



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 # 24

July 12, 2006

Windows PowerShell (MONAD) Part II

Scott brings us up to date on PowerShell, formerly named the Windows Command Shell, code named MONAD. Learn about the new features and some great applications of the technology.

(Transcription services provided by [PWOP Productions](http://www.pwop.com))



Our Sponsors



<http://www.codesmithtools.com/>



XFEED
MULTI-TALENTED COMPONENTS



<http://www.peterblum.com>



(Music)

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 #24 recorded Monday, July 10th, 2006. Support for Hanselminutes is provided by CodeSmith Tools, makers of CodeSmith – an extensible template-based Code Generator for .Net. And now, Hanselminutes listeners get \$100 off CodeSmith professional with Coupon Code HM100, online at codesmithtools.com. Support is also provided by Xceedzip for .NET, which lets you handle Zip, Tar and Gzip files on FTP servers, in streams, in memory and more. Now get 20% off any Xceed component or a Suite just for listening to Hanselminutes, just go to shrinkster.com/fpt and use the code HM-20-20. And don't forget to visit Peterblum.com – "Start with better controls, finish with better sites", online at peterblum.com and .NET Developer's Journal – "The World's leading .NET developer magazine", online at www.sys-con.com. In this episode, Scott discusses advanced capabilities of Microsoft's PowerShell.

Carl Franklin: Hi, this is Carl Franklin. Welcome back to Hanselminutes. Of course, we are talking with Scott Hanselman as we always do. Scott, what's on the roster for today?

Scott Hanselman: Well, it's been a while since we talked about PowerShell, which was originally called the Monad, Microsoft Command Shell.

Carl Franklin: Right.

Scott Hanselman: And I got a number of letters from listeners saying they want to hear about some more PowerShell stuff and if you read my blog, you know that I have been messing around with lot of PowerShell. So, let's call this advanced PowerShell.

Carl Franklin: Such as it is. Okay, great.

Scott Hanselman: And have you had a chance to dig into this at all?

Carl Franklin: PowerShell, no. I have read the blogs, I have seen what it can do but I haven't done it myself. I haven't used it yet.

Scott Hanselman: PowerShell for me, I have been thinking more and more about how to describe this. What's the elevator speed when you try to explain to someone, "What's PowerShell?"

Carl Franklin: Yeah.

Scott Hanselman: And it's a .NET command line that lets you put Windows on a string.

Carl Franklin: Sweet.

Scott Hanselman: Anything that you can do by right-clicking in MMC and saying 'File New', whatever, you can do with PowerShell.

Carl Franklin: And when you script it, you can pass object references, right?

Scott Hanselman: Exactly. You pass objects from place to place so you can do all the same stuff you used to, like DIR, Pipe More but when you say DIR- you are not really getting a listing of Directory, listing of the directory as TEXT; you are getting back some Directory info objects, some file info objects.

Carl Franklin: Right.

Scott Hanselman: You are really looking at arrays of objects, which then lets you use reflections to manipulate these things.

Carl Franklin: Sweet.

Scott Hanselman: It's a fantastic command shell and there's a very vibrant community that's being built up around this, for system administrators. Also enthusiasts and developers like myself, I am not an admin. or an IT guy, but you know, I used to write a lot of small programs to test stuff out, maybe want to check to see what 'date,time.2 string' does or whatever. And more and more, I have been doing that at the command line. It's such an amazing experience to build load and assembly from your command line and start calling methods on it, directly.

Carl Franklin: That's very cool.

Scott Hanselman: Now, there's a whole series of lists and resources that I could potentially give you and I'll give a lot but there is a fantastic list of resources at del.icio.us/powershell which is at shrinkster.com/gil. del.icio.us, of course, is the social Book-Marking system online.

Carl Franklin: Automatic cream rising-to-the top-machine.

(Laughing).

Scott Hanselman: Exactly and the user, whose user name is PowerShell, has set up a series of links and then there are also all of the del.icio.us



links that are tagged as PowerShell which is at shrinkster/gin so that's del.icio.us/powershell or del.icio.us/tag/powershell. Once you started poking around in there, you see the popular tags, you see basically everything-- the depths of all the things that are being talked about around PowerShell.

Carl Franklin: Right.

