How to use Sieve

This tutorial will cover the very basics of the powerful Sieve Filtering Language. Sieve is an email filtering language. A sieve script has a set of rules, each one comprising a test, and a set of instructions.

Each incoming mail is checked against each rule: if the test matches, the instructions are performed on that mail.

Definitions

- **Header**
    - The headers of an email contain information about the mail: Sender, Recipient, which server it was sent from, which servers it travelled through, etc. FastMail adds a number of headers to allow you to filter for spam among other things. To view the headers on a piece of mail, use the `More` button in the web interface and select `Show Raw Message`.
- **Regex or Regexp**
    - Short for 'Regular Expression', a system that can be used to match parts of text, using an advanced algorithm. There are a number of tutorials and syntax checkers available online.

Getting started

The syntax of the sieve language is quite simple:

```
if <condition> {
    action1;
    action2;
    ...
 }
```

1. The `<condition>` has:
    - The part of the email you want to test (body, header, envelope) AND
    - The test you want to do against that part to perform the required actions.
2. The actions: what you want to do to the email (discard, reject, redirect, fileinto, stop, keep).

In order to learn how to write your own Sieve scripts, it's a good idea to first learn how to read a Sieve script. When looking at your Filter Source (through the Define Rules screen, go to **Advanced**, the link is at the bottom: **View filter source**) look through the rules and see if you recognize items from your rules.

Let's take a size filter as an example:

```
if size :over 1024K {
  reject "Message not delivered; size over limit accepted by recipient";
  stop;
}
```

Examining this step by step:

if size :over 1024K {

Here is where you set the test condition for this rule. If this condition is met, the action is performed; if not, the script continues until it hits a rule that does match. In this case it tests if the size of the email being received is larger than 1024 kilobytes (1 megabyte, or about 13,000 lines of unformatted text).

reject "Message ... recipient";

This is the action of the rule. The command 'reject' rejects a mail, with the text between the quotes. Notice the ';' at the end: it's necessary on any line that performs an action. Important: your script won't work if you forget it.

stop;

This action tells the Sieve engine to stop here, without checking for more rules. No further action is needed here, since the message has already been rejected, and we don't want to keep a copy of it. In some cases it's important to use a 'stop' explicitly, otherwise it will continue, find another action and perform that as well. This is one of the first things to check when mails are delivered to two of your folders.

}

A closing-curly-bracket to signify the end of the rule.

Writing your own script

There's a good sieve reference online which describes the components which make up a script. Additional information is given below.

Tests

- **size**
  - This test compares the the size of the mail currently being filtered with a number. If you do a test on this you can use MB, KB, or even Bytes. Sieve is relaxed on this, just feed it :over or :under, followed by a number, immediately followed by an indicator of what you mean (K, M or nothing, for kilobytes, megabytes or bytes respectively), and it works. The number is the size to test for (e.g., 10240 and 10K are both exactly 10 kilobytes). Use `:over` and `:under` to specify whether the message should be above or below the size for the test to return true. The number should not be put in quotes. You can see it in action in the previous example.
- **header**
  - This basically covers ALL the headers in the mail, and will be the field you're using the most.
- **address**
  - This lets you match on only an address field's email address, not its associated name. For example, I could send an email as "John Smith" jsmith@example.com. A header test such as `header :is "From" "jsmith@example.com"` would be false because that is not the entire header. But an address-based test of `address :is "From" "jsmith@example.com"` would be true because that is the entire address.
- **allof** (tests list)

    - This test lets you make a rule that applies only if all of several things are true. It takes a list of other tests to apply. For example, `allof (size :over 1024K, address :is "From" "jsmith@example.com")` would be true only if both the email is larger than a megabyte *and* it came from jsmith@example.com.
- **anyof** (tests list)
    - This test lets you make a rule that applies only if any of several things are true. `anyof (size :over 1024K, address :is "From" "jsmith@example.com")` would be true only if either the email is larger than a megabyte *or* it came from jsmith@example.com.
- **true** (no parameters)
    - The value of this test is always TRUE.
- **false** (no parameters)
    - The value of this test is always FALSE.
- **not** <test>
    - The value of a not test is TRUE when the parameter is FALSE, and vice-versa.

Comparators

A comparator is something you use to *compare* a field with a certain value. Keep in mind that a comparator always starts with a colon, the ':' character.

