Active Directory Phone Numbers and Line URIs: Together at Last!

CSPShell 1 Feb 2011 10:41 AM 1

If you were to tunnel deep enough under the Microsoft campus (note: please don't tunnel under the Microsoft campus, or at least don't tell anyone we suggested it), you would eventually run into the concrete bunker that serves as the Lync PowerShell Fortress of Solitude.

Note. You'll also find thousands of copies of Windows ME buried down there as well. But that's another story.

Inside the Fortress of Solitude, teams of dedicated Lync Server PowerShell blog writers monitor the world of Lync Server PowerShell, 24 hours a day, 7 days a week. Whenever they hear about someone who has a problem or a question involving Lync Server PowerShell, they immediately dispatch a highly-trained Lync PowerShell SWAT team, a SWAT team that can be airlifted and parachuted into your facility within hours. Once your facility has been secured, the SWAT team will fix your problem or answer your question, then disappear into the night, awaiting their next assignment.

Note. OK, it's possible that the preceding description could be a *bit* of an exaggeration. But you have to consider things from our point of view: it's mid-year review time here at Microsoft, and the things that we really *do* do – write a daily Lync Server PowerShell haiku and ask you which PowerShell cmdlet is not like the others – don't stack up very well against the things other technical writers do around here.

Oh, and in case you're wondering, no, we haven't been able to talk Microsoft into buying us parachutes. However, the company doesn't seem to have any problem with us jumping out of airplanes without those parachutes.

But while we may not have a Fortress of Solitude, from time-to-time we *do* get questions about Lync Server PowerShell, and while we might not be a real SWAT team, whenever we *do* get a question we sit down and try to come up with an answer. (And, before you ask, yes, we usually try to come up with the *right* answer.)

For example, just the other day we received this email:

"Do you have or can you post a PowerShell script that will go through AD and find all enabled Lync users and then set their LineURI as a normalized version of their Office Phone number? … This would help us out, otherwise we are going to have to sit and do this by hand, not to mention when the Office Number changes it will have to be manually changed in Lync. I'm sure I'm not the only one who could use something like this."

So *can* we do that? Can we write a script that will find each user who has been enabled for Lync Server, grab that user's phone number (as currently configured in Active Directory), and then assign that phone number to their LineUri attribute? Beats us. But we'll definitely give it the old college try.

Note. Which sounds pretty good, unless you know how hard we actually tried when we *were* in college.

Let's start with a very simple scenario. Suppose we have two user accounts that have been enabled for Lync Server. Those two users, and the Active Directory phone numbers that have been assigned to them, are shown below:

Display Name	Phone Number
Ken Myer	1-(206)-555-1219
Pilar Ackerman	1-(425)-555-0712

What we want to do is take those existing phone numbers, convert them to the line URI format, and then assign those line URIs to their respective users. For example, Ken Myer has the phone number 1-(206)-555-1219. When our script finishes running, he should also have the LineURI TEL:+12065551219. What kind of script can do that for us? This kind of script:

```
$enabledUsers = Get-CsAdUser -Filter {Enabled -ne $Null}

foreach ($user in $enabledUsers)
    {
        $phoneNumber = $user.Phone
        $phoneNumber = $phoneNumber -replace "[^0-9]"
        $phonenumber = "TEL:+" + $phoneNumber
        Set-CsUser -Identity $user.Identity -LineUri $phoneNumber
    }
```

Let's see if we can figure out what's going on here. To begin with, we use the following command to return information about all the users who have been enabled for Lync Server:

```
$enabledUsers = Get-CsAdUser -Filter {Enabled -ne $Null}
```

Now, you might be looking at the preceding command and wondering: a) why did they use Get-CsAdUser instead of Get-CsUser; and, b) why did they use the filter **{Enabled –ne $Null}**? After all, if all we want to get back are

the users who have been enabled for Lync Server, shouldn't we use this filter instead: **{Enabled –eq $True}**?