Scott Hanselman: Now, because PowerShell used to be called MONAD or the Microsoft Command Shell- a number of things are tagged as MSH because MSH was what we used to call a PowerShell.

Carl Franklin: Right.

Scott Hanselman: So, you might also take a look at the del.icio.us things that are tagged MSH so that's at shrinkster/gir.

Carl Franklin: Okay.

Scott Hanselman: Between the three of those things, you have pretty much got the full spectrum of what's being talked about within the PowerShell universe. Now there is another -- number of other places where you can get introductions and things about PowerShell and we talked about...

Carl Franklin: Including your show. I mean, we did an introductory show on PowerShell.

Scott Hanselman: Yeah and we won't redo that show here. So, we are going to think this as advanced PowerShell.

Carl Franklin: This is Part Two, yeah.

Scott Hanselman: Yeah, Part 2. So, might take a look at that last show; it was a nice quick update on what was going on in PowerShell and kind of the gestalt behind it. But we are going to talk about a little bit more advanced stuff. So, if you want the basics, take a look at our other show- there is also good site at Reskit, which is at shrinkster.com/giv. They give you some details on what's available. But things that I am interested in PowerShell are-- extending it right, making it dance.

Carl Franklin: Right.

Scott Hanselman: So there is a thing called a CMDLET, 'C-M-D-L-E-T' and CMDLETS can be written in any .NET language so you would sit down in C#-- you would make a new class library and then take either existing code that you have

or write stuff from scratch using the full power of the .NET framework and write out a new command within PowerShell.

Carl Franklin: Okay.

Scott Hanselman: These commands might take input that's been piped in or input from the command line; its totally up to you. There is a really great screen cast at shrinkster/gix and this is a video of how to make a CMDLET. There is lots of great sample code that comes with the documentation pack for PowerShell and I don't -- I personally haven't had to write a CMDLET, I kept saying to myself, "Well I am going to really get right into .NET and do it" but more and more I am finding myself writing scripts because you really have the full power of the .NET framework, you don't have IntelliSense, right out of the bed, on the command line but .NET is sitting right there. You can do anything that you want to. But a lot of people have existing libraries that they want to call or they like that the sense of packaging up all of their stuff into one single CMDLET and so, take a look at that screen cast.

Carl Franklin: Okay.

Scott Hanselman: Now, when you create a CMDLET, you package up your existing functionality or new functionality into a DLL, you can run a thing called Make-Shell, 'make-shell' and you can make your own custom PowerShell.

Carl Franklin: Are you talking about references, with references loaded up or...?

Scott Hanselman: Right, with references loaded up, environment loaded up, colors and everything-- to customize.

Carl Franklin: Environment, yeah.

Scott Hanselman: So you might have -- if you are a System Administrator for a very large company, you might want to have one for managing exchange mailboxes and one for managing FTP and have all of your different Shells set up.

Carl Franklin: I see.

Scott Hanselman: You might want to set up a custom one just for the work that you do with PWOP on -- you know, you might have BitTorrent things in the command line, you might have FTP stuff set up, environment variables passed, existing scripts prepared-- so that you have multiple shells.



Carl Franklin: Yeah.

Scott Hanselman: Right, you have two shells now, Carl. Right, you have got...

Carl Franklin: Right.

Scott Hanselman: Command.exe and you have got VSWars right, when you got on to Visual Studio command prompt.

Carl Franklin: Right, because it has different path statements in the environment, sure.

Scott Hanselman: Exactly. So, extending that how many different kind of context Windows would one need? So, we have a big application at my company called Voyager that does online finance. Why wouldn't we have a Corillian specific PowerShell? Then we can just ship that shell to folks; they put PowerShell on their system.

Carl Franklin: Hmm...

Scott Hanselman: They get, you know, Corillian.voyager.shell and everything that they need is there; available to them.

Carl Franklin: Okay.

Scott Hanselman: So, there is definitely some power there, within creating your own CMDLET and then creating your own shell.

Carl Franklin: That's pretty cool; I got to tell you.

Scott Hanselman: Yeah, it gets even cooler. So, you can get lots and lots of good information at the Microsoft PowerShell newsgroup and this is just at Microsoft/communities up at shrinkster/giz but the thing, here is the mind blowing thing, its not just about the shell, its about the engine-- the fact that it decided to display itself as a text on a black background, really just a fascia on top of this underlying engine.

