



<http://www.dotnetrocks.com>



Carl Franklin

Carl Franklin and Richard Campbell interview experts to bring you insights into .NET technology and the state of software development. More than just a dry interview show, we have fun! Original Music! Prizes! Check out what you've been missing!



Richard Campbell

Text Transcript of Show #367
(Transcription services provided by [PWOP Productions](#))



Udi Dahan Scales Web Applications!
August 12, 2008
Our Sponsors





Geoff Maciolek: The opinions and viewpoints expressed in .NET Rocks! are not necessarily those of its sponsors, or of Microsoft Corporation, its partners, or employees. .NET Rocks! is a production of Franklins.NET, which is solely responsible for its content. Franklins.NET - Training Developers to Work Smarter.

[Music]

Lawrence Ryan: Hey, Rock heads! Tell your inner child to go play outside and listen up! It's time for another stellar episode of .NET Rocks! the Internet audio talk show for .NET developers, with Carl Franklin and Richard Campbell. This is Lawrence Ryan announcing show #367, with guest Udi Dahan, recorded live, Tuesday, July 15, 2008. .NET Rocks! is brought to you by Franklins.NET - Training Developers to Work Smarter and now offering SharePoint 2007 video training with Sahil Malik on DVD, dnrTV style, order your copy now at www.franklins.net. Support is also provided by Telerik, combining the best in Windows Forms and ASP.NET controls with first class customer service, online at www.telerik.com, and by CoDe Magazine, the leading independent magazine for .NET developers, online at www.code-magazine.com. And now, the man who says, "Give me ambivalence or give me, um, whatever," Carl Franklin.

Carl Franklin: Thank you very much. Welcome back to .NET Rocks! Carl Franklin here in New London, Connecticut. Richard Campbell still on vacation but he'll be here for the interview with Udi Dahan in just a minute. How are you all doing out there? It's been a great summer for me so far. What's going on? I'm doing a lot of recording in the studio, a lot of producing and editing, and all that kind of stuff. My brother Jay and I are working on our second album. We're having lots of fun doing that, of course doing these shows for you twice a week and dnrTV and having a good old time up here in New London. Drop us a line if you have something to say, dotnetrocks@franklins.net. Now, let's get into Better-Know-a-Framework.

[Music]

Of course Better-Know-a-Framework is a little routine that I do to shine a light in various hidden corners of the .NET Framework so that you know what's there and over time by osmosis, you will learn what's in there if you don't have the time to crack a book or get online and read websites all day. So, today, I'm going to talk a little bit about the System.Data.Services namespace which is where ADO.NET Data Services lives. You might remember .NET Rocks! show #289 with Pablo Castro on Astoria, that was the codename for Microsoft ADO.NET Data Services, and also show #349 with Scott Hunter on Microsoft Dynamic Data.

Microsoft ADO.NET Data Services is a nice little namespace that uses REST style HTDP calls to do data services using WCF over the internet. It's kind of a nice addition to the web services in WS-Star style of essentially web data services but this wraps it up in a nice little REST, non-SOAP kind of tool and the main entry point for developing an ADO.NET data service is the data service of T class. So, you want to check out the data service of T class. Of course, you want to listen to those shows and go to the Astoria Team blog and we'll put some links on the website for you. That's where you should get started with that ADO.NET Data Services, System.Data.Services namespace, check it out.

On to our email, we have an email this morning from Richard Dynell and I'm sorry if I butchered your name Richard. He's from Montreal so it's probably like Dinell maybe, I don't know. I'm probably shooting myself in the foot by even trying there, but he says, "Hi! Dear Carl and Richard. I'm a history buff while still working in the computer industry since 1979. I think history is a good outlet, at least for me. Listening to show #366 was like going to an arcade and shooting once and hitting twice." He is talking, of course, about last Thursday's show with Eric Swedin who wrote a great book about the history of computing and Mark Dunn and I just totally geeked out with him. It was a lot of fun. "Knowing where things come from gives us a better understanding of what we are and where we're going. John von Neumann is a hero for me and when your guest talked about this man being no mainly for one page, he wrote about the architecture of computers, I instantly put a note on my laptop to write to you about this reflection. A page. A page, a simple page but what a page. Every invention comes from an emerging idea. I call it a spark. In this case, it all started with this ingenious idea, the Instruction Pointer. Neumann, probably some others too, mainly imagined a register, for those who remember what a register is, joking of course, containing the address of the next instruction to execute and the fetch incrementing the register by the length of the instruction after execution readying for the next instruction to execute." He adds that, "The change in the value of the register will allow for jumps, which is like go-to's and looping, opening the way to modern computing. Isn't it a genius spark, a spark of genius, writing a single sheet of paper? Thanks for taking the time to read this message. Continue producing your wonderful shows that are part of my outlet because learning is fun. Also, I insist for people to join their local user groups. It's a fun way to learn and meet nice people." Thank you very much, Richard. We'll be sending you a .NET Rocks! mug. Of course, if you have something to say, as I said before, send it to us at dotnetrocks@franklins.net.



Hey, are you thinking of changing careers? You want something new to do? Want to hang out with some very cool people in New York City? Do you want to live in an apartment in New York City rent free for a year while you're working and making a great salary there? Well, you got to check out shrinkster.com/kh6 for the New York City tour that Infusion Development, friends of ours from New York, are offering. They are looking for really good people and they know that the people who listen to this show are really good, really talented people who care about their jobs. That's why they're good friends of ours and that's why they're going through us to find you. So, if you want to do that or, hey, you want to go to Dubai? I'm talking about Dubai. That's right, the Middle East where all the money is, where they have silly sick bandwidth and every luxury that you can imagine. I'm talking about Dubai. Look it up. Well, there are opportunities there and also if you want to do any Surface development, you know, Microsoft Surface, that table that looks like Minority Report, the table computer, you know what I'm talking about, well, if you want to do that too, just send me an email, carl@franklins.net, I'll hook you up with these people. More than 17 or 18 people have already joined the Infusion team from listening to .NET Rocks! so check it out.

Okay, it's time to introduce Udi. Udi Dahan is The Software Simplist, recognized by Microsoft Corporation with the coveted Most Valuable Professional award for Solutions Architecture now 3 years running. Udi is a Connected Technologies Advisor working with Microsoft on WCF, WF, and Oslo. He also serves on the Advisory Boards of the Microsoft Software Factories Initiative and the Patterns & Practices' Prism Project. He provides clients all over the world with training, mentoring and high-end architecture consulting services, specializing in Service-Oriented, scalable and secure .NET architecture design. Udi is a member of the European Speakers Bureau of the International .NET Association, a founding member of the Architect Training Committee of the International Association of Software Architects, a Dr. Dobb's sponsored expert on Web Services, SOA, & XML, a frequent conference presenter, and a regularly published author and all around good guy. Udi Dahan, how are you?

