# The Sieve Language and a General Model for Delivery and Interoperable Filtering in Internet Mail


Matthew Wall
wall@cyrusoft.com

v. 2.00 • February, 2001

Cyrusoft International, Inc.
5001 Baum Blvd.
Pittsburgh, PA 15213
http://www.cyrusoft.com

## Contents



## Overview

This white paper is an introduction to the *Sieve Mail Filtering Language*, an explication of the need for it, and a proposal for an operating model for its application.

This paper covers four topics:

- a general model for the concept of *delivery* as a distinct architectural stage in Internet Mail;
- the relationship of filtering mechanisms to this concept of delivery;
- the need for an interoperable, standardized filtering language; and
- a description of the Sieve filtering language, with some suggestions as to how to use it in the proposed model of mail delivery.

This is not meant to be an implementer's guide to Sieve, nor a definitive model document for its use. Rather, it is meant as background information to help the reader understand the motivation behind its development and its anticipated application, and as a first introduction to the Sieve concept.

## Filters and Delivery: Some Definitions

Mail *filters* are a part of almost every Internet mail system. In the basic understanding that most end users have of the concept, a filter is a two-part beast.

The first part is a test, usually set up manually by an end user, involving comparisons of textual values in mail message headers or bodies against a pre-defined list of potential matches.

Practically, this test-match is useful only when paired with the second part of a filter (again, by popular understanding within the context of a mail system), a consequent action: changing the destination folder of a message, rejecting the message, supplying an automated reply, or similar processes. There are dozens of useful potential actions, but there are four very common ones: filing a message into a specified folder; rejecting or discarding a message; generating a reply to a message (such as a 'vacation' program); and re-directing a message to a third party.

We'll talk about the current state of mail filtering in internet mail a bit further below.

Mail filtering is most usually (but not always) done at the stage of *delivery* in the lifecycle of an email message. Our contention is that in order to do the things users and administrators need to be able to do with delivery, a standardized basic mail filtering language is necessary. Understanding some nuances of delivery is helpful in describing why there should be a linkage between the concepts.

### *Delivery as a Layer*

The basic premise of this paper is that *delivery* is a separate layer in an abstract model of an Internet mail system or architecture with its own distinct functional purposes and operational requirements. While this may seem stunningly obvious, there's a subtle distinction to be made between the usual concept of delivery as a *process*, or more to the point an incidental part of the mail transport process, and our concept of delivery as an *event*. As an event, one of the aspects of delivery is to trigger potential filtering operation, and it is desirable (as we'll see below) to approach this using an interoperable standard, much as mail transport and end-user access use interoperable standards.

Most discussions of how Internet mail works on an abstract layer focus on two aspects of the end-to-end process of generating a message on the sender's side to reading it (and disposing of it) on the recipient's side. The two basic software elements of this process are encapsulated in the concepts of a *Mail Transport Agent* (MTA) and a *Mail User Agent* (MUA). In the simplest, long-standing model of Internet mail delivery, an MUA creates an RFC822-format message, submits it to an MTA that is compliant with the SMTP (Simple Mail Transfer Protocol, or RFC821) standard, which in turn transfers it dynamically over the internet to another SMTP server for delivery (or fails to do so, and notifies the sending agent of this fact).

By *delivery*, we usually mean the distinct moment at which an Internet mail message comes to "rest" by being successfully written to a mailstore, almost always in some form on disk. In the basic model above, delivery has traditionally meant simply writing out the message in a local mailbox file format for later inspection and disposition by the end user. When most internet email was, in the original instantiation of this model, simply stored in common file systems on host-based operating systems – such as in standard mailbox format on Unix -- host-based MUAs were relegated to simple examination of those contents as part of a real-time interactive login session to the host. In other words, the SMTP server handled delivery, and the end user's mail program – and by extension the end user – had nothing to do with the manner of delivery.

### *Models for Internet Mail and Delivery*

Delivery is a tricky concept, though, since these days a message can be reintroduced into the transport system, and delivered and re-delivered.

The original Interactive Mail Access Protocol (IMAP), which has evolved into the sophisticated Internet Message Access Protocol version 4 (now IMAP4rev1), was the first standard internet

protocol to develop a working model in which a client might access a mail message without a direct interactive terminal session. “Delivery” under the original IMAP-based internet mail model wasn’t changed per se, but the additional element of a client program remote to the file system on which the user’s mail is stored did complicate the overall process of modifying delivery, as well as introducing the possibilities of new functionality via extended delivery system (as we’ll discuss below).

The Post Office Protocol (aka POP, now described by the POP3 standard) introduced an even simpler model for the “access” layer by a remote client program, in which the client program initiates a one-way, one-time request to transfer the mail from a temporary delivery/storage point on a mail storage system to the computer on which the MUA resides and is used. The ‘interactive’ portion of accessing mail was limited to what amounts to a re-delivery. POP3 essentially divided the process of the mail getting to the end user into two distinct delivery events: one at the receiving SMTP server’s end of things, the other on the user’s local disk file system.

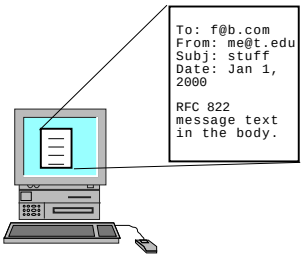
Transfers of mail messages within either the POP or IMAP-based mail-access model – copying, moving, or marking them for deletion – might be considered another instance of ‘delivery’, in that they take a message to a new ‘resting’ point. This is somewhat beyond the original model of delivery, and has additional implications for filtering of the delivery of those messages, as we’ll see. The IMAP protocol actually allows protocol-based transfer of messages between IMAP servers and separate mailstores without going through an SMTP-based MTA. This feature can be used within a given site or server for “bulletin board”- style applications, but can potentially be used for interactive direct transfer between two completely different hosts, bypassing the usual delivery checks and balances of SMTP.

Web-based mail clients, which are already extremely popular for many ‘free’ email sites and are growing in popularity for enterprise use, are another interesting case in point. The sending and receiving of mail from the user’s perspective is frequently done via CGI applications, which have their own requirements for checking validity of mail and so forth. The filter capabilities commonly available are also usually constructed by html-forms, CGI-, or other web-mediated forms of building up filter sets and rules. The point at which these rules are generated, syntax-checked, and executed may vary, and frequently there may be reasons why the filters should be modified by external agents (for instance, an automated agent that checks filters to enforce site policy, either incoming or outgoing, or displays an ad based on a user’s profile). In this case, it may be more difficult to isolate ‘delivery’ as a distinct event related to the state of the message, so much as the event of the user’s access.

The evolution in recent years of IMAP to the point where IMAP servers are now a practical option for primary management of, and access to, messages on an enterprise-scale also raises the stakes for filtering tasks, adding in the element of organizational priorities and requirements. Because a message is stored on a server, and can be moved from container to container, from local disk cache back to the server, and even within (by sequence) a given mail folder, the initial delivery may be only the first of many changes in state of a message. Conversely, in a pure on-line model of IMAP, the requirements for filtering may apply without transfer of a message at all: messages might be “filtered” for transfer to an IMAP client using IMAP’s partial download capabilities, without actually affecting a permanent transfer.