- **:is**
    - The value in the field must match the value you specified EXACTLY. Down to the letter precisely. No more letters, but certainly nothing less. This is the most precise comparator available, and tends to cause the least incorrect matches.
- **:matches**
    - The entire string must match the wildcard expression. `*` matches zero or more characters. `?` matches a single character. They may be escaped with a *pair* of backslashes: `\\?` matches a literal question mark. (The first backslash is to escape the second, to pass it through the string literal parser to the wildcard parser.)
- **:contains**
    - The value in the field must exactly contain the value you specify (wildcards are not allowed). For a field with the value 'thisismylife', any subsection of that will match

('sis', 'his','my','life', for example will all match). *Be very careful*; it's easy to create a rule that generously matches stuff you don't want matched.

- **:over**
  - This one works with fields containing numbers, like the size field. If the specified value is over the value in the header, it matches.
- **:under**
  - Same as above, only this time it should be below the value. Note that both :over and :under do NOT work with headers. Only :over and :under may be used with the size test. To match a numerical value exactly, you need to specify both :over and :under.
- **:count**
  - This allows you to count how many items there are in a certain field, for example To or Cc. Or both, using the language construct with the square brackets, as explained below. Here's some sample code: `address :count "lt" :comparator "i;ascii-numeric" ["to", "cc"] "5"`. This tests if the total number of To and CC fields is less than 5 (Note that this tests the fields in the message, not the addresses in those fields. Indeed, this test will always be true if the message is RFC 2822 compliant).

Language Constructs

A script has a certain structure. You need a way to indicate such a structure, so there are some building blocks for that too. These are called 'Language Constructs'.

- **[...]**
  - The square-brackets are used to indicate a group of items. For example ["joe","jane","jennifer","jack"] indicates the four people shown between [and]. Unlike the English language there is always a comma between the elements.
- **{ ... }**
  - The curly-brackets are used to indicate the set of actions that is to be performed if the test preceding it is TRUE.
- **if**
  - One of the most powerful tools in your arsenal. The IF rule basically forms the heart, soul and brain of the entire Sieve language. Without it you'd have a pretty

simple language that can do only one thing, and then stop. By using if you can actually say stuff like 'IF a matches b, then do c'.

- **elsif**
  - This is a contracted version of "else if". Else if means 'If the previous rule did not match, try this instead'. You can have a whole list of elsif's. It's pretty important that you put them in the right order; the one you want to match first goes first. Sieve will just keep on checking your elsif's until it finds one that works, or reaches the end of your script, or reaches an else.
- **else**
  - If your 'if' didn't match, and your elseif's didn't match, there's only one last thing for your script to try. The Else action. The Else action can be used to say things like 'If a = b then do w, if that doesn't work try if a = c and then do x, if that doesn't work try if a = d and then do y, and if that doesn't work, just put it in a folder 'Undecided' '. See the example script below.
- **stop**
  - Don't execute the rest of the script. Useful only inside curly brackets.This will stop the script continuing to look for more matches on this email, thereby preventing multiple actions from taking place on this email.

Actions

- **keep**
  - Keep a copy of the message in the default folder. Cancels discards.
- **fileinto "foldername"**
  - Store the message in this folder. If the folder doesn't exist, the message will be filed in the Inbox and no other action will be executed.
- **discard**
  - Deletes the message permanently without notifying the sender. This bypasses any FastMail Trash/delete/backup actions. Any mail deleted via discard is gone without possibility of resurrection.
- **reject "reason string"**
  - Return the message to the sender with an optional message.

- **redirect "email_address"**
  - Forward the message to that address and don't keep a copy (unless a keep or fileinto are also executed).

Comments

Comments cannot start inside strings.

- **Single-line comment**
  - Starts with a #, continues until the end of the line.
- **Multi-line comment**
  - Starts with /*, ends with */. Cannot be nested.

Example Script

```
/*
   Example last updated:
   2005-01-18
*/
require ["fileinto", "reject", "vacation", "regex", "relational", "comparator-i;ascii-numeric"];

if a :matches b {
  Do W; #an action
  stop;
}
elsif a :matches c {
  Do X;
  stop;
}
elsif a :matches d {
  Do Y;
  stop;
}
else {       # Nothing matches, put it into the Undecided folder and stop
  fileinto "INBOX.Undecided";
  stop;
}
```

Testing

There's two ways to test:

1. Save your sieve script changes and send yourself emails to test the various clauses. Inefficient particularly if you want to test matches against external senders, or against generated headers.

2. Use the sieve testing tool. Paste your full sieve script and a raw message (this allows you to modify the headers to suit) to test the matches and actions in your script.

Help & Support Technical Information **How to use Sieve**