Global Knowledge™

Expert Reference Series of White Papers

Tips on Understanding Microsoft Regular Expressions

Tips on Understanding Microsoft Regular Expressions

Joseph Parlas, CCSI, CCVP, CCNP, CCNA, A+, MCSE

Introduction

Microsoft has introduced regular expressions for the main purpose of normalizing E.164 numbers and allowing users to dial numbers by a pattern they are accustomed to and to define routes to send to an external gateway for PSTN connectivity. Regular expressions are also used for Address Book translations of numbers in users contact database that would have to be converted to the E.164 format.

This white paper focuses on the regular expression process and the syntax used by the Microsoft OCS (Optional Component Manager) Expert to create a dial plan and normalization rules that will properly be interpreted and executed.

We also will be introducing tool sets that can be used right on your XP or Vista computer to test regular expression constructs without disturbing the corporate production environment.

Let's start by looking at some of the basic constructs of the regular expression itself by homing on some of the basic symbols used. These examples are from a pdf document that can be downloaded from http://www.addedbytes.com/.

The first building block symbols are **^** and **$**.

> **^** means the start of the string or "must start here"
> **$** end of the string or "must end here

The starting point of your regular expression should be ^$, then add the rest of the constructs between them. These symbols are also referred to as anchors, and represent the start and end of whatever you are looking for.

The next series of symbols to consider, which are part of groups or ranges, are listed below.

( ) The parentheses represent a set or a **group** reference where **(**defines the beginning of a group declaration an**d)** defines an end of a group declaration. An example of grouping is (\d{3}). For now don't worry about the \d{3} within the group; that will be discussed below. Just understand we create the grouping of our expressions using the () symbols.

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

After declaring a group, it is referenced on replacement as $1 where 1 represents the first group placement. We will use this in an example to better understand the relationship.

[ ] Brackets represent a **range** of items you are looking for. Only one item is matched within a range specification. As an example, all North America Numbering codes for service are 211, 311, 411, 511, 611, 711, 811, and 911. I could represent all these variations using the range specification [2-9]11 where the range will fall on 2 - 9.

**** The back slash represents an **escape character**, which means to escape a meaning of something because you are trying to match it as a character and not use it as an regular expression option. For instance, + means 1 or more in regular expression language; however, I need to match + as part of an E.164 number. To prevent it from being used as a regular expression verb, we add a '\' before the value as in this example : '^\+404'. We are looking for +404 as a number and do not want to use the actual noun one or more.

\d This is a **character class** that is used to represent the number of digits, regardless of the actual value. For instance, if I am looking for 3 digits in the range of 0-9, I could use '/d/d/d' since /d would represent one digit condition. This could be awkward if we want to find, for example, a combination of 9 digits. The alternative is to use a number representation in curly braces {N} where N is an integer value of the number of digits you are trying to match. So in our previous example to match any 3 [0-9] combinations I could write this two ways. The first way, described earlier, is '/d/d/d' but an easier approach would be to use'/d{3}'.

Now let's look at other symbols that are used routinely in OCS.

Quantifiers

***** - means 0 or more digits to follow

+ - means at least 1 or more digits to follow

? - means exactly 0 or 1 more digits to follow

{6} - means exactly 6 digits to follow

{3,} - means at least 3 or more digits to follow

{5,9} - means 5, 6, 7,8 or 9

Note that the quantifiers with {} are typically used after the \d character class.

Assertions

?= - this is a look ahead

?| - negative look ahead

?() - if then condition

?()| - if then else condition

?# - place a comment

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

There are other regular expression symbols, but these tend to be the most commonly used ones you need to have in your tool chest in order to provide the necessary constructs for successful enterprise voice integrations.

Now that we understand the meaning of these symbols, let's look at common constructs for normalization rules in OCS. As a reminder, a normalization rule deals with a phone number construct a user would dial, then the rule will take that number and convert it to E.164. Remember, OCS only deals with E.164 telephony numbers by default.

Normalization

Example 1. My company, ABC Inc., uses 5-digit dial rules within the headquarters location. The number range is 12000 to 12999. Let's say that the full E.164 dial rule is North America using +18583412XXX where the Xs represent the DID number range.