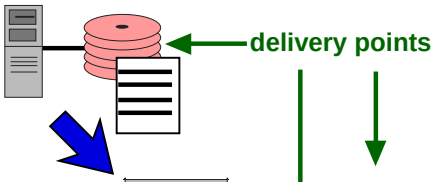
A simple case in point is the ability of some IMAP servers to understand *subaddressing*. A subaddress is an additional part to the left-hand-side of an RFC822-formatted address, related to a recipient ID but separated from it by a token. An example might be wall+spam@cyrusoft.com –the addressee is wall@cyrusoft.com, but the destination is one of user wall’s folders called ‘spam’. The Cyrus IMAP server from Carnegie Mellon (and many of its commercial descendants) supports this form of re-routing for both personal mailbox space and for public delivery (via a separate prefix, ‘bb’, followed by the foldername). One practical application is to deliver mail from mailing lists directly into a folder specifically for that list. Such servers essentially break out a distinction between users of mail systems and folders within the mail system to which one or more users may have access; the process of determining the mail folder is really a form of filtering at delivery time inherent to the specific server. (See [MURCHISON] for a description of sub-addressing and a proposed Sieve-based treatment as a practical exemplar.)

1. Originating MUA composes message

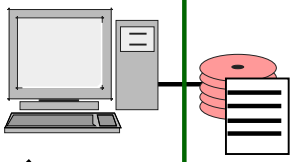



2. Message is submitted to SMTP server

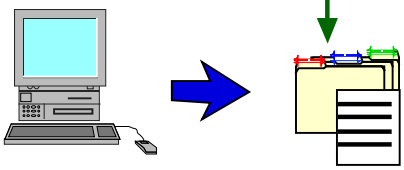
3. Message arrives at receiving SMTP server




4. SMTP server decides where to deliver it



5. Recipient's MUA access the message



6. Recipient's MUA files the message on a local disk store

Figure 1: Delivery in Internet Mail
Each point at which a message comes to "rest" might be considered a "delivery point"

Delivery, therefore, can be more accurately viewed as a distinct action that can occur multiple times throughout the "life" of a message rather than a single event. Architecturally speaking, this element can be layered onto three distinct processes: SMTP-MTA delivery to mailstore, transfer from a temporary mailstore to a remote client agent (as with POP and purely offline-IMAP), and at either entrance to or exit from a more permanent mailstore (as with IMAP mailbox delivery) (see figure 1 for a diagram).

### Filters and Internet Mail Now

At present, the end user's mail client application software is by far the most common layer at which mail filters are used and seen, especially in POP3-based mail application programs. The filter model is fairly simple: users set up the tests with dialogues that allow them to specify search criteria based on the values of strings that are to map onto specific parts of a standard mail messages. The examination of the message parts, and evaluation of the tests, is done directly by the client application program at the time messages are retrieved from a remote server or otherwise examined by the program. The resulting actions that are possible as a result of a test-criterion match are wholly defined by the capabilities of the client mail program, and limited in scope to what data and actions the mail program may access. In practice, this usually means either simple transfer of the mail message to a specific local disk mail folder, or possibly submission of a new message to the mail transport system, if in fact the mail transport system is available at that time.

Filters, however, actually operate at many layers.

Most SMTP-based mail transport agents have capabilities for evaluating aspects of a message and rejecting them, if the local mailstore cannot accept them or if they're in the wrong format, or for other (syntactic) technical reasons. SMTP-based mail servers also provide an increasingly-important layer for prevention of delivery of mail based on the semantic implications of the message, such as unsolicited email, (aka junkmail or spam), or for more sophisticated content-based rejection of messages inappropriate for final delivery to an end user (as of certain forms of electronic communication to minors, for example).

A recent but increasingly important development in the mail tools market is a category of product specifically intended to filter *outgoing* mail by content. Some organizations do this to try to prevent proprietary or private information from being leaked out of the organization; a more altruistic reason to have such a mechanism is to prevent the spread of MIME-based mail viruses.

There are also existing tools for users to specify filters outside of the scope of their mail client, such as the popular Procmail program, which is run in between the SMTP server and the final storage of mail on many time-sharing systems. There are many reasons for wishing to do this. Users not using a POP-based mail client may wish to have the same level of functionality that POP-style filters provide. Administrators may wish to apply filtering criteria for a very specific group of users, criteria that ought not apply to the general population of users, and are thus not appropriate as generic SMTP delivery filters. Even users of a POP system may wish to pre-sort their mail, to avoid the delivery costs (in network bandwidth, processing time, disk use, and most importantly, the time of the end user) of downloading messages the user has utterly no interest in receiving. Alternatively, POP users might wish to have messages filtered and actions made based on those filters without the requirement they actually perform a POP-mail check themselves: for instance, "vacation" processing, to tell correspondents the user is not actually reading his or her mail.

There are also instances where a form of filtering is used within a mail system, after initial delivery of the message to the user's inbox. For instance, a user or administrator might wish to "cull" mail older than a certain date from an IMAP-based mail system to clear out disk space, or do a periodic check of certain MIME parts for viruses. Any administrative tool performing tests and actions in this manner is doing a form of filtering -- in these examples, non-delivery-time filtering.

So, far from just something an MUA does for the convenience of a user, filters can and should operate at multiple layers in a mail system for a variety of purposes. (One might note a certain parallelism to the different delivery events, although not a complete one-to-one-mapping.)

Generally speaking, there are thus (at least) three major filtering models, which can work independently or in conjunction with one another.

- Client-side filtering, in which the end-user's mail user agent establishes the filtering criteria, stores them, and executes the filters at the time the client is used to access specific message parts. One way of describing this form of filtering is that it is *synchronous* with the point at which the end user accesses the mail.
- Server-side filtering, in which filters are stored and executed by the mail server system, usually at the time of initial delivery by the SMTP server. From the user's perspective, this is generally an *asynchronous* event.
- *Filtering by Proxy* (this is our coinage), describing a filtering activity which is performed on behalf of a user or mail user agent by a third party independent of, or not required for, initial delivery or access. We use this to describe the general category of administrators or their software agents doing filtering of mail that is not necessarily end-user directed, and which may happen during the process of transport as an extra layer or gateway, or after-the-fact.

Figure 2 illustrates where filtering occurs in the previously-described delivery model.

### The Need for a Standard Filtering Language

Each of these filtering systems and layers may work well enough in isolation – otherwise, of course, they wouldn't be used now. But virtually every filtering scheme is idiosyncratic to a specific software tool, and the scope of these software tools is almost always limited to a specific functional problem. This means the filter, once set up, can't be re-used by other programs and frequently not even by other users of the same general system.

There are also operational problems specific to each kind of filtering. For instance, in a Procmail-style server-side filtering system, users have to know how to login to a host system and edit text files, as well as understand the syntax of a specific language. In client-side filtering systems, the same filter to sort out a single spammer might have to be re-constructed by thousands of users. It would be nice if common filters could be more easily shared among users of disparate components. And if a user changes mail clients, they might have to re-create literally hundreds of filters. Even if re-created, there's no guarantee the new filters will be a semantic or functional match to the old ones, since the new filtering mechanism probably will have different syntax and structure.

