Degrees Minutes Seconds (DMS) RegEx

Asked 12 years ago Modified 12 years ago Viewed 10k times

6

I have a regular expression that I want to match a latitude/longitude pair in a variety of fashions, e.g.

```
123 34 42
-123* 34' 42"
123* 34' 42"
+123* 34' 42"
45* 12' 22"N
45 12' 22"S
90:00:00.0N
```

I want to be able to match these in a pair such that

`90:00:00.0N 180:00:00.0E` is a latitude/longitude pair.

or

`45* 12' 22"N 46* 12' 22"E` is a latitude/longitude pair (1 degree by 1 degree cell).

or

`123* 34' 42" 124* 34' 42"` is a latitude/longitude pair

etc

Using the below regular expression, when I type in 123, it matches. I suppose this is true since 123 00 00 is a valid coordinate. However, I want to use this regular expression to match pairs in the same format above

```
   "([-|\\+]?\\d{1,3}[d|D|\u00B0|\\s](\\s*\\d{1,2}['|\u2019|\\s])?"
 + "(\\s*\\d{1,2}[\"|\u201d|\\s])?\\s*([N|n|S|s|E|e|W|w])?\\s?)"
```

I am using Java.

***** denotes a degree.

What am I doing wrong in my regular expression?

java regex latitude-longitude

asked Nov 28, 2012 at 18:09
user195488

Your last example is *not* a lat/long pair. There's no such thing as 123* North. – John Nov 28, 2012 at 18:15

@John: I know, that's the problem. It's not matching pairs. As far as being a valid coordinate, that's another problem. – user195488 Nov 28, 2012 at 18:16

@AlexWien: Not concerned about decimal at the moment and don't be too concerned about the numbers, I am just concerned about the format. – user195488 Nov 28, 2012 at 18:19

I'm not sure what's wrong with your regex, but the problem is you are trying to write the whole regex in one go, which makes it horribly hard to detect error and/or maintain it. – nhahtdh Nov 28, 2012 at 18:34

1 @0A0D: ideone.com/jyaVVm Not sure if this is a good way to go, but at least, my head doesn't explode in the process. – nhahtdh Nov 28, 2012 at 19:08

3 Answers

Sorted by: Highest score (default)

12

Well, for one thing, you're filling your character sets with a bunch of unnecessary pipe characters - alternation is implied in a `[]` pair. Additional cleanup: `+` doesn't need to be escaped in a character class. Your regular expression seems to be addressing a bigger problem statement than you gave us - you make no mention of `d` or `D` as matchable character. And you've made pretty much the entire back half of your RegEx optional. Going off of what I think your original problem statement is, I built the following regular expression:

```
^\s*([+-]?\d{1,3}\*?\s+\d{1,2}'?\s+\d{1,2}"?[NSEW]?|\d{1,3}(:\d{2}){2}\.\d[NSEW]\s*){1,2}$
```

It's a bit of a doozy, but I'll break it down for you, or anyone who happens across this in the future (Hello, future!).

```
^
```

Start of string, simple.

```
\s*
```

Any amount of whitespace - even none.

```
(
```

Denotes the beginning of a group - we'll get back to that.

```
[-+]?
```

An optional sign

```
\d{1,3}
```

1 to three digits

```
\*?
```

An optional Asterisk - the escape here is key for an asterisk, but if you want to replace this with the unicode codepoint for an actual degree, you won't need it.

```
\s+
```

At least one character of whitespace

```
\d{1,2}
```

1 or two digits.

```
'?
```

Optional apostrophe

```
\s+\d{1,2}+
```

You've seen these before, but there's a new curveball - there's a plus after the `{1,2}` quantifier! This makes it a *possessive quantifier*, meaning that the matcher won't give up its matches for this group to make another one possible. This is almost exclusively here to prevent `1 1 11 1 1` from matching, but can be used to increase speed anywhere you're 100% sure you don't need to be able to backtrack.

```
"?
```

Optional double quote. You'll have to escape this in Java.

```
[NSEW]?
```

An optional cardinal direction, designated by letter

```
|
```

OR - you can match everything in the group before this, or everything in the group after this.

```
\d{1,3}
```

Old news.

```
(:\d{2})
```

A colon, followed by two characters...

```
{2}
```

twice!

```
\.\d
```

Decimal point, followed by a single digit.

```
[NSEW]
```

Same as before, but this time it's mandatory.

```
\s*)
```

Some space, and finally the end of the group. Now, the first group has matched an entire longitude/latitude denotation, with an arbitrary amount of space at the end. Followed closely by:

```
{1,2}
```

Do that one, or two times - to match a single or a pair, then finally:

```
$
```

The end of the string.