Udi Dahan: Very well. Thank you Carl. It's a pleasure to be back.

Carl Franklin: It's a pleasure to have you back.

Richard Campbell: Almost a year on the nose.

Udi Dahan: Yup, yup, it was August, wasn't it?

Richard Campbell: It was indeed.

Carl Franklin: I remember being a little intimidated by your resume back then, by your bio, yeah, so that's...

Udi Dahan: It wasn't long enough, was it?

Carl Franklin: Yeah, we had to make it longer. It's quite a lot of stuff that you do there.

Udi Dahan: Yeah, as Richard says, no rest for the wicked.

Carl Franklin: And he should know.

Richard Campbell: Yeah, thanks. Okay. So Udi, what is your day to day work like? Are you mostly dealing with just architecture? Do you get a chance to write much code?

Udi Dahan: I don't write much code, but I do get to figure out if my code isn't working as expected.

Richard Campbell: Right.

Udi Dahan: A lot of the original calls that I get from clients the first time around is, "So we've got this problem with our site. For a while it works okay and then it gets a little bit slow and then servers start crashing and somebody told us that you're the guy to talk about server questions." Then I sort of dig around the innards and try to figure out what it is that they did that was wrong and sometimes it's just innocuous as a single line of code but that really has big ripple effect that just tear apart the architecture of the site and bring it to its knees.

Richard Campbell: This is fun work for me too. I love doing this. I mean we never get to build a site from scratch. It's always you get pulled into after it's already failing.

Udi Dahan: Actually, I have one client, one that said, "We want to do it right, so we're throwing everything away and we're building it from the ground up," and unless I'm very much mistaken, this past weekend they went into production and everything is looking great, performance is wonderful, and they're scaling out and everything is stable and they have a handle on what's going on with their site. So, I'm very pleased that I was actually able to get into a Greenfield project like that and have such a big impact on the direction they went and to see that it worked, you know, just straight through...

Richard Campbell: Yeah, from version 1, that's a rarity definitely but I guess they had good figures on



how much performance and how much scale they were going to need.

Udi Dahan: Well, they had a baseline from their previous system. They also knew that -- or one of the things financing this whole effort from the business side was, "Okay, we're signing up a whole bunch of huge clients and we really got to do this right because the last time we tried doing that and just making the system work and scale however we could, we ran into lots of problems. So, we really understand that we need to do this right; otherwise, we could just lose this client."

Richard Campbell: It's funny how different you can design applications depending on the scale requirements and the reliability of the requirements that people have.

Udi Dahan: Absolutely.

Richard Campbell: So we bumped into each other at TechEd and I think you and I were pretty much the only people talking about scaling ASP.NET.

Udi Dahan: I hope that we're not the only people that are actually trying to do it.

Richard Campbell: I'm sure other people are doing it but, you know, actually doing sessions on it at TechEd is a rarity.

Udi Dahan: Right, yeah, yeah.

Richard Campbell: So, what sort of sessions were you doing?

Udi Dahan: One session I had was web scalability using an asynchronous systems architecture which also double with the issue of caching which you talked about in your session quite a bit.

Richard Campbell: Yeah.

Udi Dahan: But came at it from a slightly different direction. One of the things that I was talking about quite a bit in my session was the use of publish-subscribe as a core pattern in solving the issue of keeping the cache up-to-date or keeping the right part of the right cache up-to-date, the right amount versus a specific system because as we know, one of the problems of caching these days is that not all data behaves the same and trying to fit all your heterogeneous data that needs to behave differently into one style of cache really just causes more problems than it solves.

Richard Campbell: Well, yeah, caching is one of those things and I don't know that I actually dug as

deeply as you did in the pattern discussion of how to manage these things, but really it was just a discussion of people abuse caching because they don't have a plan.

Udi Dahan: Well, it's the new shiny toy right now with Velocity coming out. That's the new thing. It's the new kid on the block.

Richard Campbell: Caching is far from new. I've had a sense that I've seen this in a lot of different projects where caching benchmarks really, really well, that the guy sits down, figures out how to cache one set of classes and fires it up and shows dramatic performance improvement, 10 times faster, that kind of thing and he says, "Okay, we got to do this everywhere," and it gets out of hand.

Udi Dahan: Well, you know what they say about benchmarks. Benchmarks don't lie but liars definitely do benchmark.

Richard Campbell: Definitely. So, let's talk more about I mean the shiny part of caching is easy to look at, this idea that I pulled the data out of the database, store it closer to the page so that it renders that much faster. What's the ugly side of caching?

Udi Dahan: Oh, the ugly side. The ugly side is what the cache does underneath the covers. It's all the things that can go wrong, that will go wrong, and well of course when a developer runs it on their own machine, they never bring the system into those situations where that becomes a problem.

Richard Campbell: It never goes wrong.

Udi Dahan: Right. Now, once you start bringing such a system into production and you got all different kinds of users and they're all working on different kinds of data, all of a sudden naïve caching strategy will bring your entire database or try to bring all your entire database into memory with that caching strategy and the poor cache in trying to manage its memory will start paging things to disk and things that if you just went to the database might have taken you a couple of dozen of milliseconds. Now, because everything has to be swapped in and out of memory and to file and to look it back up again and to load it back into memory, those same queries that could have been fairly fast if you just went straight to the database, now that you're using caching under a highly concurrent multi-user scenario, those things can take seconds.

Richard Campbell: Just because there's so much churn.

Udi Dahan: Right, exactly, and the thing is developers never see that. It always works on their

machine just fine. They don't actually try to run the system with a thousand concurrent users, each one doing a different thing. Also most of the low testing tools, they'll make it very easy for it to set up different scenarios and to have users stick around on the site just long enough for their sessions to start getting stale before they start working again. It's a scary world out there that the tools kind of make it look like everything is rosy, you know, just add this little API and you're good to go, but there are a lot of things that need to be thought about. Just the issue of is the cache up-to-date, is what I'm showing my user up-to-date, there are certain parts of the business that no, you actually have to show the user the most up-to-date data and thinking that going to the cache is a way to do that may not always be true and keeping that of course consistent with other bits of data that are cached in different machines is a total nightmare and I've seen systems where in order to solve those problems, they've started putting transactions on top of their cache and started doing this gigantic two-faced commits that essentially locked out their entire web tier and their database while one user was working.

