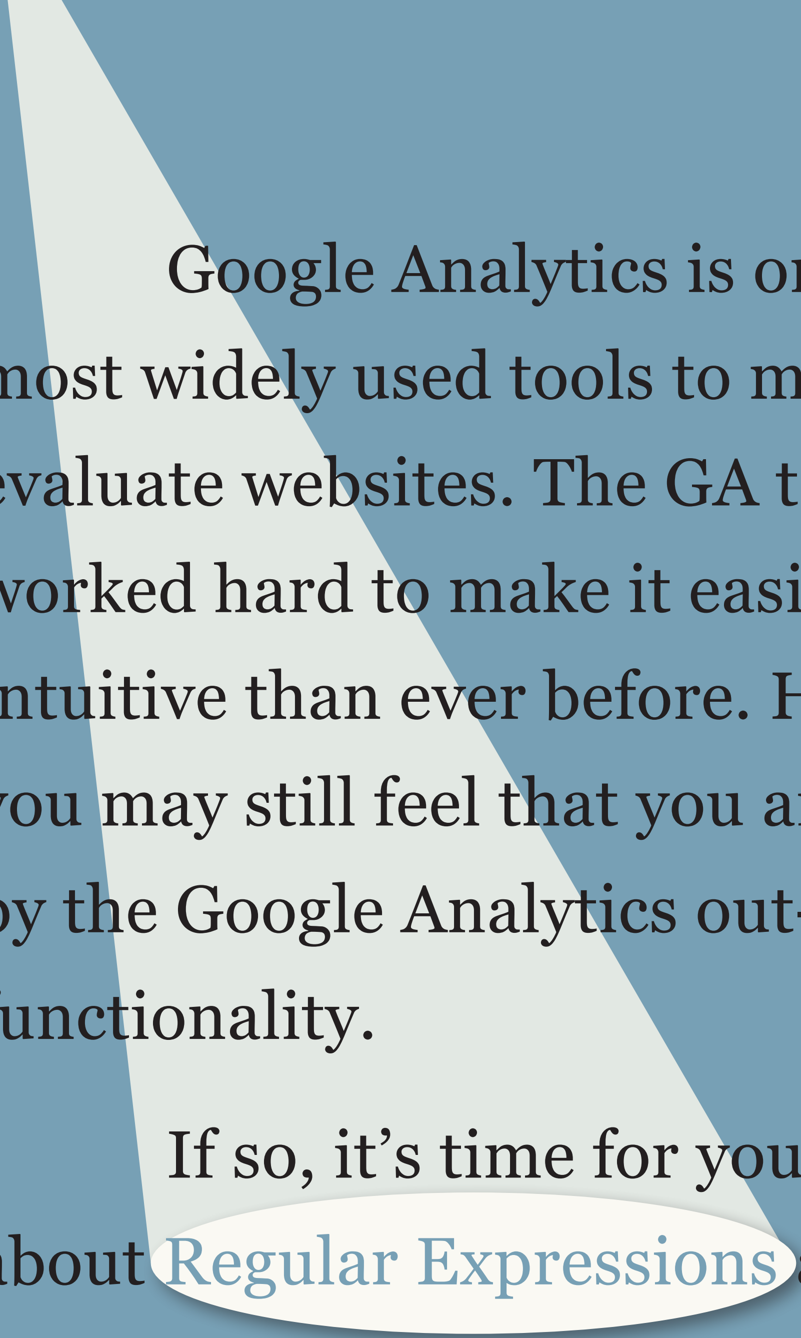
Regular Expressions for Google Analytics

By Robbin Steif



Google Analytics is one of the most widely used tools to measure and evaluate websites. The GA team has worked hard to make it easier and more intuitive than ever before. However, you may still feel that you are limited by the Google Analytics out-of-the box functionality.

If so, it's time for you to learn about Regular Expressions and how GA uses them.

JUST a story about Regular People and Regular Expressions . . .

When I first starting working with Google Analytics, I was an analyst. A marketing person. Not a techie.

Back then, the Google Analytics documentation kept referencing something called Regular Expressions. I could see that my goals and filters weren't doing what they needed to do, but, not being a techie, I didn't know how to implement RegEx and fix them.

(In fact, I knew so little about this space that when a friend referred to Regular Expressions as "RegEx," I wondered what he was talking about.)

Slowly I taught them to myself, with the help of Wikipedia and a friend in Australia. Then I began to blog about them, using non-techie language. I got a letter from a trainer on the other side of the pond who told me that when he trained people in Regular Expressions, **he** turned them loose on the LunaMetrics blog. Eventually, Google invited my company to become a Google Analytics Certified Partner. Our company helped rewrite the Google Analytics Help Center section on Regular Expressions. And to this day, I get random emails from random people, asking me to troubleshoot their RegEx.

Why Use RegEx?

In Google Analytics, you can use Regular Expressions to …

create filters. Many filters require Regular Expressions. If you don't know what filters are, you can start learning about them here.

create one goal that matches multiple goal pages. Perhaps your "thank you" page has many names, but to you, all leads are the same goal. So you can use Regular Expressions to "roll them up."

fine-tune your funnel steps so that you can get exactly what you need. Remember, Regular Expressions can be **specific**.

What are Regular Expressions, anyway? Regular Expressions are about "power matching." If you need to create a goal that matches multiple thank-you pages – that is power matching. If you need to write a filter that matches multiple URLs, but only know what a piece of each URL looks like – again, that is power matching.

But what about Advanced Segments? Can't I skip this whole RegEx thing now that Google Analytics has Advanced Segments?