Believe it or not, we have good reasons for using Get-CsAdUser and for using the filter we chose. (And yes, that might be the first time we *ever* had a good reason for doing something.) Because we're interested only in users who have been enabled for Lync Server, you might have expected us to use the Get-CsUser cmdlet; after all, those are the only kind of users Get-CsUser *can* return. However, if we want this script to actually work (which we do), then we need to retrieve each user's current Active Directory phone number. Get-CsAdUser returns that phone number for us; Get-CsUser doesn't. That's why we need to use Get-CsAdUser.

As for the filter we chose, well, the Enabled attribute tells whether or not a user has been enabled for Lync Server. The Enabled attribute can have one of three values:

- **$True**, which means the user has been enabled for Lync Server.
- **$False**, which means the user has been enabled for Lync Server, but his or her Lync account is temporarily disabled.
- **$Null**, which means that the user has *not* been enabled for Lync Server.

As you can see, if we looked for users where the Enabled attribute is equal to $True, we wouldn't get back users whose accounts have been enabled for Lync Server but are temporarily disabled. Therefore, we decided to look for all the users where the Enabled attribute is equal to either $True or $False; in other words, where the attribute is *not* equal to (-ne) $Null.

Note. Confused? We don't blame you. For more information on how the Enabled attribute works take a look at the article When is a Boolean Not a Boolean?

After we execute the first line in our script we should have information about all the users who have been enabled for Lync Server; that collection of data then gets safely tucked away in the variable $enabledUsers. In the next line of code, we set up a foreach loop that loops through all the users in that collection. Inside that loop, the first thing we do is grab the user's Active Directory phone number (for example, 1-(206)-555-1219) and stash that in a variable named $phoneNumber:

```
$phoneNumber = $user.Phone
```

This is where it gets interesting. (Or, should we say, this is where it gets even *more* interesting.) As you know, line URIs are finicky little things: they have to be formatted just right or Lync Server will reject them. In particularly, line URIs must start with the prefix *TEL:+* and then be followed by a phone number consisting of the country code, area code, and the actual phone number. Oh, and that country code, area code, and phone number? Digits only. No blank spaces, no parentheses, no hyphens, nothing but numbers.

Which means we have two problems here. First, Ken Myer's Active Directory phone number – 1-(206)-555-1219 – doesn't start with the *TEL:+* prefix. Second, that phone number contains some invalid characters: (,), and -. Before we can turn this phone number into a line URI we need to fix both of these problems.

Let's start off by tackling the second problem: the presence of invalid characters. What we need to do here is take our Active Directory phone number and remove everything that isn't a number. That sounds like a pretty daunting task; the truth is, it requires just one line of code:

```
$phoneNumber = $phoneNumber -replace "[^0-9]"
```

What we're doing here is using the –replace operator and the regular expression **[^0-9]** to locate everything that isn't a number and replace it with, well, nothing. We know that we're searching for everything that isn't a number because of the regular expression **[^0-9]**; that value tells PowerShell to use the numeric range 0 through 9. The caret symbol (**^**) tells PowerShell to look for characters that *aren't* in the range 0-9. If we wanted to look for characters that *are* in the range 0-9 (which we don't) we'd simply omit the caret:

```
"[0-9]"
```

And to make sure that non-digit characters are deleted (that is, replaced by nothing) we don't include a replacement parameter. Suppose, for some reason, we wanted to replace non-digit characters with the letter *X*. In that case, we'd use this syntax:

```
-replace "[^0-9]", "X"
```

Which is probably more than you really need to know at the moment.

