



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 # 26**

**July 26, 2006**

**Globalization/Internationalization with .NET**

Scott chats about preparing your .NET application for the global marketplace and Carl leads us into a discussion about character encoding.

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



**Our Sponsors**



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



<http://dotnet.con-sys.com>



<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 #26 recorded Monday, July 24<sup>th</sup>, 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](http://codesmithtools.com). And don't forget to visit [peterblum.com](http://peterblum.com) -- "Start with better controls, finish with better sites", online at [peterblum.com](http://peterblum.com); and .NET Developer's Journal -- "The World's leading .NET developer magazine", online at [www.sys-con.com](http://www.sys-con.com). In this episode, Scott discusses Internationalization.

(Music)

**Carl Franklin:** Hi! This is Carl Franklin, welcome back to another great episode of Hanselminutes, I am here with Scott Hanselman. Hi! Scott.

**Scott Hanselman:** Hola, Carl.

**Carl Franklin:** And I am never here with anybody else but you. It's all about Scott, this show, in case you haven't been listening. Today, the topic is 'Internationalization.'

**Scott Hanselman:** Si Carl.

**Carl Franklin:** Es verdad?.

**Scott Hanselman:** Si, es la verdad. Esta episodio es totalmente en espanol, completamente gratis para aqui.

**Carl Franklin:** Just kidding. So, what are we talking about today?

**Scott Hanselman:** Internationalization.

**Carl Franklin:** Good.

**Scott Hanselman:** Very exciting. More exciting in Spanish, but still...

**Carl Franklin:** We're in a little bit of a wacky mood.

**Scott Hanselman:** I am feeling -- rather wac... well, it was 104 degrees in Portland right now. So, I am looking for any opportunity to say a hace p punto net en espanol. And until that opportunity comes up, it's going to be in English today.

**Carl Franklin:** So, are we just talking about websites, or are we talking about Windows applications too; I know you are a web guy.

**Scott Hanselman:** Yeah I am a web guy, but everything that we are talking about for the most part is going to apply to just Internationalization as a concept; because underneath all of the stuff that we are talking about, there is the notion of the Current Threads culture -- and the Current Threads UI culture. So, we'll probably talk about ASP .NET and some of the general culture things, but mostly things apply to WinForms as well.

**Carl Franklin:** Okay, great. Let's get started.

**Scott Hanselman:** So, most people think about Internationalization as being just, you make a bunch of resources and you put all your strengths in another folder and it's all good. Does that pretty much sum it up for you?

**Carl Franklin:** Yeah, it's pretty much the way I thought about it. I haven't done it so.

**Scott Hanselman:** So, sometimes the terms Internationalization, Globalization, and Localization are all used interchangeably.

**Carl Franklin:** Yes, I have noticed this.

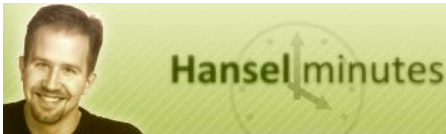
**Scott Hanselman:** Internationalization and Globalization are mostly synonyms. They are preparing your application to be localized. Localizing your application is the actual converting of your application to use strings from a particular language. So, I could write an application in English, have it globalized, but not yet localized. And then when I translate it into Spanish, it might then become a localized application.

**Carl Franklin:** So, Globalization is what you do to your application to make it localized to the user.

**Scott Hanselman:** Localizable.

**Carl Franklin:** Localizable to the user.

**Scott Hanselman:** Yeah exactly. There is all sorts of great places to start when you are learning about the stuff; there's a really great article by Wei-Meng Lee at OnDotnet, which is at [shrinkster/gve](http://shrinkster/gve). And this is a really great article; it talks about ASP.NET 2.0 localization and some of the improvements that they have made. Back in the day, if you wanted to change the current thread, the current executing thread to a different



culture, you had to do this kind of work yourself. Basically it worked like this, there is the `System.Threading.Thread.CurrentThread` tells you all sorts of information about the current executing thread. With ASP.NET, you are pulling threads of execution from a pool. So, even though Carl, might I go from `page1.aspx` to `page2` and have a sense of continuity, that sense of continuity on the server side is really artificial. You have things like the session object to maintain State -- and that thread of execution is going to be different; it needs to be setup for the user, before the page starts executing. So, in the past, it's been very common to sniff the browser and say, "What language does this browser speak?" -- because if you go to Tools options, Languages in Internet Explorer and in a similar menu in Firefox, people will setup a list of languages that they speak, in the order that they prefer to get them, and those are returned in a header, an HTTP header that's passed in every time you request something from your server. So, you look at this `Accept-Languages` Header, and on most systems it will say, `en` or `en-US` or `en-UK`, but for example, if you are visiting from Morocco, they may want English first, French second and Arabic third, or in some order like that. So, its really been the responsibility of the person on the server to look at that `Accept-Languages` Header and they typically use that with `Request.UserLanguages` and if they wanted the first one they would say, `Request.UserLanguages@zero`; they pull off that culture -- that's an ISO or International Standards Organization Standard Language Identifier, like `en-us` or `zh-hk` would be Chinese in Hong Kong. And they would take that and create what's called a `CultureInfo` Object. And that `CultureInfo` Object would get stuck on the current thread.

