Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 3, 2012

C. Daboo
A. Stone

January 31, 2012

Sieve Email Filtering: Include Extension draft-ietf-sieve-include-15

Abstract

The Sieve Email Filtering "include" extension permits users to include one Sieve script inside another. This can make managing large scripts or multiple sets of scripts much easier, and allows a site and its users to build up libraries of scripts. Users are able to include their own personal scripts or site-wide scripts.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 3, 2012.

Copyright Notice

Table of Contents

1. Introduction and Overview

It's convenient to be able to break SIEVE [RFC5228] scripts down into smaller components which can be reused in a variety of different circumstances. For example, users may want to have a default script and a special 'vacation' script, the latter being activated when the user goes on vacation. In that case the default actions should continue to be run, but a vacation command should be executed first. One option is to edit the default script to add or remove the vacation command as needed. Another is to have a vacation script that simply has a vacation command and then includes the default script.

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].

Conventions for notations are as in SIEVE [RFC5228] Section 1.1.

The following key phrases are used to describe scripts and script execution:

script
: a valid Sieve script.

script execution
: an instance of a Sieve interpreter invoked for a given message delivery, starting with the user's active script and continuing through any included scripts until the final disposition of the message (e.g. delivered, forwarded, discarded, rejected, etc.).

immediate script
: the individual Sieve script file being executed.

including script
: the individual Sieve script file that had an include statement which included the immediate script.

3. Include Extension

3.1. General Considerations

Sieve implementations that implement the "include", "return", and "global" commands described below have an identifier of "include" for

use with the capability mechanism. If any of the "include", "return", or "global" commands are used in a script, the "include" capability MUST be listed in the "require" statement in that script.

Sieve implementations need to track the use of actions in included scripts so that implicit "keep" behavior can be properly determined based on whether any actions have executed in any script.

Sieve implementations are allowed to limit the total number of nested included scripts, but MUST provide for a total of at least three levels of nested scripts including the top-level script. An error MUST be generated either when the script is uploaded to the Sieve repository, or when the script is executed, if any nesting limit is exceeded. If such an error is detected whilst processing a Sieve script, an implicit "keep" action MUST be executed to prevent loss of any messages.

Sieve implementations MUST NOT allow recursive script inclusion. Both direct recursion, where script A includes script A (itself), and indirect recursion, where script A includes script B which includes script A once again, are prohibited.

Sieve implementations MUST generate an error at execution time if an included script is a recursive inclusion. Implementations MUST NOT generate errors for recursive includes at upload time, as this would force an upload ordering requirement upon script authors and generators.

Sieve implementations MUST generate an error at execution time if an included script does not exist, except when the ":optional" parameter is specified. Implementations MUST NOT generate errors for scripts missing at upload time, as this would force an upload ordering requirement upon script authors and generators.

If the Sieve "variables" extension [RFC5229] is present, an issue arises with the "scope" of variables defined in scripts that may include each other. For example, if a script defines the variable "${status}" with one particular meaning or usage, and another defines "${status}" with a different meaning, then if one script includes the other there is an issue as to which "${status}" is being referenced. To solve this problem, Sieve implementations MUST follow the scoping rules defined in Section 3.4 and support the "global" command defined there.

3.2. Control Structure include

Usage: include [LOCATION] [":once"] [":optional"] <value: string>

LOCATION = ":personal" / ":global"

The "include" command takes an optional "location" parameter, an optional ":once" parameter, an optional ":optional" parameter, and a single string argument representing the name of the script to include for processing at that point. Implementations MUST restrict script names according to MANAGESIEVE [RFC5804], Section 1.6. The script name argument MUST be a constant string as defined in VARIABLES [RFC5229], Section 3; implementations MUST NOT expand variables in the script name argument.

The "location" parameter MUST default to ":personal" if not specified. The "location" parameter MUST NOT be specified more than once. The "location" has the following meanings:

:personal
Indicates that the named script is stored in the user's own personal (private) Sieve repository.

:global
Indicates that the named script is stored in a site-wide Sieve repository, accessible to all users of the Sieve system.

The ":once" parameter tells the interpreter only to include the named script if it has not already been included at any other point during script execution. If the script has already been included, processing continues immediately following the include command. Implementations MUST NOT generate an error if an "include :once" command names a script whose inclusion would be recursive; in this case, the script MUST be considered previously included and therefore "include :once" will not include it again.

Note: It is RECOMMENDED that script authors and generators use the ":once" parameter only when including a script that performs general duties such as declaring global variables and making sanity checks of the environment.