Richard Campbell: Nice.

Udi Dahan: For that user to see consistent data and it's one of those cases where, you know, that silver bullet just blew their site just out of the water.

Carl Franklin: Just for the record, I am still here but I'm listening. I'm going to listen on this show because this is clearly Richard's area of expertise.

Richard Campbell: Scaling is a different world. We do so much work in ASP.NET that's got nothing to do with any of this. You can do so much and never deal with scaling problems. Most people I find don't encounter scaling problems until the site is already shipped and then it's like going to the grocery store hungry. The site is now failing under load and you're trying to come up with intelligent solutions how to fix that.

Udi Dahan: Right. Well, at that point, if a system wasn't designed to scale, again at this point, if you want to call out the point, it's a design issue. It's not a technology issue. It's not something that you're going to plug into your site that's going to auto magically make everything better, but it really is understanding what it is that the users are doing and what they try to do and what has to be consistent and what can be done in the background and what should be done in the background, all these things, it's a design process. It's not a technology question and a lot of times developers are really just looking for that just show me the API to plug into the current system that's currently failing in order to make things better

and that is so difficult to do and I'm not even talking about rolling it out because in a case that the site is failing, sometimes the last thing that you want to do is to take down your entire web farm for a few minutes while you're trying to deploy that fix and of course it's one of those cases where if you're lucky enough to get into the situation where lots of users are hammering your site, it's very difficult to bring the system up. It's like after you've already been knocked down, they keep hitting you. It's that much more difficult to bring the new site up under load than it was the first time going into production.

Richard Campbell: Yeah, you know, somebody's punching you in the face, you fall down. Do you think it's easy to get back up while they're still punching you in the face?

Udi Dahan: I hope I never have to be in that situation.

[Laughter]

Richard Campbell: It's a valid point. We often do that. The funny part is looking at the switch while this is going on. You've seen those lights pounding away and thinking that's a lot of request that aren't being served which is maybe that's a segue into this idea of the scalability is one thing in serving all those customers, but what about reliability. Web to me has never been a particularly reliable service. We tend to let pages just fail. I mean even most failover strategies I've seen in web farms, you lose a few connections along the way.

Udi Dahan: Right, right. Well, I think that kind of has to do with the, if I were to use a bad word, web 1.0 design that so many sites still are based on. Even when they are using AJAX, they sort of come at it from a different way. The thing is that HTTP is inherently unreliable. There's really nothing that can be done about that from a broad internet perspective. However, there are a number of things that can be done to make that problem have less of an impact. One of those is to understand that the browser is your friend and to specifically design the requests that are coming from the browser to your server so that if the browser doesn't get a response, you can via JavaScript on the browser site resend that same request over again. Now, if we're talking about submitting an order, cancelling an order, or doing any kind of action towards the data where you want your data to be consistent and you want to maintain reliability, you really can't let users do that because they've been trained for years on the internet. You do not push that submit button again.

Richard Campbell: Right.

Udi Dahan: No matter what and if it failed the first time, run away. That site is not deserving of any of your personal data because it's probably not going to end up somewhere good.

Richard Campbell: Yeah. It's obviously not working. Why would you want to feed more into it?

Udi Dahan: Right. So, designing for that and coding for that on the browser saying, "Okay, the user submitted it once," and then handled that whole sending the request to the server, did I get a response back in a reasonable period of time and if I didn't send the request again and then design the request in such a way so that even if it is submitted a number of times, the system will still be correct. So, if we're talking about ordering something over the internet, that's something that your browser could generate a GUID saying, "Yeah, I'm submitting an order here and this is my number," and when the web server receives that -- so, the first time, there are a number of ways that things can fail. The first time is from the browser to the server. The request just doesn't get across.

Richard Campbell: Right.

Udi Dahan: Another side is when the server's returning a response back to the browser and that gets lost. So, with this retry protocol, we can actually have a scenario where the server receives the same request twice. Now, by tagging that request with a GUID, the server can say, "Oh, I already processed that," and in that way, you can maintain its correctness. You can just send that response again.

Richard Campbell: Yeah, it checks to see the GUID is there and says, "Oh, it's already here. Just send back. Yup, I got it. Thanks for playing."

Udi Dahan: Exactly. Now, from the user's perspective, they don't see any of this.

Richard Campbell: Right. This should all be under the hood.

Udi Dahan: Exactly, and because HTTP itself is unreliable, it doesn't matter if the first request got lost or the second response got lost or whatever happened in the middle. We can make sure that from the user's perspective, until that point where we say, "Yeah, we've got your data," we can be fairly reliable even on top of an unreliable protocol.

Carl Franklin: Now, are you saying that you want to use a GUID because it's possible that two requests could be sent within the same second or within the same time instead of looking at like the date stamp of the request?

Udi Dahan: Oh, there is such a problem with date. It's so insidious and developers use it way too much. First of all, it's the assumption that all machines in the world have the same clock. They just don't.

Carl Franklin: Well, we're talking about trying to discern the first request from the second one and every request has a time stamp when it was received by the server which comes from the server, right?

Udi Dahan: Right.

Carl Franklin: In the HTTP headers, and that is not enough to discern between two requests?

Udi Dahan: Well, most of the time until your servers move from regular time to daylight savings time and then back again.

Carl Franklin: Right, so there's always some problem with dates.

Udi Dahan: Right, and there's also the network time protocol that keeps servers in sync that can move clocks a bit so even if you are sending a request one second after another because of some updates to the server's time, that is really fairly low level. It's not something that an IT operator or administrator would even notice.

Carl Franklin: Right.

Udi Dahan: We just assume that this is the second request, that the second request is actually a first request or vice versa.

Richard Campbell: Yeah. It's just weird enough that you would occasionally get these glitches and try to figure out, but I think the main reason you were talking about a GUID, Udi, was so that you could send the same request several times and have the server able to see these are actually all the same requests because they all have the same GUID.

Udi Dahan: Exactly, right.

Carl Franklin: Oh I see, I see.

Udi Dahan: And it's also fairly simple to generate GUIDs. I mean it doesn't require a very fancy library.

Carl Franklin: No, no, it's easy, yeah.

Udi Dahan: It doesn't come with any difficult performance requirements on the browser side that something even a mobile device can do.



