



# **SIEVE PROGRAMMING GUIDE**

**MODUS VERSION 4.2**



Latest revision: May, 2005

This manual is Copyright © 1995-2005 Vircom, Inc. Reproduction or deletion, in whole or in part, of this document is strictly prohibited without prior written consent from Vircom, Inc.

VOP, Vircom Online Platform, VOP Mail, VOP modusMail, Modus, Modus, SCA, VOP Radius, Modus Gate, VOP Anti-Spam Gate, VOP Mail Web, Professional VOP Mail Web, VOP Migration Services, Modus Sieve are trademarks of Vircom, Inc. Norman Virus Control and NVC are trademarks of Norman Data Defense Inc. Windows, Windows NT, Windows 2000, IIS, Internet Information Server and Data Access Components are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Platypus, RODOPI, Emerald, EcoBuilder and Worldgroup are trademarks of their respective owners. All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations.

McAfee and NetScan are registered trademarks of Network Associates, Inc. and/or its affiliates in the US and/or other countries. ©2002 Networks Associates Technology, Inc. All Rights Reserved.

Modus is based on the Professional Internet Mail Services product licensed from the University of Edinburgh.

Certain algorithms used in parts of this software are derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

## TABLE OF CONTENTS

|  |           |
|--|-----------|
| <b>Sieve Script Overview</b> .....           | <b>3</b>  |
| Sieve Command Type.....                      | 6         |
| Sieve Commands.....                          | 8         |
| #.....                                       | 8         |
| /*.....                                      | 8         |
| :contains.....                               | 8         |
| :content_type.....                           | 9         |
| :is.....                                     | 10        |
| :matches.....                                | 10        |
| :regex.....                                  | 10        |
| :over.....                                   | 11        |
| :under.....                                  | 12        |
| address.....                                 | 12        |
| allof.....                                   | 13        |
| anyof.....                                   | 13        |
| attachment.....                              | 14        |
| body.....                                    | 14        |
| discard.....                                 | 14        |
| elsif.....                                   | 15        |
| else.....                                    | 15        |
| exists.....                                  | 16        |
| envelope.....                                | 16        |
| false.....                                   | 17        |
| fileinto.....                                | 17        |
| header.....                                  | 17        |
| if.....                                      | 18        |
| not.....                                     | 19        |
| redirect.....                                | 19        |
| reject.....                                  | 19        |
| require.....                                 | 20        |
| size.....                                    | 20        |
| stop.....                                    | 21        |
| true.....                                    | 21        |
| <b>SIEVE Variables Extension</b> .....       | <b>22</b> |
| Introduction.....                            | 22        |
| Interpretation of strings.....               | 22        |
| Quoting.....                                 | 22        |
| Numbered variables.....                      | 23        |
| Action set.....                              | 24        |
| Modifier.....                                | 24        |
| Test string.....                             | 25        |
| <b>SIEVE MIME Extension</b> .....            | <b>26</b> |
| Test mime.....                               | 26        |
| <b>SIEVE EDITHEADER Extension</b> .....      | <b>28</b> |
| Introduction.....                            | 28        |
| Action replaceheader.....                    | 28        |
| Action addheader.....                        | 29        |
| Action deleteheader.....                     | 29        |
| Interaction with Other Sieve Extensions..... | 30        |

## Sieve Script Overview

**Sieve is a scripting language for mail filtering. The proposed standard for the Sieve scripting language is specified in RFC 3028.**

The Modus implementation of Sieve varies slightly from the proposed standard. As such, Modus Sieve is not fully compliant with RFC 3028. This was required to adapt Sieve to anti-spam filtering for an entire mail system.

There are three major differences between RFC 3028 and Vircom's implementation of Sieve:

- The body extension feature was added, to allow to test against the raw body of the message, and the message without HTML tags.
- Modus Sieve supports the **regex** command.
- Commands such as **fileinto** have a different syntax to specify the physical folder where the mail will be stored instead of the user folder.
- The action command **vacation** is not implemented for now since it has no impact on spam filtering. For now, it is silently ignored, i.e., it is not processed as a command but does not generate an error either.
- Two new test commands were added: **body** and **attachment**.

### Sieve script structure

Since Sieve is a scripting language, it means commands are executed sequentially starting at the beginning of the script and interpreting line per line. Each line (i.e. segment of text separated by a newline character) represents one Sieve command. Note that CR or LF can both be used as valid newline characters.

