

How can I validate an email address using a regular expression?

Asked 16 years, 2 months ago Modified 7 months ago

Viewed 2.7m times



4252



Over the years I have slowly developed a [regular expression](#) that validates *most* email addresses correctly, assuming they don't use an IP address as the server part.

I use it in several PHP programs, and it works most of the time. However, from time to time I get contacted by someone that is having trouble with a site that uses it, and I end up having to make some adjustment (most recently I realized that I wasn't allowing four-character [TLDs](#)).

What is the best regular expression you have or have seen for validating emails?

I've seen several solutions that use functions that use several shorter expressions, but I'd rather have one long complex expression in a simple function instead of several short expression in a more complex function.

regex

Share


edited May 4, 2022 at 13:51


Improve this question

Follow

community wiki


19 revs, 15 users 52%
acrosman

11 The regex that can validate that an IDNA is correctly formatted does not fit in stackexchange. (the rules on canonicalisation are really tortuous and particularly ill-suited to regex processing) – [Jasen](#) Aug 29, 2017 at 23:51 

15 Why you should not do this: [Can it cause harm to validate email addresses with a regex?](#) – [klutt](#) Jan 9, 2018 at 14:30 

The regexes may be **variable** as in some cases, an email can contain a space, and in other times, it cannot contain any spaces. – [Mũtĩgǎñácěơữ\\$](#) Jul 23, 2018 at 4:21

You can check Symfonys regex for loose and strict check: github.com/symfony/symfony/blob/5.x/src/Symfony/Component/... – [Did](#) May 16, 2021 at 16:15

Using just regex can harm server security but if it is just as an input pattern, i suggest use this: stackoverflow.com/questions/5601647/... – [MMMahdy-PAPION](#) Jun 7, 2021 at 21:42 

77 Answers

Sorted by:

Highest score (default)



1

2

3

Next



3465



+50



The [fully RFC 822 compliant regex](#) is inefficient and obscure because of its length. Fortunately, RFC 822 was superseded twice and the current specification for email addresses is [RFC 5322](#). RFC 5322 leads to a regex that can be understood if studied for a few minutes and is efficient enough for actual use.

One RFC 5322 compliant regex can be found at the top of the page at <http://emailregex.com/> but uses the IP address pattern that is floating around the internet with a bug that allows `00` for any of the unsigned byte decimal values in a dot-delimited address, which is illegal. The rest of it appears to be consistent with the RFC 5322 grammar and passes several tests using `grep -Po`, including cases domain names, IP addresses, bad ones, and account names with and without quotes.

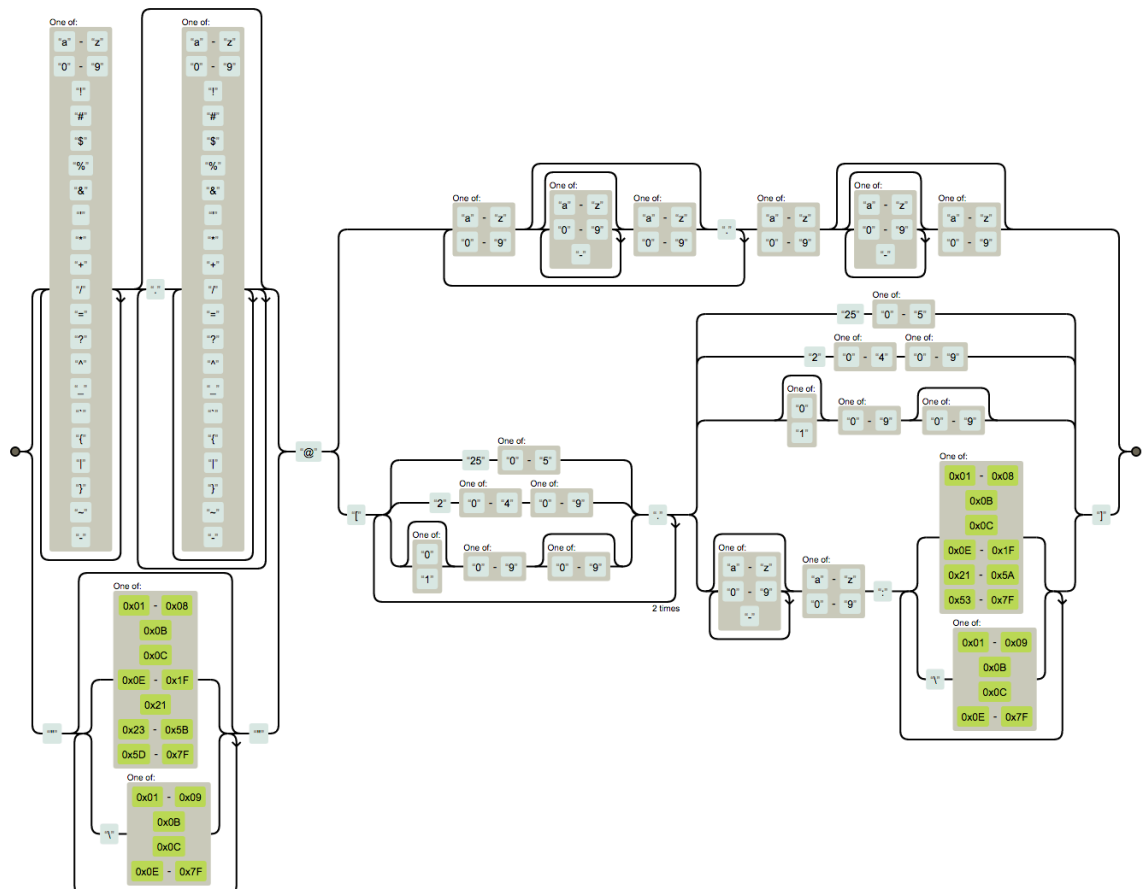
Correcting the `00` bug in the IP pattern, we obtain a working and fairly fast regex. (Scrape the rendered version, not the markdown, for actual code.)

```
(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*)"(?:\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f)|\x01-\x09\x0b\x0c\x0e-\x7f))*")@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)|\[(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|1[0-9][0-9][1-9]?[0-9])\]{3}(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|1[0-9][0-9][1-9]?[0-9])|[a-z0-9-]*[a-z0-9](?:\x01-
```

or:

```
(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\. [a-z0-9!#$%&'*/+=?^_`{|}~-]+)*)"(?:[\x01-
\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\[\x01-
\x09\x0b\x0c\x0e-\x7f])*)" @ (?: (?: [a-z0-9]
(?: [a-z0-9-]* [a-z0-9])? \. ) + [a-z0-9] (?: [a-z0-9-]*
[a-z0-9])? |\[(?: (?: (2(5[0-5]| [0-4][0-9])| 1[0-9]
[0-9]| [1-9]?[0-9])) \. ) {3} (?: (2(5[0-5]| [0-4][0-9])| 1[0-9]
[0-9]| [1-9]?[0-9])| [a-z0-9-]* [a-z0-9]:
(?: [\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-
\x7f]|\[\x01-\x09\x0b\x0c\x0e-\x7f])+) \]) )
```

Here is [diagram](#) of [finite state machine](#) for above regexp which is more clear than regexp itself



The more sophisticated patterns in Perl and PCRE (regex library used e.g. in PHP) can [correctly parse RFC 5322 without a hitch](#). Python and C# can do that too, but they use a different syntax from those first two. However, if you are forced to use one of the many less powerful pattern-matching languages, then it's best to use a real parser.

It's also important to understand that validating it per the RFC tells you absolutely nothing about whether that address actually exists at the supplied domain, or whether the person entering the address is its true owner. People sign others up to mailing lists this way all the time. Fixing that requires a fancier kind of validation that involves sending that address a message that includes a confirmation token meant to be entered on the same web page as was the address.

Confirmation tokens are the only way to know you got the address of the person entering it. This is why most mailing lists now use that mechanism to confirm sign-ups. After all, anybody can put down

`president@whitehouse.gov`, and that will even parse as legal, but it isn't likely to be the person at the other end.

For PHP, you should *not* use the pattern given in [Validate an E-Mail Address with PHP, the Right Way](#) from which I quote:

There is some danger that common usage and widespread sloppy coding will establish a de

facto standard for e-mail addresses that is more restrictive than the recorded formal standard.

That is no better than all the other non-RFC patterns. It isn't even smart enough to handle even [RFC 822](#), let alone RFC 5322. [This one](#), however, is.

If you want to get fancy and pedantic, [implement a complete state engine](#). A regular expression can only act as a rudimentary filter. The problem with regular expressions is that telling someone that their perfectly valid e-mail address is invalid (a false positive) because your regular expression can't handle it is just rude and impolite from the user's perspective. A state engine for the purpose can both validate and even correct e-mail addresses that would otherwise be considered invalid as it disassembles the e-mail address according to each RFC. This allows for a potentially more pleasing experience, like

The specified e-mail address
'myemail@address.com' is invalid. Did you mean
'myemail@address.com'?

See also [Validating Email Addresses](#), including the comments. Or [Comparing E-mail Address Validating Regular Expressions](#).



[Debuggex Demo](#)

Share Improve this answer

edited May 4, 2022 at 13:53

Follow

community wiki

25 revs, 20 users 17%

[bortzmeyer](#)

-
- 269 You said "There is no good regular expression." Is this general or specific to e-mail address validation? – [Tomalak](#) Oct 14, 2008 at 14:33
-
- 65 @Tomalak: only for email addresses. As bortzmeyer said, the RFC is extremely complicated – [Luk](#) Oct 14, 2008 at 16:23
-
- 48 The linux journal article you mention is factually wrong in several respects. In particular Lovell clearly hasn't read the errata to RFC3696 and repeats some of the errors in the published version of the RFC. More here: dominicsayers.com/isemail – [Dominic Sayers](#) Apr 8, 2009 at 15:56
-
- 10 Note that [the current HTML5 spec](#) includes a regex and ABNF for email-type input validation that is deliberately more restrictive than the original RFCs. – [Synchro](#) Sep 9, 2014 at 16:14
-
- 19 RFC 822, section 6.2.4. specifically and explicitly allows capital letters, but this answer does not. w3.org/Protocols/rfc822/#z26 Perhaps the author of this answer intended for their regex to be applied insensitively. If so, that should be made explicit in the body of the answer. – [Jared Beck](#) Apr 10, 2019 at 22:36
-



You should not use regular expressions to validate email addresses.

854

Instead, in C# use the [MailAddress](#) class, like this:



```
try {  
    address = new MailAddress(address).Address;  
} catch (FormatException) {  
    // address is invalid  
}
```



The `MailAddress` class uses a [BNF](#) parser to validate the address in full accordance with RFC822.

If you plan to use the `MailAddress` to validate the e-mail address, be aware that this approach accepts the display name part of the e-mail address as well, and that may not be exactly what you want to achieve. For example, it accepts these strings as valid e-mail addresses:

- "user1@hotmail.com; user2@gmail.com"
- "user1@hotmail.com; user2@gmail.com; user3@company.com"
- "User Display Name user3@company.com"
- "user4 @company.com"

In some of these cases, only the last part of the strings is parsed as the address; the rest before that is the display name. To get a plain e-mail address without any display name, you can check the normalized address against your original string.


```
bool isValid = false;

try
{
    MailAddress address = new MailAddress(emailAddress);
    isValid = (address.Address == emailAddress);
    // or
    // isValid = string.IsNullOrEmpty(address.DisplayName);
}
catch (FormatException)
{
    // address is invalid
}
```

Furthermore, an address having a dot at the end, like `user@company.` is accepted by `MailAddress` as well.

If you really want to use a regex, [here it is](#):

```
(?: (?: \r\n)? [ \t] ) * (?: (?: (?: [^ ( ) <> @, ; : \\ " . \ [ \ ] \000-\031] + (?: (?: (?: \r\n)? [ \t] ) + | \Z | (?: = [ \ [ " ( ) <> @, ; : \\ " . \ [ \ ] ] ) ) | " (?: [ ^ \ " \r \ \ ] | \\. | (?: (?: \r\n)? [ \t] ) ) * " (?: (?: \r\n)? [ \t] ) * ) (?: \. (?: (?: \r\n)? [ \t] ) * (?: [ ^ ( ) <> @, ; : \\ " . \ [ \ ] \000-\031] + (?: (?: (?: \r\n)? [ \t] ) + | \Z | (?: = [ \ [ " ( ) <> @, ; : \\ " . \ [ \ ] ] ) ) | " (?: [ ^ \ " \r \ \ ] | \\. | (?: (?: \r\n)? [ \t] ) ) * " (?: (?: \r\n)? [ \t] ) * ) * @ (?: (?: \r\n)? [ \t] ) * (?: [ ^ ( ) <> @, ; : \\ " . \ [ \ ] \000-\031] + (?: (?: (?: \r\n)? [ \t] ) + | \Z | (?: = [ \ [ " ( ) <> @, ; : \\ " . \ [ \ ] ] ) ) | \ ( ( [ ^ \ [ \ ] \r \ \ ] | \\. ) * \ ] (?: (?: \r\n)? [ \t] ) * ) (?: \. (?: (?: \r\n)? [ \t] ) * (?: [ ^ ( ) <> @, ; : \\ " . \ [ \ ] \000-\031] + (?: (?: (?: \r\n)? [ \t] ) + | \Z | (?: = [ \ [ " ( ) <> @, ; : \\ " . \ [ \ ] ] ) ) | \ ( ( [ ^ \ [ \ ] \r \ \ ] | \\. ) * \ ] (?: (?: \r\n)? [ \t] ) * ) ) * | (?: [ ^ ( ) <> @, ; : \\ " . \ [ \ ] \000-\031] + (?: (?: (?: \r\n)? [ \t] ) + | \Z | (?: = [ \ [ " ( ) <> @, ; : \\ " . \ [ \ ] ] ) ) | " (?: [ ^ \ " \r \ \ ] | \\. | (?: (?: \r\n)? [ \t] ) ) * " (?: (?: \r\n)? [ \t] ) * ) * \< (?: (?: \r\n)? [ \t] ) * (?: @ (?: [ ^ ( )
```

```
<>@,;:\\".\\[\\ \000-\031]+(?::(?:\r\n)?[ \t])+|Z|(?=[\"()<>@,;:\\".\\[\\]))|\\([\\^\\[\\]\\\\r\\\\\\|\\\\\\.)*\\(?::(?:\r\n)?[ \t])*(?:\\.\\(?::(?:\r\n)?[ \t])*(?:\[\"()<>@,;:\\".\\[\\ \000-\031]+(?::(?:\r\n)?[ \t])+|Z|(?=[\"()<>@,;:\\".\\[\\]))|\\([\\^\\[\\]\\\\r\\\\\\|\\\\\\.)*\\(?::(?:\r\n)?[ \t])*)*)*(?:,|@(?::(?:\r\n)?[ \t])*(?:\[\"()<>@,;:\\".\\[\\ \000-\031]+(?::(?:\r\n)?[ \t])+|Z|(?=[\"()<>@,;:\\".\\[\\]))|\\([\\^\\[\\]\\\\r\\\\\\|\\\\\\.)*\\(?::(?:\r\n)?[ \t])*)*)*(?::(?:\r\n)?[ \t])*(?:\[\"()<>@,;:\\".\\[\\ \000-\031]+(?::(?:\r\n)?[ \t])+|Z|(?=[\"()<>@,;:\\".\\[\\]))|\\([\\^\\[\\]\\\\r\\\\\\|\\\\\\.)*\\(?::(?:\r\n)?[ \t])*)*)*(?::(?:\r\n)?[ \t])*(?:\[\"()<>@,;:\\".\\[\\ \000-
```

Share Improve this answer

edited Feb 13, 2022 at 15:22

Follow



Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Dec 14, 2009 at 20:43



SLaks

886k ● 181 ● 1.9k ● 2k

53 You'll find that the MailAddress class in .NET 4.0 is far better at validating email addresses than in previous versions. I made some significant improvements to it. – [Jeff Tucker](#) Dec 15, 2009 at 9:56

10 I think it sort of... doesn't work... for simpler ids. a@b doesn't validate. ar@b.com matches only till ar@b , the .com is not matched. However, something like "I am me"@[10.10.10.10] does work! :) – [Raze](#) Dec 15, 2009 at 11:24

14 Be warned that these RFC compliant regex validators will let through a lot of email addresses that you probably wouldn't want to accept such as "a<body/onload=alert('lol.com?'+document.cookies) @a.a>"

which is a valid email address in perl's Email::Valid (which uses that huge regex), and can be exploited for XSS
rt.cpan.org/Public/Bug/Display.html?id=75650

– [Matthew Lock](#) Sep 28, 2012 at 6:03 

16 @MatthewLock: That is no worse than `fake@not-a-real-domain.name`. You **must not** rely on email validation to prevent XSS. – [SLaks](#) Sep 28, 2012 at 17:19

16 @MatthewLock: **No**. You need to *escape* SQL queries (or, better yet, use parameters). Sanitization is not a proper defense. – [SLaks](#) Mar 2, 2016 at 14:49



621

This question is asked a lot, but I think you should step back and ask yourself *why* you want to validate email addresses syntactically? What is the benefit really?