Richard Campbell: I generally have the sense that we're still under utilizing the browser side in a web application, that there's more horsepower, there's more capability there. It's not being used very well and most times -- I mean even Microsoft's implementation of AJAX does its rendering on the server.

Udi Dahan: Right, right.

Richard Campbell: So, it just seems like we're ignoring the potential of the browser to cooperate in a sophisticated web app.

Udi Dahan: I think that's very much the case but it's not only an issue of horsepower, it's more an issue of management, getting back to that reliability that you were talking about. First of all, with the fact that we can't keep the browser responsive in the meantime AJAXy and all that kind of stuff, but more in terms of, well, we can actually design protocols for our system on top of HTTP to maintain its correctness. So, until we tell the user, "Yeah, we've got your data," they're not going to assume that we've received it. However, if the page goes blank or some things start spinning or something like that and we don't have a response to them, that web 1.0 style, that blank page that you get back is the scariest thing for users especially when they've submitted data or again because they've been taught don't refresh the page and if you go back, it might break the site so be careful and they kind of get into this freeze mode, "What do I do?" "I don't know." "Maybe I'll email somebody and ask what I should do, but I don't want to refresh. I don't want to resubmit and if I go back and I look at my status, I may not be up to date," and all sorts of other problems just by turning the browser into you might call it our user proxy. So, the way that our server talks to the browser, it actually is a distributed system and I think that that's one of the forward looking things that's going to be coming in web development is that this isn't just a web app. This is a distributed system. There is a client site component, there is a server side component, and the way these things talk to each other, yes, it may be going over HTTP but we do actually get to decide what the protocol is and then we can design that protocol in such a way that it will be reliable and robust and even if HTTP loses some data, we'll be able to get around that and there are a whole host of things that we can do. One of the talks that I gave, a talk following up on my scalability talk at TechEd dealt with one of the very difficult problems with web applications today is what do you do when the processing of some requests take a very long time and when I say a very long time, it could be because a user has to authorize something or just we're talking about some serious number crunching in the background, something that in the background we're going to be doing asynchronously, but how do we actually connect that

to the user's expectation of a regular request response semantic.

Richard Campbell: How long are we talking when we say very long, five minutes?

Udi Dahan: Very long can be from one minute to a day, to a week. One of the things that I've been seeing in web apps, or rather they call them composite web apps, is that similar to something like iGoogle or something like that, users have various windows on to the world. So, while one window, they may submit a request saying, "I want to go on vacation," until they get a response back saying "your vacation has been processed. It's been accepted," or "It's been denied," another actually has to take action. In the meantime, that window shouldn't allow the user to do anything with vacation, but for all those other little windows or web parts, they can let the user continue doing other work as well. So, when the user is presented with a web app that's more a dashboard on to various domains, well, each domain can be different, right?

Richard Campbell: Right.

Udi Dahan: And we really need to design our system in such a way that, well, even though some things can be happening fairly quickly, say in the period of a minute, if we were to pull every 10 seconds, that will just totally bring a server that's processing things that takes days and days to its knees and vice versa. We wouldn't want things that can be happening quickly just to pull them every hour just because there are some things that can take days.

Richard Campbell: Generally speaking, pulling is just not a good idea.

Carl Franklin: Especially on a browser.

Udi Dahan: Well, pulling has been somewhat misaligned. It's considered a bad choice but over the web, we don't really have very much other choice because we can't initiate the connection from the server back to the client.

Richard Campbell: Right.

Udi Dahan: So, keeping a connection open is one option and IIS doesn't like that.

Richard Campbell: Nope.

Udi Dahan: Another option is for the browser to pull. The thing is what does it pull? If it were to simply pull a static resource, that wouldn't load up our servers very much at all. Apparently, web servers are designed extremely well when it comes to



servicing up static content and if the static content that you're serving is the way that you encode responses, apparently you can handle a huge amount of load even with pulling.

Richard Campbell: So, painting this in sort of a realistic scenario, are you essentially saying in order to provide a monitoring of a long running process, what you should be doing is writing out a static resource file periodically?

Udi Dahan: That's one way to look at it. Think about it like this. What we have is our browser calls a web service on the server. The web service generates a kind of ticket and says to the client, "You are customer number 1,235," and your response is going to be at this resource right here and it gives it some URI to a static resource. On top of that, you can also say, "And you should be pinging this resource once every one second, 10 seconds, one hour, one day."

Richard Campbell: Right.

Udi Dahan: At that point, when the browser sees that, it turns around to the server and says, "Do you have my resource ready for me?" Now, that kind of request, that basic HTTP request for static resource is something that doesn't even get to ASP.NET.

Richard Campbell: Yeah, it's IIS.

Udi Dahan: It's now entirely by IIS on the native pipeline. It is blazingly fast. Okay? Now, by also adding that bit of telling the client, "Check this resource once every T seconds," we can modulate dynamically what kind of native load we're getting. So, for things that are going to take days and days, you can say, "Oh, just ping me every hour or so. For things that can take up 10 minutes, ping me every 10 seconds." In that way, we can actually take a look at things that go fast to make the user get a quick response. For things that go slow, don't overload my server, "Oh, and by the way, my IT operators are going to want fiddle with these numbers to see what kind of load correctors they're going to have on the site."

Richard Campbell: Right. Yeah, this is good instrumentation for how things are going.

Udi Dahan: Right and you have all the IIS performance monitors. You can see what kind of requests you're getting and from which client you're getting to roll them up into a dashboard and your operators can see what's going on, who's received your request, how timely it was, and then they can fiddle around with the parameters, but again this style of programming is by saying, "I'm going to take

control over the protocol between the browser and the server. I'm not going to just advocate that to ASP.NET."

Richard Campbell: Right. Well, this idea that I'm going to deliberately do it in such a way that I don't get to ASP.NET, which in some ways for a developer feels like giving up control, now I'm not in my common codebase. I'm looking somewhere else.

Udi Dahan: Right.

Richard Campbell: I'm just wondering how comfortable users are with a day-long running process against the web browser if they want to reboot or do anything like that, I guess you've got to think through. Can I give them a link that they can save so that they can come back here later? I can't count on any session cookies. I mean all of that, it's so long that all of those dynamics change.

Udi Dahan: Well, right, but then you've got again the newer generation of browser site libraries like Google Gears that enable you to take control of what data is actually presented. The browser is becoming displaced where not only can you program, but it's becoming easier to program. It's becoming easier to manage your state because more and more people are hating this boundary saying, "If we do everything server side, we just can't scale."