This isn't perfect, but it's pretty close, and I think it answers the original problem statement. Plus, I feel my explanation has demystified it enough that you can edit it to further suit your needs. The one thing it doesn't (and won't) do, is enforce that the first coordinate matches the second in style. That's just too much to ask of Regular Expressions.

Doubters: Here it is in action. Please, enjoy.

Share

Improve this answer

Follow

edited Nov 29, 2012 at 1:59

answered Nov 28, 2012 at 23:03

FrankieTheKneeMan

6,790 ● 2 ● 28 ● 39

`The one thing it doesn't (and won't) do, is enforce that the first coordinate matches the second in style. That's just too much to ask of Regular Expressions.` --> It is possible to do so without exerting too much effort - check my code in the comment to the question. I even do checking on the coordinate to make sure it is in valid range. – nhahtdh Nov 29, 2012 at 4:36

Yes, but yours use three regular expressions, and must check every one of them. There's argument there for stringing them together with alternation, but it's still not really a great regular expression. The best choice here is to see if it's a lat/lon formatted pair, then extract the values and do error checking on those. – FrankieTheKneeMan Nov 29, 2012 at 5:09

`It's still not really a great regular expression` - I don't get how you judge "great" here. Performance - yeah, may not be great. But being succinct is not a very good criteria for this. `The best choice here is to see if it's a lat/lon formatted pair, then extract the values and do error checking on those.` There are pros and cons to using a relaxed (and extract candidate) and a strict regex (directly get a valid token) - can't say without looking at the data. – nhahtdh Nov 29, 2012 at 5:50

1 Modified the regular expression to support decimal degrees too.. `^([+-]?\d{1,3}\*?\s+\d{1,2}'?\s+\d{1,2}+"?[NSEW]?|\d{1,3}(:\d{2}){2}\.\d[NSEW]\s*){1,2}$|\d{1,3}(.\d{1,9})` – user195488 Nov 29, 2012 at 19:16

1

Generally, I dont think that this is a good approach. In your interface try to have DMS coordinates in one specific format.
The User should enter this in 3 separate text fields.

Further this regex is not very maintainable.

There are much more possibilities to notate a DMS coordinate, you even cannot imagine. Humans are creative.

Eg:

Put N,S in front
or: North, 157 deg 50 min 55.796 sec
or: from wiki: The NGS now says in 1993 that point was 21-18-02.54891 N 157-50-45.90280 W

Share
Improve this answer
Follow

edited Nov 28, 2012 at 18:29

answered Nov 28, 2012 at 18:23

AlexWien
28.7k ● 6 ● 55 ● 84

Yes, this is true, however, this user has a desired range of options. I am aware of the limitations. – user195488 Nov 28, 2012 at 18:32

Then let the user choose from a set of x DMS notations, and send the dmsNotationID along with the DMS coordinates. Otherewise this will end up in chaos. – AlexWien Nov 28, 2012 at 18:36

0

I'm not a RE wizard but with your formats you'd need to have some kind of convention for which pair comes first (probably latitude) if you're doing parsing from a single text box.

From there, you have six numeric fields (deg, min, sec for each, possibly with a decimal point), two signs (+ or - for each) and up to two hemispheres (one for each).

As far as I can see, parsing these 8-10 fields from your input would occur in the same order each time if you demanded only that the latitude is first, and the longitude second. The rest of the symbols (save the decimal point(s)) can be treated essentially as separators.

Does that make it easier?

Share Improve this answer Follow

answered Nov 28, 2012 at 18:35

John
16k ● 10 ● 74 ● 113