- It will not catch common typos.
- It does not prevent people from entering invalid or made-up email addresses, or entering someone else's address for that matter.




If you want to validate that an email is correct, you have no choice than to send a confirmation email and have the user reply to that. In many cases you will *have* to send a confirmation mail anyway for security reasons or for ethical reasons (so you cannot e.g. sign someone up to a service against their will).

Share Improve this answer

[edited Jul 2, 2021 at 9:18](#)

Follow

-
- 129 It might be worth checking that they entered something@something into the field in a client side validation just to catch simple mistakes - but in general you are right. – [Martin Beckett](#) Aug 25, 2009 at 16:25
-
- 159 @olavk: if someone enters a typo (eg: `me@hotmail`), they're obviously not going to get your confirmation email, and then where are they? They're not on your site any more and they're wondering why they couldn't sign up. Actually no they're not - they've completely forgotten about you. However, if you could just do a basic sanity check with a regex while they're still with you, then they can catch that error straight away and you've got a happy user. – [nickf](#) Jun 2, 2010 at 13:53 
-
- 6 @JacquesB: You make an excellent point. Just because it passes muster per the RFC doesn't mean it is really that user's address. Otherwise all those `president@whitehouse.gov` addresses indicate a very netbusy commander-in-chief. :) – [tchrist](#) Nov 7, 2010 at 20:09
-
- 63 It doesn't have to be black or white. If the e-mail looks wrong, let the user know that. If the user still wants to proceed, let him. Don't force the user to conform to your regex, rather, use regex as a tool to help the user know that there might be a mistake. – [ninjaneer](#) Feb 18, 2014 at 2:56
-
- 3 Security. If you validate email addresses properly and only allow safe characters chances of the email address being used in some sort of malicious way reduces drastically. For example in email header exploits. – [Gellweiler](#) Nov 5, 2021 at 10:49
-



578



It all depends on how accurate you want to be. For my purposes, where I'm just trying to keep out things like `bob` `@ aol.com` (spaces in emails) or `steve` (no domain at all) or `mary@aolcom` (no period before .com), I use

```
/^\\S+@\\S+\\.\\S+$/
```

Sure, it will match things that aren't valid email addresses, but it's a matter of getting common simple errors.

There are any number of changes that can be made to that regex (and some are in the comments for this answer), but it's simple, and easy to understand, and is a fine first attempt.

Share Improve this answer

edited May 6, 2020 at 22:26


Follow

community wiki
6 revs, 5 users 72%
Andy Lester

9 It does not match `foobar@dk` which is a valid and working email address (although probably most mail servers won't accept it or will add something.com.) – [bortzmeyer](#) Oct 14, 2008 at 19:30

7 @Richard: `.` is included in `\\S` . – [David Thornley](#) Dec 17, 2009 at 18:48

73 JJJ: Yes, it will match a lot of crap. It will match &\$*#\$(@\$0(%))\$#.)&*)(*\$, too. For me, I'm more concerned with catching the odd fumble-finger typo like `mary@aolcom` than I am complete garbage. YMMV. – [Andy Lester](#) Oct 16, 2012 at 16:03

17 Just to control for @ signs: `/^[^\\s@]+@[^\\s@]+\\. [^\\s@]{2,}$/` [jsfiddle.net/b9chris/mXB96](#) – [Chris Moschini](#) Aug 4, 2014 at 21:32 

33 And another common typo: two consecutive dots in domain name or a comma instead of a dot.
`^[^\\s@]+@([^\\s@.,]+\\.)+[^\\s@.,]{2,}$`
– [Piskvor left the building](#) Sep 24, 2015 at 9:12



It depends on what you mean by best: If you're talking about catching every valid email address use the following:

384



```
(?: (?: \r\n )? [ \t ] ) * (?: (?: (?: [^ ( ) < > @ , ; : \ \" . \ [ \ ] \ 000 - \ 0
) + | \ Z | ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ] ) ) | " (?: [ ^ \" \r \ \ ] | \ \ . | (?:
\r \ n )? [ \ t ] ) * ) (?: \ . (?: (?: \r \ n )? [ \ t ] ) * (?: [ ^ ( ) < > @ , ; : \ \"
?: \r \ n )? [ \ t ] ) + | \ Z | ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ] ) ) | " (?: [ ^ \"
\ t ] ) ) * " (?: (?: \r \ n )? [ \ t ] ) * ) * @ (?: (?: \r \ n )? [ \ t ] ) * (?: [ ^
31 ] + (?: (?: (?: \r \ n )? [ \ t ] ) + | \ Z | ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ]
) (?: (?: \r \ n )? [ \ t ] ) * ) (?: \ . (?: (?: \r \ n )? [ \ t ] ) * (?: [ ^ ( ) < >
(?: (?: (?: \r \ n )? [ \ t ] ) + | \ Z | ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ] ) ) | \
(?: \r \ n )? [ \ t ] ) * ) ) * | (?: [ ^ ( ) < > @ , ; : \ \" . \ [ \ ] \ 000 - \ 031 ] + (
| ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ] ) ) | " (?: [ ^ \" \r \ \ ] | \ \ . | (?: (?: \r
? [ \ t ] ) * ) * \ < (?: (?: \r \ n )? [ \ t ] ) * (?: @ (?: [ ^ ( ) < > @ , ; : \ \" . \ [
r \ n )? [ \ t ] ) + | \ Z | ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ] ) ) | \ ( [ ^ \ [ \ ] \r
\ t ] ) * ) (?: \ . (?: (?: \r \ n )? [ \ t ] ) * (?: [ ^ ( ) < > @ , ; : \ \" . \ [ \ ] \
? [ \ t ] ) + | \ Z | ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ] ) ) | \ ( [ ^ \ [ \ ] \r \ \ ] |
) * ) ) * (?: , @ (?: (?: \r \ n )? [ \ t ] ) * (?: [ ^ ( ) < > @ , ; : \ \" . \ [ \ ] \ 00
\ t ] ) + | \ Z | ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ] ) ) | \ ( [ ^ \ [ \ ] \r \ \ ] | \ \
) (?: \ . (?: (?: \r \ n )? [ \ t ] ) * (?: [ ^ ( ) < > @ , ; : \ \" . \ [ \ ] \ 000 - \ 0
) + | \ Z | ( ? = [ \ [ " ( ) < > @ , ; : \ \" . \ [ \ ] ] ) ) | \ ( [ ^ \ [ \ ] \r \ \ ] | \ \ . ) * \
* : (?: (?: \r \ n )? [ \ t ] ) * ) (?: [ ^ ( ) < > @ , ; : \ \" . \ [ \ ] \ 000 - \ 031
```



|\\Z|(?=[\\["()<>@,;:\\\\".\\[\\]])|"(?:[\\^\\\\"r\\\\"|\\\\.|(?:(?\\n)?[\\t]))*(?:\\. (?: (?:\\r\\n)?[\\t]))*(?:[\\^()<>@,;:\\\\".\\r\\n)?[\\t])+|\\Z|(?=[\\["()<>@,;:\\\\".\\[\\]])|"(?:[\\^\\\\"r\\"])*"(?: (?:\\r\\n)?[\\t]))*@(?: (?:\\r\\n)?[\\t]))*(?:[\\^()]+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\["()<>@,;:\\\\".\\[\\]]))?: (?:\\r\\n)?[\\t]))*(?:\\. (?: (?:\\r\\n)?[\\t]))*(?:[\\^()<>@,; (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\["()<>@,;:\\\\".\\[\\]]))|\\[(?:\\r\\n)?[\\t]))*\\>(?: (?:\\r\\n)?[\\t]))*|(?:[\\^()<>@,;:\\ (?:\\r\\n)?[\\t])+|\\Z|(?=[\\["()<>@,;:\\\\".\\[\\]]))|"(?:[\\^ [\\t]))*"(?: (?:\\r\\n)?[\\t]))*:(?: (?:\\r\\n)?[\\t]))*(?: (\\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\["()<>@,;:\\\\ \\. |(?: (?:\\r\\n)?[\\t])))*"(?: (?:\\r\\n)?[\\t]))*(?:\\. (?: (@,;:\\\\".\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[(?:[\\^\\\\"r\\\\"|\\\\. |(?: (?:\\r\\n)?[\\t])))*"(?: (?:\\r\\n)?[\\t])*(?:[\\^()<>@,;:\\\\".\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]".\\[\\]))|\\[(\\^\\[\\]r\\\\"|\\\\.)*\\](?: (?:\\r\\n)?[\\t]))*(?: [\\^()<>@,;:\\\\".\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+| \\]))|\\[(\\^\\[\\]r\\\\"|\\\\.)*\\](?: (?:\\r\\n)?[\\t]))*| (?: \\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\["()<>@,;:\\\\".\\[\\ ?:(?:\\r\\n)?[\\t])))*"(?: (?:\\r\\n)?[\\t]))*\\<(?: (?:\\r\\n) :\\\\".\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\[" ^\\[\\]r\\\\"|\\\\.)*\\](?: (?:\\r\\n)?[\\t]))*(?:\\. (?: (?:\\r\\n) .\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\["()< >\\r\\\\"|\\\\.)*\\](?: (?:\\r\\n)?[\\t]))*(?:, @ (?: (?:\\r\\n)? [\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\["()<>@, r\\\\"|\\\\.)*\\](?: (?:\\r\\n)?[\\t]))*(?:\\. (?: (?:\\r\\n)?[\\t] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\["()<>@,;:\\|\\\\.)*\\](?: (?:\\r\\n)?[\\t]))*))*:(?: (?:\\r\\n)?[\\t]))*? 00-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\["()<>@,;:\\\\". .|(?: (?:\\r\\n)?[\\t])))*"(?: (?:\\r\\n)?[\\t]))*(?:\\. (?: (?: ;:\\\\".\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z|(?=[\\[:[\\^\\\\"r\\\\"|\\\\. |(?: (?:\\r\\n)?[\\t])))*"(?: (?:\\r\\n)?[\\t]) (?:[\\^()<>@,;:\\\\".\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]) \\[\\]))|\\[(\\^\\[\\]r\\\\"|\\\\.)*\\](?: (?:\\r\\n)?[\\t]))*(?:\\ ^()<>@,;:\\\\".\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]))+|\\Z])|\\[(\\^\\[\\]r\\\\"|\\\\.)*\\](?: (?:\\r\\n)?[\\t]))*\\>(?: (?:(?:[\\^()<>@,;:\\\\".\\[\\] \\000-\\031)+(?: (?: (?:\\r\\n)?[\\t]".\\[\\]))|"(?:[\\^\\\\"r\\\\"|\\\\. |(?: (?:\\r\\n)?[\\t])))*"(?:(? ?:\\r\\n)?[\\t]))*(?:[\\^()<>@,;:\\\\".\\[\\] \\000-\\031)+(?: (?: \\["()<>@,;:\\\\".\\[\\]))|"(?:[\\^\\\\"r\\\\"|\\\\. |(?: (?:\\r\\n)? []))*))*@(?: (?:\\r\\n)?[\\t]))*(?:[\\^()<>@,;:\\\\".\\[\\] \\000-\\])+|\\Z|(?=[\\["()<>@,;:\\\\".\\[\\]))|\\[(\\^\\[\\]r\\\\"|\\\\.)* :\\. (?: (?:\\r\\n)?[\\t]))*(?:[\\^()<>@,;:\\\\".\\[\\] \\000-\\031] \\Z|(?=[\\["()<>@,;:\\\\".\\[\\]))|\\[(\\^\\[\\]r\\\\"|\\\\.)*\\](?

(<http://www.ex-parrot.com/~pdw/Mail-RFC822-Address.html>) If you're looking for something s

```
"^[a-zA-Z0-9_+.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+$."
```

This regular expression will only validate addresses that have had any comments stripped and replaced with whitespace (this is done by the module).

community wiki
8 revs, 6 users 91%
Good Person

-
- 66 Can you give me an example of some `email address` that wrongly passes through the second one, but is caught by the longer regex? – [Lazer](#) May 15, 2010 at 18:32
-
- 4 Much though I did once love it, that's an RFC 822 validator, not [an RFC 5322](#) one. – [tchrist](#) Nov 7, 2010 at 20:17 
-
- 28 @Lazer in..valid@example.com would be a simple example. You aren't allowed to have two consecutive unquoted dots in the local-part. – [Randal Schwartz](#) Dec 6, 2011 at 18:04
-
- 5 @Mikhail perl but you shouldn't actually use it. – [Good Person](#) Jan 8, 2013 at 18:48
-
- 3 @RSC that is a FQDN which is fine – [Good Person](#) May 28, 2014 at 23:16 
-

Per [the W3C HTML5 specification](#):

```
^[a-zA-Z0-9.!#$%&'*/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9](?:[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*)$
```

Context:

A **valid e-mail address** is a string that matches the ABNF production [...].



351



Note: This requirement is a [willful violation](#) of [RFC 5322](#), which defines a syntax for e-mail addresses that is simultaneously too strict (before the “@” character), too vague (after the “@” character), and too lax (allowing comments, whitespace characters, and quoted strings in manners unfamiliar to most users) to be of practical use here.

The following JavaScript- and Perl-compatible regular expression is an implementation of the above definition.

```
/^[a-zA-Z0-9.!#$%&'*/=\?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-
```

Share Improve this answer

edited Feb 13, 2022 at 16:14

Follow

community wiki

10 revs, 8 users 40%

Rory O'Kane

13 This is interesting. It's a violation of RFC, but a willful one and it makes sense. Real world example: gmail ignores dots in the part before @, so if your email is test@gmail.com you can send emails to test.@gmail.com or test....@gmail.com, both of those addresses are invalid according to RFC, but valid in real world. – [valentinas](#) Jan 16, 2013 at 5:04

2 I think last part should be '+' instead of '*': ^[a-zA-Z0-9.!#\$%&'*/=\?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)+\$

– mmmmmm Jan 21, 2013 at 12:12 ✎

12 @mmmmmm `john.doe@localhost` is valid. For sure, in a real world application (i.e. a community), I'd like your suggest to replace * by + – rabudde Feb 1, 2013 at 10:03

3 @valentinas Actually, the RFC does *not* preclude these local parts, but they have to be quoted. `"test...."@gmail.com` is perfectly valid according to the RFC and semantically equivalent to `test....@gmail.com`. – Rinke Nov 17, 2014 at 9:01

I get an error while trying to send email using python through my company's relay if I try to send to an address with a `.@` or `..@`. Actually that is also the case with a `_@`. I rather remove those before sending than trusting that the recipient will do it.
– ndvo Feb 11, 2016 at 11:31



299



[UPDATED] I've collated everything I know about email address validation at <http://isemail.info>, which now not only validates, but it also diagnoses problems with email addresses. I agree with many of the comments here that validation is only part of the answer; see my essay [What is a valid email address?](#).

`is_email()` remains, as far as I know, the only validator that will tell you definitively whether a given string is a valid email address or not. I've uploaded a new version at <http://isemail.info/>

I collated test cases from Cal Henderson, Dave Child, Phil Haack, Doug Lovell, [RFC 5322](#) and [RFC 3696](#). 275 test addresses in all. I ran all these tests against all the free validators I could find.