The ":optional" parameter indicates that the script may be missing. Ordinarily, an implementation MUST generate an error during execution if an include command specifies a script that does not exist. When ":optional" is specified, implementations MUST NOT generate an error for a missing script, and MUST continue as if the include command had not been present.

The included script MUST be a valid Sieve script. Implementations MUST validate that each script has its own "require" statements for all optional capabilities used by that script. The scope of a "require" statement is the script in which it immediately appears, and neither inherits nor passes on capabilities to other scripts during the course of execution.

A "stop" command in an included script MUST stop all script processing, including the processing of the scripts that include the immediate one. The "return" command (described below) stops processing of the immediate script only, and allows the scripts that include it to continue.

The "include" command MAY appear anywhere in a script where a control structure is legal, and MAY be used within another control structure, e.g., an "if" block.

Examples:

The user has four scripts stored in their personal repository:

"default"

This is the default active script that includes several others.

```
require ["include"];

include :personal "always_allow";
include :global "spam_tests";
include :personal "spam_tests";
include :personal "mailing_lists";
```

Personal script "always_allow"

This script special-cases some correspondent email addresses and makes sure any message containing those addresses are always kept.

```
if address :is "from" "boss@example.com"
{
    keep;
}
elsif address :is "from" "ceo@example.com"
{
    keep;
}
```

Personal script "spam_tests" (uses "reject" [RFC5429])

This script does some user-specific spam tests to catch spam messages not caught by the site-wide spam tests.

```
require ["reject"];

if header :contains "Subject" "XXXX"
{
    reject "Subject XXXX is unacceptable.";
}
elsif address :is "from" "money@example.com"
{
    reject "Mail from this sender is unwelcome.";
}
```

Personal script "mailing_lists"

This script looks for messages from different mailing lists and files each into a mailbox specific to the mailing list.

```
require ["fileinto"];

if header :is "List-ID" "sieve.ietf.org"
{
    fileinto "lists.sieve";
}
elsif header :is "List-ID" "ietf-imapext.imc.org"
{
    fileinto "lists.imapext";
}
```

There is one script stored in the global repository:

Site script "spam_tests" (uses "reject" [RFC5429])

This script does some site-wide spam tests which any user at the site can include in their own scripts at a suitable point. The script content is kept up to date by the site administrator.

```
require ["reject"];

if anyof (header :contains "Subject" "$$",
          header :contains "Subject" "Make money")
{
    reject "No thank you.";
}
```

3.3. Control Structure return

Usage: return

The "return" command stops processing of the immediately included script only and returns processing control to the script which includes it. If used in the main script (i.e., not in an included script), it has the same effect as the "stop" command, including the appropriate "keep" action if no other actions have been executed up to that point.

3.4. Interaction with Variables Extension

In order to avoid problems of variables in an included script "overwriting" those from the script that includes it, this specification requires that all variables defined in a script MUST be kept "private" to the immediate script by default - that is, they are not "visible" to other scripts. This ensures that two script authors cannot inadvertently cause problems by choosing the same name for a variable.

However, sometimes there is a need to make a variable defined in one script available to others. This specification defines the new command "global" to declare that a variable is shared among scripts. Effectively, two namespaces are defined: one local to the immediate script, and another shared among all scripts. Implementations MUST allow a non-global variable to have the same name as a global variable but have no interaction between them.

3.4.1. Control Structure global

Usage: global <value: string-list>

The "global" command accepts a string list argument that defines one or more names of variables to be stored in the global variable space. Each name MUST be a constant string and conform to the syntax of variable-name as defined in VARIABLES [RFC5229], Section 3. Match variables cannot be specified and namespace prefixes are not allowed. An invalid name MUST be detected as a syntax error.

The "global" command is only available when the script has both "include" and "variables" in its require line. If the "global" command appears when only "include" or only "variables" has been required, an error MUST be generated when the script is uploaded.

If a "global" command is given the name of a variable that has previously been defined in the immediate script with "set", an error MUST be generated either when the script is uploaded or at execution time.

If a "global" command lists a variable that has not been defined in the global namespace, the name of the variable is now marked as global, and any subsequent "set" command will set the value of the variable in global scope.

A variable has global scope in all scripts that have declared it with the "global" command. If a script uses that variable name without declaring it global, the name specifies a separate, non-global variable within that script.

Interpretation of a string containing a variable marked as global, but without any value set, SHALL behave as any other access to an unknown variable, as specified in VARIABLES [RFC5229], Section 3 (i.e., evaluates to an empty string).