The number of tabs and white spaces between commands or command parameters are irrelevant. Additionally, Sieve scripts are case-insensitive.

### Example

Here is a simple example of a Sieve script:

```
if size :over 100K {  
    discard;  
    stop;  
}
```

### Comments

Single-line and multi-line comments are allowed in Sieve scripting. The two types of comments are:

```
# single line comment

/* Comment on multiple lines
   at the same time. */
```

### *Variable declarations*

#### *Numbers*

Numbers can be specified in straightforward decimal format. However, for large values, the suffixes “K”, “M” and “G” can be appended to the number. These suffixes represent the power of 2 by which the number is multiplied:

| Suffix | Power of 2 | Value         |
|--------|------------|---------------|
| K      | 10         | 1,024         |
| M      | 20         | 1,048,576     |
| G      | 30         | 1,073,741,824 |

#### *Example*

$2K = 2 \times 1,024 = 2,048$

$5M = 5 \times 1,048,576 = 5,242,880$

#### *Strings*

A string starts and ends with a double quote (“”).

Escape codes can be entered by typing “\” followed by the one-character escape code. To enter a backspace in the actual text, the escape code “\ ” can be used.

For large amounts of text, a multi-line form is allowed. This can be done using the keyword **text:**, as follows:

```
text:
This is a multi-line
text entry, which can
be used for specifying
the body of a message.
.
```

The last dot signals the end of the multi-line text entry.

Note that brackets (“[ ]”) are not allowed in this entry mode.

#### *String lists*

When it is necessary to specify a list of strings, the list can be encapsulated in brackets (“[ ]”) as follows:

```
[“item-1”, “item-2”, ..., “item-n”]
```

Example:

```
header :contains [“To”, “Cc”]
```

Note that a one-string list is equivalent to a string. This means [*“item”*] is the same as *“item”*.

### *Errors*

To prevent compilation errors, Modus parses the entire script before executing it. This means the script can never execute an invalid set of actions.

In the case of a run-time error (the script gives an error even though the syntax was valid), the messages being filtered are simply kept.

## Sieve Command Type

**There are three different types of commands that can be used in Sieve scripting: control, action and test. Add to this are two comment commands, as well as five comparison commands.**

### *Control commands*

Control commands control the flow of the code. They are not commands per se, but they affect how the commands will be carried out.

```
else
elsif
if
require
stop
```

### *Action commands*

Action commands are direct commands that perform an action. They can be preceded by a control command, or on their own.

```
reject
fileinto
redirect
discard
```

### *Test commands*

Test commands are used in conjunction with if/elsif to specify the condition under which the following commands will be carried out.

```
address
allof
anyof
envelope
header
false
exists
not
size
true
body
attachment
```

### *Comment commands*

These commands mark a comment in the script that will be ignored by the interpreter.

```
#  
/*
```

### *Comparison commands*

These commands are used to compare variables between them, to be used with **if/elsif**.

```
:contains  
:is  
:matches  
:over  
:under  
:regex
```

### *Special commands*

This command is a special implementation, adding Body Purification to Sieve.

```
:content_type
```

### *Example*

|   |
|---|
| <pre>if      size      :over      500K      { discard;<br/>stop;}</pre> |
| <p><i>control test comparison value action</i></p>                      |

## Sieve Commands

**This section details the various commands that can be used when building Sieve scripts under Modus3. They are sorted alphabetically.**

### #

#### Type

Comment

#### Syntax

**# *comment***

#### Description

Marks the beginning of a one-line comment in the code. Modus ignores the rest of the line.

```
# This is a comment.
```

### /\*

#### Type

Comment

#### Syntax

**/\* *comment* \*/**

#### Description

Marks the beginning of a multi-line comment in the code. Modus ignores the text between “/\*” and “\*/”.

```
/*  
This is a multi-line comment.  
There is no limit to the number of lines you enter.  
*/
```

### :contains

#### Type

Comparison

#### Syntax

***value1* :contains *value2***

## Description

Evaluates to **true** if *value2* is a substring of *value1*; otherwise, returns **false**.

```
if header :contains ["from"] "coyote" {
    fileinto "INBOX.harassment";
}
```

## :content\_type

### Type

Special

### Syntax

```
if body :content_type <contentlist> <comparison>
<bodytransform> <string>
```

if body :content\_type ["text/plain", "text/html"] :contains :text ["foo", "bar"]

### Description

:content\_type is the Vircom implementation of the Body Transform extension to Sieve. It allows to purify the body of a message to compare against either the raw text of a message, or a stripped-down version without HTML tags. In order to use the Body Transform, you must specify the following parameters:

**contentlist:** Can be either "text/plain" or "text/html", to specify which part of the body message must be tested against. If you wish to test against both, you can use "text".

