



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 #96

January 11, 2008

### Starting Small with F# with Dustin Campbell

Scott is at CodeMash in Ohio this week chatting with CodeRush/Refactor developer Dustin Campbell about his recent obsession with F#. Is it a functional language and object-oriented language or an imperative language? Why should you care?

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



*Our Sponsors*

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

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

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





**Lawrence Ryan:** From [hanselminutes.com](http://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 #96, recorded live Friday, January 11, 2008. 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](http://www.telerik.com). In this episode, Scott talks F# with Dev Express developer Dustin Campbell.

**Scott Hanselman:** Hi, this is Scott Hanselman and this is another episode of Hanselminutes. I'm sitting here in the hallowed halls of the Kalahari Resort in Sandusky, Ohio, halfway between Toledo and Cleveland. I'm sitting here in the middle of the CodeMash conference with Dustin Campbell, famed developer of Dev Express and arch nemesis of Mark Miller.

**Scott Hanselman:** How are you, Dustin?

**Dustin Campbell:** I'm doing great, man.

**Scott Hanselman:** Yesterday, you gave a fantastic talk on F#. I thought that was really interesting.

**Dustin Campbell:** Thanks, man. I really appreciate that.

**Scott Hanselman:** One of the things that really struck me about your talk was that you said this is an immense language and you said that at least three times. You had this diagram up on the screen. I know that we're in a podcast so I can't show people the diagram, maybe I'll blog about it, but explain to me these different modalities that F# encompasses.

**Dustin Campbell:** Well, F#, where some programming paradigms can be -- you can work with different programming paradigms in different languages, F# actually fully embraces three, so functional programming, object-oriented programming and imperative. All right? Now, what goes along with that is a massive number of features. So, you've got your functional programming features, which are fully in there like automatic generalization and extreme type inference; but then you've got object-oriented features that some of these are unique to F#. On top of that, things like compiling to Mono and standalone executables and all sorts of things are just like, there are a lot of features in this language. It's a huge, huge thing.

**Scott Hanselman:** You keep using really enthusiastic adjectives. You keep using things like immense and extreme. You said extreme type inference. That seems like the X Games. What is an example of an extreme type inference? I mean you're pulling information from as much context as you can

from other places. Where can you pull this context from?

**Dustin Campbell:** Well, for example, the operators that are being used, functions that are being used within the body of another function, those might have some type indentations, but the F# compiler can also say, "You know what? I'm not sure what these types are, but I can make them generic. I know that I can make these generic types for you." So, you might declare a function and it might not actually say, "Hey, this primary is going to be an int." It's going to say, "You know what? This is going to be generic type parameter A," that sort of thing.

**Scott Hanselman:** Now, when you say generic, I mean think generic, I think angle bracket generics like listed int.

**Dustin Campbell:** Sure.

**Scott Hanselman:** Is that the kind of generic that we're talking about?

**Dustin Campbell:** Exactly that kind of generic, but at a function level where your function is actually declaring generics into itself that some of its parameters will have generic type parameters.

**Scott Hanselman:** Ah, so it's saying that I don't know enough that this is an int or long, but I know that it's going to be some kind of a type so I'll just loosen up a little bit and allow these types to go through.

**Dustin Campbell:** Exactly.

**Scott Hanselman:** Now, can you provide it with ints?

**Dustin Campbell:** Yes. You can add your own type annotations if you need to and sometimes when you're writing library code if you're writing a public API that might be nice. Maybe if you're giving out the source code and people want to look at this, they can see what the types are, but generally as I'm writing code, I can write it very succinctly, not use types, and maybe like 97% of the time, it just gives what I want.

**Scott Hanselman:** Is this a dynamic language? I mean people think of dynamic languages as moving types in a very liquid form from here to there.

**Dustin Campbell:** No. It's not a dynamic language. It's very strongly, very statically typed, but the type inference allows you to write in a way that feels like that kind of fluid dynamic language. If I declare a variable and the compiler infers the type for that variable, I can't now go and set that variable to a value of a different type.



**Scott Hanselman:** You can't change it to a value of another type, but I go and say, "Let i=0." I can't go "let i=string."

**Dustin Campbell:** Well, yes and no.

**Scott Hanselman:** Or I can't change i at all.

**Dustin Campbell:** You can't change i but there are scoping rules within F#, so you can actually re-declare i, but that's actually a new i. So, I'm not actually changing the original variable i to a new type; I'm declaring a new variable i. If you had a function, say, that work with i, if you had a function that consumed the original i, that function would still continue to use the original i with the original type and if you re-declare i to a new type, that i now exists for the rest of the scope.