I'll try to keep this page up-to-date as people enhance their validators. Thanks to Cal, Michael, Dave, Paul and Phil for their help and cooperation in compiling these tests and constructive criticism of [my own validator](#).

People should be aware of the [errata against RFC 3696](#) in particular. Three of the canonical examples are in fact invalid addresses. And the maximum length of an address is 254 or 256 characters, **not** 320.

Share Improve this answer

edited Feb 13, 2022 at 15:00

Follow

community wiki

9 revs, 4 users 74%

[Dominic Sayers](#)

[This validator](#) also seems correct. [...time passes...] Hm, looks like it is just RFC 5322, not 3693 or errata thereto.
– [tchrist](#) Nov 7, 2010 at 20:11

2 Very nice. Here we not only get a nice essay, we get a validation tester as well as a library to download. Nice answer! – [bgmCoder](#) Apr 9, 2013 at 20:49

2 Your validator doesn't support punycode (RFC 3492). name@öäü.at can be a valid address. (it translates to name@xn--4ca9at.at) – [Josef](#) Mar 25, 2015 at 7:28

Hi @Josef. You should try to validate `name@xn--4ca9at.at` since this code is about validation, not interpretation. If you'd like to add a punycode translator then I'm happy to accept a pull request at github.com/dominicsayers/isemail – Dominic Sayers Apr 27, 2015 at 18:19



It's easy in Perl 5.10 or newer:

207



```
/(?(DEFINE)
  (?<address>          (?&mailbox) | (?&group))
  (?<mailbox>          (?&name_addr) | (?&addr_spec))
  (?<name_addr>        (?&display_name)? (?&angle_addr
  (?<angle_addr>       (?&CFWS)? < (?&addr_spec) > (?&
  (?<group>            (?&display_name) : (?:(?&mailbo
                                   (?&CFWS)?))

  (?<display_name>     (?&phrase))
  (?<mailbox_list>     (?&mailbox) (?: , (?&mailbox))*

  (?<addr_spec>        (?&local_part) \@ (?&domain))
  (?<local_part>       (?&dot_atom) | (?&quoted_string
  (?<domain>           (?&dot_atom) | (?&domain_litera
  (?<domain_literal>   (?&CFWS)? \[ (?:(?&FWS)? (?&dc
                                   \] (?&CFWS)?))

  (?<dcontent>         (?&dtext) | (?&quoted_pair))
  (?<dtext>            (?&NO_WS_CTL) | [\x21-\x5a\x5e-

  (?<atext>           (?&ALPHA) | (?&DIGIT) | [!#\${}%&
  (?<atom>             (?&CFWS)? (?&atext)+ (?&CFWS)?
  (?<dot_atom>         (?&CFWS)? (?&dot_atom_text) (?&
  (?<dot_atom_text>    (?&atext)+ (?: \. (?&atext)+)*)

  (?<text>             [\x01-\x09\x0b\x0c\x0e-\x7f])
  (?<quoted_pair>      \\ (?&text))

  (?<qtext>            (?&NO_WS_CTL) | [\x21\x23-\x5b\x
  (?<qcontent>         (?&qtext) | (?&quoted_pair))
  (?<quoted_string>    (?&CFWS)? (?&DQUOTE) (?:(?&FWS)
                                   (?&FWS)? (?&DQUOTE) (?&CFWS)?)
```

```

(?<word>                (?&atom) | (?&quoted_string))
(?<phrase>              (?&word)+)

# Folding white space
(?<FWS>                  (?: (?&WSP)* (?&CRLF))?) (?&WSP)
(?<ctext>                (?&NO_WS_CTL) | [\x21-\x27\x2a-
(?<ccontent>            (?&ctext) | (?&quoted_pair) | (
(?<comment>              \( (?: (?&FWS)? (?&ccontent))*
(?<CFWS>                 (?: (?&FWS)? (?&comment))*
                           (?: (?: (?&FWS)? (?&comment)) |

# No whitespace control
(?<NO_WS_CTL>           [\x01-\x08\x0b\x0c\x0e-\x1f\x7f

(?<ALPHA>               [A-Za-z])
(?<DIGIT>               [0-9])
(?<CRLF>                \x0d \x0a)
(?<DQUOTE>              ")
(?<WSP>                 [\x20\x09])
)

(?&address)/x

```

Share Improve this answer

edited Feb 13, 2022 at 15:42

Follow

community wiki

3 revs, 3 users 69%

Abigail

23 Would love to see this in Python – [tdc](#) Dec 15, 2011 at 16:36

4 I think that only a subset of the `addr spec` part is really relevant to the question. Accepting more than that and forwarding it though some other part of the system that is not ready to accept full RFC5822 addresses is like shooting is your own foot. – [dolmen](#) Dec 17, 2011 at 13:53

4 Great (+1) but technically it's not a regex of course... (which would be impossible since the grammar is not regular).
– Rinke Jan 3, 2013 at 21:41

12 regexes stopped being regular some time ago. It is a valid Perl 'regex' though! – rjh Mar 10, 2014 at 15:00

4 I set up a test for this regex on IDEone: ideone.com/2XFecH
However, it doesn't fair "perfectly." Would anyone care to chime in? Am I missing something? – Mike Jul 30, 2014 at 17:56



I use

205

```
^\w+([-+.']\w+)*@\w+([-.\]\w+)*\.\w+([-.\]\w+)*$
```



Which is the one used in ASP.NET by the
RegularExpressionValidator.



Share Improve this answer edited Oct 27, 2014 at 23:32

Follow


community wiki

3 revs, 3 users 73%

Per Hornshøj-Schierbeck

38 Boo! My (ill-advised) address of `!@mydomain.net` is rejected. – Phrogz Jan 19, 2011 at 21:35

6 According to this page data.iana.org/TLD/tlds-alpha-by-domain.txt there is no domains with just a single character in top level e.g. "something.c", "something.a", here is version that support at least 2 characters: "something.pl",

"something.us": `^\\w+([-.']\\w+)*@\\w+([-.]\\w+)*\\.\\w{2,}([-.]\\w+)*$` – [Tomasz Szulc](#)
Nov 19, 2015 at 12:53 

6 @Wayne Whitty. You have hit upon the primary issue of whether to cater for the vast majority of addresses, or ALL, including ones that nobody would use, except to test email validation. – [user4188092](#) Nov 28, 2015 at 3:13

2 We want explanation about this :) . People come here to see Why it is the way it is. Please consider Regex explanation too! Not everyone is advanced enough to know what you wrote there without explanation. Thanks – [Pratik Joshi](#) Dec 6, 2015 at 10:47

3 this fails on `simon-@hotmail.com` which is in fact valid (a customer of ours had a similar address)` – [Simon_Weaver](#) Feb 10, 2017 at 0:43



149

I don't know about best, but [this one](#) is at least correct, as long as the addresses have their comments stripped and replaced with white space.



Seriously. You should use an already-written library for validating emails. The best way is probably to just send a verification e-mail to that address.



Share Improve this answer

[edited Jul 21, 2021 at 9:50](#)

Follow

community wiki
[3 revs, 2 users 55%](#)
[Peter Mortensen](#)

3 As far as I know, some libraries are wrong, too. I vaguely remember that PHP PEAR had such a bug. – [bortzmeyer](#)
Oct 14, 2008 at 14:34

That page also has a disclaimer at the bottom about a couple of things from the spec. that the regexp does not support. – [Chris Vest](#) Oct 14, 2008 at 14:37

7 That's an RFC 822 spec, not an [RFC 5322](#) spec. – [tchrist](#)
Nov 7, 2010 at 20:12

14 Ultimately, he's right in that the only way to truly *validate* an email address is to send an email to it and await a reply.
– [Blazemonger](#) Oct 26, 2011 at 19:43



Quick answer

143

Use the following regex for input validation:



```
([ -!#- '*+/-9=?A-Z^_~]+(\. [ -!#- '*+/-9=?A-Z^_~]+)*|"  
([ ]!#- [^_~ \t]|(\\ [ \t -~]))+">@[0-9A-Za-z]([0-9A-Za-  
z-]{0,61}[0-9A-Za-z])?(\. [0-9A-Za-z]([0-9A-Za-z-]  
{0,61}[0-9A-Za-z])?)+
```



Addresses matched by this regex:

- have a local part (i.e. the part before the @-sign) that is strictly compliant with RFC 5321/5322,
- have a domain part (i.e. the part after the @-sign) that is a host name with at least two labels, each of which is at most 63 characters long.

The second constraint is a restriction on RFC 5321/5322.

Elaborate answer

Using a regular expression that recognizes email addresses could be useful in various situations: for example to scan for email addresses in a document, to validate user input, or as an integrity constraint on a data repository.

It should however be noted that if you want to find out if the address actually refers to an existing mailbox, there's no substitute for sending a message to the address. If you only want to check if an address is grammatically correct then you could use a regular expression, but note that `""@[]` is a grammatically correct email address that certainly doesn't refer to an existing mailbox.

The syntax of email addresses has been defined in various [RFCs](#), most notably [RFC 822](#) and [RFC 5322](#). RFC 822 should be seen as the "original" standard and RFC 5322 as the latest standard. The syntax defined in RFC 822 is the most lenient and subsequent standards have restricted the syntax further and further, where newer systems or services should recognize obsolete syntax, but never produce it.

In this answer I'll take "email address" to mean `addr-spec` as defined in the RFCs (i.e. `jdoe@example.org`), but

not "John Doe"<jdoe@example.org>, nor some-group:jdoe@example.org,mrx@example.org;).

There's one problem with translating the RFC syntaxes into regexes: the syntaxes are not regular! This is because they allow for optional comments in email addresses that can be infinitely nested, while infinite nesting can't be described by a regular expression. To scan for or validate addresses containing comments you need a parser or more powerful expressions. (Note that languages like Perl have constructs to describe context free grammars in a regex-like way.) In this answer I'll disregard comments and only consider proper regular expressions.

The RFCs define syntaxes for email messages, not for email addresses as such. Addresses may appear in various header fields and this is where they are primarily defined. When they appear in header fields addresses may contain (between lexical tokens) whitespace, comments and even linebreaks. Semantically this has no significance however. By removing this whitespace, etc. from an address you get a semantically equivalent *canonical representation*. Thus, the canonical representation of `first. last (comment) @ [3.5.7.9]` is `first.last@[3.5.7.9]`.

Different syntaxes should be used for different purposes. If you want to scan for email addresses in a (possibly very old) document it may be a good idea to use the syntax as defined in RFC 822. On the other hand, if you

want to validate user input you may want to use the syntax as defined in RFC 5322, probably only accepting canonical representations. You should decide which syntax applies to your specific case.

I use POSIX "extended" regular expressions in this answer, assuming an ASCII compatible character set.

RFC 822

I arrived at the following regular expression. I invite everyone to try and break it. If you find any false positives or false negatives, please post them in a comment and I'll try to fix the expression as soon as possible.

```
( [^] [ ( ) < > @ , ; : \ " . \x00-\x1F\x7F ] + | " ( \n | ( \\ \\r ) *  
 ( [ ^ " \\ \\r \n ] | \\ [ ^ \r ] ) ) * ( \\ \\r ) * " ) ( \. ( [ ^ ] [ ( ) < > @ , ; : \ " .  
\x00-\x1F\x7F ] + | " ( \n | ( \\ \\r ) * ( [ ^ " \\ \\r \n ] | \\ [ ^ \r ] ) ) *  
( \\ \\r ) * " ) ) * @ ( [ ^ ] [ ( ) < > @ , ; : \ " . \x00-\x1F\x7F ] + | \ [ ( \n |  
( \\ \\r ) * ( [ ^ ] [ \\ \\r \n ] | \\ [ ^ \r ] ) ) * ( \\ \\r ) * ] ) ( \. ( [ ^ ] [ ( )  
< > @ , ; : \ " . \x00-\x1F\x7F ] + | \ [ ( \n | ( \\ \\r ) * ( [ ^ ]  
[ \\ \\r \n ] | \\ [ ^ \r ] ) ) * ( \\ \\r ) * ] ) ) *
```

I believe it's fully compliant with RFC 822 including the [errata](#). It only recognizes email addresses in their canonical form. For a regex that recognizes (folding) whitespace see the derivation below.

The derivation shows how I arrived at the expression. I list all the relevant grammar rules from the RFC exactly as they appear, followed by the corresponding regex.

Where an erratum has been published I give a separate expression for the corrected grammar rule (marked "erratum") and use the updated version as a subexpression in subsequent regular expressions.

As stated in paragraph 3.1.4. of RFC 822 optional linear white space may be inserted between lexical tokens.

Where applicable I've expanded the expressions to accommodate this rule and marked the result with "opt-lwsp".

```
CHAR          = <any ASCII character>
               =~ .

CTL           = <any ASCII control character and DEL>
               =~ [\x00-\x1F\x7F]

CR            = <ASCII CR, carriage return>
               =~ \r

LF            = <ASCII LF, linefeed>
               =~ \n

SPACE         = <ASCII SP, space>
               =~

HTAB          = <ASCII HT, horizontal-tab>
               =~ \t

<">          = <ASCII quote mark>
               =~ "

CRLF          = CR LF
               =~ \r\n

LWSP-char     = SPACE / HTAB
               =~ [ \t]

linear-white-space = 1*([CRLF] LWSP-char)
```

=~ ((\r\n)?[\t])+

specials = "(" / ")" / "<" / ">" / "@" / "," / ";"
"." / "[" / "]"
=~ [] [() < > @ , ; : \ \ " .]

quoted-pair = "\" CHAR
=~ \\..

