Network Working Group	N. Freed
Internet-Draft	S. Vedam
Expires: September 10, 2009	Sun Microsystems
	March 09, 2009

TOC

Sieve Email Filtering: Sieves and display directives in XML
draft-freed-sieve-in-xml-03

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 10, 2009.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This document describes a way to represent Sieve email filtering language scripts in XML. Representing sieves in XML is intended not as an alternate storage format for Sieve but rather as a means to facilitate manipulation of scripts using XML tools.

The XML representation also defines additional elements that have no counterparts in the regular Sieve language. These elements are intended for use by graphical user interfaces and provide facilities for labeling or grouping sections of a script so they can be

displayed more conveniently. These elements are represented as specially structured comments in regular Sieve format.

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

Changed representation of comments in XML to use a comment element.

Update references.

Added an IANA registration of a URN for the Sieve namespace.

Updated XML Schema to allow largely unrestricted use of material in other namespaces.

Add compact Relax NG schema.

Updated example stylesheet to handle material in other namespaces.

Corrected stylesheet handling of <comment> elements.

Added a section defining the structured comment convention.

Moved the examples section to an appendix.

Added text to clarify that the examples in the various appendices are in fact code components and may therefore be reused.

Added a section on validation requirements.

Clarified various editor requirements and trust issues, restricted the use of "*/" in non-Sieve XML content.

Table of Contents

1. Introduction

TOC

Sieve [RFC5228] (Guenther, P. and T. Showalter, "Sieve: An Email Filtering Language," January 2008.) is a language for filtering email messages at or around the time of final delivery. It is designed to be implementable on either a mail client or mail server. It is meant to be extensible, simple, and independent of access protocol, mail architecture, and operating system and it is intended to be manipulated by a variety of different user interfaces.

Some user interface environments have extensive existing facilities for manipulating material represented in XML. While adding support for alternate data syntaxes may be possible in most if not all of these environments, it may not be particularly convenient to do so. The obvious way to deal with this issue is to map sieves into XML, possibly on a separate backend system, manipulate the XML, and convert it back to normal Sieve format.

The fact that conversion into and out of XML may be done as a separate operation on a different system argues strongly for defining a common XML representation for Sieve. This way different front end user interfaces can be used with different back end mapping and storage facilities.

Another issue with the creation and manipulation of sieve scripts by user interfaces is that the language is strictly focused on describing email filtering operations. The language contains no mechanisms for indicating how a given script should be presented in a user interface. Such information can be represented in XML very easily so it makes sense to define a framework to do this as part of the XML format. A structured comment convention is then used to retain this information when the script is converted to normal Sieve format.

Various sieve extensions have already been defined, e.g., [RFC5183] (Freed, N., "Sieve Email Filtering: Environment Extension," May 2008.) [RFC5229] (Homme, K., "Sieve Email Filtering: Variables Extension," January 2008.) [RFC5230] (Showalter, T. and N. Freed, "Sieve Email Filtering: Vacation Extension," January 2008.) [RFC5231] (Segmuller, W. and B. Leiba, "Sieve Email Filtering: Relational Extension," January 2008.) [RFC5232] (Melnikov, A., "Sieve Email Filtering: Imap4flags Extension," January 2008.) [RFC5233] (Murchison, K., "Sieve Email Filtering: Subaddress Extension," January 2008.) [RFC5235] (Daboo, C., "Sieve Email Filtering: Spamtest and Virustest Extensions," January 2008.) [RFC5293] (Degener, J. and P. Guenther, "Sieve Email Filtering: Editheader Extension," August 2008.), and more are planned. The set of extensions available varies from one implementation to the next and may even change as a result of configuration choices. It is therefore essential that the XML representation of Sieve be able to accommodate Sieve extensions without requiring schema changes. It is also desirable that Sieve extensions not require changes to the code that converts to and from the XML representation.

This specification defines an XML representation for sieve scripts and explains how the conversion process to and from XML works. The

XML representation is capable of accommodating any future Sieve extension as long as the underlying Sieve grammar remains unchanged. Furthermore, code that converts from XML to the normal Sieve format requires no changes to accommodate extensions, while code used to convert from normal Sieve format to XML only requires changes when new control commands are added - a rare event. An XML Schema, Relax NG Schema, and a sample stylesheet to convert from XML format are also provided in the appendices.

---

2. Conventions used in this document

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.) [RFC2119].

---

3. Grammatical structure of Sieve

TOC

The Sieve language is designed to be highly extensible without making any changes to the basic language syntax. Accordingly the syntax of Sieve, defined in section 8 of [RFC5228] (Guenther, P. and T. Showalter, "Sieve: An Email Filtering Language," January 2008.), is entirely structural in nature and employs no reserved words of any sort.

Structurally a sieve script consists of a series of commands. Each command in turn consists of an identifier, zero or more arguments, a optional test or test-list, and finally an optional block containing another series of commands. Commands are further broken down into controls and actions, although this distinction cannot be determined from the grammar.

