Groupe de travail Réseau
Request for Comments : 5228
RFC rendue obsolète : 3028
Catégorie : Sur la voie de la normalisation

P. Guenther, éditeur, Sendmail, Inc.
T. Showalter, éditeur
janvier 2008
Traduction Claude Brière de L'Isle

Sieve : un langage de filtrage de messagerie électronique

Résumé
Le présent document décrit un langage pour filtrer les messages de messagerie électronique au moment de leur livraison finale. Il est conçu pour être mis en œuvre sur tout client ou serveur de messagerie électronique. Il est destiné à être extensible, simple, et indépendant du protocole d'accès, de l'architecture de messagerie, et du système d'exploitation. Il convient pour un fonctionnement sur un serveur de messagerie où les utilisateurs pourraient ne pas avoir la permission d'exécuter des programmes arbitraires, comme sur des serveurs de boîte noire du protocole d'accès au message Internet (IMAP, *Internet Message Access Protocol*) car le langage de base n'a pas de variables, de boucles, ou de capacité d'abriter des programmes externes.

Table des Matières

1. Introduction

Le présent mémoire documente un langage qui peut être utilisé pour créer des filtres pour la messagerie électronique. Il n'est pas lié à un système d'exploitation particulier ni à une architecture de messagerie. Il exige l'utilisation de messages conformes à la [RFC2822], mais devrait autrement se généraliser sur de nombreux systèmes.

Le langage est assez puissant pour être utile mais limité afin de permettre un système de filtage sûr côté serveur. L'intention est de rendre impossible aux utilisateurs de faire quelque chose de plus complexe (et dangereux) que d'écrire de simples filtres de messagerie, tout en facilitant l'utilisation d'interfaces graphiques d'utilisateur (GUI, *Graphical User Interface*) pour la création et la manipulation de filtres. Le langage de base n'a pas été conçu pour être complet au sens de Turing : il n'a pas de structure ou fonctions de contrôle de boucle.

Les scripts écrits en Sieve sont exécutés durant la livraison finale, quand le message est passé à la boîte aux lettres accessible à l'utilisateur. Dans les systèmes où l'agent de transfert de messagerie (MTA, *Mail Transfer Agent*) fait la livraison finale, comme dans la messagerie Unix traditionnelle, il est raisonnable de filtrer quand le MTA dépose les messages dans la boîte aux lettres de l'utilisateur.

Il y a un certain nombre de raisons pour utiliser un système de filtrage. Le trafic de messagerie pour la plupart des utilisateurs va croissant du fait de l'usage accru de la messagerie électronique, de l'émergence de messages non sollicités comme forme de publicité, et de l'usage croissant des listes de diffusion.

L'expérience faite à Carnegie Mellon a montré que si un système de filtrage est disponible aux utilisateurs, beaucoup vont l'utiliser afin de mettre dans des fichiers les messages provenant d'utilisateurs ou listes de diffusion spécifiques. Cependant, de nombreux autres n'ont pas utilisé le langage de filtrage FLAMES [FLAMES] du système Andrew à cause de la difficulté de son établissement.

À cause de l'espoir que les utilisateurs se servent du filtrage si il est offert et facile à utiliser, ce langage a été fait assez simple pour permettre à de nombreux utilisateurs de l'utiliser, mais assez riche pour qu'il puisse être utilisé de façon productive. Cependant, on s'attend à ce que les éditeurs fondés sur GUI soient la façon préférée d'éditer des filtres pour un grand nombre d'utilisateurs.

1.1 Conventions utilisées dans ce document

Dans les sections de ce document qui discutent les exigences des divers mots clés et opérateurs, les conventions suivantes ont été adoptées.

Les mots clés "DOIT", "NE DOIT PAS", "EXIGE", "DEVRA", "NE DEVRA PAS", "DEVRAIT", "NE DEVRAIT PAS", "RECOMMANDE", "PEUT", et "FACULTATIF" en majuscules dans ce document sont à interpréter comme décrit dans le BCP 14, [RFC2119].

Chaque paragraphe relatif à une commande (essai, action, ou contrôle) a une ligne intitulée "Usage:". Cette ligne décrit l'usage de la commande, incluant son nom et ses arguments. La liste des arguments requis est donnée entre des crochets angulaires ("<" et ">"). Les arguments facultatifs sont inscrits entre des crochets ("[" et "]"). Chaque argument est suivi de son type, donc "<clé: chaîne>" représente un argument appelé "clé" qui est une chaîne. Les chaînes littérales sont

représentées entre guillemets. Les solutions de remplacement sont séparées par des barres obliques, et les parenthèses sont utilisées pour grouper, comme dans la [RFC4234].

Dans la ligne "Usage:", il y a trois éléments de syntaxe spéciaux qui sont fréquemment répétés, MATCH-TYPE, COMPARATOR, et ADDRESS-PART. Ils sont discutés respectivement aux paragraphes 2.7.1, 2.7.3, et 2.7.4.

La grammaire formelle de ces commandes est définie à la Section 8 et est la référence d'autorité sur la façon de construire les commandes,mais la grammaire formelle ne spécifie pas l'ordre, la sémantique, le nombre ou les types d'arguments des commandes, ou les noms des commandes légales. L'intention est de permettre des extensions sans changer la grammaire.

1.2 Exemples de messages de messagerie

Les messages suivants vont être utilisés tout au long du présent document dans les exemples.

Message A

```
-----------------------------------------------------------
Date: Tue, 1 Apr 1997 09:06:31 -0800 (PST)
From: coyote@desert.exemple.org
To: roadrunner@acme.exemple.com
Subject: J'ai un cadeau pour toi

Vois tu, je suis désolé pour l'histoire de l'enclume, et je n'avais réellement pas l'intention d'essayer de te la laisser tomber dessus depuis le haut de la falaise. Je veux essayer de me racommoder avec toi. J'ai un tas de graines pour les oiseaux chez moi--le truc du haut du panier--et si tu passes, cela sera déjà tout emballé pour toi. Je suis vraiment désolé pour tous les problèmes que je t'ai causés toutes ces années, mais je sais qu'on peut y arriver.
--
Wile E. Coyote  "Super Genius"  coyote@desert.exemple.org
-----------------------------------------------------------
```

Message B

```
-----------------------------------------------------------
From: youcouldberich!@reply-by-postal-mail.invalid
Sender: b1ff@de.res.exemple.com
To: rube@landru.exemple.com
Date:  Mon, 31 Mar 1997 18:26:10 -0800
Subject: $$$ TOI AUSSI, TU PEUX DEVENIR MILLIONNAIRE ! $$$

TU PEUX AVOIR DÉJA GAGNÉ DIX MILLIONS D'EUROS, MAIS J'EN DOUTE ! ENVOI SEULEMENT CECI À SIX CENTS CORRESPONDANTS ! CELA VA GARANTIR QUE TU AURAS AU MOINS CINQ RÉPONSES AVEC L'ARGENT ! L'ARGENT ! L'ARGENT ! DU BON ARGENT ! TU VA RECEVOIR PLUS DE 20 000 EUROS EN MOINS DE DEUX MOIS ! ET C'EST LÉGAL !!!!!!!!!!!!!!!!!!!!!!!!!!!!111111111!!!!!!!11111111111!!1 ENVOIE JUSTE 5 € EN PETITES COUPURES, NON MARQUÉES AUX ADRESSES CI-DESSOUS !
-----------------------------------------------------------
```

2. Conception

2.1 Forme du langage

Le langage consiste en un ensemble de commandes. Chaque commande consiste en un ensemble de jetons délimités par des espaces. L'identifiant de commande est le premier jeton et il est suivi par zéro, un ou plusieurs jetons d'argument. Les arguments peuvent être des données littérales, des étiquettes, des blocs de commandes, ou des commandes d'essai.

À l'exceptions des chaînes et commentaires, le langage est limité aux caractères US-ASCII. Les chaînes et commentaires peuvent contenir des octets en dehors de la gamme de l'US-ASCII. Précisément, ils vont normalement être en UTF-8, comme spécifié dans la [RFC3629]. Le caractère NUL (US-ASCII 0) n'est jamais permis dans les scripts, tandis que CR et LF peuvent seulement apparaître comme la terminaison de ligne CRLF.

Note : bien que la présente spécification permette que des octets arbitraires apparaissent dans les chaînes et commentaires de scripts Sieve, cela rend difficile d'avoir un traitement robuste des scripts Sieve dans les programmes qui sont sensibles au codage utilisé. La capacité "encoded-character" (paragraphe 2.4.2.4) donne un moyen de remplacement pour représenter ces octets dans les chaînes en utilisant juste des caractères US-ASCII. À ce titre, l'utilisation de

texte non UTF-8 dans les scripts devrait être considérée comme une caractéristique déconseillée qui peut être abandonnée.

Les jetons autres que des chaînes sont considérés comme insensibles à la casse.

2.2 Espace

L'espace est utilisée pour séparer les jetons. Les espaces blanches sont des tabulations, des sauts à la ligne (CRLF, jamais juste CR ou LF) et le caractère espace. La quantité d'espaces utilisée n'est pas significative.

2.3 Commentaires

Deux types de commentaires sont offerts. Les commentaires sont sémantiquement équivalents aux espaces et peuvent être utilisés partout où les espaces le sont (à une exception près dans les chaînes multi-lignes, comme décrit dans la grammaire).

Les commentaires hachés commencent par un caractère "#" qui n'est pas contenu dans une chaîne et se continuent jusqu'au prochain CRLF.

Exemple : si taille :supérieure à 100 k { # c'est un commentaire éliminer;
}

Les commentaires entre crochets commencent par le jeton "/*" et se terminent avec "*/" en dehors d'une chaîne. Les commentaires entre crochets peuvent s'étendre sur plusieurs lignes. Les commentaires entre crochets ne sont pas incorporés.

Exemple : si taille :supérieure à 100 k { /* c'est un commentaire c'est encore un commentaire */ éliminer /* c'est un commentaire */ ;
}

2.4 Données littérales

Les données littérales signifient que les données ne sont pas exécutées, simplement évaluées "telles qu'elles", pour être utilisées comme arguments des commandes. Les données littérales sont limitées aux nombres, chaînes, et listes de chaînes.