qtext = <any CHAR excepting "<>", "\" & CR, and
space>
=~ [^"\\r]|((\r\n)?[\t])+

dtext = <any CHAR excluding "[", "]", "\" & CR,
white-space>
=~ [^][\\r]|((\r\n)?[\t])+

quoted-string = "<" *(qtext|quoted-pair) "<"
=~ "([^\\"r]|((\r\n)?[\t]))|\\.)*"
(erratum) =~ "(\n|(\\"r)*([^\\"r\n]|\\[^\r]|(\r\

domain-literal = "[" *(dtext|quoted-pair) "]"
=~ \[([^\\"r]|((\r\n)?[\t]))|\\.)*]
(erratum) =~ \[(\n|(\\"r)*([^\\"r\n]|\\[^\r]|(\r\

atom = 1*<any CHAR except specials, SPACE and
=~ [^][() < > @ , ; : \ \ " . \x00-\x1F\x7F]+

word = atom / quoted-string
=~ [^][() < > @ , ; : \ \ " . \x00-\x1F\x7F]+ | "(\n | (
[^\r]|(\r\n)?[\t]))*(\\"r)*"

domain-ref = atom

sub-domain = domain-ref / domain-literal
=~ [^][() < > @ , ; : \ \ " . \x00-\x1F\x7F]+ | \[(\n | (
[^\r]|(\r\n)?[\t]))*(\\"r)*]

local-part = word *("." word)
=~ ([^][() < > @ , ; : \ \ " . \x00-\x1F\x7F]+ | "(\n | (
[^\r]|(\r\n)?[\t]))*(\\"r)*") \. ([^][() < > @ , ; : \ \ " . \x0
([^\\"r\n]|\\[^\r]|(\r\n)?[\t]))*(\\"r)*") *
(opt-lwsp) =~ ([^][() < > @ , ; : \ \ " . \x00-\x1F\x7F]+ | "(\n | (
[^\r]|(\r\n)?[\t]))*(\\"r)*") (((\r \n) ? [\ t]) * \. ((\r \n

```

<>@,;:\\". \x00-\x1F\x7F]+|"(\n|(\\"r)*([^\\"r\n]|\\"(\\"r)*"))*

domain      =  sub-domain *("." sub-domain)
              =~ ([^][()<>@,;:\\". \x00-\x1F\x7F]+|\\"(\n|
[^\r]|(\r\n)?[ \t]))*(\\"r)*)(\.(^[^][()<>@,;:\\". \x0
(\\"r)*([^\\"r\n]|\\"[^\r]|(\r\n)?[ \t]))*(\\"r)*)))*
(opt-lwsp)  =~ ([^][()<>@,;:\\". \x00-\x1F\x7F]+|\\"(\n|
[^\r]|(\r\n)?[ \t]))*(\\"r)*)(((\r\n)?[ \t])*\.((\r\n
<>@,;:\\". \x00-\x1F\x7F]+|\\"(\n|(\\"r)*([^\\"r\n]|\\"
(\\"r)*)))*)

addr-spec   =  local-part "@" domain
              =~ ([^][()<>@,;:\\". \x00-\x1F\x7F]+|"(\n|
[^\r]|(\r\n)?[ \t]))*(\\"r)*)(\.(^[^][()<>@,;:\\". \x0
([^\\"r\n]|\\"[^\r]|(\r\n)?[ \t]))*(\\"r)*))*@([^[^][()
\x1F\x7F]+|\\"(\n|(\\"r)*([^\\"r\n]|\\"[^\r]|(\r\n)?[
<>@,;:\\". \x00-\x1F\x7F]+|\\"(\n|(\\"r)*([^\\"r\n]|\\"
(\\"r)*)))*)
(opt-lwsp)  =~ ([^][()<>@,;:\\". \x00-\x1F\x7F]+|"(\n|
[^\r]|(\r\n)?[ \t]))*(\\"r)*)(((\r\n)?[ \t])*\.((\r\n
<>@,;:\\". \x00-\x1F\x7F]+|"(\n|(\\"r)*([^\\"r\n]|\\"[
(\\"r)*))((\r\n)?[ \t])*)*@((\r\n)?[ \t])*([^[^][()<>@,;
[(\n|(\\"r)*([^\\"r\n]|\\"[^\r]|(\r\n)?[ \t]))*(\\"r)
((\r\n)?[ \t])*([^[^][()<>@,;:\\". \x00-\x1F\x7F]+|\\"(\n
[^\r]|(\r\n)?[ \t]))*(\\"r)*)))*)
(canonical) =~ ([^][()<>@,;:\\". \x00-\x1F\x7F]+|"(\n|
[^\r]))*(\\"r)*)(\.(^[^][()<>@,;:\\". \x00-\x1F\x7F]+|
([^\\"r\n]|\\"[^\r]))*(\\"r)*))*@([^[^][()<>@,;:\\". \x
(\\"r)*([^\\"r\n]|\\"[^\r]))*(\\"r)*)(\.(^[^][()<>@,;
[(\n|(\\"r)*([^\\"r\n]|\\"[^\r]))*(\\"r)*)))*)

```

RFC 5322

I arrived at the following regular expression. I invite everyone to try and break it. If you find any false positives or false negatives, please post them in a comment and I'll try to fix the expression as soon as possible.

```
([-!#-'*+/-9=?A-Z^_~]+(\.[-!#-'*+/-9=?A-Z^_~]+)*|"  
([ ]!#-[^_~ \t]|(\\\[ \t -~]))+">@([-!#-'*+/-9=?A-Z^_~]+  
(\.[-!#-'*+/-9=?A-Z^_~]+)*|\\\[ \t -Z^_~]*))
```

I believe it's fully compliant with RFC 5322 including the [errata](#). It only recognizes email addresses in their canonical form. For a regex that recognizes (folding) whitespace see the derivation below.

The derivation shows how I arrived at the expression. I list all the relevant grammar rules from the RFC exactly as they appear, followed by the corresponding regex. For rules that include semantically irrelevant (folding) whitespace, I give a separate regex marked "(normalized)" that doesn't accept this whitespace.

I ignored all the "obs-" rules from the RFC. This means that the regexes only match email addresses that are strictly RFC 5322 compliant. If you have to match "old" addresses (as the looser grammar including the "obs-" rules does), you can use one of the RFC 822 regexes from the previous paragraph.

VCHAR	=	%x21-7E
	=~	[!-~]
ALPHA	=	%x41-5A / %x61-7A
	=~	[A-Za-z]
DIGIT	=	%x30-39
	=~	[0-9]
HTAB	=	%x09
	=~	\t


```

CR          =  %x0D
           =~  \r

LF          =  %x0A
           =~  \n

SP          =  %x20
           =~

DQUOTE      =  %x22
           =~  "

CRLF        =  CR LF
           =~  \r\n

WSP         =  SP / HTAB
           =~  [\t ]

quoted-pair =  "\" (VCHAR / WSP)
           =~  \"[\t -~]

FWS         =  ([*WSP CRLF] 1*WSP)
           =~  ([\t ]*\r\n)?[\t ]+

ccontent    =  %d33-39 / %d42-91 / %d93-126
           =~  [ ]!- '*-[^\t -~]

("comment" is left out in the regex)
cccontent   =  ccontent / quoted-pair / comment
           =~  [ ]!- '*-[^\t -~]|(\"[\t -~])

(not regular)
comment     =  "(" *([FWS] ccontent) [FWS] ")"

(is equivalent to FWS when leaving out comments)
CFWS        =  (1*([FWS] comment) [FWS]) / FWS
           =~  ([\t ]*\r\n)?[\t ]+

atext       =  ALPHA / DIGIT / "!" / "#" / "$" /
"+ " / "-" / "/" / "=" / "?" / "^" / "_" / "`" / "{" /
           =~  [-!#- '*+/-9=?A-Z^_~]

dot-atom-text =  1*atext *("." 1*atext)
           =~  [-!#- '*+/-9=?A-Z^_~]+(\.[-!#- '*+/-

```

```

dot-atom          =  [CFWS] dot-atom-text [CFWS]
                   =~  (([\t ]*\r\n)?[\t ]+)?[-!#-'*+/-9=A-Z^~]+)*(([\t ]*\r\n)?[\t ]+)?
(normalized)      =~  [-!#-'*+/-9=?A-Z^~]+(\.[-!#-'*+/-

qtext             =  %d33 / %d35-91 / %d93-126
                   =~  []!#-[^~]

qcontent          =  qtext / quoted-pair
                   =~  []!#-[^~]|(\[\t -~])

(erratum)
quoted-string     =  [CFWS] DQUOTE ((1*([FWS] qcontent)
[CFWS]
                   =~  (([\t ]*\r\n)?[\t ]+)?"(((([\t ]*\
(\[\t -~]))))+(([\t ]*\r\n)?[\t ]+)?|(([\t ]*\r\n)?[\t
]+)?
(normalized)      =~  "([]!#-[^~ \t]|(\[\t -~])))+"

dtext             =  %d33-90 / %d94-126
                   =~  [!-Z^~]

domain-literal    =  [CFWS] "[" *([FWS] dtext) [FWS] "]"
                   =~  (([\t ]*\r\n)?[\t ]+)?\[(((([\t ]*\
([\t ]*\r\n)?[\t ]+)?)(([\t ]*\r\n)?[\t ]+)?
(normalized)      =~  \[[\t -Z^~]*]

local-part        =  dot-atom / quoted-string
                   =~  (([\t ]*\r\n)?[\t ]+)?[-!#-'*+/-9=
A-Z^~]+)*(([\t ]*\r\n)?[\t ]+)?|(([\t ]*\r\n)?[\t ]+
)+)?([]!#-[^~]|(\[\t -~])))+(([\t ]*\r\n)?[\t ]+)?|
([\t ]*\r\n)?[\t ]+)?
(normalized)      =~  [-!#-'*+/-9=?A-Z^~]+(\.[-!#-'*+/-
\t]|(\[\t -~])))+"

domain            =  dot-atom / domain-literal
                   =~  (([\t ]*\r\n)?[\t ]+)?[-!#-'*+/-9=
A-Z^~]+)*(([\t ]*\r\n)?[\t ]+)?|(([\t ]*\r\n)?[\t ]+
)+)?[!-Z^~]*(([\t ]*\r\n)?[\t ]+)?|(([\t ]*\r\n)?[\t
(normalized)      =~  [-!#-'*+/-9=?A-Z^~]+(\.[-!#-'*+/-
~]*)

addr-spec         =  local-part "@" domain

```

```

= ~ ( ( ( [ \t ] * \r \n ) ? [ \t ] + ) ? [ - ! # - ' * + / - 9
' * + / - 9 = ? A - Z ^ - ~ ] + ) * ( ( [ \t ] * \r \n ) ? [ \t ] + ) ? | ( ( [ \t ] * \r \n )
] * \r \n ) ? [ \t ] + ) ? ( [ ] ! # - [ ^ - ~ ] | ( \ [ \t - ~ ] ) ) + ( ( [ \t ] * \r \n
[ \t ] + ) ? ) " ( ( [ \t ] * \r \n ) ? [ \t ] + ) ? @ ( ( ( [ \t ] * \r \n ) ? [ \t ]
( \ . [ - ! # - ' * + / - 9 = ? A - Z ^ - ~ ] + ) * ( ( [ \t ] * \r \n ) ? [ \t ] + ) ? | ( ( [ \t
] * \r \n ) ? [ \t ] + ) ? [ ! - Z ^ - ~ ] ) * ( ( [ \t ] * \r \n ) ? [ \t ] + ) ? ) ( ( [ \t
(normalized)      = ~ ( [ - ! # - ' * + / - 9 = ? A - Z ^ - ~ ] + ( \ . [ - ! # - ' * + /
~ \t ] | ( \ [ \t - ~ ] ) ) + ) " @ ( [ - ! # - ' * + / - 9 = ? A - Z ^ - ~ ] + ( \ . [ - ! # - ' *
Z ^ - ~ ] * ] )

```

Note that some sources (notably [W3C](#)) claim that RFC 5322 is too strict on the local part (i.e. the part before the @-sign). This is because `".."`, `"a..b"` and `"a."` are *not* valid dot-atoms, while they may be used as mailbox names. The RFC, however, *does* allow for local parts like these, except that they have to be quoted. So instead of `a..b@example.net` you should write `"a..b"@example.net`, which is semantically equivalent.

Further restrictions

SMTP (as defined in [RFC 5321](#)) further restricts the set of valid email addresses (or actually: mailbox names). It seems reasonable to impose this stricter grammar, so that the matched email address can actually be used to send an email.

RFC 5321 basically leaves alone the "local" part (i.e. the part before the @-sign), but is stricter on the domain part (i.e. the part after the @-sign). It allows only host names in place of dot-atoms and address literals in place of domain literals.

The grammar presented in RFC 5321 is too lenient when it comes to both host names and IP addresses. I took the liberty of "correcting" the rules in question, using [this draft](#) and [RFC 1034](#) as guidelines. Here's the resulting regex.

```
([ -!#- '*+/-9=?A-Z^_~]+(\.[ -!#- '*+/-9=?A-Z^_~]+)*|"
([ ]!#- [^_~ \t]|(\\[ \t -~]))+)"@([0-9A-Za-z]([0-9A-Za-
z-]{0,61}[0-9A-Za-z])?(\.[0-9A-Za-z]([0-9A-Za-z-
]{0,61}[0-9A-Za-z])?)*|\[((25[0-5]|2[0-4][0-9]|1[0-9]
{2}|[1-9]?[0-9])(\.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-
9]?[0-9])){3}|IPv6:((((0|[1-9A-Fa-f][0-9A-Fa-f]
{0,3})):){6}|::((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3})):){5}|
[0-9A-Fa-f]{0,4}::((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3})):
{4}|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3})):)?(0|[1-9A-Fa-
f][0-9A-Fa-f]{0,3}))?:::((0|[1-9A-Fa-f][0-9A-Fa-f]
{0,3})):){3}|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3})):){0,2}
(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?:::((0|[1-9A-Fa-f][0-
9A-Fa-f]{0,3})):){2}|(((0|[1-9A-Fa-f][0-9A-Fa-f]
{0,3})):){0,3}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::(0|
[1-9A-Fa-f][0-9A-Fa-f]{0,3}):|(((0|[1-9A-Fa-f][0-9A-
Fa-f]{0,3})):){0,4}(0|[1-9A-Fa-f][0-9A-Fa-f]
{0,3}))?:::((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):(0|[1-9A-
Fa-f][0-9A-Fa-f]{0,3})|(25[0-5]|2[0-4][0-9]|1[0-9]
{2}|[1-9]?[0-9])(\.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-
9]?[0-9])){3})|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3})):)
{0,5}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::(0|[1-9A-Fa-
f][0-9A-Fa-f]{0,3})|(((0|[1-9A-Fa-f][0-9A-Fa-f]
{0,3})):){0,6}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::)|
(?!IPv6:)[0-9A-Za-z-]*[0-9A-Za-z]:[!-Z^_~]+))
```

Note that depending on the use case you may not want to allow for a "General-address-literal" in your regex. Also note that I used a negative lookahead `(?!IPv6:)` in the final regex to prevent the "General-address-literal" part to match malformed IPv6 addresses. Some regex processors don't support negative lookahead. Remove the substring `|(?!IPv6:)[0-9A-Za-z-]*[0-9A-Za-z]:[!-Z^--~]+` from the regex if you want to take the whole "General-address-literal" part out.

Here's the derivation:

```
Let-dig      =  ALPHA / DIGIT
              =~ [0-9A-Za-z]
```

```
Ldh-str      =  *( ALPHA / DIGIT / "-" ) Let-dig
              =~ [0-9A-Za-z-]*[0-9A-Za-z]
```

(regex is updated to make sure sub-domains are max. 63
1034 section 3.5)

```
sub-domain   =  Let-dig [Ldh-str]
              =~ [0-9A-Za-z]([0-9A-Za-z-]{0,61}[0-9
```

```
Domain       =  sub-domain *("." sub-domain)
              =~ [0-9A-Za-z]([0-9A-Za-z-]{0,61}[0-9
([0-9A-Za-z-]{0,61}[0-9A-Za-z])?)*
```

```
Snum         =  1*3DIGIT
              =~ [0-9]{1,3}
```

(suggested replacement for "Snum")

```
ip4-octet    =  DIGIT / %x31-39 DIGIT / "1" 2DIGIT
"25" %x30-35
              =~ 25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9
```

```
IPv4-address-literal =  Snum 3("." Snum)
                      =~ [0-9]{1,3}(\.[0-9]{1,3}){3
```

(suggested replacement for "IPv4-address-literal")

```

ip4-address      =  ip4-octet 3("." ip4-octet)
                  =~ (25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-4][0-9]|1[0-9]{2}|[1-9]?[0-9])){3}

(suggested replacement for "IPv6-hex")
ip6-h16          =  "0" / ( (%x49-57 / %x65-70 /%x97-1
/%x97-102) )
                  =~ 0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}

(not from RFC)
ls32             =  ip6-h16 ":" ip6-h16 / ip4-address
                  =~ (0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):(0
{0,3})|(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])(\. (
9){2}|[1-9]?[0-9])){3}

(suggested replacement of "IPv6-addr")
ip6-address      =
                  /
                  / [ ip6-h16 ] ":"
                  / [ *1(ip6-h16 ":") ip6-h16 ] ":"
                  / [ *2(ip6-h16 ":") ip6-h16 ] ":"
                  / [ *3(ip6-h16 ":") ip6-h16 ] ":"
                  / [ *4(ip6-h16 ":") ip6-h16 ] ":"
                  / [ *5(ip6-h16 ":") ip6-h16 ] ":"
                  / [ *6(ip6-h16 ":") ip6-h16 ] ":"
                  =~ (((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):
9A-Fa-f]{0,3}):){5}|[0-9A-Fa-f]{0,4}::((0|[1-9A-Fa-f][
(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):)?(0|[1-9A-Fa-f][0-9
9A-Fa-f][0-9A-Fa-f]{0,3}):){3}|(((0|[1-9A-Fa-f][0-9A-F
9A-Fa-f][0-9A-Fa-f]{0,3}))?:)((0|[1-9A-Fa-f][0-9A-Fa-f
Fa-f][0-9A-Fa-f]{0,3}):){0,3}(0|[1-9A-Fa-f][0-9A-Fa-f]
[0-9A-Fa-f]{0,3}):|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):)
Fa-f){0,3}))?:)((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):)(0|[1
{0,3})|(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])(\. (
9){2}|[1-9]?[0-9])){3}|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,
[0-9A-Fa-f]{0,3}))?:)((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3})|(
f){0,3}):){0,6}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::

IPv6-address-literal =  "IPv6:" ip6-address
                      =~ IPv6:(((0|[1-9A-Fa-f][0-9
[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){5}|[0-9A-Fa-f]{0,4}::((
{0,3}):){4}|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):)?(0|[1-
{0,3}))?:)((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){3}|(((0|[
{0,3}):){0,2}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?:)((0|[1