Some example Sieve controls are:

```
stop;               <-- No arguments, test, or command block
require "fileinto"; <-- Control with a single argument
if true {stop;}     <-- Control with test and command block
```

Some examples of Sieve actions are:

```
discard;            <-- Action with no args, test, or command block
fileinto "folder";  <-- Action with an argument
```

At the time of this writing there are no controls defined that accept both arguments and a test. Similarly, there are currently no

defined actions that allow either a test or a command block. Nevertheless, the Sieve grammar allows such constructs to be defined by some future extension.

A test consists of an identifier followed by zero or more arguments, then another test or test-list. Unlike commands, tests cannot be followed by a command block.

Here are some examples of Sieve tests. Note that such tests have to appear as part of a command in order to be syntactically valid:

```
true                <-- Test with no argument or subordinate test
envelope "to" "me@example.com" <-- Test with several arguments
header :is "from" "you@example.com" <-- Test with tagged argument
```

Command or test arguments can be either string lists, whole numbers or tags. (Tags are simply identifiers preceded by a colon.) Note that although the Sieve grammar treats single strings as a degenerate case of a string list, some tests or actions have arguments that can only be individual strings, not lists.

Here is an example showing the use of both a test-list and a string list:

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

Extensions can add new controls, actions, tests, or new arguments to existing controls or actions. Extensions have also changed how string content is interpreted, although this is not relevant to this specification. However, it is especially important to note that so far no Sieve extension has added a new control to the language and it seems safe to assume that due to their nature future addition of controls will be rare.

Finally, comments are allowed between lexical elements in a Sieve script. Finally, comments are allowed between lexical elements in a Sieve script. One important use case for comments is encoding meta-data about the script, a facility which is lacking in the Sieve language. Therefore comments need to be preserved in the XML representation.

---

4. XML Representation of Sieve

TOC

Sieve controls and actions are represented in XML as "control" or "action" elements respectively. The command's identifier appears as a name attribute on the element itself. This is the only attribute allowed on controls and actions - arguments, tests, test-lists, and

nested command blocks are all represented as nested elements. While naming the element after the control or action itself may seem like a better choice, doing so would result in extensions requiring corresponding schema changes.

The example Sieve controls shown in the previous section would be represented in XML as:

```
<control name="stop"/>
<control name="require"><str>fileinto</str></control>
<control name="if">
  <test name="true"/><control name="stop"/>
</control>
```

The example Sieve actions shown above would appear in XML as:

```
<action name="discard"/>
<action name="fileinto"><str>folder</str></action>
```

The separation of controls from actions in the XML representation means that conversion from normal Sieve format to XML has to be able to distinguish between controls and actions. This is easily done by maintaining a list of all known controls since experience indicates new controls are rarely added.

Tests are represented in the same basic way as controls and actions, that is, as a "test" element with a name attribute giving the test identifier. For example:

```
<test name="true"/>
<test name="envelope"/>
  <str>to</str><str>me@example.com</str>
</test>
<test name="header"/>
  <tag>is</tag><str>from</str><str>you@example.com</str>
</test>
```

String, number, and tag arguments are represented as "str", "num", and "tag" elements respectively. The actual string, number, or tag identifier appears as text inside the element. None of these elements have any defined attributes. Several examples of arguments have already appeared in the preceding control, action and test examples. Any whitespace in the str body content MUST be preserved by the processor.

String list arguments are represented as a "list" element which in turn contains one or more str elements. Note that this allows the distinction between a single string and a string list containing a single string to be preserved. This is not essential since a list containing a single string could simply be mapped to a string, but

it seems prudent to maintain the distinction when mapping to and from XML.

Nested command blocks appear as a series of control or action elements inside of an outer control or action element. No block element is needed since an inner command block can only appear once and only after any arguments, tests, or test-lists. For example:

```
<control name="if>
  <test name="anyof">
    <test name="not">
      <test name="exists>
        <list><str>From</str><str>Date</str></list>
      </test>
    </test>
    <test name="header">
      <tag>contains</tag>
      <str>from</str>
      <str>fool@example.edu</str>
    </test>
  </test>
  <action name="discard"/>
</control>
```

Finally, Sieve comments are mapped to a special "comment" element in XML. Both hash and bracketed comments are mapped to the same construct so the distinction between the two is lost in XML. XML comments are not used because some XML tools do not make it convenient to access comment nodes.

4.1. XML Display Directives

TOC

Sometimes graphical user interfaces are a convenient way to provide sieve management functions to users. These interfaces typically summarize/annotate/group/display sieve script(s) in an intuitive way for end users.

To do this effectively, the graphical user interface may require additional information about the sieve script itself. That information or "meta-data" might include, but is not limited to - a sieve name (identifying the current sieve), whether the sieve is enabled or disabled, the order in which the part of the sieve are presented to the user. The graphical user interface may also choose to provide mechanisms to allow the user to modify the script.