Well, no. Advanced Segments are lovely, and they often make filters unnecessary. But they don't work the same way as filters do. And you will still need Regular Expressions to create interesting and complicated goals, and to accommodate your website designer who doesn't do things the Google Analytics-friendly way. And sometimes, you will want to use Regular Expressions in your Advanced Segments.

A word about language.

What would a how-to guide be without some dictionary-type advice? Here are some of the conventions we may use:

GA: The abbreviation for Google Analytics

RegEx: the abbreviation for Regular Expressions (singular and plural)

Plain text: Not Regular Expression Text

String: any assembly of characters and/or spaces. A word could be a string, a sentence could be a string, a URL could be a string.

Target String: the string you are attempting to match with your RegEx.

*Example: when I use robb?(y|i)n to match my name, Robbin (the one with all the funny characters), **robb?(y|i)n** is the Regex, and my name, **Robbin**, is a target string.*

A word about format.

I just hate when I read a post or book and they write that the keyword is "sodapop." Or "vanilla." Or anything that has quotation marks around it. Because you never know if the quotation marks are part of the stuff you are working on, or are just used to separate that word from the rest of the sentence. Consequently, I put all target strings and all RegEx in boldface – no quotation marks needed.

INDEX

Regular Expressions match as much as possible.

Most of this ebook will be about all the characters that make up the Regular Expressions "toolset." But first – you'll be lost if you don't understand the concept of matching as much as possible. Sometimes you will be very surprised at how much it matches, just as I was when I was new at RegEx.

Let's start with an example: Our company often wants to see all the keywords that include our company name (branded search) and, more often, we want to see all the keywords that don't include our company name (unbranded search). Below are two of the Regular Expressions we use when we are including or excluding our company name:

Luna
Metric

A friend wrote a question on a forum. He wanted to exclude all traffic that was coming from Google referring links, all around the world. He tried to create a really fancy RegEx that would include all countries, something like

www\.google\.(com|co\.(au|uk|il))/(docs|analytics|reader) *etc.*

I wrote back: ***"Why don't you just use the word google as your RegEx?"***

There are two interesting questions here, and you might feel some righteous indignation as you ask them. I know that I felt this way when I was first learning RegEx:

1. Why would they match? Not a single one of those phrases includes the entire name.

2. How can those possibly be Regular Expressions? There are no RegEx characters!

ANSWERS

1. They match because Regular Expressions in GA will match and match until they aren't allowed to any more. That's why **Metric** matches the target string, **LunaMetrics** – if it matches any part of the word, it will match the whole word.

2. And the characters? You don't need to have those characters just to have a Regular Expression, and having the characters doesn't necessarily make it a RegEx. All you need to do is put the expression into a field that is sensitive to RegEx. For example, when you write a Google Analytics goal, you get to choose "head match," "exact match" or "Regular Expression." As soon as you choose "Regular Expression," the field becomes sensitive to RegEx, and all the rules of RegEx apply. You often need to use little RegEx characters – but not always.

How Do I Learn About RegEx?

of course, reading this eBook will help you, but you can only get so far by reading. Ultimately, understanding and writing Regular Expressions (RegEx) is a little bit like getting your first job. You can't get hired without experience, and you can't get experience without getting hired.

With RegEx, you don't really understand them until you use them, and you can't really use them until you understand them. So you have to learn a little bit, and then use a little bit and get them wrong, and then go back to the book and learn a little bit more.

The other problem you will have with RegEx is that each character is easy. Put them all together and you get this:

```
/\?cid=[0-9]{3,3}.
```

And that one wasn't very hard. The more you work with them, the easier they'll get. So master each step, put a couple together, make some mistakes and get going.

Soon you'll be a RegEx pro.

There are a lot of great resources to check your Regular Expressions. If you use the PC, I strongly recommend the **RegEx Coach:**

http://www.weitz.de/regex-coach/

Unfortunately, it is unavailable for Mac, so an alternative would be the **RegEx Match Maker:**

http://sourceforge.net/projects/quregexmm/

There are many others.

THE BACKSLASH

I always encourage people to start their "RegEx career" by learning the characters, and the best one to start with is the backslash. A backslash is different from all the other characters, as you will see. It provides a bridge between Regular Expressions and plain text.

A backslash "escapes" a character. What does "escape" mean? It means that it turns a Regular Expression character into plain text. If that doesn't make sense to you yet, hold on – I have a few examples coming.

Perhaps **/folder?pid=123** is your goal page. The problem we have is that the question mark already has another use in Regular Expressions – but we need for it to be an ordinary question mark. (We need it to be plain text.)

We can do it like this:

