Internet Engineering Task Force	I. Radu, Ed.
Internet-Draft	November 17, 2011
Intended status: Informational	
Expires: May 20, 2012	

TOC

Advanced Groupware Access Protocol draft-iulian-advanced-groupware-access-protocol-04

Abstract

The Advanced Groupware Access Protocol, (AGAP) allows a client to access and store electronic mail messages, contacts, events, files, and configurations on a server. The electronic mail messages can be grouped in folders. AGAP also provides the capability for an offline client to resynchronize with the server.

AGAP does not specify a means of posting electronic mail messages; this function is handled by a mail transfer protocol such as **SMTP** [RFC2821]. It also does not specify a means for exchanging messages with contacts that are reported as being online; this function is handled by an instant messaging protocol such as **XMPP** [RFC3921].

AGAP includes the following operations for electronic mail messages: creating, deleting, renaming, moving and coping mail folders; checking for new messages; permanently removing messages; moving and coping messages between folders; fetching information about a message; setting and clearing tags for messages; searching in messages; retrieving only a part of a message; marking messages as SPAM; deleting attachments from a message.

AGAP includes the following operations to manipulate the contacts: creating, deleting, moving, coping, tagging, and searching contacts; checking if a contact is online; fetching information about a contact.

AGAP includes the following operations related to the use of the events: creating, deleting, moving, coping and tagging events in calendar; fetching events details; searching for events.

All entries are read and written in format XML encoded **UTF-8** [RFC3629] and each entry is identified by a unique alphanumeric identifier.

AGAP is designed to support access only to a single server per connection. It is also designed to balance the volume of text exchanged between the server and clients and its readability by humans for debugging.

Status of this Memo

Copyright Notice

Table of Contents

1. How to Read This Document

TOC

1.1. Organization of This Document

TOC

This document is written from the point of view of someone implementing an AGAP client or server, and also from the point of view of a server administrator. The protocol overview (chapter 2) presents all aspects related to a correct implementation (like the maximum length of a command or response line, charset used). The material in chapter 3 through 5 provides the states in which can be a connection at a moment, respectively what commands are valid in each state and their valid responses. Chapter 6 makes a summary of the return codes for each command. The implementers find in chapter 7 samples of conversations so that they can test the compliance of their applications with this standard.

1.2. Conventions Used in This Document

TOC

Document conventions are noted in this chapter. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT","MAY", and "OPTIONAL" in this document are to be interpreted as described in **'Key words for use in RFCs to Indicate Requirement Levels'** [RFC2119]. The word "CAN" (not "MAY") is used to refer to a possible circumstance or situation, as opposed to an optional facility of the protocol.

"User" is used to refer to a human user. "Client" refers to the software being run by the user. "Server" refers to the software responding to the client requests. In examples, "C:" and "S:" indicate lines sent by the client and server respectively. "Connection" refers to the entire sequence of client/server interaction from the initial establishment of the network connection until its termination. "Conversation" is an exchange of commands and responses between the client and the server. "Account" defines all folders and their content that can be accessed from Authenticated State. All references to characters order is according to the **UTF-8** [RFC3629] specification.

2. Protocol Overview

TOC

2.1. Charset Used for Commands and Responses

TOC

All data exchanged between the server and the client is done using strings encoded **UTF-8** [RFC3629]. If the server or client send a string incorrect encoded then the other side can close immediately the connection.

2.2. Maximal Length of a Command or Response Line

TOC

A command or response consists of a line of text that has a maximal length of 1024 characters (including line end). A line of text is ended with the character LF (0x0A). There can be optionally a CR character (0x0D) before the LF character. If the server or client sends a line with a length greater of 1024 then the other side can close immediately the connection.

2.3. Numbers in Commands and Responses

TOC

The numbers that are used in commands are signed integers on 32 bits. The valid values are between -2,147,483,648 and 2,147,483,647.