Translation Phone Pattern: ^12(\d{3})$

Replacement: +18583412$1

In this example, the rule says find a 5-digit number always beginning with 12 (i.e., ^ "must start with" and "12") and must have only 3 additional digits after the 12, which is represented with \d{3}. Also notice that we grouped the number that can vary by using the parentheses (). This is done to easily call back that group for a possible replacement criteria.

Now the replacement says add +18583412 in front of $1, which represents the grouping of (\d{3}).

So, when a user dials 12002, it is match and translated to +18583412002.

Figure 1. Illustration of OCS Normalization Rules

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

Example 2. My company, ABC Inc., wants users to dial 911 emergency numbers but route the number without any E.164 conversion.

Translation Phone Pattern: ^911$

Replacement: 911

There are some special circumstances where we have numbers that we do not want to convert to full E.164, and those may deal with numbers to route out a PSTN gateway with no manipulation. Emergency numbers are in that category.

Example 3. My company wants a 0 or operator option to be sent to the tanjay phone with the E.164 address of +18583412099.

Translation Phone Pattern: ^0$

Replacement: +18583412099

A system operator is a very common request. If anyone dials 0, it will be routed to the tanjay phone with that E.164 address assigned.

Example 4. My company wants users to dial a 7-digit number using 9 as a trunk access code.

Translation Phone Pattern: ^9([2-9]\d{6})$

Replacement: +1858$1

This uses more items in our tool chest. First, I purposely grouped the 7 digits except for the trunk access code since I want that number not to be included after normalization. This makes your gateway configuration much easier. Also the prefix code or the first 3 numbers of a 7-digit pattern can never begin with a 0 or 1: 0 could mean operator and 1 means North America Country code 1, starting a long-distance call. Therefore, we do not want to match either a 0 or 1 in the prefix portion of the NAMP (North American Numbering Plan). The NAMP has the construct or 3-digit area code, 3-digit prefix code, followed by a 4 digit subscriber code. So the range [2-9] is used for the first digit to rule out 0 or 1 as a match. Then we are looking for any additional 6-digit combination with \d{6}.

When a user dials 93564578, it would be changed to +18583564578. Notice the "9" gets removed.

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

Example 5. My company wants users to dial 10-digits, but must be only the 10 local digits in the San Diego area. Let's say San Diego has 619, 858, and 951 area codes. Now we will build a normalization rule so when the user dials 9 plus the local 10-digit number, it gets translated into a full E.164 value.

Translation Phone Pattern: ^9(619|858|951)(\d{7})$

Replacement: +1$1$2

Now this rule looks for a trunk access code 9 in front of area codes 619, or 858, or 951, followed by any 7-digit combination. The pipe symbol or | is used for the OR condition and is normally applied within grouping. Adding the 3 digit area code with the 7-digit combination afterwards gives 10-digit total.

The replacement says add +1 then add group 1 and then group 2, so if a user dials 96193651234, it would be translated to +18586193651234.

Routes

We also use regular expressions in defining routes or numbers that must be sent to a mediation server then to a gateway to the PSTN. In this case, where there is no replacement, we are only trying to match enough of the number to decide if it can be reached outside of OCS, since it is not a number associated with any users within the OCS environment.

Example 1. The company wants to route all emergency calls to PSTN gateway via mediation server.

Target Regular Expression: ^911$

Notice this has the same pattern as the normalization rule with one big exception; you point this route to a mediation server then it gets sent to your PSTN gateway.

Figure 2. Illustration of OCS Routes

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

Example 2. The company wants to route long-distance calls except for 900 calls!

Target Regular Expression: ^\+1(?!(900))

Now, let's look at this expression much closer. We are looking for any pattern first beginning with a '+' character and not one or more digits to follow, since the + symbol is preceded by the \ escape character. Also we are negatively looking ahead to the next 3-digit values matching 900. If the number matches 900, it is dismissed, since it is a negative assertion and is not included within the scope of the routing match.

Example 3. The company wants to route toll free numbers to the PSTN.

Target Regular Expression: ^\+1(800|876|888)(\d{7})$

In this expression, we are assuming when a user dials 918002345678, then the normalization rules will translate it to +18002345678, which would match this target expression. This expression looks for the +1 and will only consider 800, or 876, or 888 with 7 digits thereafter.

Example 4. The company wants to route International number to the PSTN.