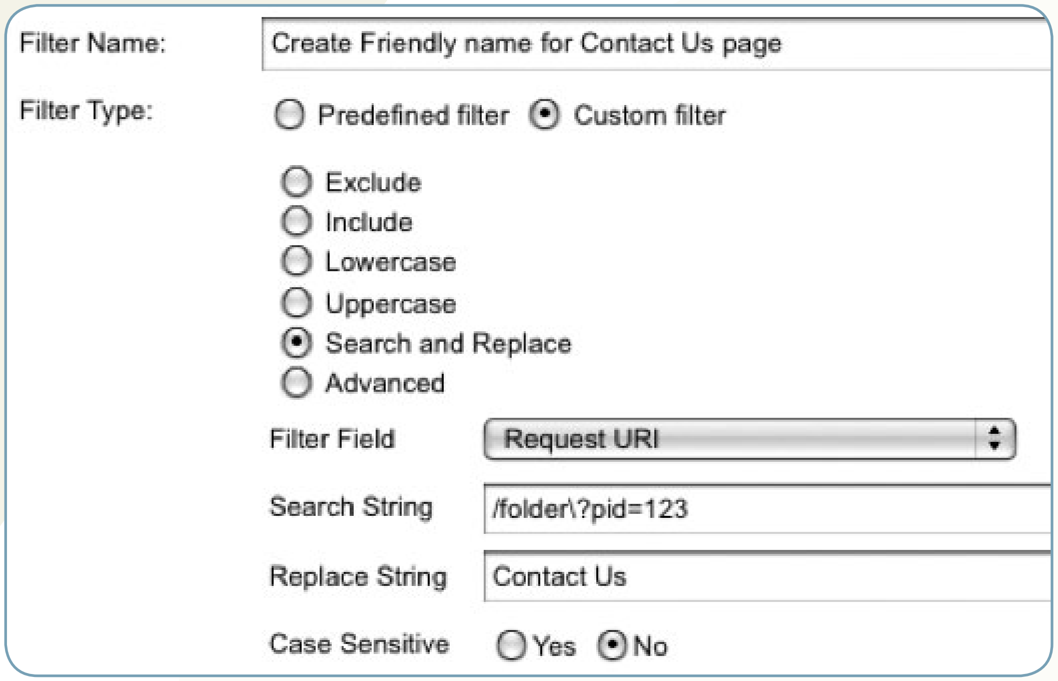
/folder\?pid=123

Notice how there is a backslash in front of the question mark – it turns it into a plain question mark.

Why are backslashes about getting started?

1. You will use them more than any other RegEx character.
2. They turn special RegEx characters into everyday, plain characters.

Example Alert: Now, when we are creating a Google Analytics search and replace filter for the page above, we can use that backslash and know that GA understands we are looking to match to a real question mark, like so:



Filter Name: Create Friendly name for Contact Us page

Filter Type: Predefined filter [✓ Custom filter]

Exclude
Include
Lowercase
Uppercase
[✓ Search and Replace]
Advanced

Filter Field: Request URI

Search String: /folder\?pid=123

Replace String: Contact Us

Case Sensitive: Yes [✓ No]

THE PIPE

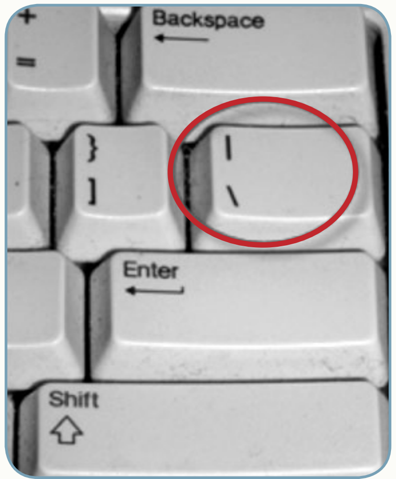
The pipe is the simplest of Regular Expressions, and it is the one that all Regular People (that's you and me) should learn. It means *or* .

Here's a simple example: **Coke|Pepsi** . A soft-drink blog might use that expression with their Google Analytics keyword report to find all examples of searches that came to their blog using either the keyword **Coke** or the keyword **Pepsi**.

Here's another example: Let's say you have two thank-you pages, and you need to roll them up into one goal. The first one is named **thanks**, and the second is named **confirmation**. You could create your goal like this:

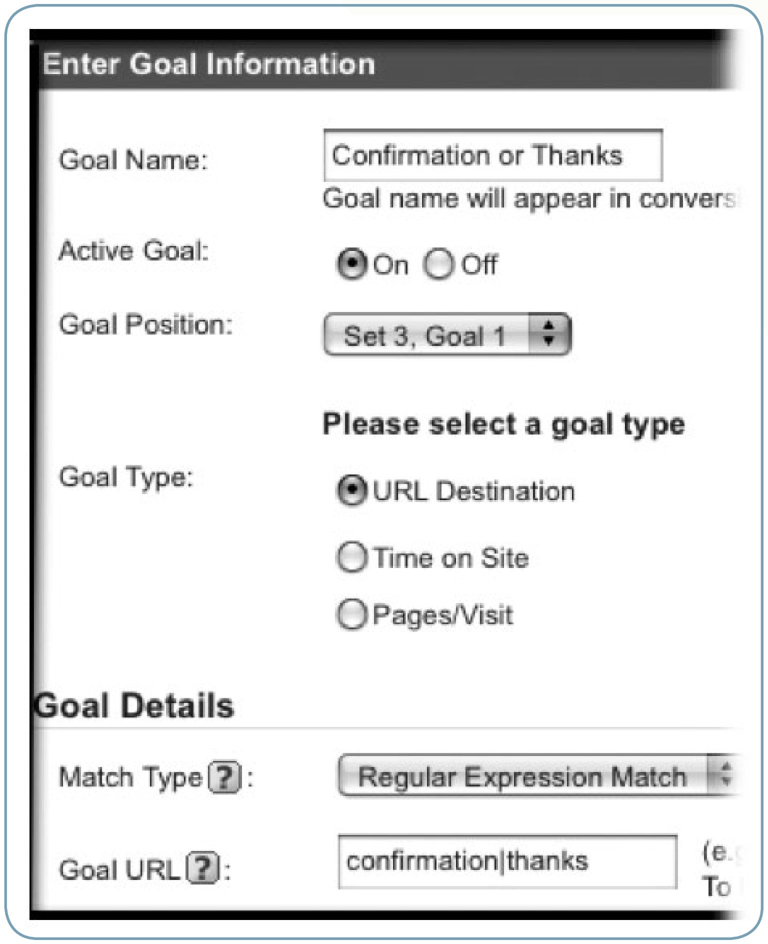
confirmation|thanks

That says, match either of those pages. Notice that the pipe matches everything on either side of it.