2.4. Regular Expressions in Commands

TOC

Following is a resume of all regular expression rules that CAN be used by the commands defined in this standard:

```
    Logical operators:
XY       X followed by Y
X|Y      Either X or Y

    Predefined character class:
.        Any character (does not match line terminators)

    Characters:
x        The character x
\\       The backslash character
\xhh     The character with hexadecimal value 0xhh
\uhhhh   The character with hexadecimal value 0xhhhh
\t       The tab character ('\x09')
\n       The newline (line feed) character ('\x0A')
\r       The carriage-return character ('\x0D')

    Character classes:
[abc]     a, b, or c (simple class)
[^abc]    Any character except a, b, or c (negation)
[a-zA-Z]  a through z or A through Z, inclusive (range)

    Boundary matchers:
^        The beginning of a line
$        The end of a line
\b       A word boundary
\B       A non-word boundary

    Greedy quantifiers:
X?       X, once or not at all
X*       X, zero or more times
X+       X, one or more times
X{n}     X, exactly n times
X{n,}    X, at least n times
X{n,m}   X, at least n but not more than m times

    Reluctant quantifiers:
X??      X, once or not at all
X*?      X, zero or more times
X+?      X, one or more times
X{n}?    X, exactly n times
X{n,}?   X, at least n times
X{n,m}?  X, at least n but not more than m times
```

Figure 1

2.5. Unique Identification Numbers (UID)

TOC

The length of an UID is between 1 and 32 characters.

The UIDs MUST to be unique only between items from the same folder.

The characters accepted for building an UID are only all 26 Latin letters (A-Z) in lowercase and uppercase and all 10 Latin digits (0-9). An UID is case sensitive and it is the same for each connections, except after server change it and announce the change by chaning the UCID assigned to the corresponding folder.

Any new message/contact/event MUST have a bigger UID as all other existing items in the selected folder. The sorting is made according **UTF-8** [RFC3629] (digits before letters and uppercase letters before the lowercase letters - 0..9A..Za..z). A shorter UID is before a longer one (9234 before 02345) and any zero (0) before a number is take into account by the server when two UIDs are compared.

We get an approximately maximum number of 4.50+e+17 unique combinations for 32 characters long UIDs. We get a maximum number of 3381098545 unique combinations for 8 characters long UIDs.

TOC

2.6. Folder Change Identification Numbers (FCID)

An FCID has the same format as a normal UID and each new value of an FCID is bigger as the precedent one (as is described for UIDs). An FCID is changed for a folder when the structure of the folder is changed (subfolders are added or removed) and when the items are changed (items are added or removed). The FCID of a folder is not changed if it is changed the (sub)child of one of its children.

TOC

2.7. UIDs Change Identification Numbers (UCID)

An UCID has the same format as a normal UID and each new value of UCID should be bigger as the precedent one (as is described for UIDs). If the last UCID had already had the biggest UID valid value then its new value can be the first valid UID value. An UCID is changed for a folder each time the server had changed the UIDs assigned to the items. This can be necessary if, for example, there is a new item and already last valid UID was assigned to an other item. The new UIDs must keep the items in the same order as before the renumbering.

TOC

2.8. Representation of Text and Binary Content in XML Bodies

Binary content must be encoded using the **BASE64** [RFC4648] method and the corresponding tag must have the ENCODED attribute set to "base64".

A text content can be passed as it is (**UTF-8** [RFC3629]) or it can be encoded using the **BASE64** [RFC4648] method. The corresponding tag must have the ENCODED attribute set to "utf-8", in case of plain text, and to "base64", if the content was encoded using the BASE64 method.

TOC

2.9. Folders

TOC

2.9.1. Naming

All folder names are case sensitive and they are encoded according to **UTF-8** [RFC3629].

A backslash (\) does not escape the character after it (it has no special meaning).

For building a folder name, the user CAN use all **UTF-8** [RFC3629] characters with a value

bigger then 0x1f (white space is the first allowed character), but with the exception of the slash (/ 9x2F), back slash (\ 0x5C), multiplication sign (* 0x2A), and question mark (? 0x3F).