Target Regular Expression: ^\+[2-9](\d*)$

This expression is looking for any number beginning with a + in E.164 with potentially unlimited digits thereafter (\d*); however, the first digit after the plus symbol must be in the number range of 2-9. This expression will exclude all North American Plan dialing since the NANP country code is 1. All other country number codes always begins in the number range of 2-9. Now you may ask why we don't have a stricter digit count versus unlimited in the last statement. Unfortunately, other country numbering plans are variable; they are not fixed like the NAMP. That is why we just allow as many digits to be pressed as long as the first two criteria are being met.

Address Book Translations

Probably the most taxing of all normalization rules is the one for translating users' outlook contacts numbers into full E.164. This process starts with editing the Company_Phone_Number_Normalization_Rules.txt located on the OCS server c:\Program files\Microsoft Office Communications Server 2007\Web Components\Address Book Files.

It is imperative that you understand all of the possible iterations of phone numbers that users will add to their respective contact book. We will look at typical outlook address book phone entries and try to come up with a single solution to accommodate all possible combinations. Let's say the table below represents all the different combinations that people have added. Remember spaces do count.

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

(619) 384-4567
+1(619) 384 4567
1(619) 384-4567
1 (619) 384 4567
1-619-384-4567
1(619)(384)(4567)
1.619.384.4567

Table 1.

Now, I have to admit, some of these combinations are a little wild but at least it gives us a chance to explore how regular expressions can be used to normalize all these pattern types to an E.164 format.

Now notice that some of the patterns begin with special characters like (, +, or even spaces, so we need to come up with a routine that would search for any combination of special characters within these patterns to fully normalize to the proper format for routing purposes.

Within the format of the Company_Phone_Number_Normalization_Rules.txt file, it is formatted very similarly to phone normalization rules. On the first line, you create the search criteria, and the next line is the replacement pattern.

This is working on the premise of the North American Dial Plan and other dial plans will require more normalization rules within the text file.

We begin by using the first stepping stones of regular expressions, the **begin** and **end** characters **^ $**. Then we will add the sequences of area code - prefix - subscriber code matching, which would be grouped like so: ^(\d{3})(\d{3})(\d{4})$. As you can see, we are going after 3 groupings, but we also have special considerations as well. The reason we are putting the number into 3 distinct groupings is because all number in Table 1 are numbers in the NANP format. Next, we need to discover special characters like + , a space , () parentheses, etc., as displayed above.

The a common approach for this is: [\+\.\s\(0\)/-]* . We are looking for any specific character including spaces "\s" that may be before or between the groupings(). The very first part of the character references above are a +, (, or, a 1 combination.

So the very first component is a grouping like this: [\+\s*1]? Basically, this says look for a + or multiple spaces \s*; remember * means 0 or more or a 1. The **?** means this entire range [] may occur 0 or 1 time.

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

So putting the above items together our regular expression would look like this:

^[\+\s*1]?(\d{3})(\d{3})(\d{4})$

But this is not complete. What if you have special characters " spaces,),(,-, or even a ."after the +1 or 1, as we do above. We need the ability to select those special characters that we could use the following for [\+\.\s\(0\)/-]*, which is a range looking for all special character combinations represented above.

The asterisk after the range simply means zero or more combinations. Now let's put it all together and see if this will fix our combination issues.

^[1\+\.\s\(0\)/-]* (\d{3}) [\.\s\(0\)/-]* (]\d{3}) [\.\s\(0\)/-]* (\d{4})$

Our replacement statement would be: +1$1$2$3.

Also note we changed the question mark ? to an asterisk * before our first grouping. The reason for doing so is that if we had multiple special characters, we would need to remove from the beginning of a number in order to normalize it correctly for instance "1.(619)357 8901". Notice that our expression would match not only the 1 but also the period as well.

Note that spaces have been added to make the above example more readable but it would not be typed that way within the text file.

Would +1(619) 384 4567 be matched and properly normalized? Let's see.

+1 would be caught by the first statement: [1\+\.\s\(0\)/-]*. This is also going after multiple spaces, + or 1 combination 0 many times. This will accommodate matching "+ 1" or " + 1" or "(" etc., including looking for "(0)" combinations.

In our example, +1 and special character (would be matched. Then 619 is matched by (\d{3}), which is our first grouping.