And where is the **Pipe**, *anyway? Keyboards differ, but you'll most likely find it above your* **Enter** *key.*

Here you can see a screen shot of a Google Analytics goal setup that includes the two pages suggested above, with a pipe:



Enter Goal Information

Goal Name: Confirmation or Thanks
Goal name will appear in convers

Active Goal: [✓ On] Off

Goal Position: Set 3, Goal 1

Please select a goal type

Goal Type: [✓ URL Destination] Time on Site Pages/Visit

Goal Details

Match Type ?: Regular Expression Match

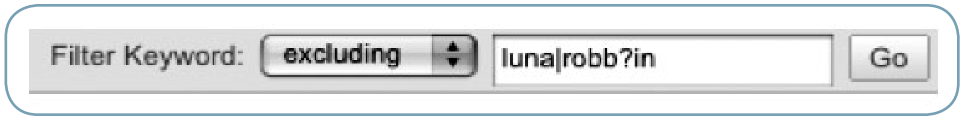
Goal URL ?: confirmation|thanks

The Question Mark

A question mark means, *"The last item (which, for now, we'll assume is the last character) is optional."* So remember how I wrote that people misspelled my name, often spelling it with one b instead of two? With a RegEx of **Robb?in**, I can capture both **Robbin** and **Robin**. That's because the question mark means that the expression will match even if the second "b" isn't in the target string – because it is optional.

In case you're wondering – I often use this RegEx to filter keywords on my company's website, so that I can quickly pull out keywords that were clearly looking for me instead of our services.

That way, I capture both the people who spell my name correctly and those who misspell. In fact, when I do a report that excludes branded keywords, I usually try to exclude both our company name and my name. Like this:


Filter Keyword: excluding luna|robb?in

Parentheses ()

Parentheses in Regular Expressions work the same way that they do in mathematics. This falls into the category of "Things I should have learned had I been paying attention in grade school."

Now, before I get into this long explanation, I want to provide an example. This is because I user-tested this whole ebook, and the testers complained, "Robbin, sometimes we just need to see the example up front." So here is an example of parentheses:

/folder(one|two)/thanks

This matches two URLs, folderone/thanks and foldertwo/thanks.

OK, on with the explanation. Remember, we were talking about things we should have learned had we been paying attention in school.

Remember how your math teacher said that if you had an equation, the division and multiplication got done before the subtraction and addition? Well, since I wasn't paying attention in Mrs. Petrowski's 4th-grade class, I pulled out my old notes, and here is what I found:

2 + 3 x 5 = 17

(Right? 3 times 5 equals 15, plus 2 equals 17.)

If you wanted it to execute differently, you had to force the equation with parentheses – like this:

(2 + 3) x 5 = 25

Above, I've changed the same numbers to become 2 plus 3 equals 5, times 5 equals 25. And that's the value of parentheses in math. I see from my very old notes that Mrs. Petrowski called it the Order of Operations.

So what about Regular Expressions? Why would we need parentheses there? In order to understand our need, we have to look at other expressions (just like we had to understand the math operations symbols in order to understand why parentheses are needed.)

PARENTHESES pt. 2

Let's use pipes as our example. I wrote that this expression:

confirmation|thanks

means everything on one side of it (**confirmation**) or everything on the other, i.e. **thanks**.

But as we start to think about why we would want to use parentheses, we can revisit that example above and ask ourselves, "What happens when we don't want to grab everything on either side of the pipe?" Like this example:

/foldertwo/thanks

/folderone/thanks

A great way to represent this in RegEx would be

/folder(one|two)/thanks

So we are allowing the RegEx to match either the thanks page in folderone or the thanks page in foldertwo – and it is the parentheses that allow us to group so that the pipe knows what to choose between.

This next example is a little different. Again, we're going to roll two URLs into one goal, but this time, we use the parentheses to tell the question mark what is optional. This website has three thank-you pages:

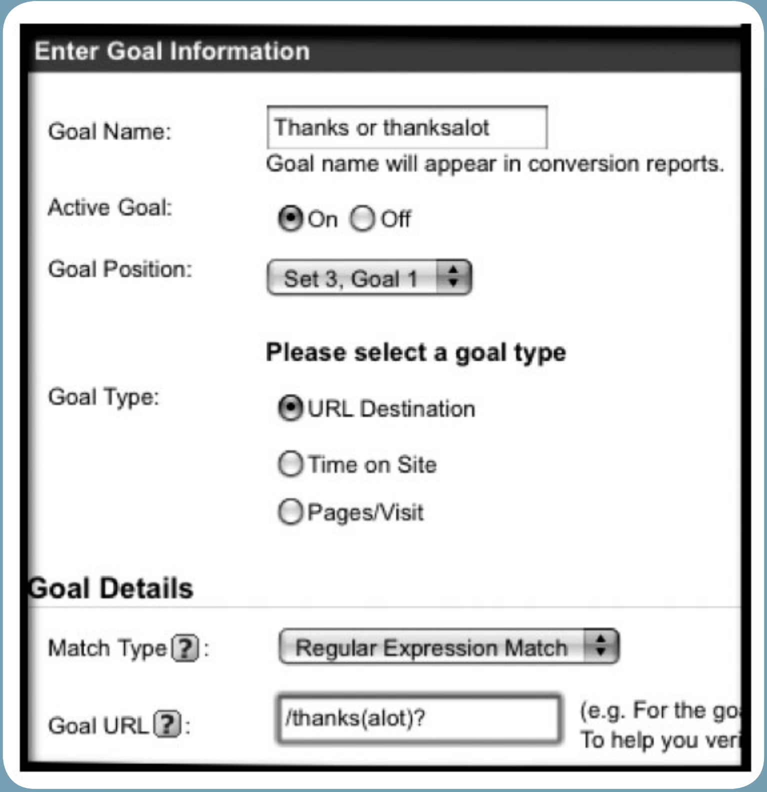
/thanks

/thankyou

/thanksalot

If we only want the **/thanks** and the **/thanksalot** pages to be part of our goal, we could do it like this: **/thanks(alot)?**

This means, the target string must include **/thanks**, but **alot** is optional. So it matches both **/thanks** and **/thanksalot**. And the **/thankyou** page will never get included, because there is no **s** in its URL (so it doesn't match the beginning of this RegEx, i.e. **thanks**).