Example:

The active script

> The included script may contain repetitive code that is effectively a subroutine that can be factored out. In this script, the test which matches last will leave its value in the test_mailbox variable and the top-level script will file the message into that mailbox. If no tests matched, the message will be implicitly kept in the INBOX.

```
require ["fileinto", "include", "variables", "relational"];
global "test";
global "test_mailbox";

set "test" "$$";
include "subject_tests";

set "test" "Make money";
include "subject_tests";

if string :count "eq" "${test_mailbox}" "1"
{
    fileinto "${test_mailbox}";
    stop;
}
```

Personal script "subject_tests"

```
This script performs a number of tests against the message, sets
the global test_mailbox variable with a folder to file the message
into, then falls back to the top-level script.


require ["include", "variables"];
global ["test", "test_mailbox"];

if header :contains "Subject" "${test}"
{
    set "test_mailbox" "spam-${test}";
}
```

3.4.2. Variables Namespace global

In addition to the "global" command, this document defines the variables namespace "global", as specified in VARIABLES [RFC5229], Section 3. The global namespace has no sub-namespaces (e.g., 'set "global.data.from" "me@example.com";' is not allowed). The variable-name part MUST be a valid identifier (e.g., 'set "global.12" "value";' is not valid because "12" is not a valid identifier).

Note that VARIABLES [RFC5229], Section 3, suggests that extensions should define a namespace that is the same as its capability string (in this case, "include" rather than "global"). Nevertheless, references to the "global" namespace without a prior require statement for the "include" extension MUST cause an error.

Example:

```
require ["variables", "include"];

set "global.i_am_on_vacation" "1";
```

Variables declared global and variables accessed via the global namespace MUST each be one and the same. In the following example script, we see the variable "i_am_on_vacation" used in a "global" command, and again with the "global." namespace. Consider these as two syntaxes with identical meaning.

Example:

```
require ["variables", "include", "vacation"];
global "i_am_on_vacation";

set "global.i_am_on_vacation" "1";

if string :is "${i_am_on_vacation}" "1"
{
    vacation "It's true, I am on vacation.";
}
```

3.5. Interaction with Other Extensions

When "include" is used with the Editheader extension [RFC5293], any changes made to headers in a script MUST be propagated both to and from included scripts. By way of example, if a script deletes one header and add another, then includes a second script, the included script MUST NOT see the removed header, and MUST see the added header. Likewise, if the included script adds or removes a header, upon returning to the including script, subsequent actions MUST see the added headers and MUST NOT see the removed headers.

When "include" is used with the MIME extension [RFC5703] "foreverypart" control structure, the included script MUST be presented with the current MIME part as though it were the entire message. A script SHALL NOT have any special control over the control structure it was included from. In the MIME example once again, a "stop" or "return" in an included script cannot directly terminate or continue flow of a "foreverypart" block. In such a case, the included script should set a global variable that the including script can test. A "stop" in an included script, even within a "foreverpart" loop, still halts all script execution, per Section 3.2.

When "include" is used with the Reject extension [RFC5429], calling "reject" or "ereject" at any time sets the reject action on the message, and continues script execution. Apropos of the MIME

extension, if an included script sees only a portion of the message and calls a reject, it is the entire message and not the single MIME part which carries the rejection.

4. Security Considerations

Sieve implementations MUST ensure adequate security for the global script repository to prevent unauthorized changes to global scripts. For example, a site policy might enable only certain users with administrative privileges to modify the global scripts. Site are advised against allowing all users to have write access to the site's global scripts.

Sieve implementations MUST ensure that script names are checked for validity and proper permissions prior to inclusion, in order to prevent a malicious user from gaining access to files accessible to the mail server software that should not be accessible to the user.

Sieve implementations MUST ensure that script names are safe for use with their storage system. An error MUST be generated either when the script is uploaded or at execution time for a script including a name that could be used as a vector to attack the storage system. By way of example, the following include commands should be considered hostile: 'include "./../..//etc/passwd"', 'include "foo$(`rm star`)"'.

Beyond these, the "include" extension does not raise any security considerations that are not present in the base SIEVE [RFC5228] document and the VARIABLES [RFC5229] extension.

