regular expressions.

Jeremy Miller: If you're going to build this yourself or I think for you to understand how these are usually implemented, there's usually something Fowler calls the semantic model directly underneath. Say we're working on Fluent NHibernate and we're building our own conventions. Say, for us we have a convention that determines the many to many table mapping names.

Scott Hanselman: Okay.

Jeremy Miller: Or many to many relationships.

Scott Hanselman: Right.

Jeremy Miller: It's very important you always want the names the exact same way. Under the hood, it's altering a property on a many to many table map property. There's actually a class.



Scott Hanselman: Can you give me a concrete example?

Jeremy Miller: It's called the many to many table map I believe.

Scott Hanselman: Oh, all right, okay.

Jeremy Miller: Let's say our convention is if you're relating one class to another class, you take the two class names, you make sure that you put it in alphabetical order so that the name would be identical going from A to B or B to A...

Scott Hanselman: Okay.

Jeremy Miller: And that first name to second name is how we determine our many to many table mapping. That's setting a property of a class inside of Fluent NHibernate and at the very last minute, this semantic model, this whole object graph of how things are mapped from class to table property to column name. Then at the last minute is used to configure NHibernate itself internally. If you ever decide to build one of these things yourself, you build a semantic model that actually performs the runtime actually performs the run time activity. You test that very well, you layer conventions or DSL on top of this semantic model. Your conventions are altering this runtime object graph or your DSL is riding to the semantic model.

Scott Hanselman: Okay. This might be stretch but because I'm doing multiple podcasts in a day here, I'm starting to attach things that I've talked to on previous shows. We talked to Ian about why I was running into problems with my application I had my view and I thought I had a view model but it was really just a bunch of sloppy code that was talking to my view. I needed multiple layers and I really just had one big ball of a mess. I didn't really have a view model. You're describing a semantic model. When you're saying semantic models, I started thinking about view models versus models. You're saying to really have that layer that is that object graph that shows your intent effectively and separating that out from where that intent was sourced from, where it came from, it came out of configuration or it came out of a fluent model or it came out of convention.

Jeremy Miller: Yes.

Scott Hanselman: It's just interesting that all these problems in computer science can be solved by one additional layer of abstraction whether it be adding a view model to a previously cluttered view or adding a semantic model to your semantics.

Jeremy Miller: That's absolutely true. If you want to think of it this way, the semantic model you can build any way that's easiest for you to make it

work. In WPF, WPF we don't really have much of a separation. What you see in XAML is pretty well a serialized copy of the raw semantic model, the controls, the tree. The DSL or conventions, it's just putting icing on the cake or if you have to say this, lipstick on a pig. It's trying to create an API that's friendly for the user where I can be very concerned about what's best for the user that just simply helps configure the things that work. It's isolating the user from the mechanics so that I can build the mechanics in a way that's easy for me as a developer and still provide a good experience for the developers using it.

Scott Hanselman: That might be the essence of why this isn't obvious to the average Joe; whether using WPF as the example or using the system in C# because there's nothing in the tooling, there's nothing in the language that enforces this. There's nothing in WPF when I go file a WPF application that forces me to have a view model and even now in Fluent NHibernate and in Structure Map I'm sure I could probably shoot myself in the foot if I wasn't using these fluent interfaces correctly, right?

Jeremy Miller: Sure.

Scott Hanselman: I mean they don't prevent me from expressing myself incorrectly.

Jeremy Miller: That's absolutely right. If I do my job well as an API designer and things are consistent and I fail fast, I have to try to detect when you do something wrong and fail fast and tell you where and why you did something wrong and I have to be consistent all the way through. That was a part of another one of my talks. I did a lessons learned talk from structure map development and I outlined where I've gotten in trouble in my structure map API from inconsistencies or making a developer do what seems like the exact same task. In structure map configuring, you try to override constructor arguments explicitly. You do strings and numbers and primitive types one way but for dependencies, non-primitive types, you do it another way and that confuses structure map users. It's something I get in trouble with right now. I know as a developer this tool that structure map handles primitives and non-primitives differently, so I made the API reflect that but the user shouldn't have to know that. I should make the API smart enough.

Scott Hanselman: You ask too much of them.

Jeremy Miller: Exactly, that's the kind of mistake you can make with these DSL APIs.

Scott Hanselman: Oh, okay, Interesting. Well cool. Thank you so much for sitting down with me today and explaining this stuff to me and I suppose that people are going to be able to see your talk online at some point. So, I'll put links up to the NDC,



the Norwegian Developers Conference talk as well as your Blog and you're also on Twitter.

Jeremy Miller: Yes.

Scott Hanselman: All right. Well this has been another episode of Hanselminutes, this week talking with Jeremy Miller and I'll see you again next week.