With increasing volumes of mail, there is also the question of the "right fit" for a mail filtering agent. Filtering at the level of an end user requires that the delivery and transport system absorb the overhead of initial storage and transfer to the end user, and the end user's own system (typically a desktop computer system, but possibly a PDA, pager/cellphone, or other 'thin' device) must bear the computational and data handling burden of doing the processing of the filtering action. This is not much of a problem for a few messages, but if each user receives hundreds of messages a day, and might reject half of them, the mail agent is performing far more work than it needs to. There's also the issue of whether users can keep up with the pace of mail filtering requirements, or generally have the sophistication to set up what amount to elementary scripts or programs, in the instances where they must build and maintain client-side filters. The more complex logical constructs for advanced filtering are out of reach to the majority of users, simply because they require detailed understanding of both mail syntax and first-order logic.

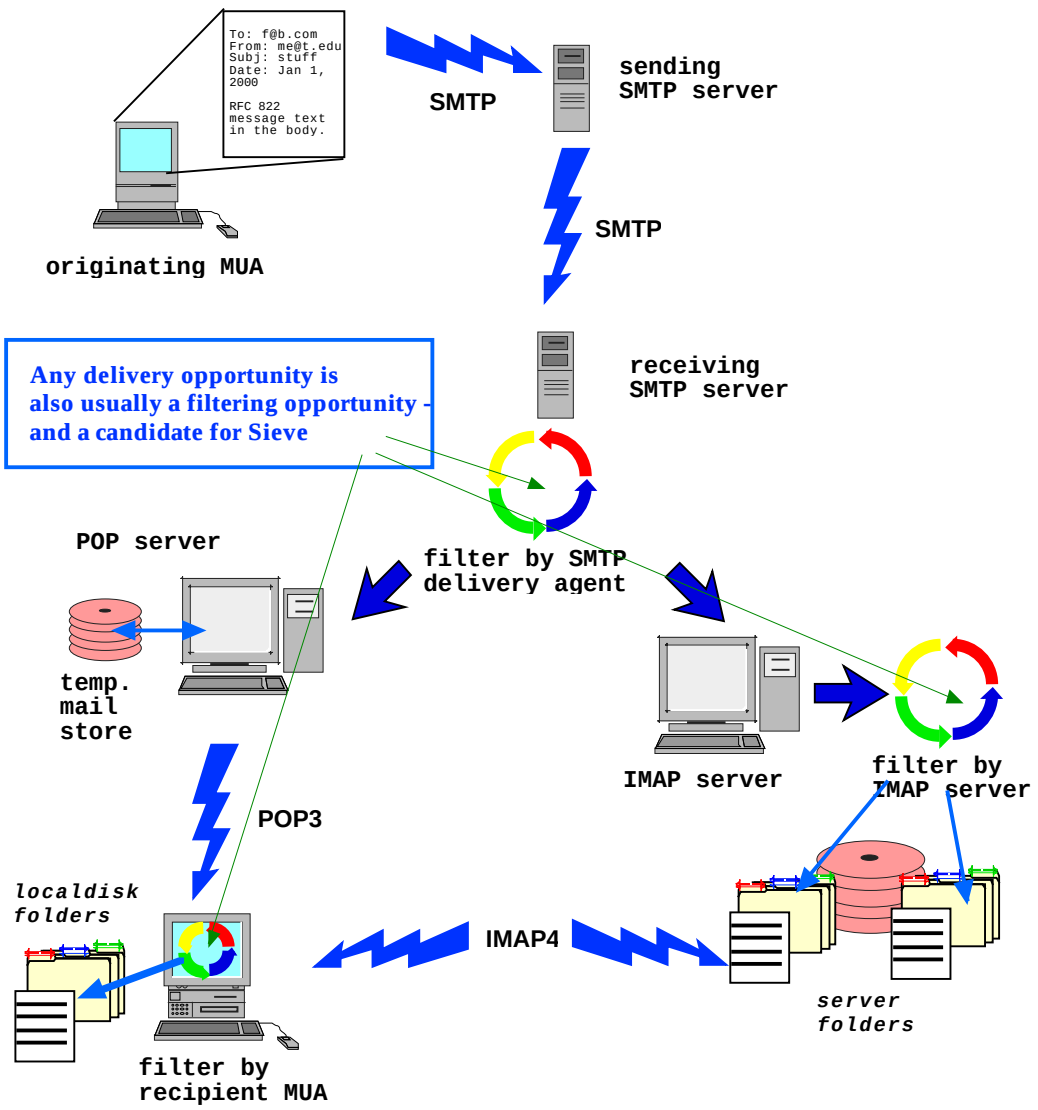



figure 2: filtering in Internet Mail

Aspects of these problems could be significantly addressed by having a standard, interoperable syntax for filtering internet mail messages. By having a common core syntax, filters could be created once and then shared and re-used among applications, users and administrators, and re-applied at many different potential layers. By having more individuals, administrators, and software developers focussing on creating tools using a common language, it should be possible for a larger and more sophisticated body of tools to progress, instead of energy being expended on a panoply of different syntaxes and implementations.

## The Sieve Mail Filtering Language

*Sieve* is a mail filtering language that has been developed as a proposed candidate for an internet standard mail filtering syntax. It is not meant to be an exclusive remedy or mechanism, but rather the basic *lingua franca* on which to build and begin to integrate interoperable filtering elements at various layers of an internet mail system.

The basic concepts for Sieve have been discussed in standards-engineering circles since 1993. In 1995, as part of Carnegie Mellon University's Project Cyrus, the first formal attempt to specify what ultimately became Sieve was made. Since then, through open discussion and development in the manner of the Internet Engineering Task Force, the evolution of Sieve has become a multi-vendor effort.

As of tJanuary, 2001, Sieve is now aProposed Internet Standard, described in RFC 3028 (see [SHOWALTER] for details. Multiple implementations of Sieve exist on both client and server, and some implementations are being used in a production capacity, including Mulberry 2.1 from Cyrusoft.

### *Requirements for Sieve*

What are the requirements, then, for a standardized mail filtering language? The experience and contributions from adminstrators, end users, and developers of both client and server products in specifying Sieve suggests the following key considerations.

- For internet email, it first and foremost must be specifically designed for RFC822-formatted messages.
- The language must be independent of the event at which it is to be used. In other words, it must be implementable on either a mail client or mail server, but dependent on neither in particular.
- It must be usable by both knowledgeable users (e.g. mail system administrators) and "normal" end users, and of course executable by software elements appropriate to the given user.
- It must provide the most basic, common-across-the-board functions of a mail filtering system to provide the desired lingua franca.
- It must provide an extensibility mechanism to provide new functionality and possible compatibility with more advanced mail filtering functions.
- It must not make the problems filtering is attempting to solve worse, or introduce new problems of its own.

There are some additional features that, while not absolutely mandatory, are highly desirable in a standardized filtering language:

- Its syntax and structure should be amenable to construction with a graphical user interface, and where possible, relatively easy readability by humans.
- The syntax should be "safe", as experience with computationally-complex languages has shown them to be a potential security nightmare when attached to a mail system. For the purposes of designing Sieve, we have interpreted this to mean that the language should not have loops, function calls, or variables, but rather be a linearly-interpreted script.
- The language should not be overly complex. The experience of developing new internet standards strongly suggests starting with simple solutions and building on them, rather than trying to develop a completely comprehensive system from scratch.

In short, the language should be specifically tuned to usefulness in the specific mail filtering context, should be simple, but should be open-ended enough so that it can be gradually built up and 'evolved' as time and experience dictate.

### *What Sieve is Not*

Sieve is not intended as a complete or complex computer language. It is specifically and narrowly defined for the purpose of filtering RFC822 messages and nothing else.