**bodytransform:** Can be either :raw or :text. If you use :raw, the raw (unpurified) body part will be used. If you use :text, the body message will be stripped of all HTML tags.

:content\_type will evaluate to **true** if the comparison is true; otherwise, returns **false**.

```
if body :content_type "body/html" :contains :text "spammer"
{
    discard; stop;
}
```

NOTE : For backward compatibility issues with Modus 1.x, the following syntax is also accepted:

```
if body :contains ["text/plain", "text/html"] ["foo",
"bar"] {
    discard; stop;
}
```

**:is****Type**

Comparison

**Syntax*****value1* :is *value2*****Description**Evaluates to **true** if *value1* is exactly *value2*; otherwise, returns **false**.

```
if header :is ["subject"] "make money fast" {  
    discard; stop;  
}
```

**:matches****Type**

Comparison

**Syntax*****value1* :matches *value2*****Description**Returns **true** if *value1* matches *value2*, **false** if they do not match. The **:matches** comparison command allows for the use of the “\*” and “?” wildcards. The “\*” wildcard matches 0 or more characters; the “?” wildcard returns 1 character.

Note that you can include a “\*” or “?” as a non-wildcard character by using a double escape code: “\\”. Thus, “\\\*” will match a single \* without employing a wildcard.

```
If header :matches ["subject"] "$$$$*" {  
    discard; stop;  
}
```

**:regex****Type**

Comparison

**Syntax*****value1* :regex *value2***

## Description

This performs a regexp comparison between *value1* and *value2*. If *value1* matches *value2*, the comparison returns **true**; otherwise, it returns **false**.

The **:regex** comparison command matches the following matches:

| Match single character      |   |
|-----------------------------|---|
| .                           | Match any single character, including newline   |
| ?                           | Match any or no single character  |
| *                           | Match any or no char string   |
| [ ]                         | Bracket expression. Match any one of the enclosed characters. A hyphen indicates a range of consecutive characters. |
| \\                          | Escape the following special character (match the literal character)  |
| Anchoring – match positions |   |
| ^                           | Match the beginning of the line or string   |
| \$                          | Match the end of the line or string   |

```
if header :regex ["subject"] "^make money$" {  
    discard; stop;  
}
```

## :over

### Type

Comparison

### Syntax

**size** **:over** *value*

### Description

This command can only be used with the **size** command. It allows for the testing of the message's size. If the message size is over *value*, it returns **true**; otherwise, it returns **false**.

Note that if the message is exactly of the specified size, it will return **false**.

```
if size :over 1M {  
    discard; stop;  
}
```

### See also

**size**

**:under****Type**

Comparison

**Syntax****size :under value****Description**

This command can only be used with the **size** command. It allows for the testing of the message's size. If the message size is under *value*, it returns **true**; otherwise, it returns **false**.

Note that if the message is exactly of the specified size, it will return **false**.

```
if size :under 4K {  
    discard; stop;  
}
```

**See also****size****address****Type**

Test

**Syntax****address** [*address-part*] [*comparator*] [*match-type*] <*header-list*> <*keylist*>**Description**

The **address** test matches Internet addresses in headers that contain addresses. It returns **true** if any header contains any key in the specified part of the address, as modified by the comparator and the match keyword.

This test returns true if any combination of the *header-list* and *keylist* arguments match.

[*address-part*] can be one of three things: **:localpart** (the user part of the email address), **:domain** (the domain name) or **:all** (the entire email address.) The **:all** option is the default if no [*address part*] is specified.

If [*comparison command*] is not specified, it defaults to **:matches**.

**address** never acts on the phrase part of an email address, nor on comments within that address. It also never acts on group names, although it does act on the addresses within the group construct.