**Carl Franklin:** Now, this is very interesting; I always thought that localization happened at the application level; you are saying it happens at the thread level.

**Scott Hanselman:** Well, all of the goodness in Localization, as far as, pulling strings out, stems from a thing called `system.resources`. And `system.resources` is the namespace that handles all of the reading and writing of resources. It's got four major classes, `Resource Manager`, `Reader`, `Resource Set` and `Resource Writer`. And you typically call this function `GetString` -- so you say, make a resource manager and you call `GetString`. `GetString` looks at the current threads, UI culture for its resource selection. So, if I go and say, `System.Threading.CurrentThread.CurrentUICulture=blah blah blah en-us`, and when I ask the

resource manager later, it's going to go and look for `en-us` type stuff.

**Carl Franklin:** It makes perfect sense, because you may be serving on a website -- every thread is a different user and every thread can be looking at a different version of the site.

**Scott Hanselman:** Exactly. Now, the current UI culture is for resource selection. The current culture is a different thing -- and this can be a little confusing; it's the thing that handles formatting of dates, numbers and string comparisons. So, these can be different. You could say, "Well I speak French, but I want my dates and numbers and stuff done in Spanish." Now, with a WinForms person, this is done automatically for you. So, whatever the current system is setup for, you'll just inherit. You could force your application to work in English on a French system, by setting the culture manually yourself. Now, I had kept saying `en-us` as an example, but the `en` part is the non-locale specific part of things; it's just saying "English" -- it does not make any judgments about English where, just like I could say, `es` and that would mean Spanish. Well, `es-mx` means Spanish in Mexico versus `es-es`, which is Spanish in Spain. So, there is a hierarchy -- and that hierarchy can become very powerful. So, let's say that my application has 100 strings and I am going to have -- by default my application will be in English. So, I might have my default resources linked in with my main application. So, I have got 100 strings, name value pairs and they are all linked in; default's English. But I want to have three different strings of those 100, be different for people in the UK; I could make an `en-uk` satellite assembly, a separate assembly that would live underneath my `bin` folder with just those three different strings. And then, we'll have what's called resource fallback; if I request a string that's in that assembly I will get it; if I request a string that's not in that assembly, we'll fall back, using a standard fallback hierarchy.

**Carl Franklin:** Scott, is there a naming convention for these assemblies? Did you -- do you have to name it `en-uk.dll` or is it?

**Scott Hanselman:** Right, so there is a naming convention, typically it's `whateveryouwant.locale.dll.resources.dll`. So, I typically say things like, `constants.en-us.resources.dll` and someone now like, Michele Bustamante has a really good article on Best Practices and Internationalization -- and that is at [shrinkster/gv8](http://shrinkster/gv8). And she suggests things like a glossary -- separate assembly, and one for global things and one for local things and one for



constants. You can lay this out really any way that you want. With ASP.NET 1.0 it was all about you; you got to make the decisions, you had to do all the work, typically you would, in the BeginRequest Event, before any page got loaded, you would go and look at their cookies, see if they had overwritten something with some profile settings that you have selected. If they didn't have a specific preference, you would look at their Request.UserLanguages. And then you'd set the current thread up; so then later on, when the page actually got around executing, your thread will be prepared for you. See what I am saying? It wouldn't be up to the page to do that work, it would really be up to the BeginRequest, a module earlier in the process. Now in ASP.NET 2.0 all that's done for you because they have added new stuff on the page at the page directory where you go like, addPage and then you have all that information at the very first line of the page. You can say Culture = auto, UICulture=auto, and ASP.NET will automatically map those AcceptLanguage HTTP headers to a culture info object and put them on the current thread.

**Carl Franklin:** Now, Scott here is a question for you and you always see, you go to these big conglomerate websites and the first thing they do is, they ask you to pick a country. Is that more of -- you know they have different content based on what country you are in, or is that language? I mean, why are they asking you to pick your culture if the browser already knows what culture you are in?