Note that Sieve does not technically require SMTP as the delivery mechanism: it speaks only to message format. It is perfectly possible that Sieve can be used outside of a mail transport system; for instance, for the post-processing of messages in RFC822 format that have been saved to local disk. However, the vast majority of applications of Sieve will be in an "active" mail transport and delivery sequence.

Sieve is not intended as a final solution for any particular mail management problem, such as spam, although it clearly has some application for specific problems.

Sieve does not currently address content-based filtering of message bodies. Future extensions to or revisions of the Sieve base specification may do so.

Sieve is intended to be used primarily for 'incoming' messages, although it is possible to fruitfully use its syntax (most likely in the context of future extensions) for 'outgoing' filtering of mail.

Sieve is transport-independent; while we've suggested some models for its use below, these are our particular opinions, not dogma. Sieve is flexible enough to be used in a variety of scenarios.

Sieve will probably not be an out-of-the-box replacement for any particular piece of software. There are many filtering implementations extant, most of which have capabilities that presently exceed those of the base Sieve syntax. At the same time, with the extensibility of Sieve, it is suggested that future modifications and extensions of existing products might be done with future compatibility with Sieve engines in mind.

The Sieve model is not either a prescriptive nor a proscriptive one, in that Sieve-compatible implementations can co-exist with non-standard filtering schemes, and Internet mail will continue to work.

### *The Sieve Language*

The Sieve language is formally described in [SHOWALTER]. This document is sufficient basis on which to develop a full implementation, although its specific use is not prescribed normatively. We'll suggest some – but by no means all – possible models for a working Sieve system in the next section. This section is intended as a very brief introduction to the basic language concepts.

Sieve is a simple ordered set of commands, represented by lines. The language is represented in UTF-8.

Sieve has the usual basics of a scripting language, such as a facility for comments, the capability for multi-line commands, and so forth.

Sieve recognizes the standard parts of an Internet Mail message: headers, addresses, and distinct MIME parts. Sieve does not have a facility for evaluating the body of messages; this is on purpose, to avoid many complications in a required implementation, but adding body comparators is envisioned as a future extension of the syntax.

Tests are given as arguments to commands, using a comparator. Sieve allows for the use of multiple comparators, but defaults to octet comparisons using ASCII unless otherwise specified. A proposed sieve extension (see [MURCHISON2]) provides for regular expressions.

There are ten tests defined in the base Sieve syntax: basic boolean tests, a size test, and other tests more specific to mail message syntax such as subparts of the message header. These can be used in combination. For example, Sieve syntax allows one to simultaneously test if any header contained the words 'Make Money Fast' or 'Get thin now!' or is sent from `'boss@yourcompany.com'`, and perform a single appropriate action.

There are three basic control structures. There is a simple `if-elseif-else` case conditional control; a structure which requires the use of certain extensions to the language before executing a command; and a `stop` primitive to manually halt execution. Control structures execute on blocks of text as arguments, which are enclosed in curly braces.

There are five basic actions for disposing of a message: `reject`, `fileinto`, `redirect`, `keep`, and `discard`. The `keep` action is defined by each implementation as the default action for a new message, most typically placing it in the user's main inbox. The `reject` action sends back a Message Delivery Notification is optional to implement (though strongly suggested). Similarly, `fileinto` is also optional (though also strongly suggested), in that not all mail systems have the concept of more than one possible location in which to file mail.

Extensions to the syntax are invoked by the use of the command `require` in a script. Extensions can include new control structures, actions, and tests. Servers supporting specific extensions provide a capability advertisement at execution time. Execution of a script is stopped if a required extension is not present.

The basic form of a Sieve command is typically an '`if`' control structure, with a message part (and perhaps subpart tag) specified, a test, and an action in curly braces. A full Sieve script can and probably will contain many such commands. Execution implicitly stops at the end of a script or file, or via an explicit `stop` control.

Some basic examples of Sieve scripts follow.

**Example 1**

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

This example tests to see if the total message size is over 100K, and if it is, the message is permanently discarded – i.e. it's not filed into the user's INBOX or any other place.

**Example 2**

```
if header :contains :comparator "i;octet" "Subject"
                "MAKE MONEY FAST" {
                   discard;
            }
```

This is a case-sensitive test to see if the Subject line contains the phrase 'MAKE MONEY FAST' – in all caps – and if it does, to delete it.