In Modus3, the following headers can be tested with address: From, To, Cc, Bcc, Sender, Resent-From, Resent-To.

```
if address :all :is "from" "tim@example.com" {
    discard; stop;
}
```

## allof

### Type

Test

### Syntax

**allof** <testlist>

### Description

This command is used to test several test patterns, for instance, in an **if** argument. This test command returns **true** if all of the tests return true. Otherwise, it returns **false**.

```
if allof
    (header :contains "subject" "money",
     header :contains "from" "fool@example.edu")
    {
    discard; stop;
    }
```

## anyof

### Type

Test

### Syntax

**anyof** <testlist>

### Description

This command is used to test several test patterns, for instance, in an **if** argument. This test command returns **true** if any of the tests return true. Otherwise, it returns **false**.

```
if anyof
    (not exists ["From", "Date"],
     header :contains "from" "fool@example.edu")
    {
    discard; stop;
    }
```

## attachment

### Type

Test

### Syntax

```
attachment [comparison-command] [match-type] <keylist>
```

### Description

This test command returns **true** if any attachment display name matches the *keylist*. Otherwise, it returns **false**.

If [*comparison-command*] is not specified, it defaults to **:matches**.

```
if attachment :matches ["*.vbs", "*.exe"] {  
    fileinto "INBOX.suspicious";  
}
```

## body

### Type

Test

### Syntax

```
body [comparison-command] [match-type] <content-type>  
<keylist>
```

### Description

This test command evaluates to **true** if the body of any MIME part with the specified content-type matches any of the keys in the *keylist*.

If [*comparison-command*] is not specified, it defaults to **:matches**.

```
if body :matches ["text/plain"] ["*make*money*fast*"  
    "*university*dipl*mas*"] {  
    discard;  
    stop;  
}
```

## discard

### Type

Action

## Syntax

### discard

## Description

This action discards the message without any further action.

```
If header :contains ["from"] ["idiot@somewhere.com"] {
    discard; stop;
}
```

## elsif

## Type

Control

## Syntax

**elsif** *condition action*

## Description

This control command is used exclusively after the **if** control command. If the condition of the **if** control command evaluates to **false**, the test specified after the **elsif** is evaluated.

```
if size :over 500K {
    discard; stop;
}
elsif size :under 4K {
    stop;
}
```

See also

**else**, **if**

## else

## Type

Control

## Syntax

**else** *action*

## Description

This control command is used exclusively after the **if** control command. If the condition of the **if** control command evaluates to **false**, the action after the **else** command is performed.

```
if size :under 4K {
    stop;
}
else {
    discard; stop;
}
```

See also  
`elsif`, `if`

## exists

### Type

Test

### Syntax

`exists` *<header-names>*

### Description

This test command returns **true** if one of the headers specified in the *header-names* string list exists. All headers must exist, or the test will return **false**.

```
if not exists ["From", "Date"] {
    discard; stop;
}
```

## envelope

### Type

Test

### Syntax

`envelope` [*comparator*] [*address-part*] [*match-type*]  
*<envelope-part>* *<keylist>*

### Description

This test is **true** if the envelope fields found in the .RCP file associated with the message match the specified keys.

If *<envelope-part>* contains “from”, matching will occur against the “Return Path” of the SMTP envelope.

If *<envelope-part>* contains “to”, matching will occur against the “Recipient” part of the SMTP envelope.

*[address-part]* can be one of three things: **:localpart** (the user part of the email address), **:domain** (the domain name) or **:all** (the entire email address.) The **:all** option is the default if no *[address part]* is specified.

This test returns **true** if any combination of the *envelope-part* and *keylist* arguments match.

```
if envelope :all :is "from" "tim@example.com" {  
    discard; stop;  
}
```

## false

### Type

Test

### Syntax

**false**

### Description

This test always returns **false**.

## fileinto

### Type

Action

### Syntax

**fileinto** <*folder*>

### Description

This action files the message into a specified folder on the disk (it only works for local drives), where the original message and associated .RCP files were copied.

```
if header :contains ["from"] "coyote" {  
    fileinto "spool\\spam\\harassment";  
}
```

## header

### Type

Test

## Syntax

```
header [comparison-command] [match-type] <header-names>  
<keylist>
```

## Description

This test evaluated to **true** if any header name matches any entry in the keylist. If no comparison-command is specified, it defaults to **:is**.

This test returns **true** if any combination of the *header-names* and *keylist* arguments match.

```
if header :matches ["From"] "joe@somewhere.com" {  
    stop;  
}
```

## if

### Type

Control

### Syntax

```
if condition action
```

### Description

If the condition of this control command evaluates to **true**, the action specified after the command will be executed.

An **elsif** must follow an **if**, and an **else** must follow either an **if** or an **elsif**.

If the condition of the **if** control command returns **false**, the condition of the first **elsif** (if any) will be evaluated. If the **elsif** condition is **true**, the **elsif** block is executed. An **elsif** may be followed by an **elsif**, in which case, the interpreter repeats this process until there are no more **elsifs**.