The following folder names are also not accepted: '.', and '..'.

2.9.2. Hierarchy

TOC

None of the reserved folders can have subfolders, exception makes the TRASH that must to store also deleted folders and FILESHARE that holds ordinary files.

The character used for delimiting path levels is the slash (/). A path that starts with '/' represents an absolute path. All other are relative to the currently selected folder (with SLCT).

If there is no folder currently selected then the client MUST use only absolute paths. It is recommended for a client to use always absolute paths.

2.9.3. Folder Types

TOC

The following folder types are defined by this standard:

- calendar - CALE - holds events;
- configuration - CONF - holds user accounts configuration (the client is free to store all information it needs for providing roaming);
- contacts - ADDR - holds contact information;
- files - FILE - holds files that have no special meanings for the server;
- filter - FILT - holds the definition of a filter (it does not allow subfolders);
- folder - FOLD - contains only subfolders;
- journal - JRNL - holds journal entries;
- message - MESG - holds e-mail messages;
- notes - NOTE - holds short texts;
- tasks - TASK - holds tasks.

Each of these types allow for subfolders in them.

2.9.4. Reserved Folders

TOC

All the following reserved folders are located in the root of the user's account:

- CALENDAR - CALE - holds the main calendar of the user (tag: PUBLIC);
- CONFIGURATION - CONF - holds account configuration;
- CONTACT - ADDR - holds the main contact list (tag: PUBLIC);
- DRAFT - MESG - holds templates for e-mail messages;
- FILESHARE - FILE - holds shared files (tag: PUBLIC);
- INBOX - MESG - holds all new e-mail messages;
- JOURNAL - JRNL - holds the main journal (tag: PUBLIC);
- JUNK - MESG - holds all e-mail messages marked as SPAM or VIRUSED by user or the server;
- NOTE - NOTE - holds short texts;
- OUTBOX - MESG - holds all e-mail messages that wait to be sent;
- SENT - MESG - holds copy of sent e-mail messages;
- TASK - TASK - holds the main tasks list (tag: PUBLIC);
- TRASH - MESG - holds all deleted e-mail messages;

All defined PUBLIC folders allow others to view and add content to them, but they are not allowed to delete or change something.

A client can use different names for these folders when display them so that the client application can use localization and standard or customized names for them. If this is the case, then the user cannot create a folder, in the root of his account, with the same name as the real (reserved) name of the folder.

2.10. Tags

TOC

2.10.1. Syntax

TOC

The client can set tags only for folder entries, but the server can set tags also for folders. The tags of a folder are reported by the STAT command.

The format of a tag is a name optionally followed by the equal sign (=) and a value. Each time a tag is set, the new value replace the old one. All tags that have no value assigned are returned only as name. Assigning an empty value to a tag makes it to return a name followed by the equal sign and no value. Setting a tag without a value for an entry which previously had the same tag with a value makes the tag to lose its value and to be returned as name only (without the equal sign).

The characters accepted for building a TAG are only all 26 Latin letters (A-Z) in uppercase, all 10 Latin digits (0-9) and the minus sign (-). A TAG is case insensitive. Its length is between 1 and 32 characters.

The characters accepted for a TAG value are only all 26 Latin letters (A-Z) in lowercase and uppercase, all 10 Latin digits (0-9), plus the minus (-), underscore (_) and dot (.) characters. A TAG value is case sensitive. Its length is between 1 and 32 characters.

The server returns always the TAG names in uppercase, even if the client set them using a lowercase version. The server should convert silently any lowercase character in a TAG name (sent by client) to its corresponding uppercase character.