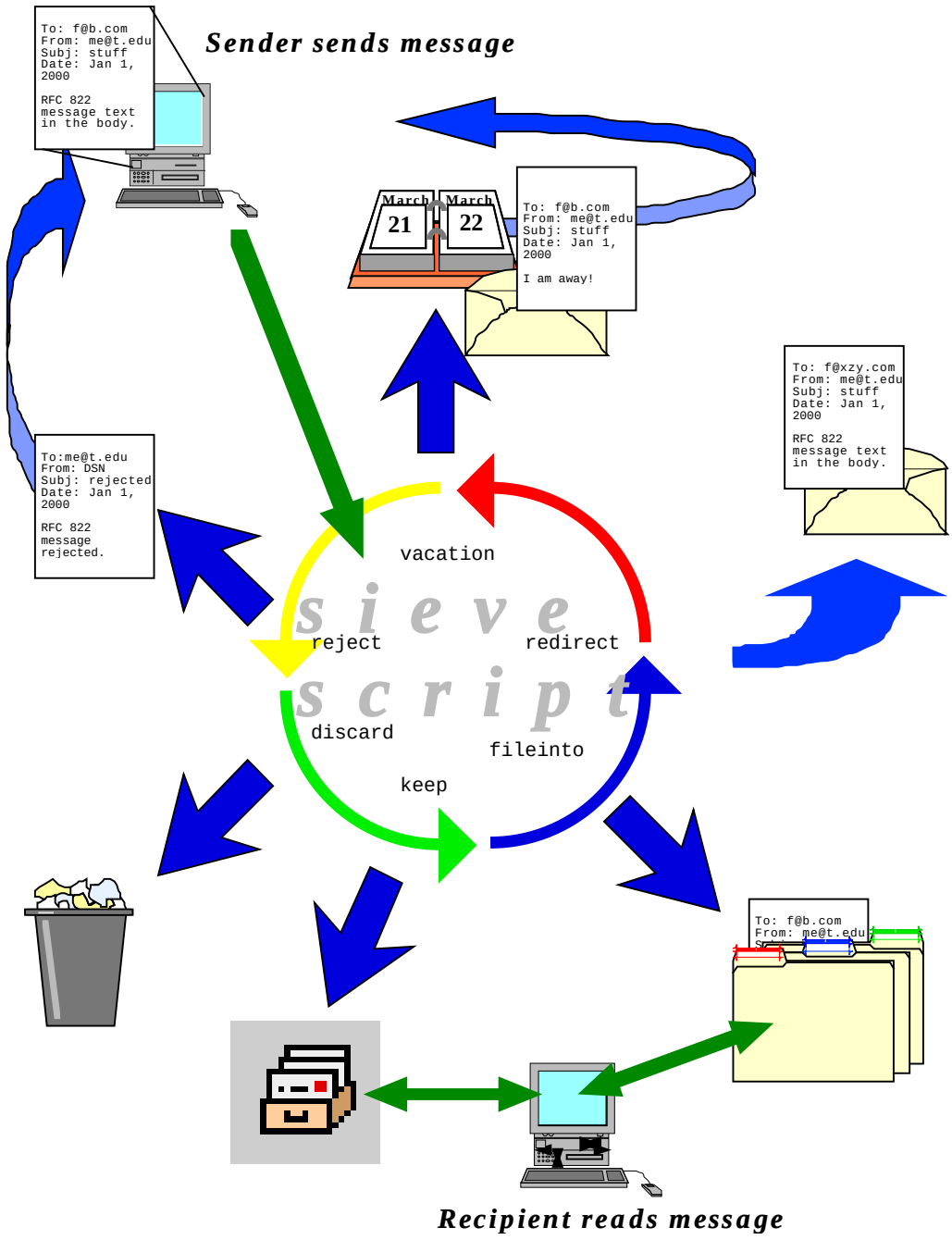



figure 3: some possible Sieve actions

**Example 3**

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

In this example, the header of the message is evaluated so that if the From address includes the string “Donald Trump”, or if the subject contains the string “$$$”, the message is discarded, and otherwise it’s properly filed into the user’s inbox. Note this script requires the ‘fileinto’ extension.

**Example 4**

```
require "vacation";
             if header :contains "from" "boss@frobnitzm.edu" {
                redirect "pleeb@xanadu.wv.us";
             } else {
                vacation "Sorry, I'm away, I'll read your message
                   when I get around to it.";
             }
```

This example is an application of the vacation extension (described in [SHOWALTER2]). It checks to see if a message is from The Boss, and if it is, the message is sent along to another email address (where one might be reading mail while on vacation). Otherwise, the vacation message is sent back to the sender of the message. (Vacation, it should be noted, of course has its own requirements and capabilities, such as the requirement there be no mail loops generated, and the capability to set a vacation message with a Sieve command.)

### *Sieve Language Concepts*

The rules and language specified in the base document [SHOWALTER] should be considered the definitive source for the semantics of the language. This section, however, is meant as a brief summation of some of the general underlying concepts and specific design choices in the Sieve language.

We generally refer to an implementation capable of executing a Sieve script as a ‘Sieve Engine’, to provide a generic way of describing Sieve execution that might take place on either client or server.

The execution of Sieve scripts is done by definition at the time of final delivery. Obviously, there may be multiple “final deliveries” as we discussed above, but Sieve narrowly defines final delivery as the movement of a mail message into a user-accessible permanent mailstore.

Sieve scripts are intended to be read, parsed, and executed linearly, left to right, from top to bottom. It is a matter of design choice as to whether the script is interpreted or compiled by the Engine, although the order of execution is implicitly message-wise. Syntax errors at compile time should be detected by the implementation and relayed back to the user or user agent if possible. All processing stops if there is a run-time failure. Obviously an interpreted script will only detect syntax errors at the point in the script they arise.

There are many restrictions on combinations of actions specified in the base document, as well as a common sense general rule of thumb that implementations should not allow combinations of actions that make no sense. When an implementation detects such a forbidden (or prohibitable) combination, it stops processing.

By default, if there is a failure of any sort, Sieve has the concept of an *implicit keep* with respect to the current message being processed. If an implementation has chosen to parse and run simultaneously, messages which were processed earlier in any given batch may have already been filed or otherwise acted upon. It is the responsibility of the implementation to provide reasonable notification to the user as to the extent of processing before failure.

The same message should not be delivered to the same mailbox more than once per script-evaluation, but if a script explicitly calls for this action, it is not treated as an error. The basic premise is the primacy of the message; the user should see one copy of any given message and no more in most circumstances. However, certain applications of a 'multiple fileinto' will be useful in some implementations.

Implementations may limit the number of actions that may be executed by a given script or command. In particular, implementations should be done with an eye towards allowing site-wide policies to be set with respect to the number of actions and which actions may be used together.

Implementations may also limit the levels of nesting of blocks and tests, but a minimum of at least fifteen levels is specified in the base document.

Some recognized differences in the semantics of filing in different types of mail systems and between client- and server-side filtering is one reason why certain implementation advice and requirements in the document are 'MAY' and 'SHOULD' and 'MUST'. For example, in many IMAP-based systems, there is the concept of a 'multiple fileinto', in which the same message is filed into more than one mailbox for purposes of automated processing or local redistribution. This is a difficult and ambiguous operation on many POP-based clients, and impossible using some MTAs.

The design of Sieve anticipates that many extensions will be written, but that for the most common extensions (e.g. 'vacation' – cf. [SHOWALTER2]) be documented publically, and ideally submitted as proposed standards through the IETF process.

However, if a given Engine does not understand a specific extension required by a script, the Engine is prohibited from doing any processing at all. As such, implementations should intelligently anticipate a graceful mechanism for both advertising new extensions to users and providing understandable feedback when the script calls for an extension not supported by the Engine.

A MIME type of `application/sieve` is defined in the base specification, although since Sieve scripts can be usefully composed, transported, and stored in plaintext format, a large variety of options are available for their transport and storage.

## Models for the Application of Sieve

We've discussed the general model and need for Sieve, and specifics of the syntax and semantics of the language itself. This final section is a discussion of some of the ways in which Sieve might be practically incorporated into specific software elements, set up and used by sites, and what the end-user experience might be.

### *Creation and Editing of Sieve Scripts*

Sieve scripts are relatively easy to read, and most keywords used in it have obvious real-world meanings. It is thus quite possible for a script to be developed and edited 'by hand', through simple editing of a file by an end user. However, the left to right reading does not necessarily translate into an English-style grammar, and like most such languages, precision is required for syntactic correctness (balancing braces, remembering colons, etc.) There are also inevitably some subtle semantics involved with creating longer scripts that might not be obvious to a neophyte. As such, this form of editing is probably appropriate only for very knowledgeable users and administrators.

The preferred mechanism for generating and editing Sieve scripts is a specialized form of graphical editor. Ideally, this would provide a point and click interface that would allow the end user to generate the filter without even having to see the text. There have been two implementations of such a GUI, both using an HTML-based interface to a CGI which returns the completed Sieve syntax to the user. One of these directly stores the scripts, and the other allows the user to then cut, copy, paste, and otherwise combine blocks of Sieve script.

One optional but clever feature of a Sieve editor would be a syntax pre-checker. This would allow validation of manually-edited scripts against the base specification and known extensions, prior to their submission to the Engine.

A Sieve editor might also include an on-board Sieve Engine, to allow actual execution of a script against sample messages once it is written.

There is at least one Sieve 'standalone' filter-builder under commercial development, and one client-side manual-edit filter-builder in internal private release.

Obviously, there needs to be a certain amount of coordination between a Sieve editor and the intended Sieve engine, but we can foresee a model wherein a single program generates and edits Sieve scripts for use by multiple engines. For instance, a mail user agent might use Sieve scripts for client-side filtering, but provide a mechanism for saving and transporting the same primitives to a compatible server-side Sieve Engine. A typical case in point might be detecting a new form of Spam. A single end-user could create a script, document it in its comments, and make it available server-wide, thus providing an optional yet widely-available mechanism for others to stop the particular kind of message wihout being required to or re-creating the filter themselves.