Carl Franklin: Right.

Scott Hanselman: What if I have an application that I want to enable for scripting. I have got an application, it's got an object model, it's got things that people might want to extend. I would like them to write just a little bit of script to manipulate the objects that are living inside of my application.

Carl Franklin: Sure.

Scott Hanselman: So, Lee Holmes has got a really great blog post at shrinkster/giz and do you remember, Logo? Did you do this in school...this is the little Turtle?

Carl Franklin: Yeah, Turtle; basic kind of stuff, yeah.

Scott Hanselman: Turn right, turn left and then you start learning about looping and the next thing you know, you have written and drawn a flower...

Carl Franklin: Right.

Scott Hanselman: It's some sort of a recursive algorithm. So he wrote Logo in C# and it is pretty straightforward. Its just a little WinForms application, he has got a little object called "Turtle" okay and Turtle has got a number of public methods like reset, turn left, turn right, things like that-- seems pretty straight forward. But how would he make it available to the user to script? So, he makes a text box where people can write in PowerShell script and then he hosts what's called a 'Runspace.' That Runspace is the PowerShell engine, okay, so at this point PowerShell hasn't yet come together, it hasn't come into the application. He has got a nice, graphical area to draw the Turtle, he has got a text box-- they write into the text box, may be they say, 'turtle.left 10.' Okay.

Carl Franklin: Yeah.

Scott Hanselman: Then in the run button on his click, he says, 'Click, do it', he makes a new Runspace and then he says to the Runspace, 'There is this object called Turtle,' and he hands it into it. So, you are creating an instance of PowerShell that Runspace, handing the Turtle object into it and then you say, 'Create pipeline' because a pipeline is really a series of commands so whatever they wrote in their text box-- that script is a pipeline of command after command after command.

Carl Franklin: Cool!

Scott Hanselman: And then he says, 'Pipeline.invoke', and draws what the Turtle did.

Carl Franklin: Very cool.

Scott Hanselman: So, think of how you could use this in existing applications.

Carl Franklin: Sure.



Scott Hanselman: You know, may be, someone could script PWOPcatcher to do different things. You have basically built in automation for the existing .NET application.

Carl Franklin: You like using PWOPcatcher as an example, I have noticed.

Scott Hanselman: Well, in this case, my audience of 1 is you, right?

Carl Franklin: (Laughing).

Scott Hanselman: So, I am putting it in terms that my audience can understand.

Carl Franklin: Sure.

Scott Hanselman: (Laughing) Just like my example is always Voyager and our application at Corillian.

Carl Franklin: Yeah.

Scott Hanselman: But PWOPcatcher is a great example because its one of these little applications that starting to become more and more fully featured.

Carl Franklin: Sure.

Scott Hanselman: And at some point, scripting might be something you might want to do to it.

Carl Franklin: Absolutely.

Scott Hanselman: Or other applications.

Carl Franklin: Yeah.

Scott Hanselman: Scripting is one of those things that one always wants to add to their application but the distance between your idea and the actually getting it done is always so great. In this example, Lee Holmes has added scripting to an application in six lines of code.

Carl Franklin: Yeah.

Scott Hanselman: And you don't see the PowerShell, right. It doesn't fire up and shell off into something, I mean, this is running in process. He is just adding a reference to system.management.automation. So, think about this, if you can start hosting the PowerShell- just in this example, doing script, what else could you do with it? You could have an administrative console written entirely in PowerShell, get it working and then write a WinForms app around it.

Carl Franklin: Yeah.

Scott Hanselman: A management application.

Carl Franklin: You know, I hear the voice of skepticism in the back of my head. Maybe, that few listeners out there are feeling this too. You know, why are you reinventing the wheel, why not just do things with Visual Studio, why not just create an application? You know, why have a limited functionality when you can have the real thing?

Scott Hanselman: Okay, so that's a very good question. So one, if I wanted to go and write some functionality that would be available to me at both the command line and at the WinForms. In WinForms, I'd have to write some DLL out there that would do the thing, right so, I'd write then C# with VB and then I'd write the command line version, right and then I'd write the Windows version. But because the command line version is forced to take strings as input and strings as output, it wouldn't be able to very easily communicate with other pieces of the command line version. I couldn't string multiple commands together.