```

```
{0,3}):){2}|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){0,3}(0{0,3}))?::(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):|(((0|[1-9A-{0,4}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::)((0|[1-9A-Fa-[1-9A-Fa-f][0-9A-Fa-f]{0,3})|(25[0-5]|2[0-4][0-9]|1[0-(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])){3})|(((0|{0,3}):){0,5}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::(0|[1-{0,3})|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){0,6}(0|[1-9{0,3}))?::))
```

```
Standardized-tag          =    Ldh-str
                           =~   [0-9A-Za-z-]*[0-9A-Za-z]
```

```
dcontent                  =    %d33-90 / %d94-126
                           =~   [!-Z^--~]
```

```
General-address-literal =    Standardized-tag ":" 1*dco
                           =~   [0-9A-Za-z-]*[0-9A-Za-z]:[
```

```
address-literal =    "[" ( IPv4-address-literal / IPv6-
General-address-literal ) "]"
                     =~   \[((25[0-5]|2[0-4][0-9]|1[0-9]{2}|
5|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])){3}|IPv6:((((0|[
{0,3}):){6}|::((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){5}|[0
9A-Fa-f][0-9A-Fa-f]{0,3}):){4}|(((0|[1-9A-Fa-f][0-9A-F
f][0-9A-Fa-f]{0,3}))?::((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}
[0-9A-Fa-f]{0,3}):){0,2}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}
9A-Fa-f]{0,3}):){2}|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):
9A-Fa-f]{0,3}))?::(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):|(((
{0,3}):){0,4}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::)((0|[
{0,3}):)(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3})|(25[0-5]|2[0-4]
9))(\.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])){3})
Fa-f]{0,3}):){0,5}(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::(
{0,3})|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){0,6}(0|[1-9
{0,3}))?::)|(?!IPv6:)[0-9A-Za-z-]*[0-9A-Za-z]:[!-Z^--~]
```

```
Mailbox                  =    Local-part "@" ( Domain / address-
                             =~   ([-!#- '*+/-9=?A-Z^--~]+(\.[-!#- '*+/-
~ \t]|(\\[ \t -~])))+)"@([0-9A-Za-z]([0-9A-Za-z-]{0,61}[
z]([0-9A-Za-z-]{0,61}[0-9A-Za-z]))?)*|\[((25[0-5]|2[0-4]
[0-9])\.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])){
[0-9A-Fa-f]{0,3}):){6}|::((0|[1-9A-Fa-f][0-9A-Fa-f]{0,
{0,4}::((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){4}|(((0|[1-9
{0,3}):)?(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::((0|[1-9A-
{3}|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){0,2}(0|[1-9A-F
```

```
( (0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){2}|(((0|[1-9A-Fa-f]
(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}))?::(0|[1-9A-Fa-f][0-9A
Fa-f][0-9A-Fa-f]{0,3}):){0,4}(0|[1-9A-Fa-f][0-9A-Fa-f]
f)[0-9A-Fa-f]{0,3}):(0|[1-9A-Fa-f][0-9A-Fa-f]{0,3})|(2
{2}|[1-9]?[0-9])(\.(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9
Fa-f][0-9A-Fa-f]{0,3}):){0,5}(0|[1-9A-Fa-f][0-9A-Fa-f]
[0-9A-Fa-f]{0,3})|(((0|[1-9A-Fa-f][0-9A-Fa-f]{0,3}):){
Fa-f]{0,3}))?::)|(?!IPv6:)[0-9A-Za-z-]*[0-9A-Za-z]:[!-
```

User input validation

A common use case is user input validation, for example on an html form. In that case it's usually reasonable to preclude address-literals and to require at least two labels in the hostname. Taking the improved RFC 5321 regex from the previous section as a basis, the resulting expression would be:

```
([-!#-'*+/-9=?A-Z^_~]+(\.[-!#-'*+/-9=?A-Z^_~]+)*|"
([!#-^_~\t]|(\\[ \t -~]))+)"@[0-9A-Za-z]([0-9A-Za-
z-]{0,61}[0-9A-Za-z])?(\.[0-9A-Za-z]([0-9A-Za-z-]
{0,61}[0-9A-Za-z])?)+
```

I do not recommend restricting the local part further, e.g. by precluding quoted strings, since we don't know what kind of mailbox names some hosts allow (like

"a..b"@example.net or even "a b"@example.net).

I also do not recommend explicitly validating against a list of literal top-level domains or even imposing length-constraints (remember how ".museum" invalidated [a-z]{2,4}), but if you must:


```
([-!#-'*+/-9=?A-Z^_~]+(\.[-!#-'*+/-9=?A-Z^_~]+)*|"  
([!#-[^-~ \t]|(\\[ \t -~]))+">@([0-9A-Za-z]([0-9A-Za-  
z-]{0,61}[0-9A-Za-z])?\.)* (net|org|com|info| etc... )
```

Make sure to keep your regex up-to-date if you decide to go down the path of explicit top-level domain validation.

Further considerations

When only accepting host names in the domain part (after the @-sign), the regexes above accept only labels with at most 63 characters, as they should. However, they don't enforce the fact that the entire host name must be at most 253 characters long (including the dots). Although this constraint is strictly speaking still regular, it's not feasible to make a regex that incorporates this rule.

Another consideration, especially when using the regexes for input validation, is feedback to the user. If a user enters an incorrect address, it would be nice to give a little more feedback than a simple "syntactically incorrect address". With "vanilla" regexes this is not possible.

These two considerations could be addressed by parsing the address. The extra length constraint on host names could in some cases also be addressed by using an extra regex that checks it, and matching the address against both expressions.

None of the regexes in this answer are optimized for performance. If performance is an issue, you should see

if (and how) the regex of your choice can be optimized.

Share Improve this answer

edited Feb 13, 2022 at 16:43

Follow

community wiki

43 revs, 3 users 94%

Rinke

-
- 3 RFC [6532](#) updates [5322](#) to allow and include full, clean UTF-8. Additional details [here](#). – user2350426 Jun 26, 2015 at 17:30
-


According to wikipedia seems that the local part, when dotted, has a limitation of 64 chars per part, and also the RFC 5322 refers to the dotted local part to be interpreted with the restrictions of the domains. For example

arbitrary-long-email-address-should-be-invalid-arbitrary-long-email-address-should-be-invalid.and-the-second-group-also-should-not-be-so-long-and-the-second-group-also-should-not-be-so-long@example.com should not validate. I suggest changing the "+" signs in the first group (name before the optional dot) and in the second group (name after the following dots) to `{1, 64}` – Xavi Montero May 22, 2017 at

0:35 

As the comments are limited in size, here is the resulting regex I plan to use, which is the one at the beginning of this answer, plus limiting the size in the local part, plus adding a back-slash prior to the "/" symbol as required by PHP and also in regex101.com: In PHP I use: `$emailRegex =`

```
'/^([!#-\'+\/-9=?A-Z^~]{1,64}(\.[!#-\'+\/-9=?A-Z^~]{1,64})*"([!#-[\^~\t]|(\\[\t-~]))+)"@[0-9A-Za-z]([0-9A-Za-z-]{0,61}[0-9A-Za-
```

z]))?(\.[0-9A-Za-z]([0-9A-Za-z-]{0,61}[0-9A-Za-z]))?)+\$/' ; – [Xavi Montero](#) May 22, 2017 at 0:39 

CAUTION: For some reason, StackOverflow adds hidden characters when copying from the rendered markdown. Copy it into the regex101.com and you'll see black dots there. You have to remove them and correct the string... Maybe if integrated in the answer, there they are correctly copiable. Sorry for the inconvenience. I don't want to add a new answer as this one is the proper one. Also I don't want to directly edit unless the community thinks this should be integrated into it. – [Xavi Montero](#) May 22, 2017 at 0:48

@XaviMontero Thaks for contributing Xavi! Do you have a reference to the RFC stating the 64 character limit on local part labels? If so, I would gladly adjust the answer. – [Rinke](#) May 22, 2017 at 11:21



119



The email addresses I want to validate are going to be used by an ASP.NET web application using the `System.Net.Mail` namespace to send emails to a list of people.

So, rather than using some very complex regular expression, I just try to create a `MailAddress` instance from the address. The `MailAddress` constructor will throw an exception if the address is not formed properly. This way, I know I can at least get the email out of the door. Of course this is server-side validation, but at a minimum you need that anyway.

```
protected void emailValidator_ServerValidate(object so
ServerValidateEventArgs args)
{
```

```
try
{
    var a = new MailAddress(txtEmail.Text);
}
catch (Exception ex)
{
    args.IsValid = false;
    emailValidator.ErrorMessage = "email: " + ex.M
}
}
```

Share Improve this answer

edited Jun 16, 2020 at 14:57

Follow

community wiki

2 revs, 2 users 65%

davcar

-
- 4 A good point. Even if this server validation rejects some valid address then it is not a problem since you will not be able to send to this address using this particular server technology anyway. Or you can try doing the same things using any third party emailing library you use instead of the default tools.
– [User](#) Jun 16, 2009 at 10:59

-
- 1 I really like how this leverages .Net framework code - no sense in reinventing the wheel. This is excellent. Simple, clean, and assures you can actually send the email. Great work. – [Cory House](#) Aug 15, 2010 at 19:43

... yes and for the those interested in how it validates have a look at the code in Reflector - there's quite a bit of it - and it ain't a regular expression! – [Tom Carter](#) Sep 17, 2010 at 8:07

-
- 2 Just a note: the MailAddress class doesn't match RFC5322, if you just want to use it for validation (and not sending as well, in which case it's a moot point as mentioned above).

See: stackoverflow.com/questions/6023589/... – [porges](#) May 31, 2011 at 5:06

Just a minor issue: if you want to make your server side validator code more reusable (either in this case or generally), I suggest to use `args.Value` instead of referencing the field like `txtEmail.Text` hard-coded. The latter one will bound your validator to the single control instance, that may be OK, as long you have a single e-mail field, but not recommended otherwise. – [pholpar](#) Aug 21, 2019 at 11:23



78

There are plenty examples of this out on the Internet (and I think even one that fully validates the RFC - but it's tens/hundreds of lines long if memory serves).



People tend to get carried away validating this sort of thing. Why not just check it has an @ and at least one `.` and meets some simple minimum length? It's trivial to enter a fake email and still match any valid regex anyway. I would guess that false positives are better than false negatives.



Share Improve this answer

edited Jul 21, 2021 at 9:40

Follow

community wiki
2 revs, 2 users 75%
[Peter Mortensen](#)

- 1 Yes, but **which** RFC? :) This [RFC-5322-validator] (stackoverflow.com/questions/201323/...) is only around forty lines long. – [tchrist](#) Nov 7, 2010 at 20:20
- 17 A . is not required. A TLD can have email addresses, or there could be an IPv6 address – [Sijmen Mulder](#) Feb 15, 2011 at 12:58
- 3 RFCs are not the end of the story: ICANN does not allow 'dotless' domains any more: icann.org/news/announcement-2013-08-30-en – [Synchro](#) Sep 9, 2014 at 16:28



This regex is from Perl's [Email::Valid](#) library. I believe it to be the most accurate, and it matches all of [RFC 822](#).

65

And, it is based on the regular expression in the O'Reilly book:



Regular expression built using Jeffrey Friedl's example in *Mastering Regular Expressions* (<http://www.ora.com/catalog/regexp/>).

```
$RFC822PAT = <<'EOF';
[\040\t]*(?:\[^\x80-\xff\n\015()]*(?:\[^\x80-\xff\n\015()]*(?:\[^\x80-\xff\n\015()]*)*\)[\040\t]*)*(?:([^\040]>@,;:".\\[\]
ff]+(?:\[^\040]>@,;:".\\[\]\000-\037\x80-\xff))|"^[
"]*(?:\[^\x80-\xff][^\x80-\xff\n\015"]*)*)[\040\t]
\xff\n\015()*(?:([^\x80-\xff]|([^\x80-\xff\n\01
-\xff][^\x80-\xff\n\015()]*))*)[^\x80-\xff\n\015(
)*(?:\. [\040\t]*(?:\[^\x80-\xff\n\015()*(?:([^\x80-\xff\n\015()*(?:\[^\x80-\xff][^\x80-\xff\n\0
x80-\xff\n\015()]*)*\)[\040\t]*)*(?:([^\040]>@,;:".\\
0-\xff)+(?:\[^\040]>@,;:".\\[\]\000-\037\x80-\xff))|
\015"]*(?:\[^\x80-\xff][^\x80-\xff\n\015"]*)*)[\04
```

80-\xff\n\015()]*(:(:\[\^x80-\xff]|\([\^x80-\xff\

\x80-\xff][\^x80-\xff\n\015()]*\))[\^x80-\xff\n\

\t]*)*)*@\040\t]*(?:\([\^x80-\xff\n\015()]*(:(:\

\^x80-\xff\n\015()]*(:\[\^x80-\xff][\^x80-\xff\n

\x80-\xff\n\015()]*\))[\040\t]*)*(?:\([\^(\040)<>@,;:".

x80-\xff]+(?:\([\^(\040)<>@,;:".\\[\]\000-\037\x80-\xff]

\xff\n\015\[\])|\[\^x80-\xff\))*\))[\040\t]*(?:\([\^x80-

\xff]|\([\^x80-\xff\n\015()]*(:\[\^x80-\xff\n\015()

\x80-\xff\n\015()]*\))[\^x80-\xff\n\015()]*\))[\040

\t]*(?:\([\^x80-\xff\n\015()]*(:(:\[\^x80-\xff]|\

n\015()]*(:\[\^x80-\xff][\^x80-\xff\n\015()]*\))\

015()]*\))[\040\t]*)*(?:\([\^(\040)<>@,;:".\\[\]\000-\0

[\^(\040)<>@,;:".\\[\]\000-\037\x80-\xff])|\[(?:\[\^x80-

\xff]|\[\^x80-\xff\))*\))[\040\t]*(?:\([\^x80-\xff\n\01

x80-\xff]|\([\^x80-\xff\n\015()]*(:\[\^x80-\xff][\^

5()]*\))[\^x80-\xff\n\015()]*\))[\040\t]*)*)*|(:

\\[\]\000-\037\x80-\xff]+(?:\([\^(\040)<>@,;:".\\[\]\000-

)|"[^x80-\xff\n\015"]*(?:\[\^x80-\xff][^x80-\xf

()<>@,;:".\\[\]\x80-\xff\000-\010\012-\037]*(?:(:\([\

15()]*(:(:\[\^x80-\xff]|\([\^x80-\xff\n\015()]*(:

\^x80-\xff\n\015()]*\))[\^x80-\xff\n\015()]*\))

n\015"]*(?:\[\^x80-\xff][^x80-\xff\n\015"]*)*)[\^

x80-\xff\000-\010\012-\037]*)*<[\040\t]*(?:\([\^x80-

:(:\[\^x80-\xff]|\([\^x80-\xff\n\015()]*(:\[\^x80-

\xff\n\015()]*\))[\^x80-\xff\n\015()]*\))[\040\t]

(?:\([\^x80-\xff\n\015()]*(:(:\[\^x80-\xff]|\([\^

()*(?:\[\^x80-\xff][^x80-\xff\n\015()]*\))[\^x80-

\xff\n\015()]*\))[\040\t]*)*(?:\([\^(\040)<>@,;:".\\[\]\000-

\037\x80-\xff])|\[(?:\[\^x80-\xf

[\^x80-\xff\))*\))[\040\t]*(?:\([\^x80-\xff\n\015()