Enter Goal Information

Goal Name: Thanks or thanksalot
Goal name will appear in conversion reports.

Active Goal: [✓ On] Off

Goal Position: Set 3, Goal 1

Please select a goal type

Goal Type: [✓ URL Destination] Time on Site Pages/Visit

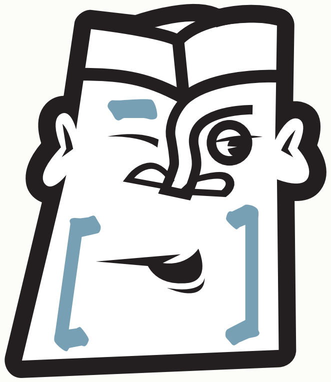
Goal Details

Match Type: Regular Expression Match

Goal URL: /thanks(alot)?

(e.g. For the go
To help you veri

Square Brackets & Dashes pt. 1



I usually like to introduce just one character at a time. But these two are a little like salt and pepper. True, you can use just salt or just pepper, but they get served together so often that they are a set. And so we have both square brackets [ ] and dashes –

With square brackets, you can make a simple list, like this: **[aiu]** . This is a list of items and includes three vowels only. Note: Unless we use other expressions to make this more complicated, only one of them will work at a single time.

So **p[aiu]n** will match pan, pin and pun. But it will not match pain, because that would require us to use two items from the **[aiu]** list, and that is not allowed in this simple example.

You can also use a dash to create a list of items, like this:

[a-z] – all lower-case letters in the English alphabet

[A-Z] – all upper-case letters in the English Alphabet

[a-zA-Z0-9] – all lower-case and upper-case letters, and digits. (Notice they are not separated by commas.)

Dashes are one way of creating a list of items quickly, as you can see above.

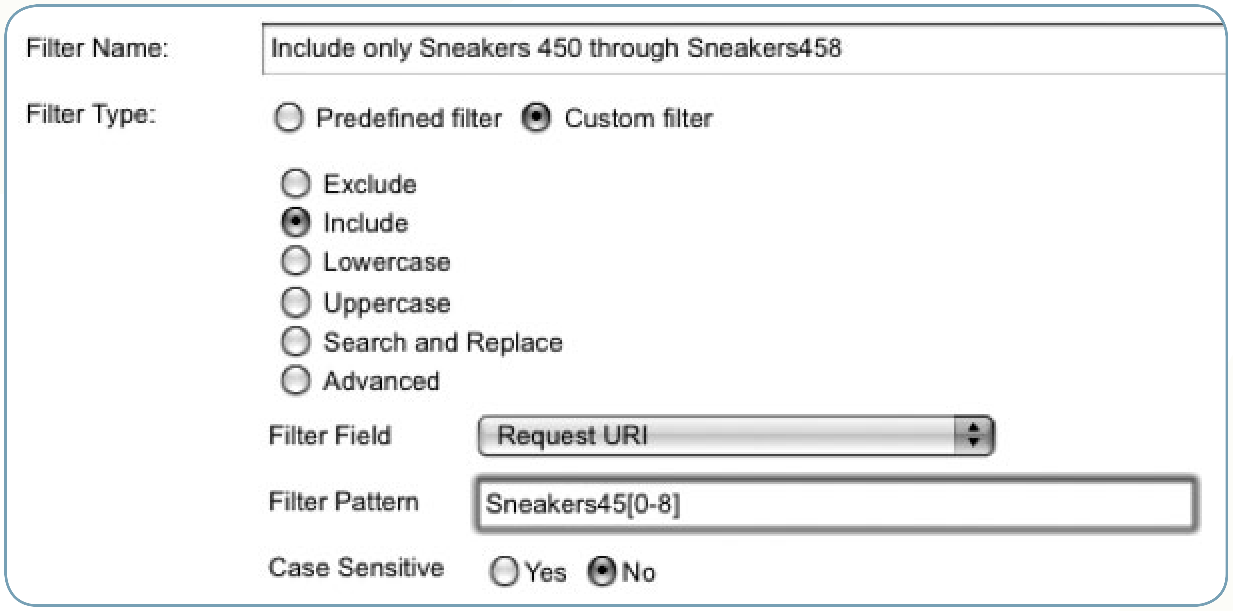
ADVANCED TIP

Characters that are usually special, like $ and ?, no longer are special inside of square brackets. The exceptions are the dash, the caret (more on this one later) and the backslash, which still works like all backslashes do inside the square brackets.

Square Brackets & Dashes pt. 2

Here is an example of how you might use square brackets by themselves. Let's say you have a product group, **sneakers**, and each product name has a number appended to it in the URL (we see this a lot with industrial products where they don't have zippy names). So you might have **sneakers450**, **sneakers101**, etc.