When the interpreter runs out of **elsifs**, there may be an **else** control command. If there is, and neither the **if** nor any of the **elsifs** conditions evaluated to **true**, the **else** command will be executed.

```
if header :contains ["From"] "idiot@somewhere.com" {  
    discard; stop;  
}  
elsif size :under 4K {  
    stop;  
}
```

### See also

**if, else**

## not

### Type

Test

### Syntax

**not** <*test*>

### Description

This test takes a test value and sets it to the opposite. Thus, **not true** evaluates to **false**, and **not false** evaluates to **true**.

## redirect

### Type

Action

### Syntax

**redirect** <*address*>

### Description

This action is used to send the message to another user instead of the intended recipients of the message. The address on the SMTP envelope is replaced with the new address, and the message is sent out again.

```
redirect "spam-report@domain.com"
```

### Description

This action is similar to the one above, except that it keeps the original recipient(s) as well as sending the message to another user. The address on the SMTP envelope is modified to include the new address, and the message is sent out again.

```
redirect "spam-report@domain.com"; keep;
```

## reject

### Type

Action

### Syntax

**reject** <*reason*>

## Description

This action rejects a message and sends back a message to the sender containing `<reason>` in the body of the message.

```
if size :over 500K {
    reject
    "This message exceeds the allowed size limit
    imposed for this system. Please reduce the size
    of the message to under 500K and send it again."
;
}
```

## require

### Type

Control

### Syntax

```
require <capabilities>
```

### Description

This control command specifies that the script will use the specified extensions. The **require** command must be used before any other command is used in the script.

```
require ["fileinto", "reject"]
require "discard"
```

## size

### Type

Test

### Syntax

```
size <:over|:under> <limit>
```

### Description

This test allows for the verification of a message size. The limit is the number of bytes of the message. This is defined as the number of octets from the initial header until the last character in the message body.

If the argument is “**:over**”, and the size of the message is greater than `<limit>`, the test returns **true**.

If the argument is “**:under**”, and the size of the message is lower than `<limit>`, the test returns **true**.

```
if size :over 500K {  
    discard; stop;  
}
```

## stop

### Type

Action

### Syntax

`stop;`

### Description

This command tells the interpreter to stop processing the current script. In Modus3, the **stop** command also stops any further Sieve scripts from being processed for the current message.

For details on the execution of multiple Sieve scripts, see [Cascading Sieve scripts](#).

## true

### Type

Test

### Syntax

`true`

### Description

This test always returns **true**.

# SIEVE Variables Extension

## Introduction

See <http://www.ietf.org/internet-drafts/draft-homme-sieve-variables-04.txt> for more information.

This extension changes the interpolation of strings, adds an action to store data in variables, and supplies a new test so that the value of a string can be examined.

It adds support for storing and referencing data in string variables. The mechanisms detailed in this document will only apply to Sieve scripts that include a require clause for the "variables" extension. The require clauses themselves are not affected by this extension.

The capability string associated with the extension defined in this document is "variables".

## Interpretation of strings

This extension changes the semantics of quoted-string, multi-line-literal and multi-line-dotstuff found in [SIEVE] to enable the inclusion of the value of variables.

When a string is evaluated, substrings matching variable-ref SHALL be replaced by the value of variable-name. Only one pass through the string SHALL be done. Variable names are case insensitive, so "foo" and "FOO" refer to the same variable. Unknown variables are replaced by the empty string.

|               |   |                                      |
|---------------|---|--------------------------------------|
| variable-ref  | = | "\${" variable-name "}"              |
| variable-name | = | num-variable / *namespace identifier |
| namespace     | = | identifier "."                       |
| num-variable  | = | 1*DIGIT                              |

Examples:

"&%\${}!" => unchanged, as the empty string is an illegal identifier  
"\${doh!}" => unchanged, as "!" is illegal in identifiers

The variable company holds the value "ACME". No other variables are set.

"\${full}" => the empty string  
"\${company}" => "ACME"  
"\${President, \${Company} Inc.}" => "\${President, ACME Inc.}"

The expanded string MUST use the variable values which are current when control reaches the statement the string is part of.

## Quoting

The semantics of quoting using backslash are not changed: backslash quoting is resolved before doing variable substitution.

Examples:

"\${fo\o}" => \${foo} => the expansion of variable foo.