It is often useful for a graphical user interface to group related sieve script elements and provide an interface that display these groups separately so they can be managed as a single object. Some examples include Sieve statements that together provide vacation responders, blacklists/whitelists and other types of filtering controls.

Some advanced graphical user interfaces may even provide a natural language representation of a sieve script and/or an advanced interface to present sieve statements directly to the user.

A graphical user interface may also choose to support only a subset of action commands in the Sieve language (and its extensions) and so a mechanism to indicate the extent of support and characterize the relationships between those supported action commands and test (with its arguments) is immensely useful and probably required for clients that may not have complete knowledge of sieve grammar and semantics.

The Sieve language contains no mechanisms for indicating how a given script should be presented in a user interface. The language also does not contain any specific mechanisms to represent other sorts of meta-data about the script. Providing support for such meta-data as part of a sieve script is currently totally implementation specific and is usually done by imposing some type of structure on comments.

However, such information can be represented in XML very easily so it makes sense to define a framework to do this as part of the XML format. Implementations MAY choose to use structured comments to retain this information when the script is converted to normal Sieve format.

The sample schemata for the XML representation of Sieve allows XML in foreign namespaces to be inserted in most places in Sieve scripts. This is the preferred means of including additional information. Alternately, the schema defines two display directives - displayblock and displaydata - as containers for meta-data needed by graphical user interfaces.

Editors MAY use displayblock, displaydata and foreign namespaces to associate meta-data. Some editors find it inconvenient to preserve this additional data during an editing session. Editors MAY preserve this data during an editing session for compatibility with other editors.

The displayblock element can be used to enclose any number of sieve statements at any level. It is semantically meaningless to the sieve script itself. It allows an arbitrary set of attributes. Implementations MAY use this to provide many simple, display related meta-data for the sieve such as sieve identifier, group identifier, order of processing, etc.

The displaydata element supports any number of arbitrary child elements. Implementations MAY use this to represent complex data about that sieve such as a natural language representation of sieve or a way to provide the sieve script directly.

4.2. Structured Comments

TOC

Since the XML representation is not intended as a storage format there needs to be a way to preserve the additional information that can be included in the XML representation in the normal Sieve

syntax. This is done through the use of three structured comment conventions:

1. XML content in other namespaces is placed in Sieve bracketed comments beginning with the string "/* [/" and ending with the string "/] */".

2. The content of displaydata elements is placed in Sieve bracketed comments beginning with the string "/* [|" and ending with the string "|] */".

3. The beginning of a displayblock element is mapped to a bracketed Sieve comment beginning with the string "/* [*" which then lists any displayblock attribute names and values in XML format. The end of a displayblock element is mapped to a comment of the form "/* *] */".

Processors MUST preserve the additional information allowed in the XML format and SHOULD use the structured comment format shown above.

Note: If "*/" is found in the XML content, when mapped into a comment it would prematurely terminate that comment. Escaping of this sequence would often be inconvenient for processors. Editors SHALL NOT include "*/" within displayblock, displaydata or foreign markup. Processors MAY regard documents containing "*/" in foreign markup, displayblock or displaydata as invalid.

4.3. Validation

TOC

A processor MAY validate documents against a schema and MAY reject any which do not conform. For any document that a processor does not reject as invalid, any markup that the processor cannot understand by reference to this specification MAY be discarded.

Note that example Relax NG and XML Schema are given in the appendices below.

---

5. Security Considerations

TOC

Any syntactically valid sieve script can be represented in XML. Accordingly, all security considerations applicable to Sieve and any extensions used also apply to the XML representation.

The use of XML carries its own security risks. Section 7 of RFC 3470 (Hollenbeck, S., Rose, M., and L. Masinter, “Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols,” January 2003.) [RFC3470] discusses these risks.

It is axiomatic that a Sieve editor must be trusted to do what the user specifies. If XML formats are used this trust necessarily must

extent to the components involved in converting to and from XML format.

Arbitrary data can be included using other namespaces or placed in the extensible displayblock and displaydata constructs defined in this specification, possibly including entire scripts and other executable content in languages other than Sieve. Appropriate security precautions should be taken when using these facilities.

6. IANA Considerations

TOC

This section registers a new XML namespace per the procedures in RFC 3688 (Mealling, M., “The IETF XML Registry,” January 2004.) [RFC3688].