**Scott Hanselman:** Okay. So, at this point, we've lost 90% percent of our listeners.

**Dustin Campbell:** I agree.

**Scott Hanselman:** Let me see if I can understand because I'm struggling to understand this as much as I'm sure some of the listeners are. So, if I go in and I say "let," this is like what I would think of as dim, I mean I'm basically just making a declaration.

**Dustin Campbell:** Sure.

**Scott Hanselman:** "Let i=0," and I'm saying that there is a name now that is bound to the value 0 such that when I refer to i later, I'm really referring to 0 but I'm referring to it with a friendly name.

**Dustin Campbell:** Yes.

**Scott Hanselman:** Did I get that right? Now I can't say i=2 later.

**Dustin Campbell:** No.

**Scott Hanselman:** That seems like a very big mental change. I mean you're saying that there's really no variable. It feels like there's only constants.

**Dustin Campbell:** They're constant. In fact, it's just that they're variables, but they're immutable. Okay? So, they can't be changed unless you declare that variable differently. For example, you could say "let mutable i=0," and now you could then use an assignment operator to change that i. So, it is a variable. It's just by default in F#, all variables are immutable. They cannot change.

**Scott Hanselman:** Okay, and that sounds like a hassle.

**Dustin Campbell:** It is kind of a hassle if you want to program in that, if you're used to that old imperative

style of programming that we've been indoctrinated with since Fortran.

**Scott Hanselman:** Right.

**Dustin Campbell:** If that's what makes you comfortable, then it will feel a bit of a hassle at first, but once you get used to it, once you get used to the idioms of a more functional style of programming, it becomes very natural.

**Scott Hanselman:** There's an interesting contrast here. You're pointing out very explicitly that there is the imperative style of programming that you refer to as the Fortran style and then there is the functional style, which is I assume from the Lisp world. I think there are a lot of people out there, a lot of people of my generation or younger who just never learned Lisp in college. They skipped that part because imperative programming has been so fundamental to everything that we've done. Can you talk a little bit about those differences? What really is the difference between functional programming and imperative programming?

**Dustin Campbell:** Well, you made a good point that it really comes from the Lisp. Really, if you think about it, the two oldest programming languages in widespread use, which would be Fortran and Lisp, are really kind of the mother languages, Lisp being the mother language of functional programming, Fortran being the mother language of what became C and then C-based languages like Java and C# and all of our imperative languages. What makes these languages different is that Lisp, while it was trying to solve math problems, it would start from the mathematical symbols. That's where it started and then it moved downwards and added abstractions below it, whereas, Fortran started with numbers and it grew upwards and added abstractions on top. That's what we live with today. What really makes these -- they're kind of buzz words - I hate using in one sense - but the words that really describe what these are doing is that Lisp is more of a declarative style or a functional programming language away from Lisp. So, it's just a functional programming language that generally allows you to write things in a more declarative way saying "this is what I want done" as opposed to an imperative style, which is more like a recipe like you're a chef and you got a list of instructions that you want to follow step by step and say, "This is how I want this done. This is exactly what I want the process to do, the steps that I want you to go through to do it," and in a functional programming language you do the opposite and say, "This is what I want you to do and I really don't care how it happens."

**Scott Hanselman:** Okay. That's really a good description. I'm going to paraphrase that because you've blown my mind there. So, a functional programming environment is where you're basically



saying, "Here's is the state of affairs as I know them initially and here's a description of what my ultimate goal is. Go ahead and get on that. Work that out. Start. I'm not concerned if it's recursion or if it's a big old loop or whatever. Just make it happen and call me when it is done."

**Dustin Campbell:** Yes.

**Scott Hanselman:** And then in the imperative format, we start out with some beginning state as we did before, but then we say very, very explicitly all of the state changes that we're going to go through in order to get something done and it's completely up to us to decide how that gets done and sometimes that how is outside of the scope of the problem itself and we may not know the best way to do that. You had a recursive function add that took x and y and then just added one and returned and you wrote it in C# and then you try to add like 38,000, 1+1+1 and you recursed and you blew the stack.

**Dustin Campbell:** Very nasty exception to have happen.

**Scott Hanselman:** And not only did you blow the stack, but you took a long time to do it because the stack got really big. There was a huge 37,000 deep stack, but when you did it in F#, you wrote basically what I perceived to be the same recursion but it happened instantly and I didn't blow the stack.