Richard Campbell: Right.

Udi Dahan: Or it's not that we can't scale because scalability is a cost function, rather it is so expensive to scale that we need to start firing our users, sending them away.

Richard Campbell: It's just not worth it.

Udi Dahan: Right.

Carl Franklin: I want to just take a minute to bring you a message from our sponsor Telerik and let you know that this portion of .NET Rocks! is brought to you by Telerik. You know, summer is in full swing now and you're probably lying on the beach, but our friends at Telerik are working hard as usual to bring you exciting new stuff for your .NET toolbox. How about two brand new control suites? RadControls for WPF and RadControls for Silverlight. That's right. If you started building next generation applications, you now have UI components with Telerik quality and Telerik reliability. Both product lines are derived from the same code base and share the same API so transition is seamless. They have many improvements in the other robust suites for ASP.NET AJAX and Windows Forms also, as well as the intuitive reporting tool, but product alone isn't everything. To jumpstart your projects and help you

easily get up to speed with these great tools, Telerik has got a couple unique training resources, the Telerik Interactive Trainer and Telerik TV, of course, which I am the host of. Now, that's what I call summer heat. Go check out all the details at telerik.com and if you happen to run into those guys, say thanks for supporting .NET Rocks!

Richard Campbell: The other one that's recent is jQuery. It's just this increasingly sophisticated libraries on the browser side that are able to create more of a smart client, goodness knows I shouldn't be saying this, smart client model on the browser.

Udi Dahan: Right. I'm not saying that it's the solution for everything but again, what I'm saying is when you're looking at scaling a site, you do need to actually take a look at all these options and make the trade-off yourself. This isn't a, okay, we're going to build an ASP.NET site, drag and drop some stuff and go, ship it to production, but really taking a look at what you need and how much is it going to cost you to develop and what kind of user experience you're going to get and when looking at what kind of response times you've got to give to your users, and again, if you're building a composite application where on that same webpage, you want to be doing a whole bunch of different things, you're going start seeing more and more of this stuff coming up. Now, users, they're getting used to more of this kind of environment and sites that offer them that kind of interactivity, yes, you can do more things at once and when you get a response back, you'll have it ready. That could be popping up inside the browser as some kind of toast or whatever kind of solution that we have. It is more I don't want to say Smart Client kind of development, but a Smart Client user experience that we're looking at offering here.

Richard Campbell: Right. Pardon me, I was thinking about this thinking, why wouldn't I just build a Smart Client here? Why am I going to jump through these hoops?

Udi Dahan: Well, it's that classic rich versus reach argument. Some users and some organizations will just not install anything.

Richard Campbell: Right, or can't install anything. The machine is not locked down.

Carl Franklin: Plus, they already have websites they want to continue developing.

Richard Campbell: Well, there's a n awful lot of infrastructure already in place with the website like there are a lot of advantages here, although, you know, if Tim Huckaby was in the room, he'd be yelling right now. You can build a pretty lightweight Smart

Client that's calling to a web service and a lot of that sort of persistent stuff becomes very easy.

Udi Dahan: Right, but then you still have to deal with where the data can be stored locally and all that. I mean, yes, you can make it lightweight but then you get into the issues of what happens when the user's machine restarts and what happens when your server restarts and in dealing with all of that, none of the problems magically go away with any solution.

Richard Campbell: Right.

Udi Dahan: Rather, you just have more tools in your belt in order to deal with them.

Richard Campbell: In some ways, with the new security models like UAC and so forth, the Smart Client is now living under similar security restrictions to the web client. It's a funny state to be in.

Udi Dahan: Exactly, which is why so many organizations are waiting with Vista because it's just going to break so many of their apps.

Richard Campbell: Yeah, because their apps are breaking rules they told them not to break ages ago, but hasn't had consequence up until now.

Udi Dahan: Yeah, yeah.

Richard Campbell: So, when you're talking about this whole message recovery thing, do you want to sort of quote some numbers? How long should you be waiting before you make another request like what's reasonable?

Udi Dahan: That really depends on the action that you're taking. If you're talking about doing some kind of calculation and the calculation can take a while to be processed, you might decide to wait longer until you resend it. However, once again, once you start designing your protocol, then you can have multiple responses that come back. One of them that says, "Yeah, I've received your request. Now, it's being calculated. No, you don't have to resend it to me."

Richard Campbell: Right. "Here is your receipt. I definitely got that."

Udi Dahan: Exactly. So, part of that is managing when the browser is going to be resending it, so actually saying, well, if I don't have only a single response, that's something bad, saying, "Okay, here is the response. This is the result of the calculation," but rather I can say, "Okay, I've received your request and now I'm processing it," or "your order has been tentatively approved," or whatever is going on in the

background. That makes it that much easier to save on those resends. One thing I do want to call out though because we started getting into the updating data, this stuff also holds true for just a regular website in how the browser gets data. So, if we look at your average website, there's a whole bunch of information around the dynamic part or you've got various dynamic parts that show things differently. If you're looking at Amazon, for example, you have your wish list and what other customers are buying and the menu that's somewhat dynamic as well. You don't have to view all those things as a single web page because the rate at which the site is going to be changing it, its menu structures are going to be quite a bit slower than your wish list.

Richard Campbell: Right.

Udi Dahan: And what other customers are buying is somewhere in the middle.

Richard Campbell: Yes.

Udi Dahan: So treating all of those things as just a single web page...

Richard Campbell: Is something to be recomputed at every request.

Udi Dahan: Exactly. So, saying, "Wait a minute. Now, I want to cache that," or "I want to cache parts of that." The thing is that you can take those same things that you use to cache. When I say cache, I mean cache from the way that ASP.NET developers are used to doing and saying, "Okay, we're going to take our menu and we're going to put that in the cache. We're going to take this data that other customers bought, these items and put that in the cache," but actually taking that out and leveraging the web and what I mean by that is saying other customers bought these kinds of similar items, say that changes once every half an hour, you can cache that in memory or what you can do is create a static resource which could be XML or it could be HTML that your browser is going to be pulling from. So, instead of actually hitting your server, what it can be doing is it could be hitting one of the proxies all over the web from that static resource.

Richard Campbell: Now, you're talking about sort of the Akamai model?

Udi Dahan: It's kind of the Akamai model but it's simpler than that. You don't even need an explicit content distribution network.