2.4.1 Nombres

Les nombres sont donnés comme des nombres décimaux ordinaires. Comme abrégé pour exprimer les grandes valeurs, comme la taille des messages, un suffixe de "K", "M", ou "G" PEUT être ajouté pour indiquer un multiple d'une puissance de deux. Pour être comparable avec les version fondées sur les puissances de deux des unités du système international qu'utilisent fréquemment les ordinateurs, "K" spécifie kilo-, ou 1 024 (2^10) fois la valeur du nombre, "M" spécifie méga-, ou 1 048 576 (2^20) fois la valeur du nombre, et "G" spécifie giga-, ou 1 073 741 824 (2^30) fois la valeur du numbre [CEI60027-2].

Les mises en œuvre DOIT supporter les valeurs d'entier dans la gamme inclusive de zéro à 2 147 483 647 (2^31 - 1) mais PEUVENT supporter des valeurs supérieures.

Seuls les entiers non négatifs sont permis par la présente spécification.

2.4.2 Chaînes

Les scripts impliquent de grands nombres de valeurs de chaînes car elles sont utilisées pour la confrontation de schémas, d'adresses, de corps de textes, etc. Normalement, de courtes chaînes entre guillemets suffisent dans la plupart des utilisations, mais une forme plus pratique est fournie pour les chaînes plus longues comme les corps de messages.

Une chaîne entre guillemets commence et se termine par des guillemets (le caractère <">, US-ASCII 34). Une barre oblique inverse ("\", US-ASCII 92) à l'intérieur d'une chaîne entre guillemets est suivie soit par une autre barre oblique, soit par des guillemets. Ces séquences de deux caractères représentent respectivement une seule barre oblique inverse ou des guillemets au sein de la valeur.

Les scripts NE DEVRAIENT PAS échapper d'autres caractères avec une barre oblique inverse.

Une séquence d'échappement indéfinie (comme "\a" dans un contexte où "a" n'a pas de signification particulière) est interprétée comme si il n'y avait pas de barre oblique inverse (dans ce cas, "\a" est juste "a") bien que cela puisse être changé par des extensions.

Les caractères non imprimables comme les tabulations, CRLF, et les caractères de commandes sont permis dans les chaînes entre guillemets.. Les chaînes entre guillemets PEUVENT s'étendre sur plusieurs lignes. Un NUL non codé (US-ASCII 0) n'est pas permis dans les chaînes ; voir au paragraphe 2.4.2.4 comment il peut être codé.

Comme les données d'en-tête de message sont converties selon la [RFC3629], pour les comparaisons (voir le paragraphe 2.7.2) la plupart des valeurs de chaîne vont utiliser le codage UTF-8. Cependant, les mises en œuvre DOIVENT accepter toutes les chaînes qui respectent la grammaire de la Section 8. La capacité d'utiliser des chaînes qui ne sont pas codées en UTF-8 correspond à la pratique existante et s'est révélée utile à la fois dans les essais pour les données invalides et dans les arguments qui contiennent des parties MIME brutes pour des actions d'extension qui génèrent des messages sortants.

Pour entrer de grandes quantités de texte, comme un message électronique, une forme multi lignes est permise. Elle commence par le mot-clé "text:", suivi par un CRLF, et se termine par une séquence d'un CRLF, un seul point, et un autre CRLF. Le CRLF avant le point final est considéré faire partie de la valeur. Afin de permettre que le message contienne des lignes avec un seul point, les lignes sont bourrées de points. C'est-à-dire, quand on compose un corps de message, un '.' supplémentaire est ajouté avant chaque ligne qui commence par un '.'. Quand le serveur interprète le script, ces points supplémentaires sont supprimés. Noter qu'une ligne qui commence par un point suivi par un caractère non point n'est pas interprétée comme bourrée de points ; c'est-à-dire, ".foo" est interprété comme ".foo". Cependant, parce que ceci est potentiellement ambigu, les scripts DEVRAIENT être correctement bourrés de points afin que de telles lignes n'apparaissent pas.

Noter qu'un commentaire haché ou une espace peut se produire entre le "text:" et le CRLF, mais pas dans la chaîne elle-même. Les commentaires entre guillemets ne sont pas permis ici.

2.4.2.1 Listes de chaînes

Quand on confronte des schémas, il est souvent pratique de confronter des groupes de chaînes plutôt qu'une seule chaîne. Pour cette raison, une liste de chaînes est permise dans de nombreux essais, ce qui implique que si l'essai est vrai en utilisant une des chaînes, l'essai est vrai.

Par exemple, l'essai "header :contains ["To", "Cc"] ["me@exemple.com", "me00@landru.exemple.com"]" est vrai si soit un en-tête To, soit un en-tête Cc, du message d'entrée contient l'adresse de messagerie "me@exemple.com" ou "me00@landru.exemple.com".

À l'inverse, dans tous les cas où une liste de chaînes est appropriée, une seule chaîne est permise sans être membre d'une liste : elle est équivalente à une liste d'un seul membre. Cela signifie que l'essai "exists "To"" est équivalent à l'essai "exists ["To"]".

2.4.2.2 En-têtes

Les en-têtes sont un sous ensemble de chaînes. Dans la spécification du message Internet [RFC2822], chaque ligne d'en-tête peut avoir des espaces presque partout dans la ligne, y compris après le champ nom de champ et avant les deux-points suivants. Les espaces supplémentaires entre le nom d'en-tête et les ":" dans un champs d'en-tête sont ignorées.

Un nom d'en-tête ne contient jamais deux-points. L'en-tête "From" se réfère à une ligne commençant "From:" (ou "From :", etc.). Aucun en-tête ne va correspondre à la chaîne "From:" à cause des deux points en queue.

De même, aucun en-tête ne va correspondre à un nom d'en-tête syntaxiquement invalide. Une mise en œuvre NE DOIT PAS causer une erreur pour des noms d'en-tête syntaxiquement invalides dans les essais.

Les lignes d'en-tête ne vont pas à la ligne, comme décrit au paragraphe 2.2.3 de la [RFC2822]. L'interprétation des données d'en-tête DEVRAIT être faite conformement au paragraphe 6.2 de la [RFC2047] (voir les détails au paragraphe 2.7.2).

2.4.2.3 Adresses

Un certain nombre de commandes invoquent des adresses de messagerie, qui sont aussi un sous ensemble de chaînes. Quand ces adresses sont utilisées dans un contexte sortant, elles doivent être conformes à la [RFC2822], mais sont de plus contraintes dans le présent document. En utilisant les symboles définis à la Section 3 de la [RFC2822], la syntaxe d'une adresse est :

```
sieve-address = addr-spec                          ; adresse simple
              / phrase "<" addr-spec ">"           ; nom & spécification d'adresse
```

C'est-à-dire, la syntaxe de "routes" et "group" n'est pas permise. Si plusieurs adresses sont nécessaires, on utilise une liste de chaînes. Les groupes désignés ne sont pas permis.

Pour un script, c'est une erreur d'exécuter une action avec une valeur à utiliser comme adresse sortante qui ne correspond pas à la syntaxe de "sieve-address".

2.4.2.4 Codage de caractères utilisant "encoded-character"

Quand l'extension "encoded-character" est en usage, certaines séquences de caractères dans les chaînes sont remplacées par leur valeur décodée. Cela arrive après que des séquences d'échappement sont interprétées et que la suppression du bourrage de points a été faite. Les mises en œuvre DEVRAIENT prendre en charge "encoded-character".

Des octets arbitraires peuvent être incorporés dans des chaînes en utilisant la syntaxe de "encoded-arb-octets". La séquence est remplacée par les octets avec les valeurs hexadécimales données par chaque paire hexadécimale.

```
blank = WSP / CRLF
encoded-arb-octets = "${hex:" hex-pair-seq "}"
hex-pair-seq = *blank hex-pair *(1*blank hex-pair) *blank
hex-pair = 1*2HEXDIG
```

Où les non terminaux WSP et HEXDIG sont définis dans l'Appendice B.1 de la [RFC4234].

Il peut n'être pas pratique ou indésirable d'entrer tels quels les caractères Unicode, et dans ce cas la syntaxe "encoded-unicode-char" peut être utilisée. La séquence est remplacée par le codage UTF-8 des caractères Unicode spécifiés, qui sont identifiés par la valeur hexadécimale de unicode-hex.

```
encoded-unicode-char = "${unicode:" unicode-hex-seq "}"
unicode-hex-seq = *blank unicode-hex *(1*blank unicode-hex) *blank
unicode-hex = 1*HEXDIG
```

C'est une erreur pour un script d'utiliser une valeur hexadécimale qui n'est pas dans la gamme de 0 à D7FF ou de E000 à 10FFFF. (La gamme de D800 à DFFF est exclue car ces caractères ne sont utilisés qu'au titre du codage UTF-16 et ne sont pas applicables au codage UTF-8 que représente ici la syntaxe.)

Note : Les mises en œuvre NE DOIVENT soulever une erreur pour une valeur Unicode hors gamme que si la séquence qui la contient est bien formée selon la grammaire.

La chaîne de capacités à utiliser avec la commande "require" est "encoded-character".

Dans le script suivant, le message B est éliminé, car l'essai spécifié est équivalent à "$$$".

```
exemple :  require "encoded-character";
           si l'en-tête :contains "Subject" "$${hex:24 24}" {
             discard;
           }
```

Les exemples suivants montrent des codages valides et invalides et comment ils sont traités :

```
"$${hex:40}" -> "$@"
"${hex: 40 }" -> "@"
"${HEX: 40}" -> "@"
"${hex:40" -> "${hex:40"
"${hex:400}" -> "${hex:400}"
```

"${hex:4${hex:30}}" -> "${hex:40}"
"${unicode:40}" -> "@"
"${ unicode:40}" -> "${ unicode:40}"
"${UNICODE:40}" -> "@"
"${UnICoDE:0000040}" -> "@"
"${Unicode:40}" -> "@"
"${Unicode:Cool}" -> "${Unicode:Cool}"
"${unicode:200000}" -> erreur
"${Unicode:DF01} -> erreur

2.5 Essais

Les essais sont donnés comme arguments aux commandes afin de contrôler leurs actions. Dans ce document, les essais sont donnés à if/elsif pour décider quel bloc de code est traité.