**Dustin Campbell:** Yes.

**Scott Hanselman:** Was there no stack? Is there no spoon?

**Dustin Campbell:** The way F# compiler took a look at that bit of recursive code, it could see that the very last thing that was done by the function was to call itself and it's called tail recursion. That's what the tail means. It means it's the very last thing that happens. It took that and it was able to say, "Well, if all I'm gonna do is go back to the front of this function again, then all that I need to do is turn this into a *while loop*." That's what happened in the talk, in the session. We brought up Reflector and we took a look at the C# code compiled exactly as I'd written it, you know, when we looked at it with Reflector and then we looked at the F# code using the C# decompiler and we saw that *while loop*. So, it really happened almost instantaneously because it was just increments and decrements within a *while loop*.

**Scott Hanselman:** Now, is that a feature of the language or a feature of the compiler? I mean the C# compiler is getting very, very smart. There's a lot more method in lining. There is some tail call optimization in the jitter of the CLR, but apparently not enough to have spotted that issue and fixed it.

**Dustin Campbell:** Sure.

**Scott Hanselman:** So, is this a feature of the language in the sense that that kind of optimization has been brought up into the language spec and said that this is something that will handle, this is an optimization that's appropriate, or is it just that the F# compiler is just so smart?

**Dustin Campbell:** Well, it is more of a language thing because functional programming in the F# language is so built upon recursion. It's such a natural fixture, an idiom that's used that it really is part of the language. So, even if you do things that you would think this isn't a tail recursive situation where you do *corecursion*, it was called corecursion, where you have one function that calls another function and that function calls the first function and back and forth to solve a problem. F# will still handle that using IL instructions appropriate. It will manage to get that so that it won't stack overflow. So, it's really part of the language. It's trying very hard. You know, you can get there. You can get there and get a stack overflow, but if you follow these functional programming idioms of tail recursion, that sort of thing, F# will pick that up for you and allow you to do these kinds of things without overflows.

**Scott Hanselman:** Now, often when I'm writing C# code or what I would think of as classic imperative .NET languages, I find myself thinking about how things are going to look at runtime. I find myself thinking, "Well, here's where I'll be in the stack at this point," and maybe recursion is appropriate here, but I don't think recursion is something that a -- unless you're calling fine control, I don't think that Joe, a .NET developer, necessarily pulls recursion out of his pocket every five minutes, but it seems like with a functional programming language like F# where you can't change a variable, then you would need to use recursion because you had to put stuff somewhere.

**Dustin Campbell:** Yes. So, you need to modify that variable. For example, with add, the way it works, we have two parameters, x and y, and when we recurse, we call add again decrementing x and incrementing y. So, we call it with one less and one more on each parameter and by the time when x=0, then we know that y is the final result and that's how recursion works. You're going to take an accumulated value at the end. When things are immutable like that, you're exactly right. So, in functional programming and in F#, one of the structures that's used most often is a list, singly linked list, and that's a recursive data structure and it has to be dealt with using recursion. When you operate on it, you have to use recursion to build up new list to operate on a list. You manipulate your manipulations on the original list, produce a new list, and you build that up through recursion using the same kind of technique we have of parameter that's going to accumulate the result.



**Scott Hanselman:** That's interesting. So a list is not a `System.Collections.List`. This is an F#-specific thing. It's not a list that I iterate over, I'm recursing over?

**Dustin Campbell:** You can recurse over, yes.

**Scott Hanselman:** I guess what I'm saying is kind of like for loop over these lists?

**Dustin Campbell:** F# provides some structures that look like that, but that's not what's happening because an F# list, it's a beautiful structure really because it's a recursive structure where a list is simply just a head and tail. It's like a value and then the rest of the list, but that rest of the list is another list with a value and the rest of the list and so on all the way down to the very end and there's a little marker called the empty list that ends it.

**Scott Hanselman:** See, this is interesting. I keep thinking about computer science class because you're talking about things that -- I mean, really, honestly, when was the last time a C# programmer who wasn't in academia had to whip out a singly linked list?

**Dustin Campbell:** Yeah.

**Scott Hanselman:** I mean it happens and I'm sure that I'll get email from listeners that will say like, "Hey, I do singly linked lists all the time," but when you look in `System.Collections`, that wasn't the first thing that doubly linked list and singly linked lists were not the first things that leap out of the collections. It's mostly just "here's a chunk of memory and here's how we're gonna think about it."

**Dustin Campbell:** Sure, but today in programming in C# 3.0, we don't use a singly linked list per se. We use `IEnumerable`, which is essentially the same kind of thing.

**Scott Hanselman:** That's a very good point, yeah.

**Dustin Campbell:** It's iteration structure that is used in LINQ in a very similar way to the way that an F# list will be used.