Carl Franklin: What is Akamai model, Richard?

Richard Campbell: I'm sorry. We're talking about content distribution networks which is exactly where Udi was headed where when sites get to a certain size, it makes sense to move those static resources into these services that will serve them faster and closer to the customer. Akamai owns like 30,000 servers scattered around the world so they were able to on the fly figure out where the nearest server is to you and so your images, your CSS, your JIS, those kinds of things all come from there and it just sort of unload to server, but that's only one flavor of content delivery network and it's one of these things that until you get into scaling, you just don't ever hear about.

Udi Dahan: Right.

Carl Franklin: Okay.

Udi Dahan: Now, the thing is that even regular ISPs when they have static resources going through them, if you're just hitting a regular HTML page and the server says, "Okay, this HTML page is going to be valid for the next half an hour," all the other requests coming to that same ISP for that same resource are going to be served by the ISP. It's not even going to get to this source server and we can make use of that behavior of the internet, of the web when we're designing our webpage. So, instead of looking at that webpage that has a whole bunch of different parts that have different levels of dynamicness as a single web page, we can look at it as the collection of static resources that each of them has a specific period of time that it's being cache. So one of them could be refreshed once a day, another one once a week, and another one once every hour, Slashdot for instance generates their homepage once every 10 minutes and what that does is that for ISPs all over the world, they actually maintain a copy of Slashdot for 10 minutes and every single request from every single hacker and developer wannabe all over the world to that site is not even going to get to the slashed out servers. It's just going to hit their localized ping get response back. So when we're talking about scaling to massive numbers of users that all want to be looking at similar data, what we do is we take all that information and cache it, not on our servers, but all over the internet and that's how we actually with a small number of servers can be serving millions of users.

Richard Campbell: This is really just markup. This is a way of marking the page so that these high request pages, you can tell these other servers you're allowed to cache this and here's how long.

Udi Dahan: Right. Now, the thing is what we're doing here is taking that to the next step not only at the page level. The page that we're showing to the users is actually to use a web 2.0 word, it's a mashup. It's saying, "Okay, I'm going to go get this

static resource and I'm going to put it over here," then you get that static resource and put it over there. This is what the browser is doing, the same way it picks up on images, for instance, if those images were on a content delivery network. So, we can do that for our data as well and when we take those two things together the way that we update it and the way that we get data and we leverage the web in our design and we make use of these protocols, we can scale to ridiculous numbers of users and give them really good performance if we're talking about how quickly they get a response back and just maintaining the stability of our servers because, again, like we said before, if there was a problem with our site and it's not scaling well, what we've done is we've offloaded a whole bunch of load making it easier to roll out patches so as we want to version our system and put out new functionality, we won't have that problem of millions of users punching us in the face while we're trying to get up.

Richard Campbell: Right. Now, I'm trying to envision, you were talking about, say, a menu system which changes relatively infrequently utilizing this sort of content delivery model. Is it like a descriptor file for the menu that the browser renders?

Udi Dahan: Oh, it's a lot simpler than that. It's just XML.

Richard Campbell: Right.

Udi Dahan: The menu itself is -- okay, the structure of menu in the links, that's just an XML file, okay?

Richard Campbell: But because we've generated this file and it's now a static resource, the CDNs will pick it up and browsers can now request and render the entire menu without ever touching the original server.

Udi Dahan: Exactly and we can do that for all sorts of pieces of data in our sites. I'm not talking about master pages here, I'm talking to the next level. If you look at big newspaper sites, what they're needing to do these days, they're getting a lot more dynamic.

Richard Campbell: Right.

Udi Dahan: Those sorts of things saying, "Well, we can't do the whole page because so much of the page is specific on who the user is, that that's a problem for us."

Richard Campbell: And that really comes down to individual stories so the unit of granularity for a newspaper has got to be the story.

Udi Dahan: Well, right, but then you've got things like how many comments were on the story.

Richard Campbell: Right.

Udi Dahan: Now, comments are something that even though they're dynamic, we don't really have to go all the way to our web servers and all the way to our database to get that information. That's something that makes sense to cache not only on our site but all over the internet.

Richard Campbell: Comments are tolerant to latency.

Udi Dahan: Exactly and there are all sorts of tricks we can do so that when the user posts their comment, they'll see their comments again by making use of the browser and other users that are coming in won't see that user's comment until, say, half an hour later. So, users, whenever they post a comment, they want to see it right away. They're not going to be saying, "Oh yeah, we're going to be show you your comment in half an hour." They don't want that.

Richard Campbell: No.

Udi Dahan: They want to see it right away, but that user's browser says, "Well, I know that this user put this comment in so I can take that comment as well as the static resource with all the other 20 comments on there and show to this specific user, 'Yeah, you're comment #21 and here's your comment right here.'"

Richard Campbell: But half an hour from now when it actually gets back and gets pushed to the cache, it might be comment 25.

Udi Dahan: Exactly but at that point, the user won't care anymore, will they?

Richard Campbell: No, they want to see that when they wrote their comment, the comment showed up. They probably won't look at it again.

Udi Dahan: Exactly. There are all sorts of, I don't want to call them tricks that we can do, but all sorts of techniques that we can make use of when we're designing our system dynamic as they may be, that when we start talking about caching and the use of CDNs and use of static resources and things like pulling and using publish-subscribe, there are so many tools that we have in our belt that most developers aren't even used to seeing and then the question becomes given that I've designed a website in a straightforward ASP.NET AJAX request-response manner and now my system is just falling over, how do I solve that problem? That's a difficult problem to

solve. The smart thing, don't get yourself into that situation in the first place.

Richard Campbell: Yeah, don't get there. This is so far removed. When I start talking to groups of ASP.NET developers about these sorts of technologies, it's so far removed from the dropping a web control on a page and hooking it to a data table model they've learned. I don't think we have good tools right now to break out that granularity of data and allow CDNs to take on that work.

Udi Dahan: Well, actually the code itself isn't that complicated.

Richard Campbell: No but it isn't written automatically either.