2.5.1 Listes d'essais

Certains essais ("allof" et "anyof", qui mettent en œuvre respectivement le "ET" logique et le "OU" logique) peuvent exiger plus d'un seul essai comme argument. L'élément de syntaxe test-list donne un moyen pour grouper les essais comme une liste séparée de virgules entre crochets.

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

2.6 Arguments

Pour spécifier quoi faire, la plupart des commandes prennent des arguments. Il y a trois types d'arguments : positionnels, étiquetés, et facultatifs.

C'est une erreur pour un script, d'utiliser sur une seule commande, des arguments contradictoires ou d'utiliser plus d'une fois un argument étiqueté ou facultatif.

2.6.1 Arguments positionnels

Les arguments positionnels sont donnés à une commande qui discerne leur signification sur la base de leur ordre. Quand une commande prend des arguments positionnels, tous les arguments positionnels doivent être fournis et doivent être dans l'ordre prescrit.

2.6.2 Arguments étiquetés

Le présent document fournit des arguments étiquetés dans le style de CommonLISP. Ils sont aussi similaires aux fanions donnés aux commandes dans la plupart des systèmes à ligne de commande.

Un argument étiqueté est un argument pour une commande qui commence par ":" suivi par une étiquette désignant l'argument, comme ":contains". Cet argument signifie que zéro, un ou plusieurs des jetons suivants ont une signification particulière selon l'argument. Ces prochains jetons peuvent être des données littérales, mais ne sont jamais des blocs.

Les arguments étiquetés sont similaires aux arguments positionnels, sauf qu'au lieu que la signification soit déduite de la commande, elle est déduite de l'étiquette.

Les arguments étiquetés doivent apparaître avant les arguments positionnels, mais ils peuvent apparaître dans n'importe quel ordre avec d'autres arguments étiquetés. Pour simplifier la spécification, ceci n'est pas exprimé dans les définitions de syntaxe avec les commandes, mais elles peuvent quand même être réordonnées arbitrairement pourvu qu'elles apparaissent avant les arguments positionnels. Les arguments étiquetés peuvent être mélangés à des arguments facultatifs.

Les arguments étiquetés NE DEVRAIENT PAS prendre des arguments étiquetés comme arguments.

2.6.3 Arguments facultatifs

Les arguments facultatifs sont exactement comme les arguments étiquetés sauf qu'ils peuvent être omis, et dans ce cas, une valeur par défaut est impliquée. Parce que les arguments facultatifs tendent à résulter en scripts plus courts, ils ont été beaucoup plus utilisés que les arguments étiquetés.

Un cas particulièrement notable est l'argument ":comparator", qui permet à l'utilisateur de spécifier quel comparateur [RFC4790] va être utilisé pour comparer deux chaînes, car différents langages peuvent imposer des ordres différents sur les chaînes UTF-8 [RFC3629].

2.6.4 Types d'arguments

Abstraitement, les arguments peuvent être des données littérales, des essais, ou des blocs de commandes. De cette façon, une structure de contrôle "if" est simplement une commande qui se trouve prendre un essai et un bloc comme arguments et peut exécuter le bloc de code.

Cependant, cette abstraction est ambiguë du point de vue de l'analyse.

La grammaire du paragraphe 8.2 présente de cela une version analysable : les arguments sont des listes de chaînes *(string-lists)* des nombres, et des étiquettes, qui peuvent être suivies par un essai ou une liste d'essais *(test-list)* qui peut être suivie par un bloc de commandes. Pas plus d'un essai ou liste d'essais, ou plus d'un bloc de commandes, ne peut être utilisé, et les commandes qui se terminent par un bloc de commandes ne se terminent pas par un point-virgule.

2.7 Comparaison de chaînes

Quand on confronte une chaîne à une autre, il y a un certain nombre de façons d'effectuer la confrontation. Elles sont accomplies avec trois types de confrontation : une correspondance exacte, une correspondance de sous chaîne, et une correspondance de caractère générique de style global. Elles sont décrites ci-dessous.

Afin d'assurer des correspondances entre les jeux de caractères et l'insensibilité à la casse, Sieve utilise les comparateurs définis dans la [RFC4790].

Cependant, quand une chaîne représente le nom d'un en-tête, le comparateur n'est jamais spécifié pour l'utilisateur. Les comparaisons d'en-tête sont toujours faites avec l'opérateur "i;ascii-casemap", c'est-à-dire, des comparaisons insensibles à la casse, parce que c'est la façon dont les choses sont définies dans la spécification de message de la [RFC2822].

2.7.1 Type de confrontation

Les commandes qui effectuent des comparaisons de chaînes peuvent avoir un argument de type de correspondance facultatif. Les trois types de correspondance dans cette spécification sont ":contains" *(contient)*, ":is" *(est)*, et ":matches" *(correspond)*.

Le type de correspondance ":contains" décrit une correspondance de sous chaîne. Si l'argument de valeur contient l'argument de clé comme sous chaîne, la correspondance est vraie. Par exemple, la chaîne "frobnitzm" contient "frob" et "nit", mais pas "fbm". La clé vide ("") est contenue dans toutes les valeurs.

Le type de correspondance ":is" décrit une correspondance absolue ; si le contenu de la première chaîne est absolument le même que le contenu de la seconde chaîne, elles correspondent. Seule la chaîne "frobnitzm" est la chaîne "frobnitzm". La clé vide ("") ":is" correspond seulement à la valeur vide.

Le type de correspondance ":matches" spécifie une correspondance de caractère générique en utilisant les caractères "*" et "?" ; la valeur entière doit être satisfaite. "*" correspond à zéro, un ou plusieurs caractères dans la valeur et "?" correspond à un seul caractère dans la valeur, où le comparateur utilisé (voir le paragraphe 2.7.3) définit ce qu'est un caractère. Par exemple, les comparateurs "i;octet" et "i;ascii-casemap" définissent un caractère comme étant d'un seul octet, donc "?" va toujours correspondre à exactement un octet quand un de ces comparateurs est utilisé. À l'opposé, un comparateur fondé sur Unicode va définir un caractère comme étant toute séquence d'octets UTF-8 codant un caractère Unicode et donc "?" peut correspondre à plus d'un octet. "?" et "*" peuvent être échappés comme "\\?" et "*" dans les chaînes pour correspondre à elles-mêmes. La première barre oblique inverse échappe la seconde ; ensemble, elles échappent le "*". C'est bizarre, mais c'est courant pour plusieurs langages de programmation qui utilisent des globs et des expressions régulières.

Afin de spécifier le type de correspondance qui est supposé se produire, les commandes qui prennent en charge la correspondance prennent des arguments facultatifs ":matches", ":is", et ":contains". Les commandes utilisent par défaut la correspondance ":is" si aucun argument de type de correspondance n'est fourni. Noter que ces modificateurs interagissent avec les comparateurs ; en particulier, seuls les comparateurs qui prennent en charge l'opération "correspondance de sous chaîne" conviennent pour la correspondance avec ":contains" ou ":matches". C'est une erreur d'utiliser un comparateur avec ":contains" ou ":matches" qui n'est pas compatible avec eux.

C'est une erreur de donner plus d'un de ces arguments à une commande donnée.

L'élément de syntaxe "MATCH-TYPE" est défini comme suit :

Syntaxe : ":is" / ":contains" / ":matches"

2.7.2 Comparaisons à travers des jeux de caractères

Les messages peuvent impliquer un certain nombre de jeux de caractères. Afin que les comparaisons fonctionnent à travers les jeux de caractères, les mises en œuvre DEVRAIENT avoir le comportement suivant :

Les comparaisons sont effectuées sur les octets. Les mises en œuvre convertissent le texte des champs d'en-tête de tous les jeux de caractères [RFC2047] en Unicode, codé comme UTF-8, comme entrée au comparateur (voir le paragraphe 2.7.3). Les mises en œuvre DOIVENT être capables de convertir les jeux de caractères US-ASCII, ISO-8859-1, le sous ensemble US-ASCII de ISO-8859-*, et UTF-8. Le texte que la mise en œuvre ne peut pas convertir en Unicode pour une raison quelconque PEUT être traité comme de l'US-ASCII pur (incluant toute syntaxe de la [RFC2047]) ou conformément à des conventions locales. Un octet NUL codé (zéro caractère) NE DEVRAIT PAS causer une terminaison prématurée de la comparaison au contenu d'en-tête.

Si les mises en œuvre échouent à prendre en charge le comportement ci-dessus, elles DOIVENT se conformer à ce qui suit : deux chaînes en peuvent pas être considérées comme égales si l'une contient des octets supérieurs à 127.

2.7.3 Comparateurs

Afin de permettre des confrontations indépendantes du langage et de la casse, le type de correspondance peut être couplé à un nom de comparateur. Le registre de collation des protocoles d'application Internet [RFC4790] donne le cadre pour décrire et désigner les comparateurs.

Toutes les mises en œuvre DOIVENT prendre en charge le comparateur "i;octet" (compare simplement les octets) et le comparateur "i;ascii-casemap" (qui traite comme les mêmes les caractères majuscules et minuscules dans le sous ensemble US-ASCII de l'UTF-8). Si il est non spécifié, il est par défaut "i;ascii-casemap".

Certains comparateurs peuvent n'être pas utilisables avec des confrontations de sous chaînes ; c'est-à-dire, ils peuvent seulement fonctionner avec ":is". C'est une erreur d'essayer d'utiliser un comparateur avec ":matches" ou ":contains" qui n'est pas compatible avec eux.