And so what do we end up with when we remove all the non-digit characters? This:

```
12065551219
```

As you can see, that's *almost* a line URI; the only problem is that it doesn't start with the *TEL:+* prefix. But that's easy enough to fix:

```
$phonenumber = "TEL:+" + $phoneNumber
```

Needless to say, there's nothing too terribly complicated about *that* command. All we're doing is adding the prefix *TEL:+* to the value stored in the variable $phoneNumber. That means $phoneNumber now looks like this:

```
TEL:+12065551219
```

And that value *does* look like a line URI, doesn't it? Which, in turn, means that all we have to do now is assign this value to the LineUri attribute. You know, like this:

```
Set-CsUser -Identity $user.Identity -LineUri $phoneNumber
```

If we take a peek at Active Directory right now we should see something that looks like this:

Display Name	Phone Number	Line URI
Ken Myer	1-(206)-555-1219	TEL:+12065551219
Pilar Ackerman	1-(425)-555-0712	

As you can see, Ken Myer now has a line URI based on his Active Directory phone number. Granted, Pilar Ackerman doesn't have a line URI yet. But that's because we've only made one trip through our foreach loop. As soon as we loop around we'll repeat the process with Pilar's account and she'll end up with a line URI based on *her* Active Directory phone number:

Display Name	Phone Number	Line URI
Ken Myer	1-(206)-555-1219	TEL:+12065551219
Pilar Ackerman	1-(425)-555-0712	TEL:+14255550712

Etc., etc.

Now, we should note that there *is* one potential complication here. Our script assumes that you've been storing complete phone numbers in Active Directory; that is, phone numbers that have the country code, area code, *and* the phone number. As long as you use that format our script will work; for example, it can convert any of these phone numbers to line URIs:

```
1-(206)-555-1219
1-206-555-1219
1.206.555.1219
1(206)555-1219
```

And so on.

Now that's great, but what if you don't use the same format we used in our preceding example? Well, in that case, you're going to have to modify the script a little. But don't despair; we'll make a deal with you. We'll show you two additional examples for normalizing your Active Directory phone numbers to the E.164 format used in a line URI. If those examples don't work for you, send an email to cspshell@microsoft.com, making sure to give us an example of how your phone numbers *are* formatted in Active Directory. We'll hand that email over to one of the SWAT teams and with any luck, within a day or two someone will parachute into your office and hand you the solution.

Or, depending on the budget, someone will email that solution to you.

Note. And we'll also publish any additional scripts here in the Lync Server PowerShell blog, just in case anyone else is interested.

Let's start by looking at an organization that formats phone numbers like this:

Display Name	Phone Number
Ken Myer	51219
Pilar Ackerman	60712

As you can see, this organization doesn't store complete phone numbers in Active Directory; instead, they store only extension numbers. And just to make things a little more complicated, it turns out that the area code varies depending on your extension number: if your extension starts with the number *5* then you're in area code 206. If your extension starts with the number *6* then you're in area code 425.

So how are going to go about assigning line URIs in *this* organization? We're going to go about it like this:

```
$enabledUsers = Get-CsAdUser -Filter {Enabled -ne $Null}

foreach ($user in $enabledUsers)
    {
        $phoneNumber = $user.Phone
        $phoneNumber = $phoneNumber -replace "[^0-9]"
        $extension = $phoneNumber -match "^5"
        if ($extension)
            {
                $phoneNumber = "TEL:+120655" + $phoneNumber
            }
        else
```

```
            {
                $phoneNumber = "TEL:+142556" + $phoneNumber
            }

        Set-CsUser -Identity $user.Identity -LineUri $phoneNumber
    }
```

Most of that script should look familiar to you; after all, much of it is copied line-for-line from our previous script. (Of course, we're assuming that you *did* look at our previous script.) The only real difference comes here:

```
$extension = $phoneNumber -match "^5"
if ($extension)
    {
        $phoneNumber = "TEL:+120655" + $phoneNumber
    }
else
    {
        $phoneNumber = "TEL:+142556" + $phoneNumber
    }
```

In this block of code we're checking to see if the extension number starts with a 5; that's what the regular expression **^5** is for. Suppose the first number in the extension *is* a 5. In that case, that means that the user is in the 206 area code and also has the telephone prefix *555*. In that case, and in order to turn this phone extension into a line URI, we need to prefix the extension number with *TEL:+120655*. Like this:

```
$phoneNumber = "TEL:+120655" + $phoneNumber
```