Then special characters) and spaces have to be matched with this statement [\.\s\(0\)/-]*, which basically says "look for any special characters including spaces 0 or more times." This will again match the), or close parenthesis, and space before the next sequence of grouping characters.

384 is matched by (\d{3}) grouping, then we catch the space or spaces with [\.\s\(0\)/-]*.

And finally, the 4567 is being matched by (\d{4})$.

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

This is as hard as it gets, and I hope you found this presentation easy to follow. Now the one normalization statement above should work with all combinations, but how can I check that to make sure? Now let's look at different tools we can use to learn and build complex regular expressions.

Tool Set

The tool sets I describe here are the three that I routinely use and demonstrate in classes. The first toolset is Microsoft's Route Helper, which is an OCS resource kit tool, and it is invaluable, especially when looking at normalization and routes together and troubleshoot why they are not working correctly.

The next tool is the RAD Software Regular Expression Designer, which can be downloaded from www.radsoftware.com.au, an Australian company that provides free toolsets for OCS.

The last tool that I will explain is power shell scripting. I use this as a means to determine if my one regular expression rule will work with all of the instances we explained for address book translations.

The following screen shot shows the route helper. With route helper, we can test user normalization rules.

Figure 2. Test numbers that represent routes to dial to the PSTN through gateways

We can also verify location profiles we have designed through normalization rules assigned to our users.

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

Figure 3. Verifying Location Profiles

We can even test all of our normalization rules for outside numbers for enterprise dialing with route helper.

With RAD Software Regular Expression Designer, we can plug in the example we worked on above and with the replacement criteria to see if we achieve E.164 format. Here is our example running in the RAD designer software with one statement.

Figure 4. RAD Software Regular Expression Designer

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

The only problem with the RAD Software Regular Expression Designer is that we can only enter one pattern at a time and test, and this could take a long time if we have, let's say, 20 pattern combinations.

Another tool that makes it easier to test address book normalizations is using power shell scripting. Here is a sample script with our input with an output testing result. In order to use this script, you need to add the power shell command to your XP or Vista PC by downloading it from http://www.microsoft.com/windowsserver2003/technologies/management/powershell/download.mspx .

Now, let's create a power shell script based on our above address normalization solution and run it against all of our combinations to see if E.164 normalization will hold up on all those different combinations.

All you have to do is copy the power shell routine below in a text document and save it with a ps1 extension, then execute it within power shell command prompt.

The script should look like this:

```
$Norm = "^[1\+\.\s\(0\)/-]*(\d{3})[\.\s\(0\)/-]*(\d{3})[\.\s\(0\)/-]*(\d{4})$";

$Numbers =
"(619) 384-4567",
"+1(619) 384 4567",
"1(619) 384-4567",
"1 (619)  384 4567",
"1-619-384-4567",
"1(619)(384)(4567)",
"1.619.384.4567";

For each ($Number in $Numbers)
    {
    $res = $Number -match $Norm;
    write-host "+","1",$matches[1],$matches[2],$matches[3]" : ",$Number -Separator "";

    }
```

End of Script File

If we break down this script, we can see we are declaring variables $Norm, which is our regular expression string we are trying to test, and variable $Numbers, which represent a series of values we would like to test.

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

We set up a loop and declare the variable $Number to increment through each string set and then apply a write condition to the result of each line using the write-host statement. Now each $matches[1] etc., represents a array of values that are defined within the grouping parentheses. In the $Norm expression, the first sequence of (\d{3}) is placed in array position [1] and so forth.

After putting the script together and executing it, I get this result.

