Abstract

This document describes the "regex" extension to the Sieve email filtering language. In some cases, it is desirable to have a string matching mechanism which is more powerful than a simple exact match, a substring match or a glob-style wildcard match. The regular expression matching mechanism defined in this draft provides users with much more powerful string matching capabilities.

Change History (to be removed prior to publication as an RFC)

Changes from draft-murchison-sieve-regex-08:
- Updated to XML source.
- Documented interaction with variables.

Changes from draft-ietf-sieve-regex-00:
- Various cleanup and updates.
- Added trial text specifying comparator interactions.

Open Issues (to be removed prior to publication as an RFC)

- The major open issue with this draft is what to do, if anything, about localization/internationalization. Are [IEEE.1003-2.1992] collating sequences and character equivalents sufficient? Should we reference the Unicode technical specification? Should we punt and publish the document as experimental?
- Is the current approach to comparator integration the right one to use?
- Should we allow shorthands such as \b (word boundary) and \w (word character)?
1. Introduction

Sieve [RFC5228] is a language for filtering email messages at or around the time of final delivery. It is designed to be implementable on either a mail client or mail server.
The Sieve base specification defines so-called match types for tests: is, contains, and matches. An "is" test requires an exact match, a "contains" test provides a substring match, and "matches" provides glob-style wildcards. This document describes an extension to the Sieve language that provides a new match type for regular expression comparisons.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The terms used to describe the various components of the Sieve language are taken from Section 1.1 of [RFC5228].

3. Capability Identifier

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

4. Regex Match Type

When the regex extension is available, commands that support matching may take the optional tagged argument ":regex" to specify that a regular expression match should be performed. The ":regex" match type is subject to the same rules and restrictions as the standard match types defined in [RFC5228].

The "MATCH-TYPE" syntax element defined in [RFC5228] is augmented here as follows:

MATCH-TYPE =/ ":regex"

5. Interaction with Sieve comparators

In order to provide for matches between character sets and case insensitivity, Sieve uses the comparators defined in the Internet Application Protocol Collation Registry [RFC5228]. The comparator used by a given test is specified by the :comparator argument.

The interaction between collators and the match types defined in the Sieve base specification is straightforward. However, the nature of regular expressions does not lend itself to this usage for the :regex
match type.

A component of the definition of many collators is a normalization operation. For example, the "i;octet" comparator employs an identity normalization; whereas the "i;ascii-casema" normalizes all lower case ASCII characters to upper case.

The :regex match type only uses the normalization component of the associated comparator. This normalization operation is applied to the key-list argument to the test; the result of that normalization becomes the target of the regular expression comparison. The comparator has no effect on the regular expression pattern or the underlying comparison operation.

It is an error to specify a comparator that has no associated normalization operation in conjunction with a :regex match type.

6. Regular expression comparisons

Implementations MUST support extended regular expressions (EREs) as defined by [IEEE.1003-2.1992]. Any regular expression not defined by [IEEE.1003-2.1992], as well as [IEEE.1003-2.1992] basic regular expressions, word boundaries and backreferences are not supported by this extension. Implementations SHOULD reject regular expressions that are unsupported by this specification as a syntax error.

The following tables provide a brief summary of the regular expressions that MUST be supported. This table is presented here only as a guideline. [IEEE.1003-2.1992] should be used as the definitive reference.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Match any single character except newline.</td>
</tr>
<tr>
<td>[ ]</td>
<td>Bracket expression. Match any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters.</td>
</tr>
<tr>
<td>[^ ]</td>
<td>Negated bracket expression. Match any one character NOT in the enclosed list. A hyphen (-) indicates a range of consecutive characters.</td>
</tr>
<tr>
<td>\</td>
<td>Escape the following special character (match the literal character). Undefined for other characters.</td>
</tr>
<tr>
<td>NOTE: Unlike [IEEE.1003-2.1992], a double-backslash is required as per section 2.4.2 of [RFC5228].</td>
<td></td>
</tr>
</tbody>
</table>
Table 1: Items to match a single character

<table>
<thead>
<tr>
<th>Expression</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>[: :]</td>
<td>Character class (alnum, alpha, blank, cntrl, digit, graph, lower, print, punct, space, upper, xdigit).</td>
</tr>
<tr>
<td>[= =]</td>
<td>Character equivalents.</td>
</tr>
<tr>
<td>[. .]</td>
<td>Collating sequence.</td>
</tr>
</tbody>
</table>

Table 2: Items to be used within a bracket expression (localization)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>Match zero or one instances.</td>
</tr>
<tr>
<td>*</td>
<td>Match zero or more instances.</td>
</tr>
<tr>
<td>+</td>
<td>Match one or more instances.</td>
</tr>
<tr>
<td>{n,m}</td>
<td>Match any number of instances between n and m (inclusive). {n} matches exactly n instances. {n,} matches n or more instances.</td>
</tr>
</tbody>
</table>