Sieve traite un résultat de comparateur de "indéfini" comme un résultat de "no-match" *(pas de correspondance)*. C'est-à-dire, cette spécification de base ne fournit aucun moyen de détecter directement une entrée de comparateur invalide.

Un comparateur est spécifié par l'option ":comparator" avec les commandes qui prennent en charge la confrontation. Cette option est suivie par une chaîne qui donne le nom du comparateur à utiliser. Pour des raisons pratiques, la syntaxe d'un comparateur est abrégée en "COMPARATOR", et (répétée dans plusieurs essais) est comme suit :

Syntaxe : ":comparator" <nom de comparateur: chaîne>

Ainsi, dans cet exemple,

```
exemple:  si en-tête :contains :comparator "i;octet" "Subject"
             "GAGNER DE L'ARGENT FACILE" {
          discard;
        }
```

va éliminer tout message avec des sujets comme "Vous pouvez GAGNER DE L'ARGENT FACILE", mais pas "Vous pouvez Gagner de l'Argent Facilement", car le comparateur utilisé est sensible à la casse.

Les comparateurs autres que "i;octet" et "i;ascii-casemap" doivent être déclarés avec "require", car ce sont des extensions. Si un comparateur déclaré avec "require" n'est pas connu, c'est une erreur, et l'exécution va échouer. Si le comparateur n'est pas déclaré avec "require", c'est aussi une erreur, même si le comparateur est pris en charge (voir le paragraphe 2.10.5.)

Les types de correspondance ":matches" et ":contains" sont tous deux compatibles avec les comparateurs "i;octet" et "i;ascii-casemap" et peuvent être utilisés avec eux.

C'est une erreur de donner plus d'un de ces arguments à une commande donnée.

2.7.4 Comparaisons d'adresses

Les adresses sont une des choses les plus fréquentes représentées comme des chaînes. Elles sont structurées, et être capable de comparer la partie locale ou le domaine d'une adresse est utile, de sorte que certains essais qui agissent exclusivement sur les adresses prennent un argument facultatif supplémentaire qui spécifie sur quoi agit l'essai.

Ces arguments facultatifs sont ":localpart", ":domain", et ":all", qui agissent respectivement sur la partie locale (côté gauche) la partie domaine (côté droit), et l'adresse entière.

Si une adresse n'est pas syntaxiquement valide, elle ne va pas être confrontée par les essais qui spécifient ":localpart" ou ":domain".

La sorte de comparaison faite, comme si l'essai effectué est ou non insensible à la casse, est spécifiée comme un argument de comparateur à l'essai.

Si une partie d'adresse facultative est omise, la valeur par défaut est ":all".

C'est une erreur de donner plus d'un de ces arguments à une commande donnée.

Pour des raisons pratiques, l'élément de syntaxe "ADDRESS-PART" est défini ici comme suit :

Syntaxe : ":localpart" / ":domain" / ":all"

2.8 Blocs

Les blocs sont des ensembles de commandes enclos dans des accolages et fournis comme argument final d'une commande. Une telle commande est une structure de contrôle : quand elle est exécutée, elle contrôle le nombre de fois que les commandes du bloc sont exécutées.

Avec les commandes fournies dans le présent mémoire, il n'y a pas de boucles. Les structures de contrôle fournies -- if, elsif, et else -- fonctionnent sur un bloc une fois ou pas du tout.

2.9 Commandes

Les scripts Sieve sont des séquences de commandes. Les commandes peuvent prendre tout jeton ci-dessus comme argument, et les arguments peuvent être étiquetés ou positionnels. Toutes les commandes ne prennent pas tous les arguments.

Il y a trois sortes de commandes : les commandes d'essai, les commandes d'action, et les commandes de contrôle.

La plus simple est une commande d'action. C'est un identifiant suivi de zéro, un, ou plusieurs arguments, terminé par un point virgule. Les commandes d'action ne prennent pas d'essais ou de blocs comme arguments. Les actions référencées dans le présent document sont :
- keep, pour sauvegarder le message dans la localisation par défaut,
- fileinto, pour sauvegarder le message dans une boîte aux lettres spécifique,
- redirect, pour transmettre le message à une autre adresse,
- discard, pour éliminer en silence le message.

Une commande de contrôle affecte d'une certaine façon l'analyse ou le flux d'exécution du script Sieve. Une structure de contrôle est une commande de contrôle qui se termine par un bloc au lieu d'un point virgule.

Une commande d'essai est utilisée au titre d'une commande de contrôle. Elle est utilisée pour spécifier si le bloc de code donné à la commande de contrôle est exécuté ou non.

2.10 Évaluation

2.10.1 Action Interaction

Certaines actions ne peuvent pas être utilisées avec d'autres actions parce que le résultat serait absurde. Ces restrictions sont notées tout au long du présent mémoire.

Les actions d'extension DOIVENT déclarer comment elles interagissent avec les actions définies dans cette spécification.

2.10.2 Garde implicite

L'expérience des systèmes de filtrage précédents suggère que certains cas tendent à être oubliés dans les scripts. Pour prévenir les erreurs, Sieve a un "implicit keep" *(garde implicite)*.

Une garde implicite est une action "keep" (voir au paragraphe 4.3) effectuée en l'absence de toute action qui annule le "implicit keep".

Une garde implicite est effectuée si un message n'est pas écrit à une boîte aux lettres, redirigé sur une nouvelle adresse, ou explicitement éliminé. C'est-à-dire, si un "fileinto", un "keep", un "redirect", ou un "discard" est effectué, un "implicit keep" ne l'est pas.

Certaines actions peuvent être définies pour ne pas annuler le "implicit keep". Ces actions peuvent ne pas directement affecter la livraison d'un message, et sont utilisés pour leurs effets collatéraux. Aucune des actions spécifiées dans le présent document ne satisfait ce critère, mais des actions d'extension le peuvent.

Par exemple, avec tout court message présenté ci-dessus, le script suivant ne produit aucune action.

exemple : si taille :over 500K { discard; }

Il en résulte qu'un "keep" implicite est effectué.

2.10.3 Unicité de message dans une boîte aux lettres

Les mises en œuvre NE DEVRAIENT PAS livrer plus d'une fois un message à la même boîte aux lettres, même si un script demande explicitement qu'un message soit livré deux fois à une boîte aux lettres.

L'essai d'égalité de deux messages est défini par la mise en œuvre.

Si un script demande qu'un message soit livré deux fois à une boîte aux lettres, cela NE DOIT PAS être traité comme une erreur.

2.10.4 Limites du nombre d'actions

La politique d'un site PEUT limiter le nombre d'actions effectuées et PEUT imposer des restrictions aux actions qui peuvent être utilisées ensemble. Dans le cas où un script touche une limite de politique sur le nombre des actions effectuées sur un message particulier, une erreur se produit.

Les mises en œuvre DOIVENT permettre au moins un "keep" ou un "fileinto". Si fileinto n'est pas mis en œuvre, on DOIT permettre au moins un "keep".

2.10.5 Extensions et caractéristiques facultatives

À cause des capacités différentes de nombreux systèmes de messagerie, plusieurs caractéristiques de la présente spécification sont facultatives. Avant qu'aucune de ces extensions puisse être exécutée, elles doivent être déclarées avec l'action "require".

Si une extension n'est pas activée avec "require", les mises en œuvre DOIVENT la traiter comme si elles ne la prenent pas du tout en charge. Cela protège les scripts contre l'altération de leur comportement par des extensions dont l'auteur du script pourrait n'avoir même pas connaissance.

Les mises en œuvre NE DOIVENT PAS exécuter un essai ou commande de script Sieve suivant un "require" si une des extensions requises n'est pas disponible.

Note : la raison de cette restriction est que les expériences antérieures de langages comme LISP et Tcl suggèrent que c'est une façon acceptable de noter qu'un certain script utilise une extension.

Les extensions qui définissent des actions DOIVENT déclarer comment elles interagissent avec les actions discutées dans la spécification de base.

2.10.6 Erreurs

Dans tout langage de programmation, il y a des erreurs au moment de la compilation et des erreurs au moment de l'exécution.

Les erreurs au moment de la compilation sont celles de syntaxe qui sont détectables si une vérification de syntaxe est faite.

Les erreurs au moment de l'exécution ne sont détectables qu'au moment de l'exécution dulscript. Cela inclut des défaillances transitoires comme des conditions de disque plein, mais aussi des problèmes comme des combinaisons invalides d'actions.

Quand une erreur survient dans un script Sieve, tout traitement est arrêté.

Les mises en œuvre PEUVENT choisir de faire une analyse complète, puis d'évaluer le script, puis d'effectuer toutes les actions. Les mises en œuvre pourraient même aller jusqu'à s'assurer que l'exécution est atomique (soit toutes les actions sont exécutées, soit aucune n'est exécutée).

D'autres mises en œuvre peuvent choisir d'analyser et d'exécuter en même temps. De telles mises en œuvre sont plus simples, mais ont des problèmes avec la défaillance partielle (certaines actions arrivent, d'autres pas).

Les mises en œuvre DOIVENT effectuer des vérifications syntaxiques, sémantiques, et de démarrage sur le code qui est en fait exécuté. Les mises en œuvre PEUVENT effectuer ces vérifications sur toute partie du code qui n'est pas touchée durant l'exécution.

Quand une erreur arrive, les mises en œuvre DOIVENT notifier à l'utilisateur qu'une erreur s'est produite et quelles actions (si il en est) ont été prises, et faire un "keep" implicite.

2.10.7 Limites sur l'exécution

Les mises en œuvre peuvent limiter certaines constructions. Cependant, la présente spécification place une borne inférieure à certaines de ces limites :
Les mises en œuvre DOIVENT prendre en charge quinze niveaux de blocs incorporés.
Les mises en œuvre DOIVENT prendre en charge quinze niveaux de listes d'essais incorporés.

3. Commandes de contrôle

Des structures de contrôle sont nécessaires pour permettre des actions multiples et conditionnelles.

3.1 Commande "if"

Il y a trois formes de si : "if", "elsif", et "else". Chacune est en fait une commande distincte en termes de grammaire. Cependant, un "elsif" ou un "else" DOIT seulement suivre un "if" ou "elsif". Une erreur se produit si ces conditions ne sont pas remplies.