URI: urn:ietf:params:xml:ns:sieve

Registrant Contact: IETF Sieve working group
<ietf-mta-filters@imc.org>

XML:

BEGIN

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
  "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
        content="text/html;charset=iso-8859-1"/>
  <title>Sieve Namespace</title>
</head>
<body>
  <h1>Namespace for Sieve Language objects expressed in XML</h1>
  <h2>urn:ietf:params:xml:ns:sieve</h2>
  <p>See <a href="http://www.rfc-editor.org/rfc/rfcXXXX.txt">
   RFC XXXX</a>.
  </p>
</body>
</html>
```

END

7. References

TOC

7.1. Normative References

TOC

[OASISRNC]	Clark, J., "RELAX NG Compact Syntax," OASIS Committee Specification rnc, November 2002.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC3470]	Hollenbeck, S., Rose, M., and L. Masinter, "Guidelines for the Use of Extensible Markup Language (XML) within IETF Protocols," BCP 70, RFC 3470, January 2003 (TXT, HTML, XML).
[RFC3688]	Mealling, M., "The IETF XML Registry," BCP 81, RFC 3688, January 2004 (TXT).
[RFC5228]	Guenther, P. and T. Showalter, "Sieve: An Email Filtering Language," RFC 5228, January 2008 (TXT).

7.2. Informative References

TOC

[RFC5183]	Freed, N., "Sieve Email Filtering: Environment Extension," RFC 5183, May 2008 (TXT).
[RFC5229]	Homme, K., "Sieve Email Filtering: Variables Extension," RFC 5229, January 2008 (TXT).
[RFC5230]	Showalter, T. and N. Freed, "Sieve Email Filtering: Vacation Extension," RFC 5230, January 2008 (TXT).
[RFC5231]	Segmuller, W. and B. Leiba, "Sieve Email Filtering: Relational Extension," RFC 5231, January 2008 (TXT).
[RFC5232]	Melnikov, A., "Sieve Email Filtering: Imap4flags Extension," RFC 5232, January 2008 (TXT).
[RFC5233]	Murchison, K., "Sieve Email Filtering: Subaddress Extension," RFC 5233, January 2008 (TXT).
[RFC5235]	Daboo, C., "Sieve Email Filtering: Spamtest and Virustest Extensions," RFC 5235, January 2008 (TXT).
[RFC5293]	Degener, J. and P. Guenther, "Sieve Email Filtering: Editheader Extension," RFC 5293, August 2008 (TXT).

Appendix A. Extended Example

TOC

The example sieve script given in section 9 of [RFC5228] (Guenther, P. and T. Showalter, "Sieve: An Email Filtering Language," January 2008.) would be represented in XML as the following code component:

```
<sieve xmlns="urn:ietf:params:xml:ns:sieve">
  <comment>
    Example Sieve Filter
    Declare any optional features or extensions used by the script
  </comment>

  <control name="require">
    <str>fileinto</str>
  </control>

  <comment>
    Handle messages from known mailing lists
    Move messages from IETF filter discussion list to filter mailbox
  </comment>
  <control name="if">
    <test name="header">
      <tag>is</tag>
      <str>Sender</str>
      <str>owner-ietf-mta-filters@imc.org</str>
    </test>
    <action name="fileinto">
      <str>filter</str>
    </action> <comment>move to "filter" mailbox</comment>
  </control>

  <comment>
    Keep all messages to or from people in my company
  </comment>
  <control name="elsif">
    <test name="address">
      <tag>domain</tag>
      <tag>is</tag>
      <list>
        <str>From</str>
        <str>To</str>
      </list>
      <str>example.com</str>
    </test>
    <action name="keep"/>
  </control>

  <comment>
    Try and catch unsolicited email.  If a message is not to me,
    or it contains a subject known to be spam, file it away.
  </comment>
  <control name="elsif">
    <test name="anyof">
      <test name="not">
        <test name="address">
          <tag>all</tag>
          <tag>contains</tag>
          <list>
            <str>To</str>
            <str>Cc</str>
            <str>Bcc</str>
          </list>
          <str>me@example.com</str>
```

```
        </test>
      </test>
      <test name="header">
        <tag>matches</tag>
        <str>subject</str>
        <list>
          <str>*make*money*fast*</str>
          <str>*university*dipl*mas*</str>
        </list>
      </test>
    </test>
    <action name="fileinto">
      <str>spam</str>
    </action>
  </control>
  <control name="else">
    <comment>
      Move all other (non-company) mail to "personal"
      mailbox.
    </comment>
    <action name="fileinto">
      <str>personal</str>
    </action>
  </control>

</sieve>
```

The same script could be annotated with graphical display hints in a variety of ways. Three possible code components that do this are:

```
<sieve xmlns="urn:ietf:params:xml:ns:sieve">

  <control name="require">
    <str>fileinto</str>
  </control>

  <displayblock name="File filter list mail" order="1"
                group="FILE_TO_FOLDER" enable="true">
    <control name="if">
      <test name="header">
        <tag>is</tag>
        <str>Sender</str>
        <str>owner-ietf-mta-filters@imc.org</str>
      </test>
      <action name="fileinto">
        <str>filter</str>
      </action>
    </control>
  </displayblock>