**Scott Hanselman:** You have absolutely nailed my number one pet peeve about this. It is so stupid to get to a site and have them ask me what language I speak when they know darn well what language I speak. I also don't like it when applications say, if you have an awesome browser, and if you have a big monitor, go here. They know this stuff; JavaScript knows your monitor size, your browser knows -- it's just silly. If I have actually gone to the effort to put French as my number one language, well, for goodness' sake give me French. Google does a great job about this; not only do they let you select your own -- you go into the preferences section of Google from their homepage and say, I really want my language to be Zulu; so you can go in and the very first choice they have under preferences is, select the language, go all the way to the bottom, hit 'Zulu', hit 'save preferences', visit Google and it actually stays right at the top -- Google isiZulu. And I think that's so cool; they know that stuff about you. If you visit Google, by going tools, options, languages, and select any language, Google knows it. That's

really classy of Google. It would be lame if they put every flag at every country in the entire world up on your... So, personally we like to make that default to the language they say they speak and give them the choice to overwrite it. I think that's the right thing to do.

**Carl Franklin:** wikipedia.org has that, but is that a little bit different? -- because they have...

**Scott Hanselman:** Ah, so that's a very interesting point because Wikipedia has totally different amounts of content. So, they have much less Spanish than they do in English and...

**Carl Franklin:** So content seems to -- is cultural by nature, right?

**Scott Hanselman:** Right. So, in an instance like that, it would be appropriate in my opinion to sniff the browser using request.userlanguages, and depending on what contents you have available, redirect them to that separate part of your site.

**Carl Franklin:** And still let them choose other languages. If you are a French speaking English person, you might want to look something up in French.

**Scott Hanselman:** Exactly. Now that brings up a very interesting point because as a person involved in banking, we are dealing with folks that speak other languages but are typically using a certain kind of currency. So what if you were a German guy banking with a Japanese Bank? You know, ideally you'd want to see yen, and see it in German. But then it gets to an interesting question; and this is one of the little details about internationalization a lot of people forget about, is what about input? What do the dates look like, what do the decimals look like; displaying a date using the current culture is really easy. If you use a tool like Chris Sells' Format Designer at [www.shrinkster/guq](http://www.shrinkster/guq), you can use the 2 string features in Windows, so I can say daytime.now.2string "D" -- and that capital D is a format string that will say, "I want a long date." But it doesn't just say, I want a long date; it says, "I want a long date given the current culture." Now, in the States you'll get Monday, September 25<sup>th</sup> da da. But elsewhere, you might get an entirely different string and if you're speaking a language that's not even close to English, you are speaking Arabic or Hebrew or whatever, you are going to get all those characters as well. Now, it's very ethnocentric of us as Americans to assume that month, day, year is the way to go.

**Carl Franklin:** I was just going to say this; because I was interviewing Adam Cogan and he



says, do you know that the United States has the most screwed up date format in the entire world; it's different than everyone else in the world? Is this what you are going to talk about?

**Scott Hanselman:** Yeah, I was talking to Clemens Vasters and I said -- I was telling him how 9/11 really messed me up -- and he said, yeah, November 9<sup>th</sup> was a really rough day for me too.

**Carl Franklin:** Yeah.

**Scott Hanselman:** You know, and I was like, it kind of shook me for a second and then and I realized that who are we to take over an entire date?

**Carl Franklin:** Yeah. We do that a lot as a country.

**Scott Hanselman:** So here is the question; if you set your current culture to a specific, you know, date format -- what's called a date format info object, you are not just making a contract with the user that you are going to show, month, day, year or day, month, year, but that you're going to also accept them. And this can be a real problem for folks that don't think about that within the context of JavaScript validation. This is one of the primary reasons that I started using Peter Blum's validation controls.

**Carl Franklin:** Yeah.

**Scott Hanselman:** Is that he actually detects the current culture on the server side and generates the correct JavaScript to do localized JavaScript validation based on the date format.

**Carl Franklin:** Peter, you need to raise your prices man. You're giving the stuff away.

**Scott Hanselman:** Yeah, he seriously does need to raise his prices. That feature is huge if you are doing internationalization. Internationalization in JavaScript is hard, hard, hard -- no fun.

**Carl Franklin:** Yeah.

**Scott Hanselman:** He really made that easy so, I like that. Now want a little plug for myself though, and which calls back to an article that was written in MSDN about internationalization. The article was at [shrinkster.com/gup](http://shrinkster.com/gup) and I modified it at [shrinkster.com/guv](http://shrinkster.com/guv) as in "Victor" -- we have this notion of what's called pseudo-internationalization. Now, this is really cool. Typically people will start up on the project and

they'll say, we really want some data that translates this into Spanish.

**Carl Franklin:** And they all kind of start -- kind of half assed, and they will maybe change a couple of strings to OLA and then they'll say, we are done. Well, pseudo-internationalization will take a ResX file written in English and it will convert every single character to a high Latin character.