Usage : if <test1: test> <block1: block>

Usage : elsif <test2: test> <block2: block>

Usage : else <block3: block>

La sémantique est similaire à celle de nombreux autres langages de programmation où apparaissent ces structures de contrôle. Quand l'interpréteur voit un "if", il évalue l'essai qui lui est associé. Si l'essai est vrai, il exécute le bloc associé.

Si l'essai du "if" est faux, il évalue l'essai du premier "elsif" (si il y en a un). Si l'essai du "elsif" est vrai, il exécute le bloc du "elsif". Un "elsif" peut être suivi d'un "elsif," et dans ce cas, l'interpréteur répète ce procès jusqu'à ce qu'il n'y ait plus de "elsif".

Quand l'interpréteur n'a plus de "elsif", il peut y avoir un cas de "else". Si il y en a un, et si aucun des essais de "if" ou "elsif" n'est vrai, l'interpréteur traite le bloc de "else".

Cela donne un moyen d'effectuer exactement un des blocs dans la chaîne.

Dans l'exemple suivant, les deux messages A et B sont éliminés.

```
exemple:  require "fileinto";
          if header :contains "from" "coyote" {
            discard;
          } elsif header :contains ["subject"] ["$$$"] {
            discard;
          } else {
            fileinto "INBOX";
          }
```

Quand le script ci-dessous est exécuté sur le message A, il redirige le message à acm@exemple.com, le message B, à postmaster@exemple.com ; tout autre message est redirigé sur field@exemple.com.

```
exemple :  if header :contains ["From"] ["coyote"] {
             redirect "acm@exemple.com";
           } elsif header :contains "Subject" "$$$" {
             redirect "postmaster@exemple.com";
           } else {
             redirect "field@exemple.com";
           }
```

Noter que cette définition interdit la séquence "... else if ..." utilisée par C. C'est intentionnel, parce que cette construction produit un conflit de réduction de décalage.

3.2 Commande require

Usage : require <capabilities: string-list>

L'action "require" note qu'un script utilise une certaine extension. Une telle déclaration est exigée pour utiliser l'extension, comme expliqué au paragraphe 2.10.5. Plusieurs capacités peuvent être déclarées avec un seul "require".

La commande "require", si elle est présente, DOIT être utilisée avant que toute autre chose qu'un "require" puisse être utilisé. Une erreur se produit si un "require" apparaît après une commande autre que "require".

```
exemple :  require ["fileinto", "reject"];
```

```
exemple :  require "fileinto";
           require "vacation";
```

3.3 Commande stop

Usage : stop

L'action "stop" termine tout traitement. Si le "keep" implicite n'a pas été annulé, il est alors appliqué.

4. Commandes d'action

Le présent document fournit quatre actions qui peuvent être effectuées sur un message : keep, fileinto, redirect, et discard.
Les mises en œuvre DOIVENT prendre en charge les actions "keep", "discard", et "redirect".
Les mises en œuvre DEVRAIENT prendre en charge "fileinto".
Les mises en œuvre PEUVENT limiter le nombre de certaines actions (voir le paragraphe 2.10.4).

4.1 Action "fileinto"

Usage : fileinto <mailbox: string>

L'action "fileinto" livre le message dans la boîte aux lettres spécifiée. Les mises en œuvre DEVRAIENT prendre en charge "fileinto", mais dans certains environnements cela peut être impossible. Les mises en œuvre PEUVENT mettre des restrictions sur les noms de boîte aux lettres ; l'utilisation d'un nom de boîte au lettres invalide PEUT être traité comme une erreur ou résulter en la livraison à une boîte aux lettres définie par une mise en œuvre. Si la boîte aux lettres spécifiée n'existe pas, la mise en œuvre PEUT traiter cela comme une erreur, créer la boîte aux lettres, ou livrer le message à une boîte aux lettres définie par la mise en œuvre. Si la mise en œuvre utilise un schéma de codage différent de UTF-8 pour les noms de boîte aux lettres, elle DEVRAIT recoder le nom de la boîte aux lettres de UTF-8 en son schéma de codage. Par exemple, le protocole d'accès au message Internet [RFC3501] utilise de l'UTF-7 modifié, tel qu'un argument de boîte aux letttre de "odds & ends" apparaîtrait dans IMAP comme "odds &- ends".

La chaîne de capacités à utiliser avec la commande "require" est "fileinto".

Dans le script qui suit, le message A est mis dans la boîte aux lettres "INBOX.harassment".

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

4.2 Action "redirect"

Usage : redirect <address: string>

L'action "redirect" est utilisée pour envoyer le message à un autre utilisateur à l'adresse fournie, comme le fait un dispositif de transmission de messagerie. L'action "redirect" ne fait pas de changement au corps du message ou aux en-têtes existants, mais peut ajouter de nouveaux en-têtes. En particulier, les en-têtes Received existants DOIVENT être préservés et le compte des en-têtes Received dans le message sortant DOIT être supérieur au même compte dans le message reçu par la mise en œuvre. (Une mise en œuvre qui ajoute un en-tête Received avant de traiter le message n'a pas besoin d'en ajouter un autre lors de la redirection.)

Le message est renvoyé avec l'adresse provenant de la commande "redirect" comme un receveur d'enveloppe. Les mises en œuvre PEUVENT combiner des "redirect" séparés pour un message donné dans une seule soumission avec plusieurs receveurs d'enveloppe. (Ce n'est pas une transmission de style agent d'utilisateur de messagerie (MUA, *Mail User Agent*) qui crée un nouveau message avec un envoyeur et un identifiant de message différents, enveloppant le vieux message dans un nouveau.)

L'adresse d'envoyeur d'enveloppe sur le message sortant est choisie par la mise en œuvre Sieve. Elle PEUT être copiée du message traité. Cependant, si le message traité a une adresse d'envoyeur d'enveloppe vide, le message sortant DOIT aussi avoir une adresse d'envoyeur d'enveloppe vide. Cette dernière exigence est imposée pour prévenir les boucles dans le cas où un message est redirigé sur une adresse invalide quand une notification d'état de livraison est retournée qui finit elle aussi par être redirigée sur la même adresse invalide.

Un simple script peut être utilisé pour rediriger tous les messages :

exemple : redirect "bart@exemple.com";

Les mises en œuvre DOIVENT prendre des mesures pour appliquer le contrôle de boucles, éventuellement en incluant des en-têtes au message ou en comptant les en-têtes Received comme spécifié au paragraphe 6.2 de la [RFC2821]. Si une mise en œuvre détecte une boucle, cela cause une erreur.

Les mises en œuvre DOIVENT fournir des moyens de limiter le nombre de redirections qu'un script Sieve peut effectuer. Voir les détails à la Section 10.

Les mises en œuvre PEUVENT ignorer en silence une action "redirect" pour des raisons de politique. Par exemple, une mise en œuvre PEUT choisir de ne pas rediriger sur une adresse qui est connue comme étant injoignable. Un "redirect" ignoré NE DOIT PAS annuler le "keep" implicite.

4.3 Action "keep"

Usage : keep

L'action "keep" est toute action prise à la place de toute autre action, si aucun filtrage ne se produit ; généralement, cela signifie simplement de mettre le message dans la boîte aux lettres principale de l'utilisateur. Cette commande donne un moyen d'exécuter cette action sans avoir besoin de savoir le nom de la boîte aux lettres principale de l'utilisateur, fournissant un moyen de l'appeler sans qu'il soit nécessaire de comprendre le système d'établissement de l'utilisateur ou le système de messagerie sous-jacent.

Par exemple, dans une mise en œuvre où le serveur IMAP exécute les scripts au nom de l'utilisateur au moment de la livraison, une commande "keep" est équivalente à un "fileinto" "INBOX".

Exemple : if size :under 1M { keep; } else { discard; }

Noter que le script ci-dessus est identique à celui-ci .

Exemple : if not size :under 1M { discard; }

4.4 Action "discard"

Usage : discard

"Discard" est utilisé pour éliminer en silence le message. Il le fait en annulant simplement le "keep" implicite. Si "discard" est utilisé avec d'autres actions, les autres actions se produisent. "Discard" est compatible avec toutes les autres actions. (Par exemple, fileinto+discard est équivalent à fileinto.)

"Discard" DOIT être en silence ; c'est-à-dire, il NE DOIT retourner aucune sorte de notification de non livraison ([RFC3464], [RFC3798], ou autre).

Dans le script suivant, tout message provenant de "idiot@exemple.com" est éliminé.

```
exemple : if header :contains ["from"] ["idiot@exemple.com"] {
            discard;
          }
```

Tout en étant une importante partie de ce langage, "discard" a la capacité de créer de sérieux problèmes aux utilisateurs : les étudiants qui restent connectés sur une machine non surveillée dans un laboratoire d'informatique public peuvent trouver leur script changé en juste "discard". Afin de protéger les utilisateurs dans cette situation (ainsi que dans des situations similaires) les mises en œuvre PEUVENT conserver les messages détruits par un script pendant une période infinie, et PEUVENT interdire les scripts qui éliminent tous les messages.

5. Commandes d'essais

Les essais sont utilisés dans les conditions pour décider quelle ou quelles parties de la condition exécuter.

Les mises en œuvre DOIVENT prendre en charge les essais suivants : "address", "allof", "anyof", "exists", "false", "header", "not", "size", et "true".

Les mises en œuvre DEVRAIENT prendre en charge l'essai "envelope".

5.1 Essais "address"

Usage : address [COMPARATOR] [ADDRESS-PART] [MATCH-TYPE]
<header-list: string-list> <key-list: string-list>

L'essai "address" confronte les adresses Internet dans les en-tête structurés qui contiennent des adresses. Il retourne vrai si un en-tête contient une clé dans la partie spécifiée de l'adresse, comme modifiée par le comparateur et le mot clé de confrontation. Qu'il y ait d'autres adresses présentes dans l'en-tête n'affecte pas cet essai ; cet essai ne donne aucun moyen de déterminer si une adresse est la seule adresse dans un en-tête.