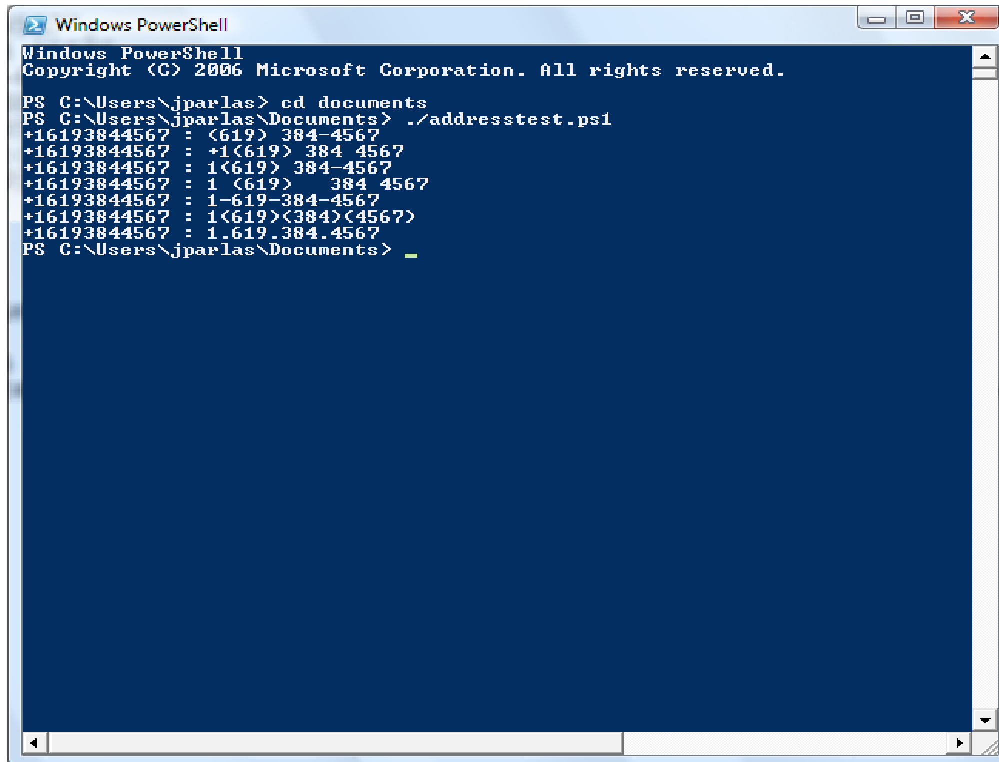

Figure 5. Powershell Script Results

So all the numbers have normalized correctly despite all the different combinations in which the numbers were saved.

In general, Route Helper is better utilized in testing or troubleshooting existing normalization and route problems on the OCS server. The RAD Software Regular Expression Designer is a great regular expression learning tool, and it can quickly aide in designing or test complex regular expressions to see if the correct results will be obtained before adding it to OCS. The power shell script is a good tool to test if a single regular expression can be used to translate different variations of numbers at once to be used for contact address book translations.

Conclusion

As you can see it, would be very beneficial to get up to speed with regular expressions as quickly as possible to help create the necessary normalization rules, route plan, and address book normalization. Regular expres-

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.

sions are needed in each of these categories and understanding the constructs will help you develop unique and efficient OCS solutions for your customers. Also, it is exciting that you have all the tools at your disposal to try out different regular expression routines without disturbing your production environment. The best tool in really learning all of the regular expression components would in, my opinion, be the RAD Software Regular Expression Designer, which was demonstrated in a TechNet webinar on understanding regular expressions.

Learn More

Learn more about how you can improve productivity, enhance efficiency, and sharpen your competitive edge. Check out the following Global Knowledge courses:

Configuring Microsoft Office Communications Server 2007 Prep

For more information or to register, visit **www.globalknowledge.com** or call **1-800-COURSES** to speak with a sales representative.

Our courses and enhanced, hands-on labs offer practical skills and tips that you can immediately put to use. Our expert instructors draw upon their experiences to help you understand key concepts and how to apply them to your specific work situation. Choose from our more than 700 courses, delivered through Classrooms, e-Learning, and On-site sessions, to meet your IT and management training needs.

About the Author

Joe Parlas has been an instructor for over eight years, concentrating specifically on Cisco Voice technologies. He has consulted for numerous Fortune 500 and 1000 companies such as Sweetheart Cup, Inc., Black and Decker, and McCormick Spice, and as a senior consultant with Symphony Health Services, Inc. in various capacities.

Joe holds the following industry certifications: CCS; CCVP; CCNP; CCNA; A+; and MCSE with Messaging 2003. He is also a contract instructor for Global Knowledge. Joe recently relocated his company Parlas Enterprises to the San Diego, CA area where he lives with his wife Parvin Shaybany.

Copyright ©2009 Global Knowledge Training LLC. All rights reserved.