Udi Dahan: Okay. If you look at the things that developers do in terms of the amount of code that they write in order to try and scale a system after it started falling over, they write massive amounts of codes and cache this and tighten loops over there and write this to this local web server and synchronize with remote web servers. They start writing tons and tons of code to solve these problems after the fact. So, if anything we're looking at comparing is what was the resulting amount of code and how complex was it doing it the so-called straightforward way and doing it the inherently scalable way. After I've seen both sides, I can tell you that if you design for these things, the code itself is simpler. It's smaller, it's bite size. I mean writing a static resource is trivial. It's trivial. I mean developers do that when trying to scale a site anyway but they do it non-strategically. It's more along the lines of can we bring up some things to make things a little bit better. Writing JavaScript, well, these days lots of developers are getting down to the JavaScript level in order to give that high level of user experience because the AJAX toolkit, as good as they are, once you step away from the grid that does just about everything by itself and you try to do other pieces like for instance when something selected in the grid changed something somewhere else, developers start writing JavaScript as well. So, it's not like that they didn't need to know JavaScript before and now all of a sudden they do. All these technologies, they're using them already. What's missing is the design component. What's missing is sort of somebody coming and saying, "Do it like this and here is why." Again, it's nothing of a tool sets because like we said before we are getting better tools. Developers are wrapping up these libraries and making them more available, but a lot of application developers are saying, "Okay, I've got this new tool. What do I use it for?"

Richard Campbell: Yeah.

Udi Dahan: That's the hard question that they have. They're saying, "Okay, I've got all these tools. What do I use them for? How do I design this site?" with this thinking in mind and none of that information is out there these days, you know. It's people like you and me that are actually doing these things, we're so busy actually doing it. We don't actually have enough time to go around talking about it.

Richard Campbell: You know, you don't find a book on this. I mean I know you just got an article out the door. I've got an article out the door recently, but that's about it like there's just not a lot of time. We're so busy working to really help people understand all of this.

Udi Dahan: Right. Well, you know, you're working on clients, projects, and then you're speaking. There's so much to do that even when you do get an article out the door, there's just so much you can talk about in 6000 words.

Richard Campbell: If you can explain one good idea in 6000 words, man, that's all you're going to get.

Udi Dahan: Right and in this conversation, we've got over like 10?

Richard Campbell: So, let's jump back a little bit here. We started out talking about caching and sort of danced around some of the big issues on that and to me, it sounds to me you're really describing a different method of caching. When I say caching, I usually think about using the caching objects in ASP.NET and now I'm struggling with memory management and also struggling with expiration, that as soon as I'm in a web farm like it's just a bear to keep all those caches in sync but it seems to me it's somewhat easier when I have relatively static resources to push them into the static form so that I'm not even touching ASP.NET anymore to cache those items.

Udi Dahan: Right, right. So, again it's one of those things that just by looking at the page itself and saying, "Well, what data is being shown here and how up-to-date does it need to be?" those kinds of thoughts or those thought processes that you go through when you think about what can I cache in ASP.NET, you can ask the same questions that, well, instead of caching it in ASP.NET, I have a number of ways of caching it and different ones make sense in different ways. So, those memory management problems, a lot of them go away just because you're not keeping them in memory.

Richard Campbell: Absolutely. Yeah, disk is so much more available and the funny part is it will still get picked up in memory by IIS, its own kernel cache.



Udi Dahan: Exactly.

Richard Campbell: Which won't be part of the .NET memory space which is a thing we're wrestling with constantly.

Udi Dahan: Yes, and also all those threading problems of keeping things up-to-date and I look for some data and it isn't there so should I block the thread or should I retry again, all those difficult problems.

Richard Campbell: Oh, I walked through this in sessions all the time, the whole check to see if the cache is populated. Oh, it's not populated. Okay, block the thread because we don't want multiple instances trying to populate the cache out at the same time. Now, go populate the item but also check again after the block. Is the item populated now? Because I've had three or four or a hundred requests held up by that thread block and then they all try to populate the item.

Udi Dahan: Yeah and if you're talking about auto-refreshing of the cache then you can have a hundred threads hitting the database...

Richard Campbell: Absolutely.

Udi Dahan: Trying to load things into memory and that also thrashes your memory. So, there are all sorts of technical gotchas down there when you try to solve the problems with the wrong tool. That actually gives you more trouble than the original thing. You can still just go to the database.

Richard Campbell: Yeah, just go back to the database. So, one of the things I've been recommending and I would be interested in your opinion on this is instrumenting your caching code. How often do you populate the cache item and how often do you expire it and how many times do you have good hits like you hit the cached item and you use it so that I could actually get a sense for under load we never get to use this cache item. We're always repopulating it.

Udi Dahan: I'd say that monitoring is the backbone of any skillful website because if you have no measures, you don't know what's wrong, you don't know what to fix, and caching is just one part of that. That's true for your database and it's true for your middle tier. It's true for everything. One of the biggest problems these days is the site is slow. Why? We don't know.

Richard Campbell: Yeah, we don't really know and looking at PerfMon...

Udi Dahan: Just speculating -- oh, PerfMon.

Richard Campbell: The number of times I've had folks say, "I know what I have to fix. I just got to get to work," and I'm like, "I don't know how you know that. I would really like to collect some data around this first." "We don't have time. We've got to fix the code." "Okay, well, you call me back when you have time to actually test." After a few weeks of churning on code and not getting any results, then they're like, "Oh okay, I guess we have time to test now."

Udi Dahan: Again, it's just one of those things that the more feature-rich a site is, the more difficult it is. If you're talking about long running background processes and just knowing if a process has completed or not is a difficult question to answer by looking at PerfMon.

Richard Campbell: Yup, PerfMon is a long way -- and the funny part is so few people even look at PerfMon at all, but then when you do finally get good at studying PerfMon and watching what's going on, it's fairly abstract from your code. So, I can see that you're churning the worker process or you're causing GCs to fire off regularly and it's tossing cache items out of memory, but which items, what process is causing these things, that is not going to be answered from PerfMon.

Udi Dahan: Right and again the relative importance of that, I mean you may have one kind of item or one class that gets regularly thrown out of the cache, but if that's something that isn't used by many pages, or if it's used by pages, those pages can actually render without that information or even if that information is shown incorrectly, then that's not a big deal.

Richard Campbell: Yeah.

Udi Dahan: Then you can say, "Well, yes, I may be treating this data poorly but that's not what we need to fix."

Richard Campbell: It's not actually hurting me.

Udi Dahan: Right and that's a big problem that I see is that again when developers get into that crunch mode. "We got to get the fix out. We got to get the fix out," and they're looking at PerfMon saying, "Okay I see a problem." They see a technical problem but they don't necessarily see that if I fix this, will it make the business problem of the site being slow go away because maybe we can just say, "Yeah, that's a technical problem. Ignore it. Go fix more important things that maybe technically less interesting or less visible, but that's not what we need to fix."