Table 3: Quantifiers - Items to count the preceding regular expression

<table>
<thead>
<tr>
<th>Expression</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>^</td>
<td>Match the beginning of the line or string.</td>
</tr>
<tr>
<td>$</td>
<td>Match the end of the line or string.</td>
</tr>
</tbody>
</table>

Table 4: Anchoring - Items to match positions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alternation. Match either of the separated regular expressions.</td>
</tr>
<tr>
<td>( )</td>
<td>Group the enclosed regular expression(s).</td>
</tr>
</tbody>
</table>

Table 5: Other constructs
7. Interaction with Sieve Variables

This extension is compatible with, and may be used in conjunction with the Sieve Variables extension [RFC5229].

7.1. Match variables

A sieve interpreter which supports both "regex" and "variables", MUST set "match variables" (as defined by [RFC5229] section 3.2) whenever the ":regex" match type is used. The list of match variables will contain the strings corresponding to the group operators in the regular expression. The groups are ordered by the position of the opening parenthesis, from left to right. Note that in regular expressions, expansions match as much as possible (greedy matching).

Example:

require ["fileinto", "regex", "variables"];

if header :regex "List-ID" "<(.*)@" {
    fileinto "lists.${1}"; stop;
}

# Imagine the header
# Subject: [acme-users] [fwd] version 1.0 is out
if header :regex "Subject" "^\[(.*)\] (.*)$" {
    # ${1} will hold "acme-users] [fwd"
    stop;
}

7.2. Set modifier :quoteregex

A sieve interpreter which supports both "regex" and "variables", MUST support the optional tagged argument ":quoteregex" for use with the "set" action. The ":quoteregex" modifier is subject to the same rules and restrictions as the standard modifiers defined in [RFC5229] section 4.

For convenience, the "MODIFIER" syntax element defined in [RFC5229] is augmented here as follows:

MODIFIER =/ ":quoteregex"

This modifier adds the necessary quoting to ensure that the expanded text will only match a literal occurrence if used as a parameter to :regex. Every character with special meaning (".", "*", "?", etc.) is prefixed with "\" in the expansion. This modifier has a precedence value of 20 when used with other modifiers.
8. Examples

Example:

require "regex";

# Try to catch unsolicited email.
if anyof (
    # if a message is not to me (with optional +detail),
    not address :regex ["to", "cc", "bcc"]
    "me(\\.+\.)?@company\\.com",

    # or the subject is all uppercase (no lowercase)
    header :regex :comparator "i;octet" "subject"
    "^[^[:lower:]]+$" ) {
    discard;    # junk it
}

9. IANA Considerations

The following template specifies the IANA registration of the "regex"
Sieve extension specified in this document:

To: iana@iana.org
Subject: Registration of new Sieve extension

Capability name: regex
Capability keyword: regex
Capability arguments: N/A
Standards Track/IESG-approved experimental RFC number: this RFC
Person and email address to contact for further information:
Kenneth Murchison
E-Mail: murch@andrew.cmu.edu

This information should be added to the list of Sieve extensions
given on http://www.iana.org/assignments/sieve-extensions.

10. Security Considerations

General Sieve security considerations are discussed in [RFC5228].
All of the issues described there also apply to regular expression matching.

It is easy to construct problematic regular expressions that are
computationally infeasible to evaluate. Execution of a Sieve that employs a potentially problematic regular expression, such as "(.*)*", may cause problems ranging from degradation of performance to and outright denial of service. Moreover, determining the computational complexity associated with evaluating a given regular expression is in general an intractable problem.

For this reason, all implementations MUST take appropriate steps to limit the impact of runaway regular expression evaluation. Implementations MAY restrict the regular expressions users are allowed to specify. Implementations that do not impose such restrictions SHOULD provide a means to abort evaluation of tests using the :regex match type if the operation is taking too long.

11. Normative References


Appendix A. Acknowledgments

Most of the text documenting the interaction with Sieve variables was taken from an early draft of Kjetil Homme’s Sieve variables specification.

Thanks to Tim Showalter, Alexey Melnikov, Tony Hansen, Phil Pennock, and Jutta Degener for their help with this document.
Authors’ Addresses

Kenneth Murchison  
Carnegie Mellon University  
5000 Forbes Avenue  
Cyert Hall 285  
Pittsburgh, PA  15213  
US

Phone: +1 412 268 2638  
Email: murch@andrew.cmu.edu

Ned Freed  
Oracle Corporation  
800 Royal Oaks  
Monrovia, CA  91016-6347  
USA

Phone: +1 909 457 4293  
Email: ned.freed@mrochek.com