\xff]|\([\^x80-\xff\n\015()](:\[\^x80-\xff][^x80-

\x80-\xff\n\015()]*\))[\^x80-\xff\n\015()]*\))[\040\t]

)(?:\.[\040-\xff\n\015()]*(:(:\[\^x80-\xff]|\([\^x80-\xff\n\

80-\xff][^x80-\xff\n\015()]*\))[\^x80-\xff\n\01

]*)*(?:\([\^(\040)<>@,;:".\\[\]\000-\037\x80-\xff]+(?:\[\^

\[\]\000-\037\x80-\xff])|\[(?:\[\^x80-\xff\n\015\[\])

\))[\040\t](?:\([\^x80-\xff\n\015()]*(:(:\[\^x80-

80-\xff\n\015()]*(:\[\^x80-\xff][^x80-\xff\n\015(-

\xff\n\015()]*\))[\040\t]*)*)*(?:,[\040\t]*(?:\([\^x80-

\xff]|\([\^x80-\xff\n\015()]*(:\[\^x80-\xff\n\015()

\x80-\xff\n\015()]*\))[\^x80-\xff\n\015()]*\))[\0

]*(?:\([\^x80-\xff\n\015()]*(:(:\[\^x80-\xff]|\([\

15()]*(:\[\^x80-\xff][^x80-\xff\n\015()]*\))[\^

()*)\))[\040\t]*)*(?:\([\^(\040)<>@,;:".\\[\]\000-\037\

\040)<>@, ;: ".\\[\\]\000-\037\x80-\xff)]|\\[(?:[^\x80-
\\[\x80-\xff])*\\)[\040\t]*(?:\\([^\x80-\xff\n\015()
-\\xff]|\\([^\x80-\xff\n\015()]*(?:\\[\x80-\xff][^\x80-
]*)*\\))^[^\x80-\xff\n\015()]*)*\\)[\040\t]*(?:\\. [\04
80-\xff\n\015()]*(?:?:\\[\x80-\xff]|\\([^\x80-\xff\
\x80-\xff][^\x80-\xff\n\015()]*)*\\))^[^\x80-\xff\n\
\t]*)*(?:^[^(\040)<>@, ;: ".\\[\\]\000-\037\x80-\xff]+(?:!
\\[\\]\000-\037\x80-\xff)]|\\[(?:[^\x80-\xff\n\015\\[\
)])*\\)[\040\t]*(?:\\([^\x80-\xff\n\015()]*(?:?:\\[\x80-
\x80-\xff\n\015()]*(?:?:\\[\x80-\xff][^\x80-\xff\n\01
80-\xff\n\015()]*)*\\)[\040\t]*))**: [\040\t]*(?:\\([\
)]*(?:?:\\[\x80-\xff]|\\([^\x80-\xff\n\015()]*(?:\
\x80-\xff\n\015()]*)*\\))^[^\x80-\xff\n\015()]*)*\\)[\
(\040)<>@, ;: ".\\[\\]\000-\037\x80-\xff]+(?:![^(\040)<>@
\037\x80-\xff)]|"^[^\x80-\xff\n\015"]*(?:\\[\x80-\xf
n\015"])*)*"[\040\t]*(?:\\([^\x80-\xff\n\015()]*(?:?:
\\([^\x80-\xff\n\015()]*(?:?:\\[\x80-\xff][^\x80-\xf
[^\x80-\xff\n\015()]*)*\\)[\040\t]*)*(?:\\. [\040\t]*(?
\n\015()]*(?:?:\\[\x80-\xff]|\\([^\x80-\xff\n\015()
ff][^\x80-\xff\n\015()]*)*\\))^[^\x80-\xff\n\015()]*
?:^[^(\040)<>@, ;: ".\\[\\]\000-\037\x80-\xff]+(?:![^(\040
000-\037\x80-\xff)]|"^[^\x80-\xff\n\015"]*(?:\\[\x80
xff\n\015"])*)*"[\040\t]*(?:\\([^\x80-\xff\n\015()]*(
ff]|\\([^\x80-\xff\n\015()]*(?:?:\\[\x80-\xff][^\x80-
\\))^[^\x80-\xff\n\015()])*\\)[\040\t]*))*@[\040\t]*
ff\n\015()]*(?:?:\\[\x80-\xff]|\\([^\x80-\xff\n\015\
\xff][^\x80-\xff\n\015()]*)*\\))^[^\x80-\xff\n\015()
*(?:^[^(\040)<>@, ;: ".\\[\\]\000-\037\x80-\xff]+(?:![^(\0
]\000-\037\x80-\xff)]|\\[(?:[^\x80-\xff\n\015\\[\\])|\\\
)][\040\t]*(?:\\([^\x80-\xff\n\015()]*(?:?:\\[\x80-\
\xff\n\015()]*(?:?:\\[\x80-\xff][^\x80-\xff\n\015()]*
ff\n\015()]*)*\\)[\040\t]*)*(?:\\. [\040\t]*(?:\\([^\x80-
?:?:\\[\x80-\xff]|\\([^\x80-\xff\n\015()]*(?:?:\\[\x
-\\xff\n\015()]*)*\\))^[^\x80-\xff\n\015()]*)*\\)[\040\t
>@, ;: ".\\[\\]\000-\037\x80-\xff]+(?:![^(\040)<>@, ;: ".\\\
0-\xff)]|\\[(?:[^\x80-\xff\n\015\\[\\])|\\[\x80-\xff])
\\([^\x80-\xff\n\015()]*(?:?:\\[\x80-\xff]|\\([^\x80-
(?:\\[\x80-\xff][^\x80-\xff\n\015()])*\\))^[^\x80-
\\)[\040\t]))*>)
EOF

community wiki
6 revs, 3 users 64%
Evan Carroll

16 O_O you would also need to be a regex master to understand what it is doing – [Chris McGrath](#) Jan 30, 2013 at 22:20

This regular expression matches parts of the MIME syntax like folding whitespace and comments; and it also allows control characters that are not permitted to be used.
– [awwright](#) Sep 13, 2020 at 1:14

The O'Reilly link is broken - "*Hrmm, we can't find that page... Sorry for the inconvenience*" – [Peter Mortensen](#) Feb 13, 2022 at 16:04

Weidly, O'Reilly seem to be as bad as Microsoft or IBM at making URLs which don't break. The current link is [oreilly.com/library/view/mastering-regular-expressions/...](#); the author has his own web site at [regex.info/book.html](#) (yes, http, not https (!)) – [tripleee](#) May 2 at 8:42



As you're writing in PHP I'd advice you to use the PHP built-in validation for emails.

49



```
filter_var($value, FILTER_VALIDATE_EMAIL)
```



If you're running a PHP version lower than 5.3.6, please be aware of this issue: [Bug #53091: Crashes when I try to filter a text of > 2264 characters](#)



If you want more information how this built-in validation works, see here: [Does PHP's filter_var FILTER_VALIDATE_EMAIL actually work?](#)

Share Improve this answer


edited Feb 13, 2022 at 16:06

Follow

community wiki

4 revs, 3 users 55%

[SimonSimCity](#)

gets a vote up, exactly what I was going to say. Doesn't handle IDN's but converting to puny code beforehand solves this. PHP>=5.3 has idn_to_ascii() for this. One of the best and easiest ways for validating an email. – [Taylor](#) Jan 25, 2012 at 23:00 



47

[Cal Henderson](#) (Flickr) wrote an article called [Parsing Email Addresses in PHP](#) and shows how to do proper RFC (2)822-compliant email address parsing.



You can also get the source code in [PHP](#), Python, and Ruby which is [Creative Commons licensed](#).



Share Improve this answer

edited Jul 21, 2021 at 9:24

Follow

community wiki

2 revs, 2 users 67%

[adnam](#)

it told me that `a@b` was valid – [dsdsdsdsd](#) Apr 16, 2014 at 11:44

- 3 @dsdsdsdsd Because `a@b` is valid... in this case `b` is the top-level domain. – [rink.attendant.6](#) Jul 31, 2015 at 21:19
-

The [original page with Python and Ruby versions](#) is down. One can still find them in the [Wayback Machine](#). – [darkdragon](#) Sep 16, 2023 at 14:01



45



I never bother creating with my own regular expression, because chances are that someone else has already come up with a better version. I always use [regexlib](#) to find one to my liking.

Share Improve this answer

answered [Oct 14, 2008 at 14:23](#)



Follow



community wiki
[Kon](#)

Nice site, however it's a bit weird that their built-in email spam protection partially hides some patterns. Especially those related to email matching. :D – [Tim Wißmann](#) Jul 9, 2021 at 0:07

- 1 That link is a search query, e.g. **"Search Results: 4128 regular expressions found."**. Which one in particular? How do you choose? – [Peter Mortensen](#) Feb 13, 2022 at 14:32
-



43



One simple regular expression which would at least not reject any valid email address would be checking for something, followed by an @ sign and then something followed by a period and at least 2 somethings. It won't reject anything, but after reviewing the spec I can't find any email that would be valid and rejected.

email =~ `/.+@[^@]+\.[^@]{2,}$/`

Share Improve this answer

edited Mar 7, 2013 at 19:57

Follow

community wiki

5 revs, 3 users 86%

spig

5 This is what I was looking for. Not very restrictive, but makes sure there is only 1 @ (as we're parsing a list and want to make sure there are no missing commas). FYI, you can have an @ on the left if it's in quotes: [Valid email addresses](#), but it's pretty fringe. – Josh Nov 11, 2011 at 6:16

3 After using it, realized it didn't work exactly.
`/^[^@]+@[^@]+\.[^@]{2}[^@]*$/` actually checks for 1 @ sign. Your regex will let multiple through because of the .* at the end. – Josh Nov 11, 2011 at 6:31

2 Right. I'm not trying to reject all invalid, just keep from rejecting a valid email address. – spig Nov 14, 2011 at 17:48

2 It would be far better to use this: `/^[^@]+@[^@]+\.[^@]{2,4}$/` making sure that it ends with 2 to 4 non @ characters. As @Josh pointed out it now allows an extra @ in the end. But you can also change that as well to:

`/^[^@]+@[^@]+\.[^a-z-A-Z]{2,4}$/` since all top level domains are a-Z characters. you can replace the `4` with `5` or more allowing top level domain names to be longer in the future as well. – [FLY](#) Jan 14, 2013 at 10:51 ✎

- 1 What if non ASCII char would used where dozens of servers support only, though ... At least, `A-Za-z0-9` then ...
– [Artfaith](#) Sep 1, 2020 at 5:02 ✎
-



40



There is not one which is really usable. I discuss some issues in my [answer to Is there a PHP library for email address validation?](#), it is discussed also in [Is regular expression recognition of an email address hard?](#).

In short, don't expect a single, usable regex to do a proper job. And the best regex will validate the syntax, not the validity of an e-mail (*jhohn@example.com* is correct, but it will probably bounce...).

Share Improve this answer

edited Jul 21, 2021 at 9:19

Follow


community wiki

4 revs, 3 users 61%

[Peter Mortensen](#)

Correct me if I'm wrong, but I believe that PHP uses PCRE patterns. If so, you should be able to craft something similar to [Abigail's RFC 5322 pattern](#). – [tchrist](#) Nov 7, 2010 at 20:24

@tchrist: not sure if PCRE has caught up to this syntax (which I discover). If so, not sure if PHP's PCRE has caught up to this version of PCRE... Well, if I understand correctly

this syntax, you can as well use a PEG parser, much clearer and complete than a regex anyway. – [PhiLho](#) Nov 10, 2010 at 14:51 

PCRE *has* caught up to it, but perhaps PHP has not caught up with PCRE. ☹ – [tchrist](#) Nov 10, 2010 at 15:09



You could use the one employed by the jQuery Validation plugin:

31



```
/^((( [a-z] | \d | [ !#$%&'*\+ \- \/= \? \^ _ ` { | } ~ ] |  
[\u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF] )+( \. ( [a-  
z] | \d | [ !#$%&'*\+ \- \/= \? \^ _ ` { | } ~ ] | [\u00A0-  
\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF] )+ )*) | (( \x22 )  
((( ( \x20 | \x09 ) * ( \x0d \x0a ) ) ? ( \x20 | \x09 ) + ) ? ( ( [ \x01-  
\x08 \x0b \x0c \x0e - \x1f \x7f ] | \x21 | [ \x23 - \x5b ] | [ \x5d-  
\x7e ] | [ \u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF ] ) |  
( \ ( [ \x01-\x09 \x0b \x0c \x0d - \x7f ] | [ \u00A0-  
\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF ] ) ) ) *  
((( ( \x20 | \x09 ) * ( \x0d \x0a ) ) ? ( \x20 | \x09 ) + ) ?  
( \x22 ) ) ) @ ( ( ( [ a - z ] | \d | [ \u00A0-\uD7FF\uF900-  
\uFDCF\uFDF0-\uFFEF ] ) | ( ( [ a - z ] | \d | [ \u00A0-  
\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF ] ) ( [ a - z ] | \d | -  
| \. | _ | ~ | [ \u00A0-\uD7FF\uF900-\uFDCF\uFDF0-  
\uFFEF ] ) * ( [ a - z ] | \d | [ \u00A0-\uD7FF\uF900-  
\uFDCF\uFDF0-\uFFEF ] ) ) ) \. ) + ( ( [ a - z ] | [ \u00A0-  
\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF ] ) | ( ( [ a - z ] |  
[ \u00A0-\uD7FF\uF900-\uFDCF\uFDF0-\uFFEF ] ) ( [ a -  
z ] | \d | - | \. | _ | ~ | [ \u00A0-\uD7FF\uF900-\uFDCF\uFDF0-  
\uFFEF ] ) * ( [ a - z ] | [ \u00A0-\uD7FF\uF900-\uFDCF\uFDF0-  
\uFFEF ] ) ) ) \. ? $ / i
```

Share Improve this answer

answered [May 23, 2009 at 18:22](#)

Follow

community wiki

this seems to be doing a good job. It allowed: `a-b'c_d.e@f-g.h` but was able to catch the inappropriate variations, such as `a-b'c_d.@f-g.h` and `a-b'c_d.e@f-.h` – [dsdsdsdsd](#) Apr 16, 2014 at 11:52

2 Do you have some reference for it? – [norok2](#) Sep 1, 2021 at 7:18



31



[Since May 2010](#), non-Latin (Chinese, Arabic, Greek, Hebrew, Cyrillic and so on) domain names exist on the Internet. Everyone has to change the email regex used, because those characters are surely not to be covered by `[a-z]/i` nor `\w`. They will all fail.



After all, the **best** way to validate the email address is still to actually *send* an email to the address in question to validate the address. If the email address is part of user authentication (register/login/etc), then you can perfectly combine it with the user activation system. I.e. send an email with a link with an unique activation key to the specified email address and only allow login when the user has activated the newly created account using the link in the email.

If the purpose of the regex is just to quickly inform the user in the UI that the specified email address doesn't look like in the right format, best is still to check if it matches basically the following regex:

```
^([\^.@]+)(\.[\^.@]+)*@([\^.@]+\.[\^.@]+)([\^.@]+)$
```

Simple as that. Why on earth would you care about the characters used in the name and domain? It's the client's responsibility to enter a valid email address, not the server's. Even when the client enters a *syntactically* valid email address like `aa@bb.cc`, this does not guarantee that it's a legit email address. No one regex can cover that.