Comme "envelope" et "header", cet essai retourne vrai si une combinaison des arguments "header-list" et "key-list" correspond, et retourne faux autrement.

Les adresses de messagerie Internet [RFC2822] ont la caractéristique un peu étrange que la partie locale à gauche de l'arobase est considérée comme sensible à la casse, et la partie nom de domaine à droite de l'arobase est insensible à la casse. La commande "address" ne traite pas elle-même de cela, mais fournit aux utilisateurs l'argument ADDRESS-PART pour le faire.

La primitive "address" n'agit jamais sur la partie phrase d'une adresse de messagerie ou sur les commentaires au sein de cette adresse. Elle n'agit jamais non plus sur les noms de groupe, bien qu'elle agisse sur les adresses au sein de la construction de groupe.

Les mises en œuvre DOIVENT restreindre l'essai "address" aux en-têtes qui contiennent des adresses, mais DOIVENT inclure au moins From, To, Cc, Bcc, Sender, Resent-From, et Resent-To, et il DEVRAIT inclure tout autre en-tête qui utilise un corps d'en-tête "address-list" structuré.

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

5.2 Essais "allof"

Usage : allof <tests: test-list>

L'essai "allof" effectue un ET logique sur les essais qui lui sont fournis.

```
exemple : allof (faux, faux) => faux
          allof (faux, vrai) => faux
          allof (vrai, vrai) => vrai
```

L'essai "allof" prend comme argument une liste d'essais *(test-list)*.

5.3 Essais "anyof"

Usage : anyof <tests: test-list>

L'essai "anyof" effectue un OU logique sur les essais qui lui sont fournis.

```
exemple : anyof (faux, faux) => faux
          anyof (faux, vrai) => vrai
          anyof (vrai, vrai) => vrai
```

5.4 Essais "envelope"

Usage : envelope [COMPARATOR] [ADDRESS-PART] [MATCH-TYPE]
<envelope-part: string-list> <key-list: string-list>

L'essai "envelope" est vrai si la partie spécifiée de l'enveloppe [RFC2821] (ou l'équivalent) correpond à la clé spécifiée. La présente spécification définit l'interprétation des parties d'enveloppe (insensibles à la casse) "from" et "to". Des parties d'enveloppe supplémentaires peuvent être définies par d'autres extensions ; les mises en œuvre DEVRAIENT considérer les parties d'enveloppe inconnues comme une erreur.

Si une des chaînes de la partie enveloppe est "from" (insensible à la casse) alors la confrontation se fait contre l'adresse FROM utilisée dans la commande SMTP MAIL. Le chemin inverse nul est confronté à la chaîne vide, sans considération de l'argument ADDRESS-PART spécifié.

Si une des chaînes de la partie enveloppe est "to" (insensible à la casse) alors la confrontation se fait contre l'adresse TO utilisée dans la commande SMTP RCPT qui a résulté en ce message livré à cet utilisateur. Noter que seul le TO le plus récent est disponible, et seul celui qui est pertinent pour cet utilisateur.

La partie enveloppe est une liste de chaînes et peut contenir plus d'un paramètre, et dans ce cas toutes les chaînes spécifiées dans la liste des clés sont confrontées à toutes les parties données dans la liste des parties d'enveloppe.

Comme pour "address" et "header", cet essai retourne vrai si une combinaison de la liste des parties d'enveloppe et des arguments liste des clés correspond, et retourne faux autrement.

Tous les essais contre les enveloppes DOIVENT éliminer les routes de source.

Si la transaction SMTP impliquait plusieurs commandes RCPT, seules les données provenant de la commande RCPT qui a causé la livraison à cet utilisateur sont disponibles dans la partie "to" de l'enveloppe.

Si un protocole autre que SMTP est utilisé pour le transport de message, les mises en œuvre sont supposées adapter cette commande de façon appropriée.

La commande "envelope" est facultative. Les mises en œuvre DEVRAIENT la prendre en charge, mais les informations nécessaires peuvent n'être pas disponibles dans tous les cas. La chaîne de capacités à utiliser avec la commande "require" est "envelope".

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

5.5 Essais "exists"

Usage : exists <header-names: string-list>

L'essai "exists" est vrai si les en-têtes mentionnés dans l'argument "header-names" existent dans le message. Tous les en-têtes doivent exister, sinon l'essai est faux.

L'exemple suivant élimine les messages qui n'ont pas un en-tête From et Date.

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

5.6 Essais "false"

Usage : false

L'essai "false" s'évalue toujours à faux.

5.7 Essais "header"

Usage : header [COMPARATOR] [MATCH-TYPE]
<header-names: string-list> <key-list: string-list>

L'essai "header" s'évalue à vrai si la valeur d'un des en-têtes désignés, en ignorant les espaces en tête et en queue, correspond à une clé. Le type de correspondance est spécifié par l'argument facultatif "match", qui est par défaut ":is" si il n'est pas spécifié, comme indiqué au paragraphe 2.6.

Comme pour "address" et "envelope", cet essai retourne vrai si une combinaison de la liste des en-têtes désignés et des arguments de liste de clés correspond, et retourne faux autrement.

Si un en-tête mentionné dans l'argument "header-names" existe, il contient la clé vide (""). Cependant, si l'en-tête désigné n'est pas présent, il ne correspond à aucune clé, y compris la clé vide. Donc, si un message contient l'en-tête X-Caffeine: C8H10N4O2, cet essai sur cet en-tête s'évalue comme suit :

header :is ["X-Caffeine"] [""] => faux
header :contains ["X-Caffeine"] [""] => vrai

L'essai de si un certain en-tête est absent ou ne contient aucun caractère non espace peut être fait en utilisant un essai "header" négatif :

not header :matches "Cc" "?*"

5.8 Essais "not"

Usage : not <test1: test>

L'essai "not" prend un autre essai comme argument, et donne le résultat opposé. "not false" s'évalue à "vrai" et "non vrai" s'évalue à "faux".

5.9 Essais "size"

Usage : size <":over" / ":under"> <limit: number>

L'essai "size" traite la taille d'un message. Il prend soit l'argument étiqueté de ":over" ou ":under", suivi par un nombre qui représente la taille du message.

Si l'argument est ":over", et si la taille du message est supérieure au nombre fourni, l'essai est vrai ; autrement, il est faux.

Si l'argument est ":under", et si la taille du message est inférieure au nombre fourni, l'essai est vrai ; autrement, il est faux.

Exactement un de ":over" ou ":under" doit être spécifié, et tout le reste est une erreur.

La taille d'un message est définie comme étant le nombre d'octets dans la représentation de la [RFC2822] du message.

Noter que pour un message qui fait exactement 4 000 octets, le message n'est ni ":over" ni ":under" 4 000 octets.

5.10 Essais "true"

Usage : true

L'essai "true" s'évalue toujours à vrai.

6. Extensibilité

De nouvelles commandes de contrôle, actions, et essais peuvent être ajoutées au langage. Les sites doivent faire connaître ces caractéristiques à leurs utilisateurs ; le présent document ne définit pas de moyen pour découvrir la liste des extensions prises en charge par le serveur.

Toute extension à ce langage DOIT définir une chaîne de capacités qui identifie de façon univoque cette extension. Les chaînes de capacités sont sensibles à la casse ; par exemple, "foo" et "FOO" sont des capacités différentes. Si une nouvelle version d'une extension change la fonctionnalité d'une extension définie précédemment, elle DOIT utiliser un nom différent. Les extensions peuvent enregistrer un ensemble de capacités en rapport en enregistrant juste un préfixe unique

pour elles. Le préfixe "comparator-" en est un exemple. Le préfixe DOIT se terminer par un "-" et NE DOIT PAS recouper d'enregistrement existant.

Dans une situation où il y a un protocole de soumission de script et un mécanisme d'annonce d'extension qui connaît les détails de ce langage, les scripts soumis peuvent être vérifiés auprès du serveur de messagerie pour empêcher l'utilisation d'une extension que le serveur ne prend pas en charge.

Les extensions DOIVENT déclarer comment elles interagissent avec les contraintes définies au paragraphe 2.10, par exemple, si elles annulent le "keep" implicite, et avec quelles actions elles sont compatibles ou non. Les extensions NE DOIVENT PAS changer le comportement de la commande de contrôle "require" ou altérer l'interprétation de l'argument du contrôle "require".

Les extensions qui peuvent soumettre de nouveaux messages ou autrement générer de nouvelles demandes de protocole DOIVENT prendre en compte la suppression de boucle, au moins pour documenter les considérations de sécurité.

6.1 Chaînes de capacités

Les chaînes de capacités sont normalement de courtes chaînes qui décrivent quelles capacités sont supportées par le serveur.

Les chaînes de capacités qui commencent par "vnd." représentent des extensions définies par un fabricant. Ces extensions ne sont pas définies par des normes de l'Internet ou des RFC, mais sont cependant enregistrées par l'IANA afin de prévenir les conflits. Les extensions qui commencent par "vnd." DEVRAIENT être suivies par le nom du fabricant et du produit, comme dans "vnd.acme.rocket-sled".

Les chaînes de capacités suivantes sont définies dans ce document :

encoded-character : la chaîne "encoded-character" indique que la mise en œuvre prend en charge l'interprétation de "${hex:...}" et "${unicode:...}" dans les chaînes.

envelope : la chaîne "envelope" indique que la mise en œuvre prend en charge la commande "envelope".

fileinto : la chaîne "fileinto" indique que la mise en œuvre prend en charge la commande "fileinto".

comparator- : la chaîne "comparator-elbonia" est fournie si la mise en œuvre prend en charge le comparateur "elbonia". Donc, toutes les mises en œuvre ont au moins les capacités "comparator-i;octet" et "comparator-i;ascii-casemap". Cependant, ces comparateurs peuvent être utilisés sans être déclarés avec "require".