Carl Franklin: Right. Now you have to write parsers and things like that.

Scott Hanselman: Then think it grows, okay.

Carl Franklin: Interoperability is really the key here.

Scott Hanselman: Right, Interop between the commands.

Carl Franklin: Between commands.

Scott Hanselman: But it really gets interesting when we talk about COM and WMI. If we are talking just .NET, you partially have a point. You could just write in .NET and it will work fine and many people have done just that and that's great. But what's cool in PowerShell is that WMI objects and COM objects and .NET objects are all peers. Now, in .NET in a regular WinForms application, COM objects have that whole Interop thing—getting caught behind you and every once in a while little COM leaks out and you go, "Ehh... that's nasty." And WMI isn't exactly easy either.

Carl Franklin: But, you have the same, you must have the same Interop issues with COM objects in PowerShell that you do in a .NET App, don't you?



Scott Hanselman: Well, so because PowerShell is doing all of its evaluation at runtime and lazily, they get to generate all this stuff and the whole fascia on top of your objects at Runtime. And the object that they hand back to you-- they can make it look like anything that they want. So, in PowerShell, if you want to make a new .NET object you can say, 'new-object system.string. If you want to make a new COM object, you say, 'new-object comshell.application and then you can loop over it and look at collections and run around inside the object exactly as if it's any old PowerShell object.

Carl Franklin: Sure, but one of the problems, especially with, you know, the disposed pattern and all that kind of stuff with COM objects is that if you really want to get rid of them, you can't just set them to nothing; you have to use the marshal object and release COM object.

Scott Hanselman: Now, I personally -- I am not sure how specifically PowerShell deals with that but I have gone and just set the COM objects to nil, to null rather, to \$ null and haven't had any trouble. We have treated WMI objects the same way.

Carl Franklin: Doesn't really mean that its working right though but...

Scott Hanselman: No, it doesn't mean its working right, but it does mean that PowerShell does everything that it possibly can, and your point is valid to the extent that its possible, to make it look like all these objects are peers; are the same thing.

Carl Franklin: And I guess, it's also what you are using it for, right. You are using PowerShell to script applications together whereas the problems with COM Interop really happen, really rear their ugly head under a big scalable back end system.

Scott Hanselman: Sure, that's a very valid point and I am not going to necessarily have PowerShell doing 10,000 somethings against a COM interface. Although I did recently write a script that would talk to Outlook and remove duplicates from my calendar. So, it spun through all my calendar objects, within the Outlook object model using COM, then looked for duplicates and removed them. It took like 200 Megs of RAM and pounded on Outlook for half an hour, but it didn't blow up, it didn't leak that I could see.

Carl Franklin: Right. So, and a leak is no problem, because it doesn't have to stay in memory for a long periods of time.

Scott Hanselman: Because I'll eventually close that shell.

Carl Franklin: Yeah, process blows away; its gone, anyway.

Scott Hanselman: Exactly, exactly.

Carl Franklin: Okay. Good points.

Scott Hanselman: So, one of the things we have been doing at Corillian is, scripting our Version Control using Subversion. We have talked before how I am a Subversion guy. And we have got some applications that need to programatically check things in and out of the system. We are doing some configuration management, we have an application that if our end user goes into our system and makes a change using a configuration tool, we want to know that they made that change; we want to have auditability. So, we actually check the files that that person changes with our content management system back into source control. So, then we can go back and say, "Oh! Gosh. On Tuesday, the client logged in and made three changes and I can see exactly what he did by doing DIFS." It's a pretty simple pattern, has nothing to do with source control; has to do with managing content. Why create a database to do all that management when you got a perfectly good journalled file system, right? And the same thing would apply to Team Foundation Server. But the Subversion system is this big opaque, C monster that I don't well understand, but a really cool guy name Arild Fines at shrinkster/gik, this is the guy who wrote Ank; Ank is the Subversion plug in for visual studio, is also a PowerShell fan and is helping us write Subversion scripts to check stuff in and out. So, right now there is scripts living at the command line, but we are going to plug those in using that same extensionability to host PowerShell within a web application or WINForms application. So, all that functionality to make these changes to our configuration system are going to be available in the WINForms app. You might put it into like an Office application, like InfoPath or the command line or in, on the web; all being hosted within PowerShell. So, PowerShell helps you level the playing field with these administrative consoles that you might have.