"\${fo}\o}" => \${fo}\o} => illegal identifier => left verbatim.  
"\\${foo}" => \${foo} => the expansion of variable foo.  
"\\\${foo}" => \\${foo} => a backslash character followed by the expansion of variable foo.

If it is required to include a character sequence such as "\${beep}" verbatim in a text literal, the user can define a variable to circumvent expansion to the empty string.

Example:

```
set "dollar" "$";  
set "text" "regarding ${dollar}{beep}";
```

## Numbered variables

The decimal value of the numbered variable name will index the list of matching strings from the most recently evaluated successful match of type ":matches" or ":regex". The list is empty if no match has been successful.

For ":matches", the list will contain one string for each wildcard ("?" and "\*") in the match pattern. Each string holds what the corresponding wildcard expands to, possibly the empty string. The wildcards expand greedily.

For ":regex", the list will contain the strings corresponding to the group operators. The groups are ordered by the position of the opening parenthesis, from left to right.

The first string in the list has index 1. If the index is out of range, the empty string will be substituted. Index 0 returns the number of strings in the list as a decimal number.

The interpreter does short-circuit tests, ie. not perform more tests than necessary to find the result. Evaluation order is left to right.

Numbered variables \${1} through \${9} are supported.

Example:

```
require [ "fileinto", "regex", "variables" ];  
  
if header :regex "List-ID" "<(*)@" {  
    fileinto "lists.${1}"; stop;  
}  
  
# this is equivalent to the above:  
if header :matches "List-ID" "*<*@*" {  
    fileinto "lists.${2}"; stop;  
}  
  
if address :matches [ "To", "Cc" ] "coyote@*.com" {  
    # ${0} is always "2", and ${2} is always the empty string.  
    fileinto "business.${1}"; stop;  
} else {  
    # control can't reach this block if any match was  
    # successful, so ${0} is always "0" here.  
    stop;
```

```

}

if anyof (true, address :domain :matches "To" "*.com") {
    # second test is never evaluated, so ${0} is still "0"
    stop;
}
    
```

## Action set

Syntax: `set [MODIFIER] [COMPARATOR] <name: string> <value: string>`

The "set" action stores the specified value in the variable identified by name. The name **MUST** be a constant string and conform to the syntax of identifier. An illegal name causes a syntax error.

The default comparator is "i;ascii-casemap". The comparator only affects the result when certain modifiers are used.

All variables have global scope: they are visible until processing stops. Variable names are case insensitive.

Example:

```

set "honorific" "Mr";
set "first_name" "Wile";
set "last_name" "Coyote";
set "vacation" text:
Dear ${HONORIFIC} ${last_name},
I'm out, please leave a message after the beep.
.
;
    
```

"set" does not affect the implicit keep.

## Modifier

Modifiers are applied on a value before it is stored in the variable. Modifier names are case insensitive. Unknown modifiers **MUST** yield a syntax error. More than one modifier can be specified, in which case they are applied according to this precedence list, highest value first:

| Precedence | Modifier                   |
|------------|----------------------------|
| 1          | :length                    |
| 2          | :lowerfirst<br>:upperfirst |
| 3          | :lower<br>:upper           |

If two or more modifiers of the same precedence are used, they can be applied in any order.

Examples:

```
set "a" "juMBIEd IETteRS";           => "juMBIEd IETteRS"
set :length "b" "${a}";              => "15"
set :lower "b" "${a}";               => "jumbled letters"
set :upperfirst "b" "${a}";          => "JuMBIEd IETteRS"
set :upperfirst :lower "b" "${a}";  => "Jumbled letters"
```

### 1.5.1 Modifier ":length"

The value is the decimal number of characters in the expansion, converted to a string.

### 1.5.2. Case modifiers

These modifiers change the letters of the text from upper to lower case or vice versa. The implementation supports US-ASCII, but not handles the entire Unicode repertoire.

### 1.5.3 Modifier ":upper"

All lower case letters are converted to their upper case counterpart.

### 1.5.4 Modifier ":lower"

All upper case letters are converted to their lower case counterpart.

### 1.5.5 Modifier ":upperfirst"

The first character of the string is converted to upper case if it is a letter and set in lower case. The rest of the string is left unchanged.

### 1.5.6 Modifier ":lowerfirst"

The first character of the string is converted to lower case if it is a letter and set in upper case. The rest of the string is left unchanged.

## Test string

Syntax: string [MATCH-TYPE] [COMPARATOR]  
<source: string-list> <key-list: string-list>

The "string" test evaluates to true if any of the source strings matches any key. The type of match defaults to "is".