The product manager for **sneakers450** through **sneakers458** wants a special profile of visits that only included his product pages. So you might include a filter like the one below, which makes it easy to include all the product names, using square brackets:



Filter Name: Include only Sneakers 450 through Sneakers458

Filter Type: Predefined filter [✓ Custom filter]

- Exclude
- [✓ Include]
- Lowercase
- Uppercase
- Search and Replace
- Advanced

Filter Field: Request URI

Filter Pattern: Sneakers45[0-8]

Case Sensitive: Yes [✓ No]

Braces

Braces are not covered in the Google Analytics documentation, but they are supported by GA. Braces are curly brackets, {like this}.

Braces repeat the last "piece" of information a specific number of times. They can be used with two numbers, like this: **{1,3}**, or with one number, like this: **{3}**. When there are two numbers in the braces, such as **{x,y}**, it means, *repeat the last "item" at least x times and no more than y times*. When there is only one number in the braces, such as **{z}**, it means, *repeat the last item exactly z times*.

Here is my two-number-in-braces example:

Lots of companies want to take all visits from their IP address out of their analytics. Many of those same companies have more than one IP address – they often have a block of numbers. So let's say that their IP addresses go from **123.145.167.0** through **123.145.167.99** – how would we capture that range with braces?

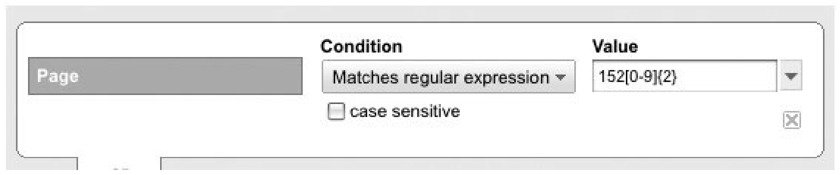
Our regular expressions would be:

```
123\.145\.167\.[0-9]{1,2}
```

Notice that we actually used four different RegEx characters: We used a backslash to turn the magic dot into an everyday dot, we used brackets as well as dashes to define the set of allowable choices, i.e. the last "item", and we used braces to determine how many digits could be in the final part of the IP address.

On the other hand, if there is only one number in the braces, the match will only work if you have exactly the right number of characters.

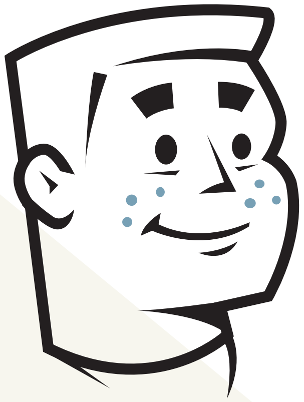
So here's an example: Let's say you are the area manager for Allegheny County (where I live), and you want to see all the visits from people who touched a page on the site that had 152XX in the URL – that's the Allegheny County ZIP code. You could create an Advanced Segment using a Regular Expression with square brackets and a brace, like this:

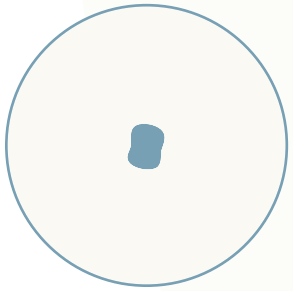


	Condition	Value
Page	Matches regular expression	152[0-9]{2}
	☐ case sensitive	

This means, if the visit included a trip to a page that included 152 and any other two digits (i.e. 152XX) in the url, include the visit in this segment.

THE DOT





A dot matches any one character.

When I was new at Regular Expressions, dots were perhaps the strangest thing for me to deal with. I couldn't figure out what "one character" meant. And I couldn't understand why they mattered much. They didn't seem very wild, as wild cards go.

Ultimately, I learned that the list of characters included all the characters I could find on my keyboard or any other – the alpha, the numeric, the special characters too. A dot even matches a whitespace. I also learned there aren't that many uses for dots by themselves, but they are very powerful when combined with other RegEx characters.

Let me start with some examples of how the dot can be used alone. Take this Regular Expression:

.ite

.... It would match **site**, **lite**, **bite**, **kite**. It would also match **%ite** and **#ite** (because **%** and **#** are characters, too.) However, it wouldn't match **ite**. Why not? A dot matches one character, and **ite** includes zero characters for the dot to match (i.e., it didn't match *any*).

So let's go to a GA example. Let's say your company owned a block of IP addresses:

123.45.67.250 through **123.45.67.255**

You want to create a Regular Expression that will match the entire block, so that you can take your company data out of your Google Analytics. Since only the last character changes, you could do it with this expression:

123\.45\.67\.25.

This would match all the required IP addresses. Note that it would also match 123.45.67.256 and 123.45.67.25% (because a dot matches any one character). However, each grouping (each octet) in the IP address only goes up to 255, and we never see percent signs or other non-numbers in IP addresses, so you would be safe with this.

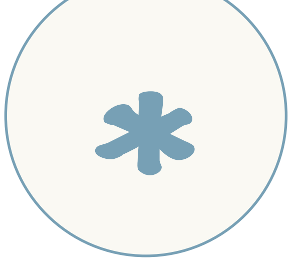
Have you ever noticed that you can type index.php into a search box that is sensitive to Regular Expressions and it works – even though you didn't escape the dot, the way you were supposed to – i.e., index\.php ? That's because the dot matches any one character, and one of the characters the dot matches is ... a dot. And since it is unlikely that a non-dot will get in there, it usually works just fine.

THE PLUS SIGN

A plus sign matches one or more of the former items, which, as usual, we'll assume is the previous character. (It can be more complicated, but let's start easy.) So the list of possible matches is clear: the former character. And the number of matches is clear: one or more.

Here's an example from the world of literature: When a character trips and falls on his face, he often says **Aaargh!** Or maybe it's **Aaaargh!** or just **Aargh!** In any case, you could use a plus sign to match the target string, like this: **aa+rgh**. That will match **aargh** and **aaargh** and **aaaaaaaaargh** … well, you understand. Notice, however, that it won't match **argh**. Remember, it is one or more of the former items.