**Scott Hanselman:** That's interesting. I didn't think about that because my next question was going to be, has F# come out of nowhere because it included nonfunctional paradigms because at the beginning of the talk here, I was asking you about how you threw up this diagram that said functional and object-oriented and imperative and then you drew this kind of circle around them all and say, "Well, F# really encompasses all of these things." One of the things that I personally feel has made functional programming languages less applicable to me in my life was that the problems I solve didn't feel like classical functional problems, but this may be one of the most accessible languages for the .NET

developer because I can do things in the old way, but put my toe in the pool I guess if I'm not quite ready to jump in *cannonball-style*. I want to just feel around and I could move from paradigm to paradigm. I wonder are there other languages that are that inclusive, that broad-reaching in their style that they would let you move comfortably from OO to functional to imperative?

**Dustin Campbell:** There are others, but I'm not finding them. I'm really not. F# for me is the one that's really spoken to me. It really makes this the most clear especially with functional programming. If you go to, you know, for example, if you want to try and learn functional programming, you go to a language like Haskell.

**Scott Hanselman:** Right. That's what I learned in college.

**Dustin Campbell:** It's a very different thing, right?

**Scott Hanselman:** Yeah.

**Dustin Campbell:** You know. It's a very different thing.

**Scott Hanselman:** Oh yeah. It was weird.

**Dustin Campbell:** And there are certain things that are forced upon you, whereas, in F# like you said earlier, yes, your variables are immutable by default, but it's fine, go ahead and declare immutable value as well because F# does embrace imperative programming. It allows you to program in imperative styles as well if that's what's comfortable to you. Hopefully, as you get more comfortable with the language, you'll do that less but you won't fall back so much on that kind of older style of thinking that it's more comfortable to you.

**Scott Hanselman:** Yeah. It seems a very inclusive language, but, you know, we had Robert Pickering on the show who wrote an F# book a couple of months ago and he pointed out that one of the things that it was attractive to him that it allows you to have significant whitespace...

**Dustin Campbell:** Yes.

**Scott Hanselman:** Or not...

**Dustin Campbell:** Yes.

**Scott Hanselman:** And that "or not" part was really very interesting because it's unusual for me I would think that a language could provide a choice like that. Languages are always so dogmatic, you know.

**Dustin Campbell:** Sure.



**Scott Hanselman:** "Here's how we do things in Scott# and if you don't like that, well, *fooi*e on you," but here's a language that said, "Here's a religious issue..." It's almost like "don't eat pork, but if you do, here are some wonderful recipes."

**Dustin Campbell:** It's that way in object-oriented programming too with F# I'm finding as I do OOP-style code in F#, I'm writing design patterns. I'm coding them in very different ways than I would have in a language like Java or C#. For example, you just made a really good example of giving you the options there, but not saying "here's what you have to do." F# supports implementation inheritance, right? But implementation inheritance has some baggage and overhead associated with it. When you descend from a class, you're essentially making a more complex object a lot of times. So, implementation inheritance can be the cause of some problems in object models. Models become brittle and that sort of thing. So, F# allows you to do implementation inheritance, but it downplays it. It gives you another language feature object called object expressions, which allows you take, say, an abstract class or an interface and just instantiate it giving it its members, giving it the implementations that you want at the time that you instantiate it. So, instead of actually having to go out and declare something and declare some descendant or implementer of something, you can just construct it right there.

**Scott Hanselman:** So, this is interesting because it's not a language that is looking down on you if you try to do something. It's saying that you can do this and rather than making you feel guilty about it, they simply provide an alternative method that if you find more attractive we would encourage you to use.

**Dustin Campbell:** Exactly.

**Scott Hanselman:** So, I think it's a very friendly language.

**Dustin Campbell:** It's a very friendly language, but also it's a research language. So, when I said it was immense, part of that is that it just keeps getting more features that some of which are very experimental. So, it's a very friendly language in the one sense, in the other sense it's very large. It seems like if you look at all the features in a big list, it would be very daunting. So, my talk earlier was about trying to start smaller with it and build from there.

**Scott Hanselman:** It's a good topic, starting small with F#.

**Dustin Campbell:** Yeah.

**Scott Hanselman:** That's our topic today. Now, this is interesting. When you say it's immense, are you saying it is immense in its feature set or is it immense in the sense that it's going to use a lot of

those if I hold down Shift and start pushing funky, swear word-looking characters at the top of the numbers row of my keyboard that it's going to use more of those in a very terse and uncomfortable way?