The "relational" extension adds a match type called ":count". The count of a single string is 0 if it is the empty string, or 1 otherwise. The count of a string list is the sum of the counts of the member strings.

## SIEVE MIME Extension

If the "mime" test is used in a SIEVE script, the corresponding capability item **MUST** appear in the script's "require" statement.

### Test mime

```
Syntax:  mime [COMPARATOR] [MATCH-TYPE]
          <header-names: string-list>
          [<parameter-names: string-list>]
          <key-list: string-list>
```

The "mime" test evaluates to true if any MIME header name, or any parameter within a MIME header name, matches any key. The type of match is specified by the optional match argument, which defaults to "is" if not provided, as specified in section 2.6 of [\[RFC3028\]](#).

If no parameter-names argument is present, then the entire content of the headers specified in the header-names argument is evaluated. So if a message contained the MIME header:

```
Content-Type: image/jpeg; filename="example.jpg"
```

These tests on that MIME header evaluate as follows:

```
mime :contains ["Content-Type"] ["image"] => true
mime :contains ["Content-Type"] ["jpg"]   => true
mime :contains ["Content-Type"] ["filename"] => true
mime :contains ["Content-Type"] ["gif"]   => false
```

If the parameter-names argument is present, then only the content of matching MIME parameters in the specified headers are evaluated. So if a message contained the MIME header:

```
Content-Type: image/jpeg; filename="example.jpg"
```

These tests on that MIME header evaluate as follows:

```
mime :contains ["Content-Type"] ["filename"] ["image"] => false
mime :contains ["Content-Type"] ["filename"] ["jpg"]   => true
mime :contains ["Content-Type"] ["filename"] ["filename"] => false
mime :contains ["Content-Type"] ["filename"] ["gif"]   => false
```

---

```
mime :contains ["Content-Type"] ["name"] ["jpg"] => false
```

If the parameter-names argument is present, but contains a single empty string, then the content of specified MIME header, excluding any parameters, will be evaluated. So if a message contained the MIME header:

```
Content-Type: image/jpeg; filename="example.jpg"
```

These tests on that MIME header evaluate as follows:

```
mime :contains ["Content-Type"] [""] ["image"] => true
mime :contains ["Content-Type"] [""] ["jpg"] => false
mime :contains ["Content-Type"] [""] ["filename"] => false
mime :contains ["Content-Type"] [""] ["gif"] => false
```

Like the "header" test in [RFC3028](#), this test returns true if any combination of the header-names, parameters-names and key-list arguments match.

Like the "header" test in [RFC3028](#), if a MIME header listed in the header-names argument exists, it contains the null key (""). However, if the named header is not present, it does not contain the null key. So if a message contained the MIME header:

```
Content-Caffeine: C8H10N4O2
```

These tests on that header evaluate as follows:

```
mime :is ["Content-Caffeine"] [""] => false
mime :contains ["Content-Caffeine"] [""] => true
```

A similar behaviour applies to MIME parameters specified by the parameter-names argument

```
Content-Caffeine: C8H10N4O2; param1=example
```

These tests on that parameter evaluate as follows:

```
mime :is ["Content-Caffeine"] ["param1"] [""] => false
mime :contains ["Content-Caffeine"] ["param1"] [""] => true
```

# SIEVE EDITHEADER Extension

## Introduction

Email headers are a flexible and easy to understand means of communication between email processors. This extension enables sieve scripts to interact with other components that consume or produce header fields by allowing the script to delete, modify, and add header fields itself.

The capability string associated with extension defined in this document is "editheader".

## Action replaceheader

Syntax:

```
"replaceheader"  
    [":index" <fieldno: number> [":last" ]]  
    [":newname" <newname: string>]  
    [":newvalue" <newvalue: string>]  
    [COMPARATOR] [MATCH-TYPE] <field-name: string>  
    [<value-patterns: string-list>]
```

The `replaceheader` action replaces all or parts of a header field, in some or all occurrences of that header field.

If `:index <fieldno>` is specified, the attempts to match a value are limited to the named header field `<fieldno>` (beginning at 1, the first named header field). If `:last` is specified, the count is backwards; 1 denotes the last named header field, 2 the second to last, and so on. As with "deleteheader", the counting happens before the `<value-patterns>` match, if any.

The header field names in the `field-name` and in the `":newname"` argument, if specified, MUST be valid 7-bit header field names as described by the RFC 2822 "field-name" nonterminal.