Sieve interfaces may provide different kinds of interface elements for different extensions. For instance, the 'vacation' extension is something that is typically edited and activated for a specific, limited period, while an anti-spam filter is probably a more permanent kind of filter with different criteria. A user interface for vacation might have a dialogue box prompting a user for the 'away' message and a list of exceptions, while an interface for an anti-spam message might ask the user to enter keywords which she or he finds likely to be associated with unsolicited email. Future extensions will likely have similar specific requirements of user interfaces.

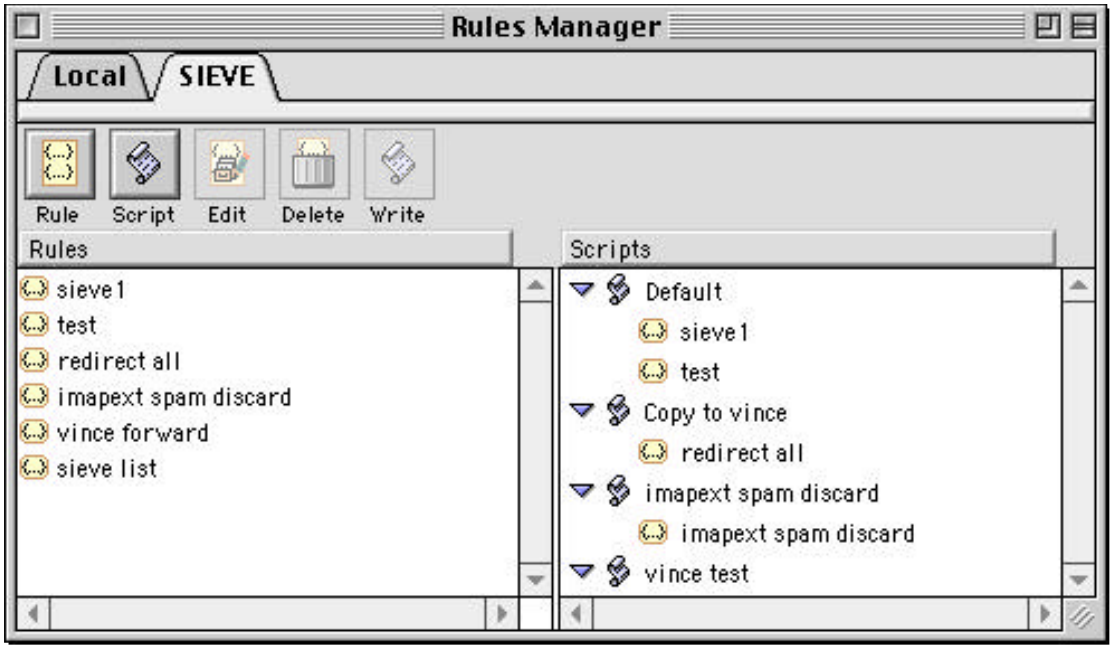



Figure 4: The Sieve Rules manager window in Cyrusoft's Mulberry 2.1, a Sieve-supporting email client.

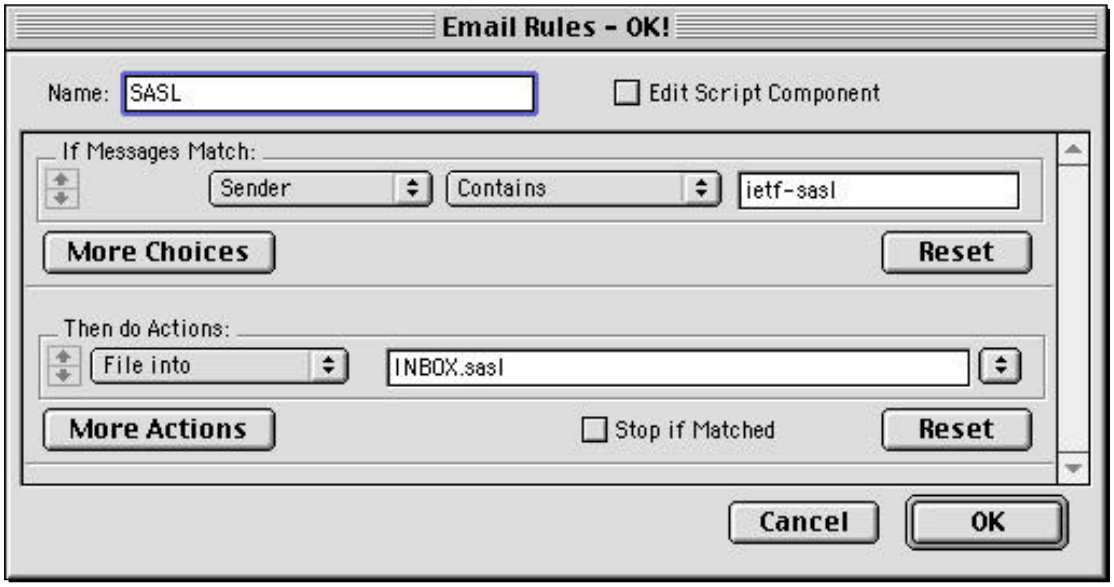



Figure 5: The graphical sieve rule editor with an example rule.

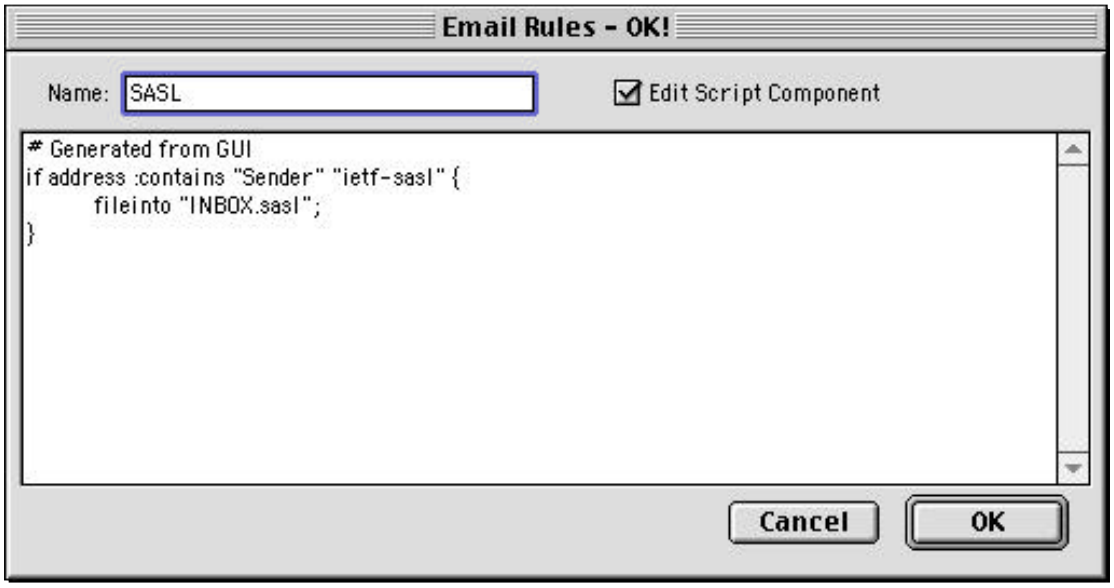



Figure 6: The rule from Figure 5 in “raw” sieve script format.

*Transport of Sieve Scripts*

As we’ve noted, Sieve is purposely designed to be independent of transport mechanism. There is no required ‘Sieve Network Protocol’[†]. This may present some initial interoperability problems in sharing and setting Sieve scripts until other products and services provide integrated support for Sieve. Some possible options for Sieve transport are listed below; however, it’s clear from the variety of desired applications that there will probably never be a single canonical transport method.