2.10.2. Reserved Tag Names

TOC

The following tag names have a meaning set by this standard for folders:

- NO-COPY - the content of this folder cannot be copied with CPYF, COPY, or FCPY but can be deleted with DELF, DELE, or FDEL or moved with MOVF, MOVE, or FMOV;
- NO-DELETE - the folder or the content of this folder cannot be deleted with DELF, DELE or FDEL but can be copied with CPYF, COPY, or FCPY or moved with MOVF, MOVE, or FMOV;
- NO-DELF - this folder cannot be deleted with DELF but its content can be deleted with DELF, DELE, or FDEL if the tag NO-DELETE is not assigned to the folder;
- NO-FOLDERS - this entry cannot have subfolders, so the user cannot create subfolders in it with MAKE;
- NO-MOVE - the content of this folder cannot be moved with MOVF, MOVE, or FMOV but can be deleted with DELF, DELE, or FDEL or copied with CPYF, COPY, or FCPY;
- NO-RENAME - the name of this folder cannot be changed with NAME;
- READ-ONLY - the user can read it with RETR and delete it with DELF, DELE, or FDEL but cannot write in it with STOR, CPYF, COPY, FCPY, MOVF, MOVE, or FMOV, create subfolders in it with MAKE or change the tags of its content with STAG, or FTAG;
- RESERVED - it is a folder reserved by this standard; the user can write in it with STOR but cannot delete it with DELF or rename it with NAME;
- PUBLIC - the content of this folder can be read by all other users;

When the user do a DELF for a folder with the tag NO-DELF but without the tag NO-DELETE then the non-folder content will gone be deleted but not the folder.

When the user do a DELF for a folder with the tag NO-DELETE then the folder and its content will not gone be deleted (the tag NO-DELF is ignored).

Implicit a folder can be read only by its owner.

The following tag names have a meaning set by this standard for messages:

- ANSWERED - it was sent a reply to this e-mail message;
- SEEN - this object was already read;
- SPAM - this e-mail message is marked as spam;

2.11. The Responses for Each Type of Folder

TOC

2.11.1. Format and Conventions

TOC

All responses are in XML format. The tags and their attributes names are written only in uppercase. The values for attributes only in lowercase. The exception are header entries for a message. The tags keep the case from the message.

The content is encoded in **UTF-8** [RFC3629] format.

Each type of folder returns its entries in a different format.

Each tag written in uppercase must to be send as it is, each tag written in lowercase will be replaced with the right value at the time of generation.

Each tag that have a question mark will be present only once if it is the case and without the question mark.

Each tag that have a star will be present, possible many times, only if it is the case and without the star.

If a command is correct but the server cannot execute it because of an internal error, then the server returns the code 401.

2.11.2. Response for Calendar Folders

TOC

The format is derived from the one defined for VEVENT and VALARM by the **iCalendar** [RFC5545] standard.

The following example corresponds to this VEVENT definition:

```
BEGIN:VEVENT
UID:20110531T114600Z-123456@agap.at
DTSTAMP:2011-05-31T12:10:00Z
DTSTART:2011-06-07T18:00:00Z
DTEND:2011-06-07T24:00:00Z
SUMMARY:AGAP RFC Party
DESCRIPTION:Celebration of a new revision!\n0.4
END:VEVENT
```

Figure 2

Example event:

```
<VEVENT>
    <UID>20110531T114600Z-123456@agap.at</UID>
    <DTSTAMP>2011-05-31T12:10:00Z</DTSTAMP>
    <DTSTART>2011-06-07T18:00:00Z</DTSTART>
    <DTEND>2011-06-07T24:00:00Z</DTEND>
```

```
    <SUMMARY>AGAP RFC Party</SUMMARY>
    <DESCRIPTION>Celebration of a new revision!
0.4</DESCRIPTION>
    <VALARM>20110607T170000Z-20110531T114600Z-123456@agap.at</VALARM>
</VEVENT>
```