**Scott Hanselman:** Right.

**Carl Franklin:** A non A-Z character like ũ or "Schwa" or just whatever and turn it into a funky character, but it's still readable. So go to [www.shrinkster.com/guv](http://www.shrinkster.com/guv) and check this out, I think you will appreciate it. And it gives you an example of what these kind of things would look like. So, I might have a string like transaction download -- and what I want to do is, not only change the string, so I can prove that my application works with non-Latin -- you know, high order characters, non-centric characters -- doesn't mess up the encoding, but for sentences of a certain length, you need to expand them; you need to make them fatter, because other languages might make a sentence as big as a 30-40 percent larger; so a pseudo-internationalizer will not only take all of your English and convert it into a funky language -- a pretend language, it's a pseudo-language, but it will also make your strings longer. This allows you to see -- if I translate this into German, is it going to make my wrapping look bad?

**Carl Franklin:** Right.

**Scott Hanselman:** If you put this in the context of a continuous integration system, where you are doing your builds automatically, you can automatically pseudo-internationalize your entire application and know what it will look like in a foreign language without having to have a really tight loop with the internationalization folks. Do you see what I'm saying?

**Carl Franklin:** Yeah.

**Scott Hanselman:** With this particular format, which is so funky but still very readable, English stands out on the page, and that's how you go, "Ooh, I missed that label. Oh, I totally forgot this button." See what I'm saying? It's a very, very nice technique.

**Carl Franklin:** So Scott, let me ask you a question about Unicode because most westerners and most people in the United States only experience the ASCII Internet, you know the



English Internet but if you go to like, a Korean or a Japanese or a Chinese website, you are going to find all sorts of strange characters in -- and it is because their alphabets are bigger than 26 letters or 256 possible combinations or characters. How does Unicode help us in .Net?

**Scott Hanselman:** Well, I will explain a little bit of it to you but for the full on version check out joelonsoftware, he has got a great article at [shrinkster/gv6](http://shrinkster/gv6). The article is entitled modestly the absolute minimum every software developer absolutely positively must know about Unicode and character sets, no excuses.

**Carl Franklin:** (Laughter).

**Scott Hanselman:** Sounds like something that I would have written but he beat me to it by about 3 years. So, you know, the idea is that back in the day right, when EBCDIC was on the way out - - when EBCDIC was on the way out, ASCII was really the way things were happening where letter A started at 65 and the kind of stuff that you would see in DOS and you'd see in Charmap if you look at the first 0-255.

**Carl Franklin:** And we are talking about the United States here.

**Scott Hanselman:** Well, we are talking about the United States to start with because we are the ones that said, "Oh, it's going to be this way."

**Carl Franklin:** Right.

**Scott Hanselman:** Right. But you bring up a very valid point because I did a lot of work in like, '91-92, we were translating applications like any "C" machines that spoke Japanese and you had to load these things called codepages up into RAM and mess with them basically on a one PC - - one character code would be NE with an "accent e gu" and on another one it would be Hebrew letter this and that. They would assign different meaning to different characters; so we might think "A" was 65 but they might disagree. They would have a code page that would map those codes for their purposes. Usually everyone would leave the 0-127 numbers to themselves but they figured above -- you know above their 128-255 was pretty much just a playground.

**Carl Franklin:** Yeah.

**Scott Hanselman:** So then you would have these code pages; and there is a whole pile of them if you can go "Google" for, you know, DOS code pages, find that there is lots of different code pages out there, mostly one for different

country. You know Israel had their own yada, yada, yada. So this was just a way of putting a pair of rose-colored glasses on top of some encoded data. So, sometimes you will do that now, you will arrive at a page and it will look like crap and then you'll go up to the View Menu and say encoding more, and there is a whole list of all the different code pages that could still be encoded out there on the internet; if you do that now, you would be surprised how big the menu is on your browser -- just go View, Encoding, More, and then this huge thing flies open that tells you all these code pages that were out there.

**Carl Franklin:** And we are still talking about mapping over a single octet, right?

**Scott Hanselman:** Exactly, exactly. This is -- the server didn't tell me enough information, I couldn't glean it so, you know better.

**Carl Franklin:** Yeah.

**Scott Hanselman:** So then we started doing the whole double byte character set. That's when I started doing the Japanese stuff myself where some of the letters were stored in 2-bytes and then it became really difficult because when someone said, what is the length of a string, or what you mean by length, do you mean characters, do you mean bytes?

**Carl Franklin:** Binary length -- and this is, I remember adding some code at Crescent Software that we had some Japanese port of some of our tools and this was a problem, it was double-byte characters, but it was still within the context of ASCII.