Client-side Sieve Editors and Engines need not worry about transport any more complicated than writing the script to a known preference or internal format, so this discussion is useful primarily in the context of server-side filtering or proxy-filtering.

Note that transport security is not explicitly discussed in each of the methods below, but clearly different approaches have the same security features and flaws of the base transport mechanism. For instance, it’s dirt simple to have an SSL / TLS-encrypted submission of a Sieve filter via http.

- FTP and file writing

  This is a more or less direct approach. The Sieve script is simply ftp’d to a designated area (the current Carnegie Mellon implementation of Sieve does it this way), or even more simply, created and edited in a host directory that the Sieve engine knows about. One problem with this approach is relaying errors detected by the Engine back to the end user, however.

- HTTP / HTML

  Web interfaces are plentiful enough these days, and submission of a Sieve filter described in HTML via http, with ultimate disposition and proper resolution by a Sieve engine as directed by an http server process (likely via CGI or similar mechanism) is the route of the first implementation by Innosoft International, Inc. This has the advantage that the syntax can be

[†] There is a proposed protocol custom-designed for remotely managing Sieve scripts described in [MARTIN], but this is in no way required. [MARTIN2] also describes a useful sieve-specific notification system for informing end users as to when a Sieve action has taken place, which may be considered a necessary companion for some of the proposed transport models described in this section.

checked at the time of submission, and an appropriate error message passed back the user. The interface and submission layers are quite distinct in this approach, allowing for a modular manner in building the two halves.

- LDAP

  An LDAP Schema for Sieve filters has already been described. Filters are vaguely directory-like data, in that they are not frequently changed once established, have the same stringent security and authenticity requirements as the editing of directory data, and are accessed by external mail systems in the same way data such as preferred mailbox can be used by LDAP. The feedback for script errors can be handled through writing to a special field within the schema. LDAP has the virtue of increasingly wide-spread availability, natural ties to the messaging system, and at least elemental support from mail clients. Interoperability between LDAP-aware clients with knowledge of the LDAP Sieve schema should be good.

- ACAP

  The Application Configuration Access Protocol is ideally suited to the transport and storage of Sieve data, in that it's designed for frequent user- and application reference, modfication, and storage. An ACAP dataset for Sieve has already been described, and the Carnegie Mellon reference ACAP server will include Sieve support. ACAP also has the merit of being capable of providing a direct notification to the end user on submit time, and of storing arbitrary intermediate data (such as is likely to happen when server-side filters are executed while the user is away) necessary to adequately apprise the end user of irregularities. Another advantage to ACAP is the ability to flexibly share, through site inheritance and access control lists, filters among multiple users, groups, and even site-wide. The major drawback with ACAP at this time is the lack of an installed base. We note that IMSP, ACAP's more widely-deployed predecessor, has been used experimentally to store Sieve filters as well.

- IMAP extension

  One proposal for handling Sieve transport has been an extension to the IMAP protocol that would allow Sieve-aware IMAP servers to advertise this fact to clients, and provide an extended IMAP syntax for submitting,verifying, and retrieving Sieve filters. This would be particularly appropriate for IMAP servers that took over the final delivery phase, such as the Cyrus-strain of server. It may also be convenient for IMAP-based mail clients that wish to share filters between client and server. However, there is not a formal proposal describing this, and there is a certain problem with overloading the base IMAP protocol with a functionality not inherently related to IMAP per se. This form of metadata is very much like user addressbook, program preference, and other user-oriented ephemera, for which IMSP and ACAP were specifically designed in preference to using IMAP.

- Submission via email

  Several existing server-side filtering schemes use a simple email submission scheme, in which a filter script is submitted to a specific email addres, verified by an automatic process, and successes or failures relayed back to the user by email. One might even describe most Listserv software as being a form of automated filter processing, albeit with restricted capabilities. This approach has the attraction of being familiar and eadily understandable by users, although the requirements of including syntactically-correct instructions via plaintext in email are considerable and probably not appropriate for most unsophisticated users. From the server implementer's side, this does have the virtue of being relatively simple to develop, but there are also serious security considerations (at least in the absence of authenticated SMTP submissions.)

  One variation on ths approach would be to use the application/sieve MIME type to submit the Sieve script as an Email attachment. The part could be generated by the local mail user agent or Sieve agent, as it were, verified, and even purposely obscured from direct editing by the end user (except via the GUI). An intelligent mail user agent might then digitally sign

such a message, for instance, with PGP, and the Sieve Engine on the server-side would then use the public key of the originating user to verify the authenticity of the filter request.

**Editors and Transport**

Ideally, a graphical editor would also include the capability to transport the completed Sieve script. The basic example is the CGI-based Sieve editor, which, when the form with the user's filtering instructions is submitted, posts the data to the CGI, which then stores it in a file particular to the mail server's Sieve Engine.

This can also, of course, be incorporated directly into an email client. **Mulberry 2.1** from Cyrusoft includes built-in hooks to allow a variety of transport methods via plug-ins, which are presented to the user in the form of simple "Save As…" and "Save to Server…" menu options.

**Transport Formats**

A plaintext file is probably acceptable for the more direct forms of submission described above, such as ftp. The specified MIME type of application/sieve should be used when transporting via email or http. Methods such as ACAP and LDAP would require translation of the Sieve script into the format appropriate for submission via protocol. It has been proposed that XML would provide a reasonable method of providing a transport-independent format for Sieve scripts and primitives.

### *Storage of Sieve Scripts*

The storage of Sieve scripts in most cases will be tied in some way to the transport method. As with mail server themselves, there will be two major models for such storage: direct file access by the end user, and the so-called "black box" storage where the storage of the Sieve scripts is obscure to the end user.

In the initial Carnegie Mellon implementation, scripts are simply stored as plaintext Unix files in a particular directory; the Sieve Engine automatically looks in each user's directory for the script.

The initial Innosoft / PMDF sieve implementation, using http as a transport mechanism, parses the HTML over the wire, then stores the verified script in a magic area accessible directly only by system administrators and the Sieve Engine associated with the mail server.

As with any such format that's peculiar to the implementation, none of these approaches is particularly interoperable. Even published specs as to the submission and storage format require that each client implement a method of access unique to that implementation. As such, while these are good interim first steps for initial deployment experience, they are not good long-term solutions. Storage and access by MTAs – for instance, a modification of Sendmail to read Sieve filters, similar to the layer that Procmail provides via a piped process – would be similarly arbitrary by server type.

In an LDAP-based Sieve storage model, the scripts would simply be stored as part of the general directory's database. In ACAP and IMSP, Sieve scripts would each get their own name within a namespace specifically set aside for Sieve, and would be stored as strings. ACAP also offers the capability of building, dynamically, a combined script, based on entries stored at site and user levels. These two approaches have the strong virtue of being regularized, potentially interoperable methods of storage that can be used by arbitrary clients without prior knowledge of the server's peculiar methods. LDAP and ACAP may work well in tandem for these purposes, LDAP providing "authoritative" filters set by proxy by administrators and automated processes, and ACAP providing Sieve scripts set directly by the user or the mail user agent / Sieve agent.