  <displayblock name="Keep all company mail" order="2"
                group="KEEP_MESSAGE" enable="true">
    <control name="elsif">
      <test name="address">
        <tag>domain</tag>
        <tag>is</tag>
        <list>
          <str>From</str>
          <str>To</str>
        </list>
        <str>example.com</str>
      </test>
      <action name="keep"/>
   </control>
  </displayblock>

  <displayblock name="File suspected spam" order="3"
                group="FILE_TO_FOLDER" enable="true">
     <control name="elsif">
       <test name="anyof">
         <test name="not">
           <test name="address">
             <tag>all</tag>
             <tag>contains</tag>
             <list>
               <str>To</str>
               <str>Cc</str>
               <str>Bcc</str>
             </list>
             <str>me@example.com</str>
           </test>
         </test>
         <test name="header">
           <tag>matches</tag>
           <str>subject</str>
           <list>
             <str>*make*money*fast*</str>
```

```
              <str>*university*dipl*mas*</str>
            </list>
          </test>
        </test>
        <action name="fileinto">
          <str>spam</str>
        </action>
      </control>
    </displayblock>

    <displayblock name="File noncompany mail as personal" order="4"
                  group="FILE_TO_FOLDER" enable="true">
      <control name="else">
        <action name="fileinto">
          <str>personal</str>
        </action>
      </control>
    </displayblock>

  </sieve>
```

Note that since displayblock elements are semantically null as far as the script itself is concerned they can be used to group structures like elsif and else that are tied to statements in other groups.

The representation of this script in regular Sieve syntax uses structured comments:

```
require "fileinto";
/* [* name="File filter list mail" order="1"
      group="FILE_TO_FOLDER" enable="true" */
if header :is "Sender" "owner-ietf-mta-filters@imc.org"
{
  fileinto "filter";
}
/* *] */
/* [* name="Keep all company mail" order="2"
      group="KEEP_MESSAGE" enable="true" */
elsif address :domain :is [ "From", "To" ] "example.com"
{
  keep;
}
/* *] */
/* [* name="File suspected spam" order="3"
      group="FILE_TO_FOLDER" enable="true" */
elsif anyof ( not ( address :all :contains [ "To", "Cc", "Bcc" ]
                    "me@example.com" ),
              header :matches "subject" [ "*make*money*fast*",
                                          "*university*dipl*mas*" ] )
{
  fileinto "spam";
}
/* *] */
/* [* name="File noncompany mail as personal" order="4"
      group="FILE_TO_FOLDER" enable="true" */
else
{
  fileinto "personal";
}
/* *] */
```

A separate namespace can be used to embed text or structured information:

```
  <sieve xmlns="urn:ietf:params:xml:ns:sieve"
         xmlns:nls="http://example.com/nls">

    <nls:interpretation>
      If the e-mail header "Sender" is owner-ietf-mta-filters@imc.org
      then file it into the "filter" folder.

      Otherwise if the address in the "From" or "To" has a domain
      that is "example.com" then keep it.

      Otherwise  messages meeting with any of these conditions:

      (1) None of the addresses in "To" or "Cc"  or "Bcc" contains
          the domain "example.com".

      (2) The "Subject" field matches the pattern *make*money*fast*
          or *university*dipl*mas* then file  it into the "spam"
          folder.

      If all else fails then file the message in the "personal"
      folder.
    </nls:interpretation>

    ... the actual sieve script ...

  </sieve>
```

Alternately, displaydata elements can be used to accomplish the same thing:

```
  <sieve xmlns="urn:ietf:params:xml:ns:sieve">

    <displaydata>
      <nls-interpretation>
        If the e-mail header "Sender" is owner-ietf-mta-filters@imc.org
        then file it into the "filter" folder.

        Otherwise if the address in the "From" or "To" has a domain
        that is "example.com" then keep it.

        Otherwise  messages meeting with any of these conditions:

        (1) None of the addresses in "To" or "Cc"  or "Bcc" contains
            the domain "example.com".

        (2) The "Subject" field matches the pattern *make*money*fast*
            or *university*dipl*mas* then file  it into the "spam"
            folder.

        If all else fails then file the message in the "personal"
        folder.
      </nls-interpretation>
    </displaydata>

    ... the actual sieve script ...

  </sieve>
```

Again, structured comments are used to represent this in regular Sieve syntax:

```
    /* [|
      <nls-interpretation>
        If the e-mail header "Sender" is owner-ietf-mta-filters@imc.org
        then file it into the "filter" folder.

        Otherwise if the address in the "From" or "To" has a domain
        that is "example.com" then keep it.

        Otherwise  messages meeting with any of these conditions:

        (1) None of the addresses in "To" or "Cc"  or "Bcc" contains
            the domain "example.com".

        (2) The "Subject" field matches the pattern *make*money*fast*
            or *university*dipl*mas* then file  it into the "spam"
            folder.