**Scott Hanselman:** Yeah. So Unicode introduces this notion I think called a code point, which is just -- you know a number that represents a glyph, right -- a particular letter in some language. But it doesn't necessarily say anything about how it's going to be encoded; it kind of leaves that part up to the implementers.

**Carl Franklin:** Yeah.

**Scott Hanselman:** So, Unicode isn't saying it shall be encoded like this, it's just a catalogue of all of these different code points that say, "A is U41; Y just 'cuz -- and what A ends up looking like on disc is -- what was up to some other people. And then it was the whole issue of high end'ian and low end'ian.

**Carl Franklin:** Can you explain that again?



**Scott Hanselman:** So, Unicode goes with -- assigns a number to every letter in all the different alphabets and celebrities that are out there.

**Carl Franklin:** So there is actually a definition of every letter in every culture in Unicode.

**Scott Hanselman:** Every one that has been submitted to the Unicode organization, they've said yes, that's cool.

**Carl Franklin:** I see.

**Scott Hanselman:** So, if you go up and look up at Unicode.org; I think Ethiopian and Harrick got in recently [Unicode.org](http://Unicode.org) -- and there is a whole list; I think Klingon's in there probably now.

**Carl Franklin:** You're probably right.

**Scott Hanselman:** So, you go click on Unicode character database; they just got version 5.0 out, you can click on it, you can go and say well, what's changed, what's new; code charts, and then go in and there is a whole online edition that you can order and check out the database full of all these different things. So if there is a particular character that you are fond of, you know, some Tamil character or whatever, that has a number that's assigned to it. Right?

**Carl Franklin:** Right.

**Scott Hanselman:** "A" is 41; that's the only one I know. I assume "B" is 42, I'd have to look it up. You can find all this with Charmap right, if you just go File - Start, Run, Charmap, that's all up there. So, those numbers though, don't necessarily say how that will look on disc. Now "A" is 41, which means that it would be really nice to go and say, "Hey, "A" is going to be 41 on disc."

**Carl Franklin:** Right.

**Scott Hanselman:** But if we go and say, "Well, Unicode is -- everything's two bytes, because we are going to do UCS-2 and everything will be two bytes, then "A" would be 0041. But are we talking about big-endian or low-endian -- at which end is the most significant bit? So then it could be 0041 or 4100; so this is when you see things like -- that refer to the Unicode BOM, the B-O-M -- that's the 'Byte Order Mark.' This is a weird thing, but it's basically, we'll put 2 bytes at the beginning of any Unicode string that will tell you whether or not we are big-endian or little-endian.

**Carl Franklin:** Its amazing to me that in 2006 we are still dealing with the difference between big-endian and little -- can we settle on this now? Come on.

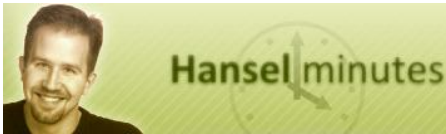
**Scott Hanselman:** You know that Mac's are big-endian and the Internet is big-endian. So Mac's don't have to do any work to translate stuff as they drop it on to the Internet, because TCP/IP at that low, low, low level is big-endian, but Intel is little-endian. So, the network cards or the software -- somewhere deep in the stack, I don't know, I think its in the card now, but it used to be in the stack or as things headed out, we had to flip it. And as it came back in, we flipped it back; so like every single byte that was going out, was getting moved, flipped around, because Intel is not the same as TCP was.

**Carl Franklin:** And it looks like that's not going to change anytime soon.

**Scott Hanselman:** Do, I care? Any problem in computer science is solved by one additional layer of abstraction; so, hide it from me, lie to me, and I am cool with that. So, this Byte Order Mark, that was really important, because sometimes you will see this in a wild; not always you will see this -- but that's one of the reasons I really like Notepad2, because if you run Notepad2, at the very bottom of Notepad2 in the status bar, it tells you whether or not you are looking at a Unicode file or UTF-8 file, and if you go file encoding within Notepad2, you can switch back and forth between these different encodings. Two different times this week I had to debug a problem where someone thought a file was saved a certain way, but it wasn't. And it turned out it was because their Text Editor wasn't advertising that fact. Right? They could go, 'Save As,' and at the bottom of the Save As dialogue select 'UTF-8' for their encoding, but it kept getting encoded as ANSI and they lost data. So, they opened up back again and they ended up getting black squares. So anyway, those code points -- sorry, you were going to say...?