Does anybody really use plus signs anymore? Sure they do – you may find that you use this expression rarely, but the one time you need it, it will be very valuable.



THE STAR

People really misuse stars. They have specific meanings in other applications. They don't mean the same thing in RegEx as they do in some of those other applications, so be careful here.

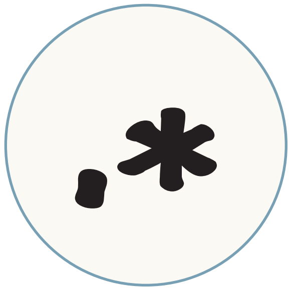
Stars will match zero or more of the previous items. They function just like plus signs, except they allow you to match ZERO (or more) of the previous items, whereas plus signs require at least one match. For the time being, let's just define "previous item" as "previous character."

Since stars are so much like plus signs, I'll start with the same example and point out the differences.

So once again, when a character trips and falls on his face, he often says Aaargh! Or maybe it is Aaaargh! or (unlike last time) just Argh! In any case, you could use a star to match the target string, like this: **aa*rgh**. That will match **aargh** and **aaargh** and **aaaaaaaaargh** – and the big difference from the plus sign is that it will also match **argh** (i.e. no extra "a's" added).

You'll notice that I don't include any screen shots for stars (or plus signs, for that matter.) While they are both very important, they are used very extensively with dots as the wildest cards.

THE DOT STAR



There are two Regular Expressions that, when put together, mean "get everything." They are a dot followed by a star, like this:

/folderone/.*index\.php

In this example, our Regular Expression will match to everything that starts with **folderone/** and ends with **index.php** . This means *if you have pages in the /folderone directory that end with .html* (I see that a lot), *they won't be a match to the above RegEx.*

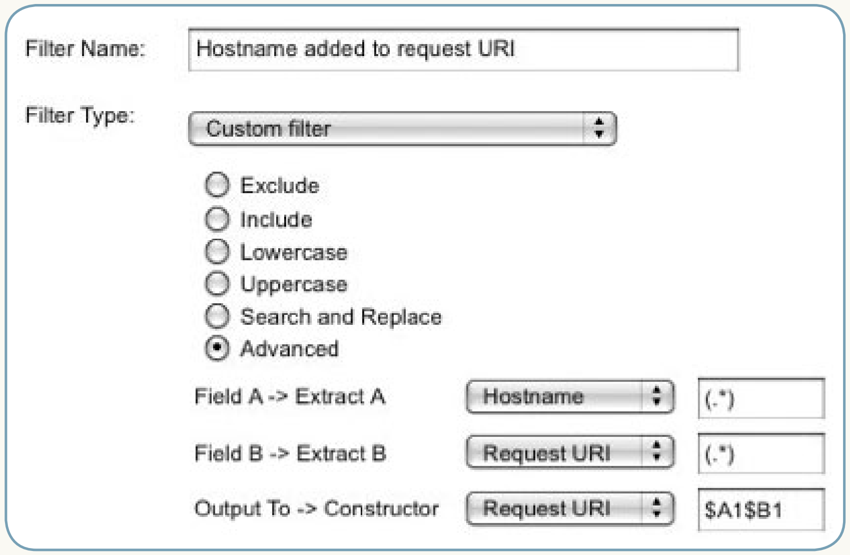
Now that you have an example, you might be interested in why this works. A dot, you may remember, means *get any character*. A star means *repeat the last character zero or more times.*

This means that the dot could match any letter in the alphabet, any digit, any number on your keyboard. And the star right after it matches the ability of the dot to match any single character, and keep on going (because it is zero or MORE) – so it ends up matching everything.

Hard to wrap your head around this one? Trust me, it works.

ADVANCED TIP

Custom Advanced filters in Google Analytics often require you to "get all" and make it a variable. For example, when you want to add the hostname to the Request URL (a very common filter), do it like this:



Filter Name: Hostname added to request URI

Filter Type: Custom filter

- ○ Exclude
- ○ Include
- ○ Lowercase
- ○ Uppercase
- ○ Search and Replace
- ● Advanced

Field A -> Extract A	Hostname	(.*)
Field B -> Extract B	Request URI	(.*)
Output To -> Constructor	Request URI	$A1$B1

Notice how we put parentheses around the ., like this: (.*) – you can see this in the boxes on the right side of the filter. To GA, in the advanced filter section, this means* get all, *and put it in a variable. So we get the entire hostname (in a variable), the entire request URL in a variable, and then in the bottom field, we are able to specify get the first variable in Field A, and to it, add the first variable in Field B.*

THE CARET

When you use a caret in your Regular Expression, you force the Expression to match only strings that start *exactly* the same way your RegEx does.

I see carets misused all the time on various Google Analytics help groups. It goes something like this:

"I want to include only subfolder2 in my profile. My URLs look like: http://www.mysite.com/folder1/subfolder2/index.html. I created the include filter and the Regular Expression, which looks like this: **^/subfolder2/index\.html**. Why isn't it working?"

The reason it doesn't work is that Google Analytics starts reading your URL right after the .com (or .edu, or .net, as it were). The GA sees the above as **/folder1/subfolder2/index.html**. So when the person who wrote the question above starts his RegEx with **^/subfolder2**, he shoots himself in the foot. In the eyes of GA, **/folder1** comes before **/subfolder2**, yet the caret mandates that **/subfolder2** *must* be at the beginning of the target string. (And it's not! So it will never match.)