        If all else fails then file the message in the "personal"
        folder.
      </nls-interpretation>
    |] */

    ... the actual sieve script ...
```

Appendix B. XML Schema for Sieves in XML

TOC

This appendix is informative. The following code component is an XML Schema for the XML representation of Sieve scripts. Most of the elements employing a complex content model allow use of elements in other namespaces, subject to lax XML Schema validation rules. Additionally, displaydata elements can be used to encapsulate arbitrary XML content. Finally, displayblock elements can be used as a general-purpose grouping mechanism - arbitrary attributes are allowed on displayblock elements.

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="urn:ietf:params:xml:ns:sieve"
            targetNamespace="urn:ietf:params:xml:ns:sieve">

  <xsd:element name="sieve">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice maxOccurs="unbounded" minOccurs="0">
          <xsd:element ref="control"/>
          <xsd:element ref="action"/>
          <xsd:element ref="displayblock"/>
          <xsd:element ref="displaydata"/>
          <xsd:element ref="comment"/>
          <xsd:any namespace="##other" processContents="lax"/>
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="comment" type="xsd:string"/>

  <xsd:complexType name="command">
    <xsd:sequence>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="str"/>
        <xsd:element ref="num"/>
        <xsd:element ref="list"/>
        <xsd:element ref="tag"/>
        <xsd:element ref="displaydata"/>
        <xsd:element ref="comment"/>
        <xsd:any namespace="##other" processContents="lax"/>
      </xsd:choice>
      <xsd:element ref="test" minOccurs="0" maxOccurs="1"/>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="control"/>
        <xsd:element ref="action"/>
        <xsd:element ref="displayblock"/>
        <xsd:element ref="displaydata"/>
        <xsd:element ref="comment"/>
        <xsd:any namespace="##other" processContents="lax"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:attribute use="required" name="name" type="identifier"/>
  </xsd:complexType>

  <xsd:element name="control" type="command"/>

  <xsd:element name="action" type="command"/>

  <xsd:element name="test">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:choice minOccurs="0" maxOccurs="unbounded">
          <xsd:element ref="str"/>
          <xsd:element ref="num"/>
```

```
        <xsd:element ref="list"/>
        <xsd:element ref="tag"/>
        <xsd:element ref="displaydata"/>
        <xsd:element ref="comment"/>
        <xsd:any namespace="##other" processContents="lax"/>
      </xsd:choice>
      <xsd:element ref="test" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute use="required" name="name" type="identifier"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="list">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="str" minOccurs="1"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="tag" type="identifier"/>

<xsd:element name="str" type="xsd:string"/>

<xsd:element name="num" type="xsd:nonNegativeInteger"/>

<xsd:simpleType name="identifier">
  <xsd:restriction base="xsd:token">
    <xsd:pattern value="[A-Za-z_][A-Za-z0-9_]*"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:element name="displayblock">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="control"/>
        <xsd:element ref="action"/>
        <xsd:element ref="displayblock"/>
        <xsd:element ref="displaydata"/>
        <xsd:element ref="comment"/>
        <xsd:any namespace="##other" processContents="lax"/>
      </xsd:choice>
    </xsd:sequence>
    <xsd:anyAttribute processContents="skip"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="displaydata">
  <xsd:complexType>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:any processContents="skip"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
</xsd:schema>
```

Appendix C. Relax NG Schema for Sieves in XML

TOC

This appendix is informative. The following code component defines a Relax NG schema using compact notation OASISRNC (Clark, J., "RELAX NG Compact Syntax," November 2002.) [OASISRNC] for the XML representation of Sieve scripts. Most of the elements employing a complex content model allow unrestricted use of elements in other namespaces. Additionally, displaydata elements can be used to encapsulate arbitrary XML content. Finally, displayblock elements can be used as a general-purpose grouping mechanism - arbitrary attributes are allowed on displayblock elements.

```
namespace sieve = "urn:ietf:params:xml:ns:sieve"

start = element sieve:sieve { ( control | action | displayblock |
                                displaydata | comment | ext )* }

comment = element sieve:comment { xsd:string }

command =
   (
      attribute name {
        xsd:token {
          pattern = "[A-Za-z_][A-Za-z0-9_]*" } },
      ( str | num | \list | tag |
        displaydata | comment | ext )*,
      test?,
      ( control | action | displayblock |
        displaydata | comment | ext )*
   ),
   empty

control = element sieve:control { command }

action = element sieve:action { command }

test =
   element sieve:test
   {
      attribute name {
        xsd:token {
          pattern = "[A-Za-z_][A-Za-z0-9_]*" } },
      ( str | num | \list | tag | comment | ext )*,
      test*
   }

\list = element sieve:list { str+ }

tag = element sieve:tag {
        xsd:token {
          pattern = "[A-Za-z_][A-Za-z0-9_]*" } }

str = element sieve:str { xsd:string }

num = element sieve:num { xsd:nonNegativeInteger }

any = ( element * { any } | attribute * { text } | text )*

ext = element * - sieve:* { any }*

displayblock =
   element sieve:displayblock
   {
      ( control | action | displayblock |
        displaydata | comment | ext )*,
      attribute * { text }*
   }

displaydata = element sieve:displaydata { any* }
```