**Carl Franklin:** I just said that's really lame.

**Scott Hanselman:** This is life though. The thing is, if you stick with UTF-8 and you read this article by Joel and you use Editors that understand UTF-8, nine times out of ten you'll be fine. Things go wrong, when you don't realize that a string is in a certain encoding; you make an assumption. Like, you might do UTF-8 everywhere, but maybe you are talking to some old database or some old back end system, and you have been thinking, "Yeah, we internationalized, we are great," but you have



never done a test, you have never taken the word, "Jose" and sent it through the system, saved it and brought it back. So you never really know -- and then one day, you send "Jose" in, and it comes back "Hose", and then you are screwed or "Hosed" at that point, because you never tested it. So, always include some internalized data in your testing, and round trip it, it's so important. Another really common thing people do is, they will make an XML file, they will retrieve an XML file from somewhere and they will look at that first line, the prolog there where it will say, `? xml encoding=UTF-8`. But there is never anything inside that file that is in fact Unicode or UTF-8. So, they will never know that their system ultimately doesn't understand it. I worked with a number of mainframes that were bringing back XML data, but they were just string concatenating the data and they just assumed that that line at the top was important and they would just stick UTF-8 at the top, but they couldn't handle anything other than ASCII.

**Carl Franklin:** Now, this might be a kind of a strange -- or I don't know, if I am talking under or over people here, but I want to know, what's the difference between all these different formats, UTF-8, ANSI/ASCII; give us a little primer.

**Scott Hanselman:** So, we get -- read this Joel On Software article, it's great, but ASCII was basically an agreement that 0 through 255 would be these characters. That became the ANSI standard, right? Everyone agrees that we are going to do one thing up to 127, and then from there on, knock yourself out. Unicode gets together and figures out all these different code points and then comes up -- and this is important, comes up with a number of ways to encode the same data. So, you can take this -- a Unicode code point and save it as a file as Unicode or save it as UTF-8 or UCS-2 or whatever.

**Carl Franklin:** And UTF-7 also, right?

**Scott Hanselman:** Yeah it's UTF-7. So, there is all these different ways, and you are not necessarily losing information, you are just encoding it differently. Now, here's a really interesting way, if you want to understand how to do this, check this out; go into, like a Notepad or Notepad2, and type in just like ABC -- I am doing this right now -- type in ABC, you go, File, Save As, and we are going to go -- I want to say that this file is going to be a Unicode file -- that usually doesn't mean UTF-8; usually means Unicode with a Byte Order Mark. I'm going to save it on my -- and I save it with a very small file name -- this is very important -- save it with like, 1.txt. Then go to the DOS prompt, go to command.exe,

go to the directory where your file is and type in 'Debug' -- just literally debug 1.txt -- and then its going to churn for a second and you are going to get a little hyphen that will pop up. Then push 'D' -- D for Dump, and you will see the contents of your file. So, I've just done that; you see the actual hex values right? So, I am looking at this, and I see, FF-FE -- that's the first 2 bytes of my file.

**Carl Franklin:** Those are those 2 bytes that tell you whether it's big-endian or little-endian right?

**Scott Hanselman:** Exactly. So, I can see that this has FF-FE and then, we know that we are dealing with a little-endian system, right, little-endian encoded at this point. Okay, so, now those first two bytes say, FF-FE, then I see 6100, I know 61 in this case is "A"; and I see "00", because I explicitly told my Editor to save this as -- 61's lowercase "A". I told my Editor to save this with 2 bytes. So, we have encoded a code point on the disk and we are ending up using 2 bytes. So now, I want to take that exact same file, I am going to go in Notepad2, I am going to say, encoding UTF-8, I want to save this exact same file as 2.txt, go back in, run debug again on the new thing debug.txt, and you can use a Hex Editor, but I like debug. So now...

**Carl Franklin:** Of course you do.

**Scott Hanselman:** Of course I do; I am just saying that its there, right, why go get a Hex Editor and try to impress somebody. So, debug 2.txt and then suddenly I see 61, 62, 63 -- ABC -- now, here's the kind of philosophical question right, like if a tree falls in the woods, does it make a sound and if no one is there to hear it -- is this UTF-8 encoded, its just 3 bytes.

**Carl Franklin:** Yeah, it looks like ASCII.

**Scott Hanselman:** Exactly. So UTF-8 really is a way to encode stuff where it says, use 2 bytes if you need 2, but it was an evil thing because it was set up such that English text is identical in UTF-8 as it was in ASCII. So, basically things worked for us.

**Carl Franklin:** Yeah that's a little weird.

**Scott Hanselman:** Yeah. So for ANSI, for ASCII -- for folks that were doing stuff before...

**Carl Franklin:** Now, if in the middle of ABC you wanted to put a Chinese character, would you have the 2-byte header at the beginning?