ADVANCED TIP

When you put a caret inside square brackets at the very beginning, it means *match only characters that are not right after the caret*. So [^0-9] means *if the target string contains a digit, it is not a match.*

The Dollar Sign

A dollar sign means *don't match if the target string has any characters beyond where I have placed the dollar sign in my Regular Expression.*

So let's say you want to include your homepage in your funnel. (Not such a great idea, if you ask me, but this book is about RegEx and not best analysis practices.) Google Analytics, by default, calls your homepage this:

/

i.e., just a slash. The problem with including just a slash in your funnel is that it will match *everything*. Every single URL on your site has a slash in it, so it automatically matches them all, since RegEx are greedy. In order to match just your homepage, you should follow it with a dollar-sign anchor, like so:

/$

That means *the page has to end with a slash.*

But wait ... aren't there pages that end with a slash that aren't your home page? Like so: **/mysite.com/folder/**

So the very best way you can be sure to get just the slash page (homepage) is with a beginning and an ending anchor:

^/$

Use this cautiously, because you might have query parameters that are "shut out" of the match. For example, let's say your affiliate marketers append code only to the end of the URL, so they can get credit for the sale:

http://www.mysite.com/productpage123 ?affiliate=storename

In the eyes of Google Analytics, that looks like:

/?affiliate=storename

... and it doesn't match the **^/$** *RegEx in the example on this page, because the target string doesn't start and end with the same slash.*

Let's Practice

When I was a newbie to RegEx, I used to see this on the RegEx Wikipedia listing (which I pored over constantly, trying to understand what these little characters were):

((great)*grand)?((fa|mo)ther)

Like most things in life, once you understand them, they are trivial ... but before you understand them, they are daunting. So I will take this apart to make it easier to understand.

The parentheses create groups, separated by a question mark. So we effectively have:

(First Expression in the first set of parentheses)?(Second Expression in the second set of parentheses)

Since a question mark means *include zero or one of the former item*, we know that the target string will be a match to this practice RegEx whether it only matches the expression in the second set of parentheses or it matches both the expressions in both sets of parentheses. (Right? That's what question marks do, allowing the target string to match the item before them OR NOT.) So let's start by looking at the second half only, which we now know should be able to stand by itself:

((fa|mo)ther)

The pipe symbol | means *or*. So this resolves to **father** or **mother**.

Now let's go back and look at the first half, the part that came before the question mark:

((great)*grand)

The star tells us to match zero, one or more than one instances of the expression before it. So it can match a string that doesn't include **great**, in which case we just have **grand**; and, of course, we always have the end of the expression, which will be either **mother** or **father**. So we are allowed to match to **grandmother** or **grandfather**. It can match a string which includes **great** just once, in which case we have **great grandmother** or **great grandfather**. And it can match a string which includes **great** more than once, so we might end up with **great great great great grandmother** or **great great grandfather**.

So there you have it – all your ancestors with just one Regular Expression.

(NOT) THE END

So instead of concluding, let's get started … with a pop quiz. I'll blog about the answers soon after this is published. (That way, you'll all have a chance to say, "No, Robbin, I found a better way to write them!") The title of the blog post will be RegEx eBook Answers, so you can search for it easily.

- Write a Regular Expression that matches both **dialog** and **dialogue**
- Write a RegEx that matches two request URLs: **/secondfolder/?pid=123** and **/secondfolder/?pid=567** (and cannot match other URLs)
- Write a single Regular Expression that matches all your subdomains (and doesn't match anything else). Make it as short as possible, since Google Analytics sometimes limits the number of characters you can use in a filter. Your subdomains are **subdomain1.mysite.com**, **subdomain2.mysite.com**, **subdomain3.mysite.com**, **subdomain4.mysite.com**, **www.mysite.com**, **store.mysite.com** and **blog.mysite.com**
- Write a funnel and goal that includes three steps for the funnel and the final goal step (four steps in all), using Regular Expressions. Notice that there are two ways to achieve Step 2. Here are the three pages:
  1. **/store/**
  2. **/store/creditcard** or **store/moneyorder**
  3. **/store/shipping**
  4. **/store/thankyou**

REGULAR EXPRESSIONS FOR GOOGLE ANALYTICS

Original ebook design: Fireman Creative

BIOS

ABOUT BOUNTEOUS

Bounteous is a Google Analytics Certified Partner in Pittsburgh, PA. The company consults in Traffic, Analysis and Action – that translates to SEO/PPC, Google Analytics, and User Testing/ Website Optimizer. Good contact information for Bounteous is getinfo@bounteous.com or directly at (877) 220-5862. If you downloaded this ebook and aren't sure where to go back to see it online, the site address is www.bounteous.com

ABOUT ROBBIN STEIF

The author, Robbin Steif, is the former owner of LunaMetrics, now Bounteous. She drove the team at Google Analytics a little crazy in her quest to get better GA documentation, and they lovingly obliged by including many of her thoughts in their Regular Expressions help sections. Steif is a frequent speaker on analytics in general and Google Analytics in particular. She has been chair of the Web Analytics Association's (WAA) Marketing committee, and has served a two year WAA Board of Directors term. Steif is a graduate of Harvard College and the Harvard Business School.

Robbin loves to hear from users, and she welcomes your questions and comments. You can reach her like this:

SKYPE robbinsteif
TWITTER robbinsteif

… or the old-fashioned way:
PHONE (412) 381-5500

SOME THANK-YOUS

To Nick M, who stood in a Palo Alto bowling alley with me and discussed Regular Expressions for Regular People (that was the first name for this ebook). To Steve in Australia and Justin, who taught me RegEx. To David Meerman Scott and Jonathan Kranz – I have never met you, but you taught me (respectively) why and how to write an ebook. A big thank you to my team at Fireman Creative for the care and feeding of this ebook.