Richard Campbell: But actually will return better results. You know, if you got rid of your 2 megabyte images, this page will perform so much better than you spending the weekend rewriting the entire page.

Udi Dahan: Exactly.

Carl Franklin: It sounds like to do what you guys are talking about just takes an awful lot of work. Scalability just takes a lot of work and to get back to your point, Udi, the original point that you made that people tend to do this, tend to think about scalability when it's too late...

Udi Dahan: Well, it tends to take a lot of thinking.

Carl Franklin: Right.

Udi Dahan: I wouldn't say that it takes a lot of work, but it takes a lot of thinking.

Carl Franklin: Smart work.

Udi Dahan: Yeah.

Richard Campbell: Well, I would argue that it's mostly about diagnostics. Once you actually know what the problem is, it's usually pretty easy to fix it, but finding the problem is the hard part.

Carl Franklin: Right.

Udi Dahan: Again, in that original view on push, to get the site into production, monitoring is usually not one of the things that companies like to invest in.

Richard Campbell: Yeah, no. They don't get around that. Okay, one more topic because we're just about out of time. Expiration, you know, one of the things that content delivery networks have in common with a lot of other basic caching techniques is they expire over time, but I find developers really wanted this expire by logic, that when the actual inventory count gets updated, then you throw out the product list.

Udi Dahan: Yup.

Richard Campbell: They don't want to do it by time and that to me becomes an utter nightmare. As soon as we're in any sense of complexity at all, as soon as we're in a web farm, as soon as the load is high, expiring over time is just so much more efficient. Any other ideas?

Udi Dahan: That's when the cases where I said, you know, just when we started out that publish-

subscribe model helps immensely. By making it possible for your mid-tier to publish an event that says inventory has dropped beneath the threshold or just inventory changed, that's something that all your servers in your web tier can get.

Richard Campbell: Right.

Udi Dahan: And then they can go and update the cache, that's if they're using a simple in-process cache. Now, if you're talking about, again, that web farm kind of caching, there are a number of other tricks that you can do in order to reliably but without turning everybody get that out there but the first part is really so critical is that being able to take that something logically interesting has happened and to push that from your backend to your front end cache is really just the lynch-pin of the solution because without it you can't do anything and by enabling whatever in your web tier to subscribe to those events, this brings you 99% of the way there. The last bit is just saying, "Okay, go update the cache."

Richard Campbell: The challenge here is if I got 10 web servers pounding away and every hundred milliseconds something is getting sold which is good, business is good, I've got to sort of modulate how frequently am I going to update that item because I'm constantly tossing it away from the cache in that sense.

Udi Dahan: Again, that's one of those issues of design and saying, "Do we really need to show the actual number of widgets in inventory?" I mean it doesn't really have to be up-to-date to the millisecond. If it was up-to-date every 10 minutes, that would be okay, wouldn't it?

Richard Campbell: That would be close enough. Well, and I solved a customer's problem like that by forcing them essentially to implement a back order system. Let us take the chance that once in a while we'll sell something we don't have to make so that in 99% of the cases, things work great and nice and fast and in that one rare case, we back order it and maybe that customer is unhappy but they will eventually get their thing and in the meantime everybody else didn't have to wait for the order to constantly keep that inventory up-to-date.

Udi Dahan: Right and of course, there are other things that we can do in the background. One thing is to implement that back order system. Another thing is to have another process that's looking at inventory coming in all the time or orders coming in all the time and doing some kind of forecasting and saying, "Okay, I see that this item is selling really well and it's going to take me, I don't know, a month to replenish the inventory and at this rate, we can be out



of it in a month and a half so I'm going to start pre-ordering things already."

Richard Campbell: Start raising order levels and so forth.

Udi Dahan: Right. So just, again, by having that ability to have our ordering system published and saying, "Okay, someone's ordered five widgets, someone has ordered 10 widgets," and having that ongoing graph of what's happening with orders being published outside the party, then they can hook into those things as well and saying, "Okay, I'm going to start doing some real time inventory management around those orders." I don't have to wait until I'm out of inventory to say, "Oh crap, now it's back ordered. How do I solve that problem as fast as possible? I can start designing processes into my organization that, yes, once in a while things will be back ordered. There's really nothing you can do about it but when they are back ordered, we already have inventory coming in on the way to be able to process those back orders that much faster." So, while users may not be thrilled that it's on back order, if they get it two days later than if it wasn't back ordered, well, that's not such a big deal after all, is it? So, it's really more of a global architecture perspective and saying, "This nuts and bolts caching expiration problem, how do we solve that?" but rather, "This overall business problem of how to handle orders and what do we do about it, how should we handle that from a global perspective while keeping all the pieces fairly simple?"

Richard Campbell: Yeah, I think this is why we make it as consultants because I'm able to spin that problem into features for the app that the customers are actually going to love.

Udi Dahan: Right.

Richard Campbell: That's really the thinking you've got to do. This is not just about solving the caching problem. It's about recognizing that a back order system and an order anticipatory system are good for business. We will sell more if we do it this way.

Udi Dahan: We will sell more, we'll be responding to orders faster, and this is going to decrease the number of customer service calls, "What's going on with my order? You said it was back ordered. Do you have the inventory yet?" It decreases the amount of customer service personnel that we have. It's just all goodness for the entire organization.

Richard Campbell: Everything is better. Udi, this is too much fun. We got to wrap it up.

Udi Dahan: Yeah.

Richard Campbell: So, Carl, did you learn anything today?

Carl Franklin: Yeah, absolutely. I learned that I kind of like sitting on the sidelines.

Richard Campbell: Oh, you do, do you?

Carl Franklin: Yeah, it's kind of fun.

Richard Campbell: So, I get the closing line today, is that what you're telling me?

Carl Franklin: Absolutely. Go for it.

Richard Campbell: All right. Udi Dahan, thanks so much for coming on .NET Rocks!

Udi Dahan: It was my pleasure.

Richard Campbell: And we'll talk to you next time on .NET Rocks!

[Music]

Carl Franklin: .NET Rocks! is recorded and produced by PWOP Productions, providing professional audio, audio mastering, video, post production, and podcasting services, online at www.pwop.com. .NET Rocks! is a production of Franklins.NET, training developers to work smarter and offering custom onsite classes in Microsoft development technology with expert developers, online at www.franklins.net. For more .NET Rocks! episodes and to subscribe to the podcast feeds, go to our website at www.dotnetrocks.com.