5. IANA Considerations

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

5.1. "include" Extension Registration

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

Capability name: include
Description:     adds the "include" command to execute other Sieve
                 scripts, the "return" action from an included
                 script, and the "global" command and "global"
                 variables namespace to access variables shared
                 among included scripts.
RFC number:      this RFC
Contact address: the Sieve discussion list <sieve@ietf.org>
```

This information has been added to the IANA registry of Sieve Extensions (currently found at http://www.iana.org).

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5228] Guenther, P. and T. Showalter, "Sieve: An Email Filtering Language", RFC 5228, January 2008.

[RFC5229] Homme, K., "Sieve Email Filtering: Variables Extension", RFC 5229, January 2008.

[RFC5804] Melnikov, A. and T. Martin, "A Protocol for Remotely Managing Sieve Scripts", RFC 5804, July 2010.

6.2. Informative References

[RFC5293] Degener, J. and P. Guenther, "Sieve Email Filtering: Editheader Extension", RFC 5293, August 2008.

[RFC5429] Stone, A., "Sieve Email Filtering: Reject and Extended Reject Extensions", RFC 5429, March 2009.

[RFC5703] Hansen, T. and C. Daboo, "Sieve Email Filtering: MIME Part Tests, Iteration, Extraction, Replacement, and Enclosure", RFC 5703, October 2009.

Appendix A. Acknowledgments

Thanks to Ken Murchison, Rob Siemborski, Alexey Melnikov, Marc Mutz, Kjetil Torgrim Homme, Stephan Bosch, Arnt Gulbrandsen, Barry Leiba, and Jeffrey Hutzelman for comments and corrections.

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

Changes from ietf-14 to ietf-15:

a. Interaction with reject, stop.

Changes from ietf-13 to ietf-14:

a. Updated for Last Call comments.

Changes from ietf-12 to ietf-13:

a. Nits from Stephan Bosch.

b. Nits from Robert Burrell Donkin.

Changes from ietf-11 to ietf-12:

a. Nits from Alexey Melnikov.

b. Nits from Barry Leiba.

Changes from ietf-10 to ietf-11:

a. Nits from Dilyan Palauzov.

b. Nits from Stephan Bosch.

c. Nits from Alexey Melnikov.

Changes from ietf-09 to ietf-10:

a. Another example script error caught by Stephan Bosch.

b. Add :optional argument to allow a missing script to be ignored.

Changes from ietf-08 to ietf-09:

a. Better variables language from Stephan Bosch.

Changes from ietf-07 to ietf-08:

a. Nits from Stephan Bosch.

b. Nits from Barry Leiba.

c. Wordsmithing and layout wrangling.

Changes from ietf-06 to ietf-07:

a. Nits from Stephan Bosch.

Changes from ietf-05 to ietf-06:

a. Nits from Barry Leiba.

Changes from ietf-04 to ietf-05:

a. Integrate review from Barry Leiba.

Changes from ietf-03 to ietf-04:

a. No changes.

Changes from ietf-02 to ietf-03:

a. Setting a variable then calling global on it is an error (something like 'use strict').

b. Specify that the 'global' keyword is only available when 'variables' has also been required.

c. Uploading a script that includes a nonexistent script is not an error at upload time.

Changes from ietf-01 to ietf-02:

a. Require that script names must be constant strings, not subject to variable expansion.

b. Try the phrase immediate script instead of current script.

c. Clarify that "global 'varname'" and "global.varname" refer to the same variable.

d. Drop the requirement the global keywords come after require and before anything else.

Changes from ietf-00 to ietf-01:

a. Replaced import/export with global.

b. Added :once modifier to include.

c. Added global namespace to see if it holds water.

Changes from daboo-06 to ietf-00:

a. None

Changes from -05 to -06:

a. Aaron Stone joins as author.

b. Removed | characters from the script examples.

c. Updated draft references to published RFCs.

Changes from -04 to -05:

a. Fixed examples.

b. Relaxed requirement that imported/exported variables be set before being used.

Changes from -03 to -04:

a. Fixed missing 2119 definitions.

b. Defined interaction with variables through use of import and export commands.

Changes from -02 to -03:

a. Refreshing expired draft (updated for nits).

b. Syntax -> Usage.

c. Updated to 3028bis reference.

Changes from -01 to -02:

a. Minor formatting changes only - refreshing expired draft.

Changes from -00 to -01:

a. Added IPR boiler plate.

b. Re-ordered sections at start to conform to RFC style.

c. Moved recursion comment into General Considerations section.

d. Switched to using optional parameter to indicate personal vs global.

e. Explicitly state that an error occurs when a missing script is included.

Authors' Addresses

Cyrus Daboo

Email: cyrus@daboo.name

Aaron Stone

Email: aaron@serendipity.cx