If we do that, we'll end up with a phone number that like this one:

```
TEL:+12065551219
```

And if the extension *doesn't* start with a 5? Well, in this somewhat-simplistic example, that can only mean that the number is in the 425 area code and must have the telephone prefix *556.* This line of code takes care of formatting extensions that don't start with a *5*:

```
$phoneNumber = "TEL:+1425556" + $phoneNumber
```

Got all that? Good. Now let's do one more. Here's a fairly typical way of storing phone numbers in Active Directory; it includes the complete phone number *plus* an extension number (the number following the *x*):

Display Name	**Phone Number**
Ken Myer	1-206-555-1111 x1219
Pilar Ackerman	1-206-555-0712

So how do we deal with Ken Myer's phone number, which is a phone number *and* an extension? Well, for starters, we need to tell you that we might have … misled … you earlier in this article, back when we said line URIs can only contain numbers. That's *mostly* true, but there are exceptions. For example, this is not only a valid line URI, but it's *the* way you need to format line URIs that include both a phone number and an extension:

```
TEL:+12065551111;ext=1219
```

We need to make Ken Myer's phone number look like that. And here's how we're going to do it:

```
$enabledUsers = Get-CsAdUser -Filter {Enabled -ne $Null}

foreach ($user in $enabledUsers)
    {
        $phoneNumber = $user.Phone
        $phoneNumber = $phoneNumber -replace "[^0-9|^x]"
        $phoneNumber = $phoneNumber -replace "x", ";ext="
        $phonenumber = "TEL:+" + $phoneNumber
        Set-CsUser -Identity $user.Identity -LineUri $phoneNumber
    }
```

There are two differences between this script and the script we originally showed you. For starters, there's this line of code:

```
$phoneNumber = $phoneNumber -replace "[^0-9|^x]"
```

What's the deal with that line of code? Well, as we know, Ken's phone number starts off looking like this:

```
1-206-555-1111 x1219
```

As we also know, we need to remove all the non-digit characters from this number – well, all of them *except* the letter *x*. Why can't remove the letter *x*? We need to keep the letter *x* in there so we know where the extension

part of Ken's phone number begins. That's why we use the regular expression [^0-9|^x]; that pattern tells PowerShell to replace everything that doesn't fall into the numeric range 0-9 *or* that isn't the letter *x* (**^x**). After we execute this line of code Ken's phone number should look like this:

```
12065551111x1219
```

That's step 1. Now we need to replace the letter *x* with the string *;ext=*. Which is what we do here:

```
$phoneNumber = $phoneNumber -replace "x", ";ext="
```

That gives Ken a phone number that looks like this:

```
12065551111;ext=1219
```

That also makes it easy for us to add the *TEL:+* prefix and then assign Ken his new line URI.

Oh, and what about Pilar's phone number, a phone number that *doesn't* use an extension? That's fine. If the script finds an extension it will deal with and, if it doesn't find an extension, it will deal with *that*.

Which, all in all, really *is* a heck of a deal, isn't it?

That's all we have time for today; it's time for our shift down in the Lync PowerShell Fortress of Solitude. If you need additional help just send an email to cspshell@microsoft.com. Remember: the SWAT teams are standing by!

Note. Well, to be honest, Greg is probably checking basketball scores and Jean is probably playing Mahjong. But, despite that hectic schedule, we'll try and answer your question as quickly as we can.

Comments

soder 2 Sep 2011 12:59 PM

I would be very cautious with the uppercase "TEL:" string: it may break compatibility with some 3rd parties, that depend strictly on the case-sensitive recognition of the lowercase "tel:" string. I dont know if the SIP standard accepts both "TEL:" and "tel:" otherwise only a fully valid representation should be advertised and used here on Technet.