Carl Franklin: Very cool.

Scott Hanselman: So, that administration stuff is really where it's at. If you look at Jeffery Snover, the architect of PowerShell, has a series of TechEd talks that are now online; you can see



them at [shrinkster/gim](#). Jeffery is the really enthusiastic guy, who helped create PowerShell. He is kind of the architect of PowerShell, the father of PowerShell. I got a chance to meet him at TechEd. We were literally there on Sunday walking around, pre conferences started and he was hanging out, talking to another blogger, named Mark from the Netherlands and they were just hanging out and I was like, "Hey, what are you guys up to?" And then we just start packing on PowerShell. I think, I talked to him for like three hours; such a cool guy. It's one of the great things about TechEd; you just bump into people like this.

Carl Franklin: I know it's awesome.

Scott Hanselman: Ended up doing two demonstrations of PowerShell for Jeffery and so he could, free him up to do a little bit of other things. We are writing more and more PowerShell content at Corillian. We need to unit-test that content. So, one of our guys Jason Scheuerman at Corillian, has written an NUnit example up at [shrinkster/gio](#) where you can use NUnit to test your PowerShell scripts to see if they work well. Very, very simple, same exact model as the Logo stuff. You are hosting PowerShell within NUnit and then NUnit itself gets hosted within a Test runner. And those concepts very much apply to whatever Test runner you are using. If you are using Team Foundation Server, it would apply as well. It's just that flexible having an in-process engine to fire a PowerShell; it makes it really, really powerful.

Carl Franklin: Nice.

Scott Hanselman: Now, one of the other things that PowerShell is really awesome at doing, is manipulating XML. It lets you look at XML in what feels like a very strongly typed way. Like, if I were going to load up, if I were to load up an RSS feed and look for attachments, right, look for enclosures- typically, I would like to, 'XMLdocument.load', give it the URL, then I would say, like, 'Select nodes' or 'Select single nodes.'

Carl Franklin: You could use XPath too.

Scott Hanselman: Right, use XPath type in, 'wackwack/enclosures' or something like that. I could use, get elements by tag name and get all of the elements of a certain tag name.

Carl Franklin: Several nested loops.

Scott Hanselman: Yeah. I could spin through it, looking at the different child nodes underneath

that dome. With PowerShell, though, you can do all those things of course, but it also looks at the tags and generates, basically I would say, synthetic properties. So, I could say, take in a variable A-- I would say, A and I go and I go off and get my RSS feed and now inside of the variable A is my RSS feed. I could say a.rss.channel.items and spin through that list. So, they don't really exist, but it looks at the document and it gleans them right there; they are synthetic.

Carl Franklin: Now, that's a beautiful model.

Scott Hanselman: Oh! It's so clean, its not serialization, right. It's almost like LINQ or XLinQ, but it's today.

Carl Franklin: Yeah, very cool.

Scott Hanselman: Yeah and I can, what's cool, check this out so I can say, 'a.rss.channel.itemspipe' and I can pipe that collection of items and then I can go and say, 'whereitem.title= this.' So, like if you almost kind of...

Carl Franklin: Very Linq like.

Scott Hanselman: It's very Linq like. And it all happens today and it's all because PowerShell is doing this lazy, reflection loose tepidness.

Carl Franklin: Interpreting on the fly.

Scott Hanselman: Right. It's very fast; performance doesn't have to be an issue, right, because it's supposed to be friendly and flexible.

Carl Franklin: Yeah and performance is less and less of an issue every year with these new processors; it's ridiculous, especially for apps.

Scott Hanselman: Exactly, exactly and for administration, it's not that big of a deal.

Carl Franklin: Not a big deal.