**Scott Hanselman:** So, let's find out right?



**Carl Franklin:** And the other question is, if you didn't have those -- okay, well, FF and FE will never be ASCII.

**Scott Hanselman:** Well, there is no Byte Order Mark on this particular UTF-8 file, because it didn't need it. Now, the reason that they did this was because they didn't want it to break existing code. Right? So, what's nice about a UTF-8 encoded string is that if it's just got low ASCII stuff in it, then existing code will work just fine. So, now I am just going to go to [china.yahoo.com](http://china.yahoo.com) and I will just grab the little "search here" text that is sitting there at the top, or just cut a little Chinese character out, just one character, and just copy it into my clipboard, and I'll go back to my 2.txt and I'll put it between A and B. So, I am looking at this in notepad, I see "A", Chinese character "B" "C" and I will save it; save it in UTF-8 and will go back and will say debug2.text, 'D' for Dump. So, I see 61-E585B3-62- 63; so that Unicode code point for that particular Chinese character was 3 bytes; because UTF-8 isn't really saying, use two bytes when you need to, it's saying, use more than one byte when you need to.

**Carl Franklin:** But I don't see any byte order of bytes -- markers.

**Scott Hanselman:** You don't see any byte markers. So, typically when you have a string, there is a couple of things you can do, it's a little complicated but some systems basically guess -- okay, on Windows we know that, you know, the processor is a certain way, so we pretty much know that all Windows code can be done a certain way, so we don't really think about it that much.

**Carl Franklin:** They don't guess, they infer.

**Scott Hanselman:** Well, they infer yeah. Now, if I go on to Notepad2 and I go to File encoding, I have two choices; I had UTF-8 and I had UTF-8 with signature. So I go and switch that to UTF-8 with signature and run debug 2.text again and look at it, then I see the EF and the different stuff at the beginning, I have the little Unicode signature at the front, and then we get into our text. So, how many choices that I have there right, and here's the trick though; if I go and I say, encoding ANSI, I get a warning; switching from ANSI to non-ANSI and back will cause loss of data; I hit 'Yes', suddenly my Chinese character just turned into three funky characters; it split apart our lost data and it turned to crap.

**Carl Franklin:** So Scott, do you know about the bugging notepad or -- I don't know if it's a bug or the behavior, that this API can break bug, you heard of this?

**Scott Hanselman:** Yeah, this is the one where you type in a certain phrase and because the phrase falls a certain way...

**Carl Franklin:** Yeah.

**Scott Hanselman:** It confuses the system because of the order of bytes.

**Carl Franklin:** Yeah, and I believe it's like, four character word, a three character word, a three character word and then a five character word.

**Scott Hanselman:** Yeah, so if you google for a quote 'This API can break' you will find lots of information about that; but if we go into Notepad and we type in, 'This API can break', we save it in a file called fu.text and then we open it, we close it and we open it again, we get back Chinese.

**Carl Franklin:** Interesting.

**Scott Hanselman:** So, if I go out to the command line and I type in debugfu.text, the file that I just put that into, and I hit 'D', I see what looks to me like some pretty regular looking ASCII code, nothing particularly interesting about that. So the deal is that there is a function in Windows that's called "Is Text Unicode?" So basically, you pass it some data and it tells you whether or not that data is encoded as Unicode; really it tells you whether it's UTF-16 or not, okay? But the deal is, like we mentioned before in the talk that it runs some -- it guesses basically.

**Carl Franklin:** Right.

**Scott Hanselman:** Guessing in Computer Science is called Heuristics, right?

**Carl Franklin:** Right.

**Scott Hanselman:** And we guess and it looks at it, and some short strings, if it doesn't have a lot of information, very short ASCII strings that have an even number of lower case letters, then it will spit it out. Now, if you look at the documentation of course, it says that it's not fool proof, and a lot of them are based on statistics; so if you pass in not enough information, it doesn't get enough data to build any kind of statistics.

**Carl Franklin:** I've actually had this bug occur to me during a demonstration where I was using



text files on the desktop to illustrate what objects are with shortcuts, and reference types for shortcuts and I would pull up the file that I had just saved like a little bit of text too, and I got 4 or 5 blocks.

**Scott Hanselman:** Oh yeah, totally.

**Carl Franklin:** Yeah.

**Scott Hanselman:** Putting in this app can break or whatever -- is a funny demo, but if you just put in the letter 'A' and then a carriage return line feed, that will fail -- it's three characters, it wasn't nearly enough.

**Carl Franklin:** Right.

**Scott Hanselman:** Now they do point out though in the documentation that this is not fool proof, it is statistical analysis; they tell you that it could break and they do point out that failure would have been preferable, you'd rather to get garbage English back than garbage Chinese because you don't want to come up with some string that might actually mean something.