The `field-name` is mandatory and always matched as a case-insensitive us-ascii string. The `value-patterns`, if specified, are matched according to the match type and comparator. (If no `value-patterns` are specified, they always match.)

If `:newname <newname>` is specified, the action changes the name of all matching header fields to `<newname>`.

If `:newvalue <newvalue>` is specified, the action changes the value of all matching header fields to `<newvalue>`.

The `MATCH-TYPE` is a type that interacts with the [VARIABLES] extension, variable references to  `${1}...${N}` in the replacement string following `":newvalue"` are evaluated according to the matched substring in the header field that is being replaced.

For example,

```
replaceheader :newvalue "[ADV] ${1}" :matches "Subject" "*";
```

would insert an "[ADV]" tag before the value of the "Subject" header field, changing

Subject: Make Money Fast!!  
to Subject: [ADV] Make Money Fast!!

If neither `:newname` nor `:newvalue` are specified, the operation still matches its argument and, in the presence of the [VARIABLES] extension and a match type that interacts with it, sets the `$$1...$$N` variables to the last header matched.

## Action addheader

Syntax:

```
"addheader" [":last"] <name: string> <value: string>
```

The `addheader` action adds a header field to the existing message header. The name **MUST** be a valid 7-bit US-ASCII header field name as described by [RFC-2822] "field-name" nonterminal.

If the specified field value does not match the RFC 2822 "unstructured" nonterminal or exceeds a length limit set by the implementation, the implementation **MUST** either flag an error or encode the field using folding white space and the encodings described in RFC 2047 or RFC 2231 to be compliant with RFC 2822.

An implementation **MAY** impose a length limit onto the size of the encoded header field; such a limit **MUST NOT** be less than 998 characters, not including the terminating CRLF supplied by the implementation.

By default, the header field is inserted at the beginning of the existing header. If the optional flag `:last` is specified, it is appended at the end.

Example:

```
/* Don't redirect if we already redirected */  
if not header :contains "X-Sieve-Filtered"  
    ["<kim@job.tld>", "<kim@home.tld>"]  
{  
    addheader "X-Sieve-Filtered" "<kim@job.tld>";  
    redirect "kim@home.tld";  
}
```

## Action deleteheader

Syntax:

```
"deleteheader"  
    [":index" <fieldno: number> [":last"]]  
    [COMPARATOR] [MATCH-TYPE]  
    <field-name: string>  
    [<value-patterns: string-list>]
```

By default, the `deleteheader` action deletes all occurrences of the named header field.

The `field-name` is mandatory and always matched as a case-insensitive us-ascii string. The `value-patterns`, if specified, are matched according to the match type and comparator. If none are specified, all values match.

The field-name MUST be a valid 7-bit header field name as described by the [RFC-2822] "field-name" nonterminal.

If `:index <fieldno>` is specified, the attempts to match a value are limited to the header field `<fieldno>` (beginning at 1, the first named header field). If `:last` is specified, the count is backwards; 1 denotes the last named header field, 2 the second to last, and so on. The counting happens before the `<value-patterns>` match, if any;

```
deleteheader :index 2 :contains "Received" "via carrier-pidgeon"
```

deletes the second "Received:" header field if it contains the string "via carrier-pidgeon" (not the second Received: field that contains "via carrier-pidgeon").

## Interaction with Other Sieve Extensions

Tests and actions such as "exist" or "header" that examine header fields MUST examine the current state of a header as modified by any actions that have taken place so far.

As an example, the "header" test in the following fragment will always evaluate to true, regardless of whether the incoming message contained an "X-Hello" header field or not:

```
addheader "X-Hello" "World";
if header :contains "X-Hello" "World"
{
    fileinto "international";
}
```

Actions that create messages in storage or in transport to MTAs MUST store and send messages with the current set of header fields.

For the purpose of weeding out duplicates, a message modified by addheader or deleteheader MUST be considered the same as the original message. For example, in an implementation that obeys the constraint in [SIEVE] section 2.10.3 and does not deliver the same message to a folder more than once, the following code fragment

```
keep;
addheader "X-Flavor" "vanilla";
keep;
```

MUST only file one message. It is up to the implementation to pick which of the redundant "fileinto" or "keep" actions is executed, and which ones are ignored.

The "implicit keep" is thought to be executed at the end of the script, after the headers have been modified. (However, a canceled "implicit keep" remains canceled.)