Share Improve this answer

edited Feb 13, 2022 at 15:49

Follow

community wiki

2 revs

BalusC

5 I agree the sending an authentication message is usually the best way for this kind of stuff, syntactically correct and valid are not the same. I get frustrated when I get made to type my email address twice for "Confirmation" as if I can't look at what I typed. I only copy the first one to the second anyway, it seems to be becoming used more and more. – [PeteT](#) Feb 2, 2010 at 15:05

2 agree! but this regex i don't think is valid because it allow spaces after the @. eg. test@test.ca com net is consider a valid email by using the above regex where as it should be returning invalid. – [CB4](#) Nov 8, 2017 at 17:54



For the most comprehensive evaluation of the best regular expression for validating an email address please

28

see this link; "[Comparing E-mail Address Validating Regular Expressions](#)"



Here is the current top expression for reference purposes:

```
/^([\w\!#\$\%\&\'\*\+\-\/\=\?\^\`\{\|\}\~]+\.)*
[\w\!#\$\%\&\'\*\+\-\/\=\?\^\`\{\|\}\~]+@((((([a-
z0-9]{1}[a-z0-9\~]{0,62}[a-z0-9]{1})|([a-z])\~)+[a-
z]{2,6})|(\d{1,3}\~){3}\d{1,3}(\~:\d{1,5})?)$/i
```

Share Improve this answer

answered May 28, 2010 at 22:03

Follow

community wiki

[Eric Schoonover](#)

1 spoon16: That link isn't really correct. Its statement that there can be no perfect pattern for validating email addresses is patently fault. You **can**, but you have to make sure that you follow the RFC right down to the letter. And you have to pick the right RFC, too. – [tchrist](#) Nov 7, 2010 at 20:27

1 The "best" right now does not work with java regex - even after properly escaping and converting the string. – [Eric Chen](#) Apr 17, 2012 at 20:57



The [HTML5 specification suggests](#) a simple regex for validating email addresses:

26

```
/^[a-zA-Z0-9.!#$%&'*\+\/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-z
9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)$
```



This intentionally doesn't comply with [RFC 5322](#).



Note: This requirement is a [wilful violation](#) of [RFC 5322](#), which defines a syntax for e-mail addresses that is simultaneously too strict (before the @ character), too vague (after the @ character), and too lax (allowing comments, whitespace characters, and quoted strings in manners unfamiliar to most users) to be of practical use here.

The total length could also be limited to 254 characters, per [RFC 3696 errata 1690](#).

Share Improve this answer

edited Feb 14, 2022 at 23:19



Follow

community wiki

6 revs, 2 users 48%

[Peter Mortensen](#)

-
- 2 Best answer! Here's a link to the w3 recommendation: [w3.org/TR/html5/forms.html#valid-e-mail-address](https://www.w3.org/TR/html5/forms.html#valid-e-mail-address) This regex is adopted by many browsers. – [Ryan Taylor](#) Nov 6, 2017 at 22:13
-
- 6 This is SO not the best answer! This pattern matches this wholly invalid address: `invalid@emailaddress`. I would urge caution and much testing before you use it! – [Sheridan](#) Mar 21, 2018 at 11:47
-

- 1 @Sheridan, if you think there is an issue with the HTML5 spec you can raise an issue here:
github.com/w3c/html/issues – Luna Mar 21, 2018 at 12:56
- 2 example@localhost is valid, but for a real world application you may want to enforce a domain extension, all you need to do is change the final * to a + to achieve this (changing that part of the pattern from 0+ to 1+) – Mitch Satchwell May 16, 2018 at 9:05 
- 5 invalid@emailaddress is a valid email address... it just doesn't exist. (And actually, depending on your network configuration, it might even be deliverable inside your local network.) And the only way to tell if an address exists or not is to send it an email. – awwright Aug 31, 2020 at 18:18 

According to the official standard, [RFC 2822](#), a valid email regex is:

16

```
(?:[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\.|(?!\.)(?:[a-z0-9!#$%&'*/+=?
\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f])|\.(?:[a-z01-
\x7f])*)@"(?:?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.|[a-
9])?|\[(?:?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.|{
[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[a-z01-\x08\x0b\
\x5a\x53-\x7f]|\.|\[a-z01-\x09\x0b\x0c\x0e-\x7f])+\)])
```

If you want to use it in Java, it's really very easy:

```
import java.util.regex.*;

class regexSample
{
    public static void main(String args[])
    {
        //Input the string for validation
        String email = "xyz@hotmail.com";
    }
}
```

```

        //Set the email pattern string
        Pattern p = Pattern.compile("(?:[a-z0-9!#$%&'
z0-9!#$%&' *+/?^_`{|}~-]+)*|"
            +"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-\\x1f
\\x7f]|\\[\\x01-\\x09\\x0b\\x0c\\x0e-\\x7f])*\\")"
            + "@(?: (?:[a-z0-9](?:[a-z0-9-]
(?:[a-z0-9-]*[a-z0-9])?)?|\\[ (?: (?:25[0-5]|2[0-4][0-9]|
(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)|[a-z0-9-]*[a-z
\\x08\\x0b\\x0c\\x0e-\\x1f\\x21-\\x5a\\x53-\\x7f]|\\[\\
\\x09\\x0b\\x0c\\x0e-\\x7f]))+\\.\\")");

        //Match the given string with the pattern
        Matcher m = p.matcher(email);

        //Check whether match is found
        boolean matchFound = m.matches();

        if (matchFound)
            System.out.println("Valid Email Id.");
        else
            System.out.println("Invalid Email Id.");
    }
}

```

Share Improve this answer

edited Nov 21, 2023 at 15:30

Follow

community wiki

6 revs, 4 users 66%

AZ_

3 Your regex does not include first uppercase letter for example **Leonardo.davinci@gmail.com** which could be annoying for some users. Use this one instead: `(?:[A-Za-z0-`

```

9!#$%&' *+/?^_`{|}~-]+(?:\\. [A-Za-z0-9!#$%&' *+/?
^_`{|}~-]+)*|"(?:[\\x01-\\x08\\x0b\\x0c\\x0e-
\\x1f\\x21\\x23-\\x5b\\x5d-\\x7f]|\\[\\x01-
\\x09\\x0b\\x0c\\x0e-\\x7f])*\\")@(?: (?:[a-z0-9](?:[a-

```

```
z0-9-]*[a-z0-9])?\.\.)+[a-z0-9](?:[a-z0-9-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\[[\x01-\x09\x0b\x0c\x0e-\x7f]\])+)\.])
```

 – [Kebab Krabby](#) Jul 17, 2019 at 15:07

@KebabKrabby Thanks, please edit the answer, I'll accept the change. – [AZ_](#) Jul 31, 2019 at 7:55

If i add that change to your answer it wont be RFC 2822 anymore so i dont know if thats correct. – [Kebab Krabby](#) Jul 31, 2019 at 22:06

- 2 @KebabKrabby: I guess we would need to apply the pattern with case insensitivity somewhere in the matching options, not change the Regex itself. – [Thomas Weller](#) Sep 25, 2020 at 11:46



15



For a vivid demonstration, the following monster is pretty good, but it still does not correctly recognize all syntactically valid email addresses: it recognizes nested comments up to four levels deep.

This is a job for a parser, but even if an address is syntactically valid, it still may not be deliverable.

Sometimes you have to resort to the hillbilly method of "Hey, y'all, watch ee-us!"

```
// derivative of work with the following copyright and
// Copyright (c) 2004 Casey West. All rights reserved
// This module is free software; you can redistribute
// modify it under the same terms as Perl itself.

// see http://search.cpan.org/~cwest/Email-Address-1.8
```

```
private static string gibberish = @"
(?-xism:(?:(?-xism:(?-xism:(?-xism:(?-xism:(?-xism:(?-
s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-x
\x0A\x0D)))|(?-xism:\s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)
|(?-xism:\\(?-xism:[^\x0A\x0D]))|)+)*\s*\)\s*)))+)*\s*\
|\s+)*[^\x00-\x1F\x7F(<>\\[\]:;@\\,.<DQ>\s]+(?-xism:(?-
s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-x
\x0A\x0D)))|(?-xism:\s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)
|(?-xism:\\(?-xism:[^\x0A\x0D]))|)+)*\s*\)\s*)))+)*\s*\
|\s+)*|(?-xism:(?-xism:(?-xism:\s*\((?:\s*(?-xism:(?-
?>[^\(\)\\\]+))|(?-xism:\\(?-xism:[^\x0A\x0D]))|(?-xism:\\
:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-xism:[^
0D]))|)+)*\s*\)\s*)))+)*\s*\)\s*)+|\s+)*<DQ>(?-xism:(?-
^\\<DQ>)]|(?-xism:\\(?-xism:[^\x0A\x0D])))+<DQ>(?-xism
sm:\s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\
m:[^\x0A\x0D]))|(?-xism:\s*\((?:\s*(?-xism:(?-xism:(?>
]+))|(?-xism:\\(?-xism:[^\x0A\x0D]))|)+)*\s*\)\s*)))+)*
s*)+|\s+)*)))+(?-xism:(?-xism:(?-xism:\s*\((?:\s*(?-x
-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-xism:[^\x0A\x0D]))|(?
\s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-
^\x0A\x0D]))|)+)*\s*\)\s*)))+)*\s*\)\s*)+|\s+)*<(?-xism
sm:(?-xism:(?-xism:(?-xism:\s*\((?:\s*(?-xism:(?-xism:
)\\\]+))|(?-xism:\\(?-xism:[^\x0A\x0D]))|(?-xism:\s*\((
?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-xism:[^\x0A\
|)+)*\s*\)\s*)))+)*\s*\)\s*)+|\s+)*(?-xism:[^\x00-\x1F\
>\\[\]:;@\\,.<DQ>\s]+(?:\\. [^\x00-\x1F\x7F(<>\\[\]:;@\\,.<
+)*)(?-xism:(?-xism:\s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)
|(?-xism:\\(?-xism:[^\x0A\x0D]))|(?-xism:\s*\((?:\s*(?
(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-xism:[^\x0A\x0D]))|
*\)\s*)))+)*\s*\)\s*)+|\s+)*|(?-xism:(?-xism:(?-xism:\\
:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-xism:[^
0D]))|(?-xism:\s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))
sm:\\(?-xism:[^\x0A\x0D]))|)+)*\s*\)\s*)))+)*\s*\)\s*)+
<DQ>(?-xism:(?-xism:[^\\<DQ>)]|(?-xism:\\(?-xism:[^\x0
)))+<DQ>(?-xism:(?-xism:\s*\((?:\s*(?-xism:(?-xism:(?>
]+))|(?-xism:\\(?-xism:[^\x0A\x0D]))|(?-xism:\s*\((?:\
ism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?-xism:[^\x0A\x0D
)*\s*\)\s*)))+)*\s*\)\s*)+|\s+)*))\@(?-xism:(?-xism:(?-
?-xism:\s*\((?:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xis
-xism:[^\x0A\x0D]))|(?-xism:\s*\((?:\s*(?-xism:(?-xism
())\\]+))|(?-xism:\\(?-xism:[^\x0A\x0D]))|)+)*\s*\)\s*)
*\)\s*)+|\s+)*(?-xism:[^\x00-\x1F\x7F(<>\\[\]:;@\\,.<DQ
?:\\. [^\x00-\x1F\x7F(<>\\[\]:;@\\,.<DQ>\s]+))*)(?-xism:(?
```

```

\S*\((?:\S*(? xism:(?-xism:(?>[^()\\]+))|(?-xism:\\(?!
^\\x0A\\x0D)))|(?-xism:\S*\((?:\S*(?-xism:(?-xism:(?>[^(
)|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|)+)*\S*\)\S*)))+*\S*
+|\S+)*|(?-xism:(?-xism:(?-xism:\S*\((?:\S*(?-xism:(?
(?>[^()\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|(?-xism:
?:\S*(?-xism:(?-xism:(?>[^()\\]+))|(?-xism:\\(?!-xism:[
x0D]))|)+)*\S*\)\S*)))+*\S*\)\S*+|\S+)*\[(?:\S*(?-xis
ism:[^\\[\\]))|(?-xism:\\(?!-xism:[^\\x0A\\x0D])))+)*\S*
sm:(?-xism:\S*\((?:\S*(?-xism:(?-xism:(?>[^()\\]+))|(?
\\(?!-xism:[^\\x0A\\x0D]))|(?-xism:\S*\((?:\S*(?-xism:(?
?>[^()\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|)+)*\S*\)
)*\S*\)\S*+|\S+)*)))>(?!-xism:(?!-xism:\S*\((?:\S*(?
xism:(?>[^()\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|(?
S*\((?:\S*(?-xism:(?-xism:(?>[^()\\]+))|(?-xism:\\(?!
\\x0A\\x0D]))|)+)*\S*\)\S*)))+*\S*\)\S*+|\S+)*))|(?-xis
ism:(?!-xism:(?!-xism:\S*\((?:\S*(?-xism:(?!-xism
())\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|(?-xism:\S*\(
(?-xism:(?!-xism:(?>[^()\\]+))|(?-xism:\\(?!-xism:[^\\x0A
]|)+)*\S*\)\S*)))+*\S*\)\S*+|\S+)*(?-xism:[^\\x00-\\x1F
<>\\[\\]:;@\\,.<DQ>\\S]+(?:\\.\\.[^\\x00-\\x1F\\x7F()<>\\[\\]:;@\\,
.]*)?(?!-xism:(?!-xism:\S*\((?:\S*(?-xism:(?!-xism:(?>[^(
)|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|(?-xism:\S*\((?:\S*(
:(?!-xism:(?>[^()\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))
S*\)\S*)))+*\S*\)\S*+|\S+)*|(?-xism:(?!-xism:(?!-xism:
?:\S*(?-xism:(?!-xism:(?>[^()\\]+))|(?-xism:\\(?!-xism:[
x0D]))|(?-xism:\S*\((?:\S*(?-xism:(?!-xism:(?>[^()\\]+)
ism:\\(?!-xism:[^\\x0A\\x0D]))|)+)*\S*\)\S*)))+*\S*\)\S*)
*<DQ>(?!-xism:(?!-xism:[^\\<DQ>])|(?-xism:\\(?!-xism:[^\\x
])))+<DQ>(?!-xism:(?!-xism:\S*\((?:\S*(?-xism:(?!-xism:(?
\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|(?-xism:\S*\(
(?-xism:(?!-xism:(?>[^()\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0
+)*\S*\)\S*)))+*\S*\)\S*+|\S+)*))\\@(?-xism:(?!-xism:(?
(?-xism:\S*\((?:\S*(?-xism:(?!-xism:(?>[^()\\]+))|(?-xi
?!-xism:[^\\x0A\\x0D]))|(?-xism:\S*\((?:\S*(?-xism:(?!-xis
^()\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|)+)*\S*\)\S*
S*\)\S*+|\S+)*(?-xism:[^\\x00-\\x1F\\x7F()<>\\[\\]:;@\\,.<D
(?:\\.\\.[^\\x00-\\x1F\\x7F()<>\\[\\]:;@\\,.<DQ>\\S]+)*)?(?!-xism:(
:\S*\((?:\S*(?-xism:(?!-xism:(?>[^()\\]+))|(?-xism:\\(?!
[^\\x0A\\x0D]))|(?-xism:\S*\((?:\S*(?-xism:(?!-xism:(?>[^
)|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|)+)*\S*\)\S*)))+*\S
+|\S+)*|(?-xism:(?!-xism:(?!-xism:\S*\((?:\S*(?-xism:(
:(?>[^()\\]+))|(?-xism:\\(?!-xism:[^\\x0A\\x0D]))|(?-xism
(?:\S*(?-xism:(?!-xism:(?>[^()\\]+))|(?-xism:\\(?!-xism:
\\x0D]))|)+)*\S*\)\S*)))+*\S*\)\S*+|\S+)*\[(?:\S*(?