Scott Hanselman: But it still is pretty fast. So, I went and I noticed that a guy had written a Windows Desktop Search thing, plug in the CMDLET for a PowerShell because he wanted to be able to search his desktop search from PowerShell. Now, I stopped using Windows Desktop Search a while back, I use Google Desktop Search. So, I know that Google Desktop has a COM API, so figured I'd go off and try to do some magic to talk to that COM API. But the Com part of my brain has long been overwritten by more interesting things; that brain cell now



holds, I don't know, information about Lost and Gray's Anatomy, you know. So, I was physically...

Carl Franklin: Amen to that.

Scott Hanselman: Oh! Yeah. I gave up on all that stuff. And then I said to myself, "Oh, crap. How am I going to talk to Google Desktop- all that I have is a COM API?" Well, it turns out that if you search Google Desktop you are doing a local host, like a mini web server, right, you are on like Port 8080. If you type in format=xml on the URL, when you talk to your Google Desktop, they will hand back XML rather than HTML.

Carl Franklin: Ooh and you pipe that right into...

Scott Hanselman: Into PowerShell and start manipulating it.

Carl Franklin: ...into PowerShell wow!

Scott Hanselman: So, what I ended up doing was, I did a local web server call. Basically I said to PowerShell, 'Search for Carl Franklin, does a web service call to the local host instance of Google Desktop format=xml' and then spun through it. And then here's the cool part- it returns back the file names, right. Well, those file names are local, they are on my real system, so I can go and say, get info and suddenly talk to those file names and I ended up returning not just a list of the file names, but the actual file info. Exactly. And then from there, I could say, search for all of these files and get their content and copy them here or delete them or whatever. So, you know, what I ended up doing, I ended up searching for my Social Security number, found it in a tax return that I had forgotten about. And then piped the resulting file info objects and deleted them.

Carl Franklin: Wow!

Scott Hanselman: Yeah, very cool stuff. And that's just one example. That applies to whatever you want to do-- any place where you have got some information you want to tie in to other information. So, PowerShell is that spackle, right, it's that putty that can tie these disparate bits of information together. So, for automating, like the workflow process, I know that you have a very complicated workflow process to make PWOP work because you have the source audio and then it fires off, all of these different Codecs and compressors and things to make all the different audio and then FTP them- put things in databases and stuff. It's probably batch files and magic in VB .NET, I don't know how you do it.

Carl Franklin: It is, no, it's not batch files. It's shelling out to different things and shelling out to convert the files to Lame and to the Windows Media Encoder and the workflow is all just logic, it's all VB.NET logic.

Scott Hanselman: So, stuff like that is totally something you could write in VB.NET and you already have the benefits of writing it and scripting or the flexibility that you can change those things very quickly. There is also the notion of a progress bar within PowerShell. You can say, 'Write progress,' and depending on what its hosted in, and it might be hosted in the command line and then bring up kind of an ANSI art progress bar or if its hosted in the Windows application, you might choose to put up an actual progress bar. So, no one has to know it's a PowerShell script. And you still get the flexibility of being able to change those scripts, that example is at shrinkster.com/gjq.

Carl Franklin: Very nice.

Scott Hanselman: Yeah, pretty cool stuff. There is a number of different advanced things that I have got on the blog to take a look at. One of them is talking to Virtual Server 2005, using WMI at shrinkster.com/gjs. And then extending DIR, actually making DIR, DIR, show more information that it does when it ships with the PowerShell- that's at shrinkster.com/git.

Carl Franklin: Nice.

Scott Hanselman: Lots of good information. I really encourage people to take the time, because it will suck for the first couple of hours; PowerShell is obtuse and confusing, but once you get it, it stops being obtuse and starts being elegant. It stops being confusing and starts working the way your brain works.

Carl Franklin: It really opens up a lot of OS things that aren't available to you. I got to tell you, Scott, that example of Google Desktop is just wild.

Scott Hanselman: Because, how would you have done that before, right.

Carl Franklin: Yeah. No, you couldn't.

Scott Hanselman: Yeah, it would have been very, very difficult. You could have done with a command-line application in .NET, but it wouldn't truly have been integrated. It's the difference between kind of McGiver style integration-- where



you are doing baling wire and Interop and really having it baked in.

Carl Franklin: Wow! That sounds fantastic, Scott. I think the listeners are really going to appreciate all your hard work in rounding up all these great resources. Check out PowerShell, get into it and we will see you next week on Hanselminutes.

(Music).