6.2 Considérations relatives à l'IANA

Afin de fournir un ensemble standard d'extensions, un registre est tenu par l'IANA. Ce registre contient les noms de capacités contrôlées par les fabricants (commençant par "vnd.") et les noms de capacités contrôlées par l'IETF. Les capacités contrôlées par les fabricants peuvent être enregistrées sur la base du premier arrivé premier servi, sur demande à l'IANA avec le formulaire du paragraphe suivant. L'enregistrement de préfixes de capacités qui ne commencent pas par "vnd." EXIGE une RFC sur la voie de la normalisation ou une RFC expérimentale approuvée par l'IESG.

Les extensions conçues pour une utilisation interopérable DEVRAIENT utiliser des noms de capacité contrôlés par l'IETF.

6.2.1 Gabarit pour les enregistrements de capacités

Le gabarit suivant est à utiliser pour les enregistrements de nouvelles extensions Sieve auprès de l'IANA.

Pour : iana@iana.org
Sujet : enregistrement d'une nouvelle extension Sieve.
Nom de capacité : [chaîne à utiliser pour la déclaration "require"].
Description : [brève description de ce que l'extension ajoute ou change].
RFC publiée : [pour les extensions publiées comme des RFC].
Adresse de contact : [mèl et/ou adresse physique du contact pour des informations supplémentaires].

6.2.2 Traitement des enregistrements de capacités existantes

Afin de mettre en ligne les enregistrements de capacités existantes avec le nouveau gabarit, l'IANA a modifié chacun comme suit :

1. Les champs "nom de capacité" et "arguments de capacités" ont été éliminés.
2. Le champ "mot clé de capacité" a été renommé en "Nom de capacité".
3. Un champ "Description" vide a été ajouté.
4. Le champ "Numéro de RFC sur la voie de la normalisation/expérimentale approuvée par l'IESG" a été renommé en "Numéro de RFC".
5. Le champ "Personne et adresse de messagerie à contacter pour plus d'informations" devrait être renommé en "Adresse de contact"

6.2.3 Enregistrements des capacités initiales

La présente RFC met à jour les entrées suivantes dans le registre IANA pour les extensions Sieve.

Nom de capacité : encoded-character
Description : change l'interprétation des chaînes pour permettre que des octets arbitraires et des caractères Unicode soient représentés en utilisant l'US-ASCII.
RFC publiée : RFC 5228 (spécification Sieve de base).
Adresse de contact : Liste de diffusion Sieve à <ietf-mta-filtres@imc.org>

Nom de capacité : fileinto
Description : ajoute l'action "fileinto" pour la livraison à une boîte aux lettres autre que par défaut.
RFC publiée : RFC 5228 (spécification Sieve de base).
Adresse de contact : Liste de diffusion Sieve à <ietf-mta-filtres@imc.org>

Nom de capacité : envelope
Description : ajoute l'essai "envelope" pour vérifier l'adresse de transport de l'envoyeur et du receveur du message.
RFC publiée : RFC 5228 (spécification Sieve de base).
Adresse de contact : Liste de diffusion Sieve à <ietf-mta-filtres@imc.org>

Nom de capacité : comparator-* (n'importe quoi qui commence par "comparator-").
Description : ajoute le comparateur indiqué à utiliser avec l'argument ":comparator".
RFC publiée : RFC 5228 (spécification Sieve de base) et [RFC4790]
Adresse de contact : Liste de diffusion Sieve à <ietf-mta-filtres@imc.org>

6.3 Transport de capacité

Une méthode pour annoncer quelles capacités prend en charge une mise en œuvre est difficile à cause de la large gamme des mises en œuvre possibles. Un tel mécanisme devrait cependant avoir la propriété que la mise en œuvre puisse annoncer l'ensemble complet des extensions qu'elle prend en charge.

7. Transmission

Le type [RFC2045] pour un script Sieve est "application/sieve".

L'enregistrement de ce type pour les exigences de la RFC 2048 est mis à jour comme suit :

Sujet : enregistrement du type de support MIME application/sieve
Nom du type de support MIME : application
Nom de sous type MIME : sieve
Paramètres exigés : aucun
Paramètres facultatifs : aucun
Considérations de codage : la plupart des scripts Sieve vont être textuels, écrits en UTF-8. Quand des caractères non 7 bits sont utilisés, "quoted-printable" est approprié pour les systèmes de transport qui exigent le codage en 7 bits.
Considérations de sécurité : discutées à la Section 10 de cette RFC.
Considérations d'interopérabilité : discutées au paragraphe 2.10.5 de cette RFC.
Spécification publiée : RFC 5228.

Applications qui utilisent ce type de support : serveur et client de messagerie à capacité Sieve.
Informations supplémentaires :
Numéro magique : aucun
Extensions de fichier : .siv .sieve
Code de type de fichier Macintosh :
Personne & adresse de messagerie à contacter pour plus d'informations : voir la liste de diffusion à ietf-mta-filtres@imc.org.
Usage prévu : COMMUN
Auteur/contrôleur des changements : goupe de travail SIEVE, sur délégation de l'IESG.

8. Analyse

La grammaire Sieve est séparée en jetons et une grammaire séparée comme le sont la plupart des langages de programmation. Des règles supplémentaires sont fournies ici pour les arguments communs aux diverses faciltés de langage.

8.1 Jetons lexicaux

Les scripts Sieve sont codés en UTF-8. Ce qui suit suppose un codage UTF-8 valide ; les caractères spéciaux dans les scripts Sieve sont tous en US-ASCII.

Les jetons de Sieve sont les suivants :
- identifiers *(identifiants)*
- tags *(étiquettes)*
- numbers *(nombres)*
- quoted strings *(chaînes citées)*
- multi-line strings *(chaînes multi lignes)*
- autres séparateurs

Les identifiants, les étiquettes, et les nombres sont insensibles à la casse, tandis que les chaînes citées et les chaînes multi lignes sont sensibles à la casse.

Les blancs, les tabulations horizontales, les CRLF, et les commentaires ("espace") sont ignorés sauf comme séparateurs de jetons. Des espaces sont exigées pour séparer des jetons par ailleurs adjacents et dans des endroits spécifiques dans les chaîne multi lignes. CR et LF peuvent seulement apparaître dans des paires CRLF.

Les autres séparateurs sont des caractères individuels et sont mentionnés explicitement dans la grammaire.

La structure lexicale de Sieve est définie dans la grammaire suivante (décrite dans la [RFC4234]) :

```
bracket-comment  = "/*" *not-star 1*STAR
                   *(not-star-slash *not-star 1*STAR) "/"
; aucun */ n'est permis dans un commentaire.; (aucune * n'est permise sauf comme dernier caractère, ou si il est suivi par un caractère autre qu'une barre oblique.)

comment = bracket-comment / hash-comment

hash-comment = "#" *octet-not-crlf CRLF

identifier = (ALPHA / "_") *(ALPHA / DIGIT / "_")

multi-line = "text:" *(SP / HTAB) (hash-comment / CRLF)
                   *(multiline-literal / multiline-dotstart) "." CRLF

multiline-literal  = [ octet-not-period *octet-not-crlf ] CRLF

multiline-dotstart = "." 1*octet-not-crlf CRLF
; ligne contenant seulement "." qui termine la multi-ligne. Supprimer un '.' en tête si il est suivi par un autre '.'.

not-star = CRLF / %x01-09 / %x0B-0C / %x0E-29 / %x2B-FF
; une paire CRLF , OU un seul octet autre que NUL, CR, LF, ou étoile.
```

not-star-slash = CRLF / %x01-09 / %x0B-0C / %x0E-29 / %x2B-2E / %x30-FF
; une paire CRLF, OU un seul octet autre que NUL, CR, LF, étoile, ou barre oblique.

number = 1*DIGIT [QUANTIFIER]

octet-not-crlf = %x01-09 / %x0B-0C / %x0E-FF
; un seul octet autre que NUL, CR, ou LF.

octet-not-period = %x01-09 / %x0B-0C / %x0E-2D / %x2F-FF
; un seul octet autre que NUL, CR, LF, ou point.

octet-not-qspecial = %x01-09 / %x0B-0C / %x0E-21 / %x23-5B / %x5D-FF
; un seul octet autre que NUL, CR, LF, guillemets, ou barre oblique inverse.

QUANTIFIER = "K" / "M" / "G"

quoted-other = "\" octet-not-qspecial
; représente juste le caractère octet-no-qspecial. NE DEVRAIT PAS être utilisé.

quoted-safe = CRLF / octet-not-qspecial
; une paire CRLF, OU un seul octet autre que NUL, CR, LF, guillements, ou barre oblique inverse.

quoted-special = "\" (DQUOTE / "\")
; représente juste un caractère guillemets ou une barre oblique inverse.

quoted-string = DQUOTE quoted-text DQUOTE

quoted-text = *(quoted-safe / quoted-special / quoted-other)

STAR = "*"

tag = ":" identifier

white-space = 1*(SP / CRLF / HTAB) / comment

8.2 Grammaire

Voici la grammaire de Sieve après son interprétation lexicale. Aucune espace ni commentaire n'apparaît ci-dessous. Le symbole étoile est "start" *(début)*.

argument = string-list / number / tag

arguments = *argument [test / test-list]

block = "{" commands "}"

command = identifier arguments (";" / block)

commands = *command

start = commands

string = quoted-string / multi-line

string-list = "[" string *("," string) "]" / string
; si il y a seulement une chaîne, les crochets sont facultatifs.

test = identifier arguments

test-list = "(" test *("," test) ")"

8.3 Éléments de déclaration

Ces éléments sont collectés des paragraphes de "Syntaxe" dans ce document, et sont fournis ici dans la syntaxe de la [RFC4234] de sorte qu'ils peuvent être modifiés par des extensions.