Appendix D. Stylesheet for conversion from XML

TOC

This appendix is informative. The following code component is a stylesheet that can be used to convert the Sieve in XML representation to regular Sieve format. Content in other namespaces, displaydata, and displayblock elements are converted to structured comments as appropriate.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Convert Sieve in XML to standard Sieve syntax -->

<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:sieve="urn:ietf:params:xml:ns:sieve">

  <xsl:output method="text" encoding="UTF-8"
              media-type="application/sieve"/>

  <!-- Only preserve whitespace in str elements -->
  <xsl:strip-space elements="*"/>
  <xsl:preserve-space elements="sieve:str"/>

  <!-- Match top level sieve node,
       start processing in sieve mode -->

  <xsl:template match="sieve:sieve">
    <xsl:apply-templates select="*" mode="sieve">
      <xsl:with-param name="prefix" select="''"/>
    </xsl:apply-templates>
  </xsl:template>

  <!-- Routine to properly literalize quotes in Sieve strings -->

  <xsl:template name="quote-string">
    <xsl:param name="str"/>
    <xsl:choose>
      <xsl:when test="not($str)"/>
      <xsl:when test="contains($str, '&quot;')">
        <xsl:call-template name="quote-string">
          <xsl:with-param name="str"
             select="substring-before($str, '&quot;')"/>
        </xsl:call-template>
        <xsl:text>\&quot;</xsl:text>
        <xsl:call-template name="quote-string">
          <xsl:with-param name="str"
             select="substring-after($str, '&quot;')"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:when test="contains($str, '\')">
        <xsl:call-template name="quote-string">
          <xsl:with-param name="str"
             select="substring-before($str, '\')"/>
        </xsl:call-template>
        <xsl:text>\\</xsl:text>
        <xsl:call-template name="quote-string">
          <xsl:with-param name="str"
             select="substring-after($str, '\')"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="$str"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
```

```
  <!-- Sieve mode processing templates -->

  <xsl:template match="sieve:control|sieve:action" mode="sieve">
    <xsl:param name="prefix"/>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:value-of select="@name"/>
    <xsl:variable name="blockbegin"
            select="generate-id(sieve:control|sieve:action)"/>
    <xsl:for-each select="*">
      <xsl:choose>
        <xsl:when test="self::sieve:str|self::sieve:num|
                        self::sieve:list|self::sieve:tag|self::sieve:test">
          <xsl:apply-templates select="." mode="sieve"/>
        </xsl:when>
        <xsl:when test="generate-id(.) = $blockbegin">
          <xsl:text xml:space="preserve">
</xsl:text>
          <xsl:value-of select="$prefix"/>
          <xsl:text>{</xsl:text>
          <xsl:apply-templates select="." mode="sieve">
            <xsl:with-param name="prefix"
               select="concat($prefix, '  ')"/>
          </xsl:apply-templates>
        </xsl:when>
        <xsl:otherwise>
          <xsl:apply-templates select="." mode="sieve">
            <xsl:with-param name="prefix"
               select="concat($prefix, '  ')"/>
          </xsl:apply-templates>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:for-each>
    <xsl:choose>
      <xsl:when test="count(sieve:control|sieve:action) &gt; 0">
        <xsl:text xml:space="preserve">
</xsl:text>
        <xsl:value-of select="$prefix"/>
        <xsl:text>}</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:text>;</xsl:text>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="sieve:test" mode="sieve">
    <xsl:text xml:space="preserve"> </xsl:text>
    <xsl:value-of select="@name"/>
    <xsl:apply-templates select="*[not(self::sieve:test)]" mode="sieve"/>
    <xsl:if test="count(descendant::sieve:test) &gt; 0">
      <xsl:text> (</xsl:text>
      <xsl:for-each select="sieve:test">
        <xsl:apply-templates select="." mode="sieve"/>
        <xsl:if test="count(following-sibling::sieve:test) &gt; 0">
          <xsl:text>,</xsl:text>
```

```
        </xsl:if>
      </xsl:for-each>
      <xsl:text> )</xsl:text>
    </xsl:if>
  </xsl:template>

  <xsl:template match="sieve:str" mode="sieve">
    <xsl:text> &amp;quot;</xsl:text>
    <xsl:call-template name="quote-string">
      <xsl:with-param name="str" select="text()"/>
    </xsl:call-template>
    <xsl:text>&amp;quot;</xsl:text>
  </xsl:template>