Client-side storage would be by choice of implementation, but implementers will bear in mind that easy interoperability would be served by having an easily interchangeable format – such as plain text or MIME type mapping.

Cyrusoft's **Mulberry 2.1** can store Sieve scripts as simple preferences, including using the IMSP and ACAP remote preferences protocols. These filters are thus potentially available through the network to other users by normal protocol streams, and can be propagated to more than one user through the customary methods available for each of these protocols.

### *Server-Side Processing*

The primary point at which Sieve should be integrated should be server-side processing, and in most cases it's appropriate for the SMTP delivery agent or a process invoked by the SMTP server to do the processing. Some specific submodels:

- Piped via the SMTP Server process

  This is the way Procmail works right now. Sendmail calls Procmail, which then executes the delivery rules. The Sieve engine would be called, or incorporated into an existing piece of software (e.g. Procmail).

- Straight to mailstore

  In this model, the SMTP agent simply delivers to the mailstore, and some post-processing by a Sieve engine is done at a later date (presumably via a scheduled job or some other user- or administrator-defined event). This has the advantage of having an initial "safe" delivery.

- Direct delivery rules

  This is the model used by the Cyrus IMAP server: a process called deliverd in this architecture takes the message handoff from the SMTP server once the deliverability has been verified, and executes certain delivery rules. Sieve can then be integrated directly into the IMAP server itself, which would allow an IMAP-centric processing model (for instance, one non-delivery-time application might be putting aging rules into an administrative tool to get rid of old mail via a Sieve script).

- LDAP delivery rules

  In this model, the Sieve rules would be retrieved from the LDAP server by a dynamic query from the SMTP server (or IMAP server, as the case may be), and the integral Sieve engine would execute.

- ACAP reference

  ACAP is an excellent repository for short, user-generated data, and direct access to rules from an ACAP server would allow a sieve-aware server (or client, for that matter) to filter at delivery time or at access time; the same rules could be used by both filtering agents, or either, depending on site policy, user preference, administrative prerogative, or compatibility level with specific clients demand.

### *Client-Side Processing*

- POP scenario

  Where a POP client would be used, the client would use the Sieve rules as either the basis for, supplement to, or replacement for its native filtering mechanism. Ideally, there would either be an export format, or a native transportable format, that would allow such Sieve scripts to be exchanged with other clients. Standard plain ASCII text should be the default lingua franca. Such client-side rules, of course, can be stored in an optimized format, e.g. a database of rules.

- IMAP on-line scenario

  For an IMAP client in “online” mode (an IMAP client in offline or disconnected mode would act very much like a POP client with respect to filtering mail to a local folder), Sieve scripts would either have to be stored as part of the client preferences, or on a suitable network cache (ACAP, IMSP, or LDAP) accessible securely to the MUA. In addition to mundane local folder activities or standard Sieve responses, the client-side Sieve engine would trigger IMAP actions or commands; a simple example would be setting flags, as described generically in [MELNIKOV]. In most cases, having a Sieve-aware IMAP server would be preferable, however.

- IMAP events to trigger filtering

  One possible use of a Sieve script is to tune a client to respond to specific IMAP activities: e.g. searching, opening or closing a folder, appending a message with the semantics of a draft message in mid-composition, etc. Some applications of Sieve to this probably require extensions, but note that the basic format of the language can describe most of these actions as a “post-delivery” event.

  Cyrusoft’s **Mulberry 2.1** has IMAP-aware filters, which can be triggered by or execute IMAP-related events. Thus it is forward-compatible with a simple Sieve extension to change IMAP flags as described in [MELNIKOV].

*Other Processing Scenarios*

- Administrator-initiated

  As described above, one use of Sieve scripts would be as a format for housecleaning-style activities for privileged administrators: deleting spam after the fact from user's mailboxes, aging old messages, and so forth.

- Agent-initiated

  Agents being used on the system for other purposes, for example, content-indexing processes, can call Sieve scripts for disposition of copies of messages or be called themselves as extensions to Sieve. Other forms of agents will want to be Sieve-aware so that the basic Sieve actions can be integrated with more advanced third-party, yet RFC822-syntax-based, activities.

## The Future: A Sieve Manifesto

We believe that the time is right to adopt an internet-standard mail filtering scheme, and that Sieve is a sufficient and well-designed basis for that standard. We therefore suggest that implementers and users of any mail software tool, internet appliance, or related RFC822-compatible tool consider adopting the following as pillars of the Sieve credo:

- All such tools be written in compliance and awareness of **RFC 3028**, Sieve, A Mail Filtering Language;
- Architecture of such tools take into account potential client / server interactions, the transport issues and models discussed in this document, and similar issues requisite to making Sieve a useful part of the internet mail architecture;
- Sieve interoperability tests be incorporated into appropriate industry interop events;
- Vendors and developers of internet mail tools continue to develop and write extensions to Sieve to provide additional layers of functionality, both converting existing tools and providing new ones as the occasion warrants;
- Vendors and users co-operate to refine and extend Sieve as a solution to problems in the mail filtering sphere.

## References and Further Information

### Sieve web sites

A website on Sieve and related material is maintained by Cyrusoft International, Inc. at

`http://www.cyrusoft.com/sieve`

### Sieve mailing list and archive

Discussions on Sieve at a technical level are currently carried out on the

`ietf-mta-filters@imc.org`

Mailing list. To join, send a note with the word 'subscribe' in the body to:

`ietf-mta-filters-request@imc.org`

Archives for this mailing list are available via the web at:

`http://www.imc.org/ietf-mta-filters`

**Documents**

*Internet draft documents are updated frequently. Query the IETF drafts page at http://www.ietf.org with the keyword 'sieve' for the most current list of documents and version numbers. Comments within this paper are based on the most recent versions available at the time, for which version numbers are detailed below.*


[SHOWALTER] "Sieve: A Mail Filtering Language", Tim Showalter**. *RFC 3028***, January, 2001. `http://www.ietf.org/rfc/rfc3028.txt?number=3028`

[IMAIL] "Standard for the Format of ARPA Internet Text Messages", Internet Standard 11 (RFC822). See also revisions of this standard under DRUMS area working group at `http://www.ietf.org`

[MARTIN] "A Protocol for Remotely Managing Sieve Scripts", Tim Martin, Internet Draft December, 2000. `http://search.ietf.org/internet-drafts/draft-martin-managesieve-02.txt`

[MARTIN2] "Sieve – An Extension for Providing Instant Notifications", Tim Martin, Internet Draft December, 2000. `http://search.ietf.org/internet-drafts/draft-martin-sieve-notify-00.txt`

[MELNIKOV] "Sieve – IMAP Flag Extension", Alexey Melnikov, Internet Draft October, 2000. `http://search.ietf.org/internet-drafts/draft-melnikov-sieve-imapflags-04.txt`

[MURCHISON] "Sieve -- Subaddress Extension", Ken Murchison. Internet Draft. September, 2000. `http://search.ietf.org/internet-drafts/draft-murchison-sieve-subaddress-02.txt`

[MURCHISON2] "Sieve – Regular Expression Extension", Ken Murchison. Internet Draft. January, 2001. `http://search.ietf.org/internet-drafts/draft-murchison-sieve-regex-03.txt`

[SHOWALTER2] "Sieve Vacation Extension", Tim Showalter. Internet Draft. August, 2000. `http://search.ietf.org/internet-drafts/draft-showalter-sieve-vacation-04.txt`