**Carl Franklin:** Right... true.

**Scott Hanselman:** Yeah, that's definitely -- an example that this kind of stuff is not easy; there is a really great blog we mentioned in the last show by Michael Kaplan, one of the internationalization wonks at Microsoft at [shrinkster/guw](#) and his blog is called "SORTING IT ALL OUT" -- really interesting stuff about some of the obscurity and craziness that happens within internationalization and just general localization problems on Windows.

**Carl Franklin:** Scott, before we wrap up, let me ask you about Vista. What's new in Vista for localization?

**Scott Hanselman:** Well, there is piles and piles of new stuff in Vista, there is the notion of being able to have a custom locale, because Vista supports something like 200 different locales in 100 different languages, but that's just a tiny fraction of all the different languages in the world. So there is a couple of things, one of them is in Beta, if you are running a Beta of Vista you can get the Locale Builder at [shrinkster/gv1](#) to create a Locale if you live in a particular island off the coast of -- I don't know, Halifax, somewhere. You might want to create your specific locale or if you decide to start your own country Carl, you are going to want to add one to the system. The Locale Builders in Beta, you can take a look at that. A couple of things that you can use now that

are really convenient; if you deal with globalization issues, which aren't just strings, but also things like time zones.

**Carl Franklin:** Sure.

**Scott Hanselman:** You've personally had a number of problems. We suffer in DasBlog dealing with time zones because there is the time zone at the server's in, the time zone at the reader's end, and the time zone at the author's end; there is a really nice time zone utility. Microsoft Time Zone Utility [shrinkster/gux](#) sits in the tray and let's you know what the time is in other time zones without having to change your own times -- that's a really convenient tool. The stuff up at the Globaldev site is really valuable; I think it's one of the little known secrets on MSDN, there is the NLS information page, is a really interesting page to look at, [shrinkster/gv2](#) which has a list of all the different languages and Locales that Windows supports, and all of this is coming off of the Global Development and Computing Portal at [shrinkster/gv3](#) including a number of glossaries that they'll give you for free at [shrinkster/gv4](#), which are basically all the different Microsoft terminologies pre-translated into other languages. So, if your translator needs to know what -- the appropriate term for File, Edit, View, Help whatever, is in the Microsoft parlance, they have got glossaries that they will give you up at [shrinkster/gv4](#)...

**Carl Franklin:** Great!

**Scott Hanselman:** ...for all the different standards because everyone has a different feel and you may not necessarily get a localizer who is familiar with the terminology that Microsoft uses.

**Carl Franklin:** One last thing; do you use the Gregorian Calendar Object for date arithmetic?

**Scott Hanselman:** Well, we do some work in Thailand, so we actually have to deal with that calendar as well as the Thai Buddhist Calendar, which is a totally different calendar.

**Carl Franklin:** It's a pretty awesome object, it's in `system.globalization.gregorianCalendar` and allows you to add epochs and eras and, you know, serious date arithmetic in globalization.

**Scott Hanselman:** All the globalization stuff is really, really good. Currency info, date format info, there is a lot of really cool stuff you can do where you could say, well I like about this culture but I want to use dollars -- or everything about



their current system except the date format, I want to be this way.

**Carl Franklin:** Yeah.

**Scott Hanselman:** You can basically hardcode these things if you want your app to work a certain way; there is a lot of good stuff in there... A couple of other things, one that hasn't been updated in a while, the Enterprise Localization Toolkit by a guy named Todd Abel has basically been dropped and forgotten about, but a lot of people use it, [shrinkster/gv9](#), basically it is a database, a SQL database that will hold all of your strings and then generate resources for you. Sometimes people find dealing with their resources in a database rather than in XML, is a lot easier to deal with; definitely we are checking out. And then one thing I didn't get a chance to talk about, which is something I am really interested in is, text direction; going left to right, right to left.

**Carl Franklin:** Oh!

**Scott Hanselman:** Up at [shrinkster/gv5](#) there is a lot of really good information, and if you are going to do any kind of HTML for Middle Eastern Content, for anything that's going to be in Arabic or Urdu or Persian, take a look at [shrinkster/gv7](#) there is a specific site called "Authoring HTML for Middle Eastern Content."

**Carl Franklin:** And that's a show, I guess Scott huh?

**Scott Hanselman:** Yeah, that was a little longer than I thought but it's a really interesting topic, I can go on for hours. I didn't get some of the stuff I wanted to talk about, maybe we will do a part 2 one day.

**Carl Franklin:** Oh, we'll fill the links up there anyway, so...

**Scott Hanselman:** Absolutely.

**Carl Franklin:** On behalf of myself and Scott, have a great week, and we will see you next week on Hanselminutes.

(Music)