  <xsl:template match="sieve:num" mode="sieve">
    <xsl:text xml:space="preserve"> </xsl:text>
    <!-- Use numeric suffixes when possible -->
    <xsl:choose>
      <xsl:when test="(number(text()) mod 1073741824) = 0">
        <xsl:value-of select="number(text()) div 1073741824"/>
        <xsl:text>G</xsl:text>
      </xsl:when>
      <xsl:when test="(number(text()) mod 1048576) = 0">
        <xsl:value-of select="number(text()) div 1048576"/>
        <xsl:text>M</xsl:text>
      </xsl:when>
      <xsl:when test="(number(text()) mod 1024) = 0">
        <xsl:value-of select="number(text()) div 1024"/>
        <xsl:text>K</xsl:text>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="text()"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>

  <xsl:template match="sieve:list" mode="sieve">
    <xsl:text> [</xsl:text>
    <xsl:for-each select="sieve:str">
      <xsl:apply-templates select="." mode="sieve"/>
      <xsl:if test="count(following-sibling::sieve:str) &amp;gt; 0">
        <xsl:text>,</xsl:text>
      </xsl:if>
    </xsl:for-each>
    <xsl:text> ]</xsl:text>
  </xsl:template>

  <xsl:template match="sieve:tag" mode="sieve">
    <xsl:text> :</xsl:text>
    <xsl:value-of select="text()"/>
  </xsl:template>

  <xsl:template match="sieve:comment" mode="sieve">
    <xsl:param name="prefix"/>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>/*</xsl:text>
```

```
    <xsl:value-of select="."/>
    <xsl:value-of select="$prefix"/>
    <xsl:text>*/</xsl:text>
  </xsl:template>

  <!-- Convert display information into structured comments -->
  <xsl:template match="sieve:displayblock" mode="sieve">
    <xsl:param name="prefix"/>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>/* [*</xsl:text>
    <xsl:apply-templates select="@*" mode="copy"/>
    <xsl:text> */</xsl:text>
    <xsl:apply-templates select="*" mode="sieve">
      <xsl:with-param name="prefix" select="$prefix"/>
    </xsl:apply-templates>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>/* *] */</xsl:text>
  </xsl:template>

  <xsl:template match="sieve:displaydata" mode="sieve">
    <xsl:param name="prefix"/>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>/* [|</xsl:text>
    <xsl:apply-templates mode="copy">
      <xsl:with-param name="prefix"
                      select="concat($prefix, '  ')"/>
    </xsl:apply-templates>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>   |] */</xsl:text>
  </xsl:template>

  <!-- Copy unrecnognized nodes and their descendants -->

  <xsl:template match="*" mode="sieve">
    <xsl:param name="prefix"/>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>/* [</xsl:text>
    <xsl:apply-templates select="." mode="copy">
      <xsl:with-param name="prefix"
                      select="concat($prefix, '  ')"/>
    </xsl:apply-templates>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>   /] */</xsl:text>
  </xsl:template>

  <!-- Copy mode processing templates -->
```

```
  <xsl:template match="*[not(node())]" mode="copy">
    <xsl:param name="prefix"/>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>&lt;</xsl:text>
    <xsl:value-of select="name()"/>
    <xsl:apply-templates select="@*" mode="copy"/>
    <xsl:text>/&gt;</xsl:text>
  </xsl:template>

  <xsl:template match="*[node()]" mode="copy">
    <xsl:param name="prefix"/>
    <xsl:text xml:space="preserve">
</xsl:text>
    <xsl:value-of select="$prefix"/>
    <xsl:text>&lt;</xsl:text>
    <xsl:value-of select="name()"/>
    <xsl:apply-templates select="@*" mode="copy"/>
    <xsl:text>&gt;</xsl:text>
    <xsl:apply-templates mode="copy">
      <xsl:with-param name="prefix"
                      select="concat($prefix, '  ')"/>
    </xsl:apply-templates>
    <xsl:if test="*[last()][not(text())]">
      <xsl:text xml:space="preserve">
</xsl:text>
      <xsl:value-of select="$prefix"/>
    </xsl:if>
    <xsl:text>&lt;/</xsl:text>
    <xsl:value-of select="name()"/>
    <xsl:text>&gt;</xsl:text>
  </xsl:template>

  <xsl:template match="@*" mode="copy">
    <xsl:text> </xsl:text>
    <xsl:value-of select="name()"/>
    <xsl:text>="</xsl:text>
    <xsl:value-of select="."/>
    <xsl:text>"</xsl:text>
  </xsl:template>

</xsl:stylesheet>
```

Appendix E. Acknowledgements

TOC

The stylesheet copy mode code is loosely based on a sample code posted to the xsl-list list by Americo Albuquerque. Robert Burrell Donkin, Andrew McKeon, Alexey Melnikov, and Aaron Stone provided useful comments on the document.

Authors' Addresses

TOC

	Ned Freed
	Sun Microsystems
	800 Royal Oaks
	Monrovia, CA 91016-6347
	USA
Phone:	+1 909 457 4293
Email:	ned.freed@mrochek.com
	Srinivas Saisatish Vedam
	Sun Microsystems
Phone:	+91 80669 27577
Email:	Srinivas.Sv@Sun.COM