ADDRESS-PART = ":localpart" / ":domain" / ":all"

COMPARATOR = ":comparator" string

MATCH-TYPE = ":is" / ":contains" / ":matches"

9. Exemple étendu

Voici un exemple étendu de script Sieve. Noter qu'il n'utilise pas de "'keep" implicite.

```
# Exemple de filtre Sieve. Déclare toutes les caractéristiques ou extensions facultatives utilisées par le script.  #

   require ["fileinto"];

# Traite les messages provenant de listes de diffusion inconnues. Déplace les messages de la liste de discussion de filtre de
l'IETF à la boîte au lettres filtre.  #

   if header :is "Sender" "owner-ietf-mta-filtres@imc.org"
         {
         fileinto "filtre";                                    # déplacer dans la boîte aux lettres "filtre".
         }
# Garder tous les messages de ou vers des gens de ma société. #

   elsif address :DOMAIN :is ["From", "To"] "exemple.com"
         {
         keep;                                                 # garder dans la boîte aux lettres "In".
         }

# Vérifier et capturer les messages non sollicités. Si un message n'est pas pour moi ou si il contient un sujet connu pour être
un pourriel, l'éliminer. #

   elsif anyof (NOT address :all :contains
                ["To", "Cc", "Bcc"] "me@exemple.com",
               header :matches "subject"
                ["*make*money*fast*", "*university*dipl*mas*"])
         {
         fileinto "spam";                                      # déplacer dans la boîte aux lettres "spam".
         }
   else
         {
# Passer tous les autres messages (pas de la société) à la boîte aux lettres "personnel". #
         fileinto "personnel";
         }
```

10. Considérations sur la sécurité

Les utilisateurs doivent obtenir leurs messages. Il est impératif que quoi que les mises en œuvre utilisent pour mémoriser les scripts de filtrage définis par l'utilisateur pour les protéger contre la modification non autorisée, elles préservent l'intégrité du système de messagerie. Un attaquant qui peut modifier un script peut causer l'élimination, le rejet, ou la transmission à un receveur non autorisé des messages. De plus, il est possible que les scripts Sieve exposent des informations confidentielles, comme des noms de boîte aux lettres, ou des adresses de messagerie de correspondants favorisés (ou défavorisés) .À cause de cela, les scripts DEVRAIENT aussi être protégés d'une restitution non autorisée.

Plusieurs commandes, comme "discard", "redirect", et "fileinto", permettent d'entreprendre des actions qui sont potentiellement très dangereuses.

L'utilisation de la commande "redirect" pour générer des notifications peut aisément submerger l'adresse cible, en particulier si elle n'a pas été conçue pour traiter de gros messages.

Permettre à un seul script de rediriger sur plusieurs destinations peut être utilisé comme moyen d'amplifier le nombre de messages dans une attaque. De plus, si la détection de boucles n'est pas mise en œuvre de façon appropriée, il serait possible d'établir des boucles de message à croissance exponentielle. En conséquence, les mises en œuvre de Sieve :

(1) DOIVENT mettre en œuvre des facilités pour détecter et casser les boucles de messages. Voir au paragraphe 6.2 de la [RFC2821] des informations supplémentaires sur les stratégies de détection de boucles de messages.

(2) DOIVENT fournir aux administrateurs des moyens pour limiter la capacité des utilisateurs à abuser de la redirection. En particulier, il DOIT être possible de limiter le nombre de redirections qu'un script peut effectuer. De plus, si aucun cas d'utilisation n'existe pour la redirection sur plusieurs destinations, cette limite DEVRAIT être réglée à 1. Des limites supplémentaires, comme la capacité de restreindre la redirection aux utilisateurs locaux, PEUVENT aussi être mises en œuvre.

(3) DOIVENT fournir des facilités pour enregistrer l'utilisation de redirections afin de faciliter le traçage des abus.

(4) PEUVENT utiliser l'analyse des scripts pour déterminer si un certain script peut ou non être exécuté en toute sécurité. Bien que le langage Sieve soit suffisamment complexe pour que l'analyse complète de tous les scripts possibles soit computationnellement infaisable, la majorité des scripts sont dans la réalité susceptibles d'analyse. Par exemple, une mise en œuvre pourrait permettre que les scripts qu'elle a déterminés comme sûrs soient utilisés sans entrave, les scripts de bloc qui sont potentiellement problématiques, et soumettre les scripts inclassables à un examen supplémentaire et un enregistrement.

Permettre même les redirections peut n'être pas approprié dans des situations où les comptes de messagerie sont librement disponibles et/ou non traçables pour une personne qui peut être tenue pour responsable de la création de bombardements de messages ou autres abus.

Comme avec tout filtre sur un flux de messages, si la mise en œuvre de Sieve et les agents de messagerie "derrière" Sieve dans le flux de messages diffèrent dans leur interprétation des messages, il se peut qu'un attaquant subvertisse le filtre. On notera en particulier qu'il y a des différences dans l'interprétation des messages mal formés (par exemple, caractères syntaxiques manquants ou supplémentaires) ou ceux qui présentent des cas particuliers (par exemple, des octets NUL codés via la [RFC2047]).

11. Remerciements

Le présent document a été révisé en partie sur la base des commentaires et des discussions qui ont eu lieu sur la liste de diffusion SIEVE. Merci à Sharon Chisholm, Cyrus Daboo, Ned Freed, Arnt Gulbrandsen, Michael Haardt, Kjetil Torgrim Homme, Barry Leiba, Mark E. Mallett, Alexey Melnikov, Eric Rescorla, Rob Siemborski, et Nigel Swinson de leur relecture et de leurs suggestions.

12. Références normatives

[RFC2045] N. Freed et N. Borenstein, "Extensions de messagerie Internet multi-objets (MIME) Partie 1 : Format des corps de message Internet", novembre 1996. *(D. S., MàJ par* 2184, 2231, 5335.)

[RFC2047] K. Moore, "MIME (Extensions de messagerie Internet multi-objets) Partie trois : extensions d'en-tête de message pour texte non ASCII", novembre 1996. (*MàJ par* RFC2184, RFC2231) *(D.S.)*

[RFC2119] S. Bradner, "Mots clés à utiliser dans les RFC pour indiquer les niveaux d'exigence", BCP 14, mars 1997. *(MàJ par* RFC8174)

[RFC2821] J. Klensin, éditeur, "Protocole simple de transfert de messagerie", STD 10, avril 2001. *(Obsolète, voir* RFC5321)

[RFC2822] P. Resnick, "Format de message Internet", avril 2001. *(Remplace la* RFC0822, STD 11, *Remplacée par* RFC5322)

[RFC3629] F. Yergeau, "UTF-8, un format de transformation de la norme ISO 10646", STD 63, novembre 2003.

[RFC4234] D. Crocker et P. Overell, "BNF augmenté pour les spécifications de syntaxe : ABNF", octobre 2005. *(Remplace* RFC2234, *remplacée par* RFC5234)

[RFC4790] C. Newman et autres, "Registre de collation des protocoles d'application de l'Internet", mars 2007. *(P.S.)*

13. Références pour information

[CEI60027-2] Norme CEI 60027-2, "Lettres symboles à utiliser dans la technologie électrique - Partie 2 : télécommunications et électronique", janvier 1999.

[FLAMES] Borenstein, N, and C. Thyberg, "Power, Ease of Use, and Cooperative Work in a Practical Multimedia Message System", Int. J. of Man-Machine Studies, avril 1991. Republié dans Computer-Supported Cooperative Work and Groupware, Saul Greenberg, éditeur, Harcourt Brace Jovanovich, 1991. Republié dans Readings in Groupware and Computer-Supported Cooperative Work, Ronald Baecker, éditeur, Morgan Kaufmann, 1993.

[RFC3028] T. Showalter, "Sieve : Langage de filtrage de messagerie", janvier 2001. *(Obsolète, voir la* RFC5228) (P.S.)

[RFC3464] K. Moore, G. Vaudreuil, "Format extensible de message pour les notifications d'état de livraison", janvier 2003. (*MàJ par* RFC4865, RFC5337, RFC6533) *(D.S.)*

[RFC3501] M. Crispin, "Protocole d'accès au message Internet - version 4rev1", mars 2003. *(P.S. ; MàJ par* RFC4466, 4469, 4551, 5032, 5182, 7817, 8314, 8437, 8474 ; *remplacée par la* RFC9051

[RFC3798] T. Hansen et G. Vaudreuil, éd., "Notification de disposition de message", mai 2004. (*MàJ par* RFC5337, RFC6533) *(D.S.; Rendue obsolète par* RFC8098)

14. Changements par rapport à la RFC 3028

La liste suivante résume les changements apportés à la spécification de base du langage Sieve de la [RFC3028].

1. Suppression de l'interdiction des essais ayant des effets collatéraux.
2. Sppression de l'extension "reject" (qui sera spécifiée dans une RFC séparée).
3. Précisé la description des comparateurs pour correspondre à la [RFC4790], leur nouvelle spécification de base.
4. Exige la suppression des espaces en tête et en queue dans l'essai "header".
5. Précise ou renforce le traitement de nombreux éléments mineurs, incluant :
   - codage invalide [RFC2047],
   - adresses invalides dans les en-têtes,
   - noms de champ d'en-tête invalides dans les essais,
   - résultat de comparateur "indéfini",
   - parties d'enveloppe inconnues,
   - chemin de retour nul dans l'essai "envelope".
6. Les chaînes de capacités sont sensibles à la casse.
7. Précisé que "fileinto" devrait recoder les noms de boîte aux lettres non ASCII pour satisfaire les conventions de mémorisation de boîte aux lettres.
8. Correction des erreurs dans l'ABNF.
9. Mise à jour des références et partage en références normatives et pour information.
10. Ajout de la capacité "encoded-character" et l'utilisation d'octets binaires arbitraires est déconseillée (mais pas supprimée) dans les scripts Sieve.
11. Mise à jour du gabarit d'enregistrement IANA, et ajout des considérations relatuives à l'IANA pour permette les enregistrements de préfixe de capacités.
12. Ajout de ".sieve" comme extension valide pour les scripts Sieve.

Adresse des éditeurs

Philip Guenther
Sendmail, Inc.
6425 Christie St. Ste 400
Emeryville, CA 94608
mél : guenther@sendmail.com

Tim Showalter
mél : tjs@psaux.com

Déclaration complète de droits de reproduction