**Dustin Campbell:** No, it doesn't. It's actually very succinct and part of that is the type inference. Part of that is just flexible syntax, but it's immense in its feature set. There are things that I haven't touched yet. I just haven't had the time. I'm spending time learning how to do certain things with the language, what solved my problem domains at the time, and I don't have time to look at these other giant piece of the language over here. For example, like computation expressions. It's something that I've not had time to look at. If there's a Haskell listener, that means that we're talking monads here in F#. I haven't had an opportunity to look at that and it's just immense.

**Scott Hanselman:** One of the things that I have trouble with when I'm learning a new language whether it be a linguistic language, a human language, or a programming language is I can get the job done at some point, but I don't know if I said it correctly. Someone said to me once that if you have a friend who says, "You know, I speak 12 languages and I speak them really well. Oh yeah, you know, French, I totally speak French," and he said, "Really? You speak French?" you should ask them how to say "flush the toilet" and if they maybe hung out in France for six weeks in a backpacking trip, they might say something in French like the toilet bowl with the water pool or whatever. They might say something that kind of sounds like and explains the process of toilet flushing, but they don't know the idiomatic specific speech. Apparently, in France, it's like "pull the chain" is the idiomatic speech that means "flush the toilet." If you ask any language, they all have a different way of expressing that, but you couldn't know that unless you lived there and worked. So, people will dance around the right way to do that. So, when I try to learn a new language like F#, I can express that bowl with the water and then it goes away, I get my point across, but I don't know if I did it right.

**Dustin Campbell:** Right.

**Scott Hanselman:** How do I know in F# that it -- I mean the fact that it worked isn't that it was the correct...

**Dustin Campbell:** Was it the most elegant way to do it? Was it the most idiomatic way to do it? Is that what you...?

**Scott Hanselman:** Yeah, I think so and then with the rise of Ruby and this notion of the programmer aesthetic and the language aesthetic, I feel more insecure about my code than ever before. I mean that's a really uncomfortable thing, I'll say it again. I feel insecure about my code more now than before



because there seems to be an unwieldy amount of peer pressure on behalf of the community no matter what language they have about, well, you know, "Here's a better way that you could have done that." Someone will write something and then another guy comes out with a C# LINQ expression that was light eight characters and it's unbelievably elegant. It doesn't feel like a really great way to learn a language and certainly not a very good way to feel good about oneself.

**Dustin Campbell:** Sure.

**Scott Hanselman:** How do you know when you're learning F# -- because I would say you're relatively new to this language, right?

**Dustin Campbell:** Yeah.

**Scott Hanselman:** How do you know that you got it right? That this was the right way, the beautiful way? Do you ask around? Do you look at examples? How can a listener jump into this language? Because so often you hear people say, "Yeah, I've been poking around with F#." I've been poking around with this language." They make it a chapter into a book, but they don't really drink deeply of the language and learn how to say "flush the toilet."

**Dustin Campbell:** It's a budding community around F# will would help, that you can go and you can ask questions and you can see examples of ways to do things. I'll go to the FS Hub, which is a community server that was set up with forum on it that people ask questions and someone will post a bit of code and say, "Hey, what's the best way to do this?" and then you'll get four different responses with different styles and different ways to do things. It's a friendly community, which is fun, and then eventually you'll see even one of the implementers of the language will come on and say, "Oh. No. Here's how. I can do this really succinctly." That's a lot of fun. It's almost like there's an engagement of people to write elegant, beautiful code with F# and they're willing to help each other do it. Another resource is that the F# Source actually ships with the language. So, you can go down to libraries, you can look at elegant ways to do things to see how the language is built up. You can look at how that F# list is implemented and see that it is in essence written in three lines of code, for example. There's a lot of beauty within the libraries themselves.

**Scott Hanselman:** It's an interesting perspective. I didn't realize that since those libraries are shipped, they would be the canonical example of a good way to do something.

**Dustin Campbell:** Sure.

**Scott Hanselman:** Fantastic! Well, I really appreciate you taking the time to sit down with me. I

know that you are missing a session right now and this was really interesting stuff. Thanks. I think that F# is going to be kind of the new .NET language. I understand it's going to be in Visual Studio. It's supported in a more fundamental way. Are we going to see refactoring tools for F#?

**Dustin Campbell:** Perhaps. That's kind of a personal goal of mine this year. I don't think I can make any announcements about things, but know that we are looking very closely at F# support for CodeRush and Refactor!, from Dev Express, so maybe on the horizon you might see something there.

**Scott Hanselman:** Fantastic! A new language on the horizon, F#. Thank you very much, Dustin, for sitting down with me here at CodeMash. This has been another episode of Hanselminutes and I'll see you again next week.