```

```
xism:[^\[\]\\\]|(?-xism:\\(?-xism:[^\x0A\x0D]))+)*\s*
ism:(?-xism:\s*\\(?:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(:\\(?:-xism:[^\x0A\x0D]))|(?-xism:\s*\\(?:\s*(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?:-xism:[^\x0A\x0D]))|)+)*\s*\\+)*\s*\\)\s*)+|\s+)*)))(?-xism:\s*\\(?:\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?:-xism:[^\x0A\x0D]))|(?-xism:\s*\s*(?-xism:(?-xism:(?>[^\(\)\\\]+))|(?-xism:\\(?:-xism:[^\x0A\x0D]))|)+)*\s*\\)\s*)))*\s*\\)\s*)*)"
.Replace("<DQ>", "\\")
.Replace("\t", "")
.Replace(" ", "")
.Replace("\r", "")
.Replace("\n", "");

private static Regex mailbox =
    new Regex(gibberish, RegexOptions.ExplicitCapture);
```

Share Improve this answer

edited Feb 13, 2022 at 15:04

Follow

community wiki

3 revs, 2 users 66%

Greg Bacon

-
- 1 What programming language? [C#](#)? – Peter Mortensen Feb 13, 2022 at 15:04
-



RFC 5322 standard:

11



Allows dot-atom local-part, quoted-string local-part, obsolete (mixed dot-atom and quoted-string) local-part, domain name domain, (IPv4, IPv6, and IPv4-mapped IPv6 address) domain literal domain, and (nested) CFWS.





```
'/^((?!(?>(?1)"?(?>\\[ -~]|["^"])"?(?1)){255,})(?!
(?>(?1)"?(?>\\[ -~]|["^"])"?(?1)){65,}@)((?>(?>(?>
((?>(?>(?>\\x0D\\x0A)?[\\t ])+|(?>[\\t ]*\\x0D\\x0A)?[\\t
 ]+)?)(\\((?>(?2)(?>[\\x01-\\x08\\x0B\\x0C\\x0E-\\'*-\\[\\]-
\\x7F]|\\[\\x00-\\x7F]|(?3)))?(?2)\\))+(?2))|(?2))?)
([!#-\\'*/-9=?^_~]+|"(?>(?2)(?>[\\x01-
\\x08\\x0B\\x0C\\x0E-!#-\\[\\]-\\x7F]|\\[\\x00-\\x7F]))*(?
2)" )(?>(?1)\\. (?1)(?4))*(?1)@(?!(?1)[a-z0-9-]{64,})
(?1)(?>([a-z0-9](?>[a-z0-9-]*[a-z0-9]))?(?>(?1)\\.
(?!(?1)[a-z0-9-]{64,})(?1)(?5)){0,126}|\\[(?:(?
>IPv6:(?>([a-f0-9]{1,4})(?>:(?6)){7}|(?!(?:.*[a-
f0-9][:\\])){8,})((?6)(?>:(?6)){0,6})?::(?7)?))|(?>
(?>IPv6:(?>(?6)(?>:(?6)){5}:|(?!(?:.*[a-f0-9]:)
{6,})(?8)??::(?>((?6)(?>:(?6)){0,4}):))?)?(25[0-
5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?[0-9])(?>\\. (?9))
{3}))\\])(?1)$/isD'
```

RFC 5321 standard:

Allows dot-atom local-part, quoted-string local-part, domain name domain, and (IPv4, IPv6, and IPv4-mapped IPv6 address) domain literal domain.

```
'/^((?!(?>"?(?>\\[ -~]|["^"])"?){255,})(?!"(?>\\[ -~]|
["^"]){65,}"?@)((?>([!#-\\'*/-9=?^_~]+)(?>\\. (?
1))*|"(?>[!#-\\[\\]-~]|\\[\\[ -~])*")@(?!(?:.*[\\^.]
{64,})(?>([a-z0-9](?>[a-z0-9-]*[a-z0-9]))?(?>\\. (?2))
{0,126}|\\[(?:(>IPv6:(?>([a-f0-9]{1,4})(?>:(?3))
{7}|(?!(?:.*[a-f0-9][:\\])){8,})((?3)(?>:(?3))
{0,6})?::(?4)?))|(?>(>IPv6:(?>(?3)(?>:(?3)){5}:|
(?!(?:.*[a-f0-9]:){6,})(?5)??::(?>((?3)(?>:(?3))
{0,4}):))?)?(25[0-5]|2[0-4][0-9]|1[0-9]{2}|[1-9]?
[0-9])(?>\\. (?6)){3}))\\])(?1)$/iD'
```

Basic:

Allows dot-atom local-part and domain name domain (requiring at least two domain name labels with the TLD

limited to 2-6 alphabetic characters).

```
"/^(?!.{255,})(?!.{65,}@)([!#-'*+\\/-9=?^_~ - ]+)(?>\\. (?1))*@(?!. *[^.] {64,})(?>[a-z0-9](?>[a-z0-9-]*[a-z0-9])?\\. ) {1,126} [a-z] {2,6} $/iD"
```

Share Improve this answer


edited Dec 19, 2012 at 12:17

Follow

community wiki

58 revs, 2 users 98%

MichaelRushton

-
- 1 What the devil language is that in?? I see a `/D` flag, and you've quoted it with single quotes yet also used slashes to delimit the pattern? It's not Perl, and it can't be PCRE. Is it therefore PHP? I believe those are the only three that allow recursion like `(?1)` . – [tchrist](#) Nov 7, 2010 at 20:32
-
- 1 It's in PHP, which uses PCRE. The slashes are used only to delimit special characters like parentheses, square brackets, and of course slashes and single quotes. The `/D` flag, if you didn't know, is to prevent a newline being added to the end of the string, which would be allowed otherwise. – [Michael](#) Feb 19, 2011 at 18:24 
-



11



Here's the PHP code I use. I've chosen this solution in the spirit of "false positives are better than false negatives" as declared by another commenter here *and* with regards to keeping your response time up and server load down ... there's really no need to waste server resources with a regular expression when this will weed out most simple



user errors. You can always follow this up by sending a test email if you want.



```
function validateEmail($email) {  
    return (bool) stripos($email, '@');  
}
```

Share Improve this answer

edited Feb 13, 2022 at 16:00

Follow

community wiki

2 revs, 2 users 57%

Mac

2 a) The "waste server resources" is infinitesimal, but if you are so inclined, you could do it client side with JS b) What is you need to send a registration mail and the user enters me@forgotthedotcom ? Your "solution" fails and you lose a user. – [johnjohn](#) Apr 3, 2012 at 9:40

2 a) Relying on a JS validation that would fail when JavaScript is disabled doesn't sound like the best idea either (just btw) – [auco](#) Dec 6, 2013 at 15:39

What do you mean by false positives? Rejecting email addresses that you shouldn't have rejected? Or accepting email addresses that you shouldn't have accepted?

– [Peter Mortensen](#) Feb 13, 2022 at 16:01



9

I've been using this touched up version of the OP's regex for a while and it hasn't left me with too many surprises. I've never encountered an apostrophe in an email yet so it doesn't validate that. It does validate



Jean+François@anydomain.museum and 试@例子.测试
مثال. آزمون یشی. , but not weird abuse of those non
alphanumeric characters .+@you.com .

```
(?!^[. +&' _ - ]*@[. *$)(^[ _\w\d+&' - ]+(\. [ _\w\d+&' - ]*)*@[ \w\{1, 3}\} | ([ \w]{2, })))$)
```

It does support IP addresses you@192.168.1.1 , but I
haven't refined it enough to deal with bogus IP address
ranges such as 999.999.999.1 .

~~It also supports all the TLDs over three characters which
stops asdf@asdf.asdf which I think the original let
through.~~ [I've been beat, there are too many TLDs now
over 3 characters.](#)

I know the OP has abandoned his regex, but this flavour
lives on.

Share Improve this answer

edited Feb 13, 2022 at 16:29

Follow

community wiki

9 revs, 3 users 54%

TombMedia

-
- 1 For all: The regular expression in the question was removed
in revision 10, in 2015 (about 7 years later).
– [Peter Mortensen](#) Feb 13, 2022 at 16:25

-
- 1 Strikeout shouldn't be necessary. That is what the revision
history is for. If something is no longer valid, it should be

removed. The answer should be as if it was written today.

– [Peter Mortensen](#) Feb 13, 2022 at 16:27 



8



The regular expression for an email address is:

```
/^(("[?:[!#-\[\]-\u{10FFFF}]|\\[\t -\u{10FFFF}])*"| [!#-\u{10FFFF}](?:\.?[!#-'*+\-/-9=?A-Z\^-\u{10FFFF}])*)@([\u{10FFFF}](?:\.?[!#-'*+\-/-9=?A-Z\^-\u{10FFFF}])*)\.[\u{10FFFF}](?:\.[\u{10FFFF}](?:[!#-'*+\-/-9=?A-Z\^-\u{10FFFF}])*)|[\u{10FFFF}](?:[!#-'*+\-/-9=?A-Z\^-\u{10FFFF}])*)$
```

This regular expression is 100% identical to the `addr-spec` [ABNF](#) for non-obsolete email addresses, as specified across [RFC 5321](#), [RFC 5322](#), and [RFC 6532](#).

Additionally, you must verify:

- The email address is well-formed UTF-8 (or ASCII, if you cannot send to internationalized email addresses)
- The address is not more than 320 UTF-8 bytes
- The user part (the first match group) is not more than 64 UTF-8 bytes
- The domain part (the second match group) is not more than 255 UTF-8 bytes

The easiest way to do all of this is to use an existing function. In PHP, see the [filter_var](#) function using `FILTER_VALIDATE_EMAIL` and `FILTER_FLAG_EMAIL_UNICODE` (if you can send to internationalized email addresses):

```
$email_valid = filter_var($email_input,  
FILTER_VALIDATE_EMAIL, FILTER_FLAG_EMAIL_UNICODE);
```

However, maybe you're building such a function—indeed the easiest way to implement this is to use a regular expression.

Remember, this only verifies that the email address will not cause a syntax error. The only way to verify that the address can receive email is to *actually* send an email.

Next, I will treat how you generate this regular expression.

I write a new answer, because most of the answers here make the mistake of either specifying a pattern that is too restrictive (and so have not aged well); or they present a regular expression that's actually matching a header for a [MIME](#) message, and not the email address itself.

It is entirely possible to make a regular expression from an ABNF, so long as there are no recursive parts.

RFC 5322 specifies what is legal to send in a MIME message; consider this the upper bound on what is a legal email address.

However, to follow this ABNF exactly would be a mistake: this pattern technically represents how you encode an email address *in a MIME message*, and allows strings not part of the email address, like folding whitespace and

comments; and it includes support for obsolete forms that are not legal to generate (but that servers read for historical reasons). An email address does not include these.

RFC 5322 explains:

Both atom and dot-atom are interpreted as a single unit, comprising the string of characters that make it up. Semantically, the optional comments and FWS surrounding the rest of the characters are not part of the atom; the atom is only the run of atext characters in an atom, or the atext and "." characters in a dot-atom.

In some of the definitions, there will be non-terminals whose names start with "obs-". These "obs-" elements refer to tokens defined in the obsolete syntax in section 4. In all cases, these productions are to be ignored for the purposes of generating legal Internet messages and MUST NOT be used as part of such a message.

If you remove `CFWS`, `BWS`, and `obs - *` rules from the `addr-spec` in RFC 5322, and perform some optimization on the result (I used ["greenery"](#)), you can produce this regular expression, quoted with slashes and anchored (suitable for use in ECMAScript and compatible dialects, with added newline for clarity):

```

/^("(?:[!#-\[\]~]|\[\t -~])*"| [!#-'*+\-/-9=?A-Z\^~]
Z\^~~])*)
@([!#-'*+\-/-9=?A-Z\^~~])(?:\.?[!#-'*+\-/-9=?A-Z\^~~])*)

```

This only supports ASCII email addresses. To support [RFC 6532 Internationalized Email Addresses](#), replace the `~` character with `\u{10FFFF}` (PHP, ECMAScript with the `u` flag), or `\uFFFF` (for UTF-16 implementations, like [.NET](#) and older ECMAScript/JavaScript):

```

/^("(?:[!#-\[\]-\u{10FFFF}]|\[\t -\u{10FFFF}])*"| [!#-
\u{10FFFF}](?:\.?[!#-'*+\-/-9=?A-Z\^-\u{10FFFF}])*)@([
\u{10FFFF}](?:\.?[!#-'*+\-/-9=?A-Z\^-\u{10FFFF}])*)|\[

```

This works, because the ABNF we are using is not recursive, and so forms a non-recursive, regular grammar that can be converted into a regular expression.

It breaks down like so:

- The user part (before the `@`) may be a dot-atom or a quoted-string
- `"([!#-\[\]~]|\[\t -~])*"` specifies the quoted-string form of the user, e.g. `"root@home"@example.com`. It permits any non-control character inside double quotes; except that spaces, tabs, double quotes, and backslashes must be backslash-escaped.
- `[!#-'*+\-/-9=?A-Z\^~]` is the first character of the dot-atom of the user.

- `(\.[!#- '*+\- /-9=?A-Z\^-\~])*` matches the rest of the dot-atom, allowing dots (except after another dot, or as the final character).
- `@` denotes the domain.
- The domain part may be a dot-atom or a domain-literal.
- `[!#- '*+\- /-9=?A-Z\^-\~](\.[!#- '*+\- /-9=?A-Z\^-\~])*` is the same dot-atom form as above, but here it represents domain names and IPv4 addresses.
- `\[[!-Z\^-\~]*\]` will match IPv6 addresses and future definitions of host names.

This regular expression allows all specification-compliant email addresses, and can be used verbatim in a MIME message (except for line length limits, in which case folding whitespace must be added).

This also sets non-capturing groups such that `match[1]` will be the user, `match[2]` will be the host. (However if `match[1]` starts with a double quote, then filter out backslash escapes, and the start and end double quotes: `"root"@example.com` and `root@example.com` identify the same inbox.)

Finally, note that [RFC 5321](#) sets limits on how long an email address may be. The user part may be up to 64 bytes, and the domain part may be up to 255 bytes. Including the `@` character, the limit for the entire address is 320 bytes. This is measured in bytes after the address is UTF-8 encoded; not characters.

Note that RFC 5322 ABNF defines a permissive syntax for domain names, allowing names currently known to be invalid. This also allows for domain names that could become legal in the future. This should not be a problem, as this should be handled the same way a non-existent domain name is.

Always consider the possibility that a user typed in an email address that works, but that they do not have access to. *The only foolproof way to verify an email address is to send an email.*

This is adapted from my article [E-Mail Addresses & Syntax](#).

Share Improve this answer

edited Feb 10, 2022 at 21:51

Follow

community wiki

8 revs, 2 users 81%

[awwright](#)

1 I can use this in javascript, but can't get it formatted for C# use. I've tried putting it into regex101 website and it says its invalid – [Post Impatica](#) May 12, 2021 at 21:52

1 @PostImpatica What is the error, exactly? Regex101 expects a regular expression that is slash-delimited. I don't know which dialect C# expects. If your dialect is slash-delimited, you'll need to escape the slashes with a backslash.
– [awwright](#) May 17, 2021 at 2:21

1 "John Smith"@example.com doesn't work with these on regexr.com – [Cees Timmerman](#) Nov 19, 2022 at 0:32 

- 2 @CeesTimmerman Spaces must be escaped in the quoted form, see rfc-editor.org/rfc/rfc5322#section-3.2.4 This is mentioned in my post: "It permits any non-control character inside double quotes; except that spaces, tabs, double quotes, and backslashes must be backslash-escaped." Note that whitespace is not considered "printing" in ASCII, see the VCHAR production in rfc-editor.org/rfc/rfc5234#appendix-B.1 – [awwright](#) Nov 20, 2022 at 2:13
-



8



Strange that you "cannot" allow 4 characters TLDs. You are banning people from *.info* and *.name*, and the length limitation stop *.travel* and *.museum*, but yes, they are less common than 2 characters TLDs and 3 characters TLDs.

You should allow uppercase alphabets too. Email systems will normalize the local part and domain part.

For your regex of domain part, domain name cannot starts with '-' and cannot ends with '-'. Dash can only stays in between.

If you used the [PEAR](#) library, check out their mail function (I forgot the exact name/library). You can validate email address by calling one function, and it validates the email address according to definition in [RFC 822](#).

Share Improve this answer

edited Feb 13, 2022 at 14:48


Follow

community wiki

2 revs, 2 users 78%

[Joseph Yee](#)

3 @Joseph Yee: Isn't RFC 822 a bit dated? – [tchrist](#) Nov 7, 2010 at 20:27

1 Re ' "cannot" allow 4 characters TLDs': What you referring to? – [Peter Mortensen](#) Feb 13, 2022 at 14:45 

1

2

3

Next



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.