Figure 3

The following example corresponds to this alarm definition related to the previous event:

```
BEGIN:VALARM
TRIGGER;RELATED=START:2011-06-07T17:00:00Z
REPEAT:3
DURATION:PT15M
ACTION:AUDIO
ATTACH;FMTTYPE=audio/mpeg;VALUE=BINARY;ENCODING=BASE64:
 ABCDEFGHIJ==
END:VALARM
```

Figure 4

Example alarm:

```
<VALARM>
    <UID>20110607T170000Z-20110531T114600Z-123456@agap.at</UID>
    <RELATED-TO SOURCE="vevent">
       20110531T114600Z-123456@agap.at</RELATED-TO>
    <TRIGGER RELATED="start">2011-06-07T17:00:00Z</TRIGGER>
    <REPEAT>3</REPEAT>
    <DURATION>PT15M</DURATION>
    <ACTION>AUDIO</ACTION>
    <ATTACH FMTTYPE="audio/mpeg" ENCODED="base64">
        ABCDEFGHIJ==</ATTACH>
</VALARM>
```

Figure 5

The BEGIN:VEVENT is replaced with <VEVENT> and END:VEVENT is replaced with </VEVENT>. The BEGIN:VALARM is replaced with <VALARM> and END:VALARM is replaced with </VALARM>. Each type in VEVENT/VALARM has a corresponding tag in uppercase. All attributes are lowercase. Any escaped semicolon or comma in VEVENT/VALARM is also passed prefixed with a backslash in XML. Any \n is replaced with a real end of line. The VEVENT/VALARM attributes LANGUAGE, VALUE and CHARSET must not be present in XML. The attribute ENCODING="BASE64" is replaced with ENCODED="base64", any other encoding scheme is silently dropped. The client can also send an attribute ENCODED="utf-8" as this is the default encoding for tags. The format for a timestamp is: yyyy-mm-ddThh:mm:ssZ. All times are UTC/GMT times according **Representation of dates and times** [ISO.8601.1988]. As VEVENT/VALARM attribute VALUE is missing then the format of the value is detected automatically based on the context. For example, a value starting with 'http://' corresponds to a 'VALUE=uri'; an 'ENCODED="base64"' corresponds to a 'VALUE=binary'. Were VEVENT/VALARM standar accepts a date or a date-time then we expect to receive a timestamp (date-time). There are not accepted tags that corresponds to VEVENT/VALARM types having 'VALUE=uri:CID:', so referincing parts that cannot exists in this XML. The DURATION must be always a relative value (starts with a P or -P). As a VALARM is no more a child of a VEVENT we need to link the two entities. A VEVENT with an associated VALARM must include a VALARM tag having the UID of the associated VALARM. A VALARM must have an UID defined and a RELATED-TO tag holding the UID of the associated VEVENT. Optionally RELATED-TO can have a SOURCE attribute with a value 'vevent'. An VEVENT can have only one VALARM associated.

2.11.3. Response for Configuration Folders

TOC

A response holding the configuration has the following structure:

```
<CONFIGURATION>
    <name>value</name>...
</CONFIGURATION>
```

Figure 6

Example:

```
<CONFIGURATION>
    <CHECK-EACH-MIN>10</CHECK-EACH-MIN>
    <QUOTA>1024</QUOTA>
</CONFIGURATION>
```

Figure 7

2.11.4. Response for Contact Folders

TOC

The format is derived from the one defined by the **vCard** [RFC2426] standard.

The following example corresponds to this VCARD definition:

```
BEGIN:VCARD
VERSION:3.0
FN:Iulian Radu
N:Radu;Iulian;;Dipl.Ing.;
ORG:Example Com\, Inc.;European Division
EMAIL;TYPE=internet,home:iulian.radu@gmx.at
TZ:+01:00
REV:2011-05-31T18:46:00Z
LOGO;TYPE=JPEG;ENCODING=b:ABCDEFGHIJ==
LABEL;TYPE=work;TYPE=parcel,intl:1. Operngasse\n1010 Wien\nAustria
END:VCARD
```

Figure 8

Example:

```
<VCARD>
   <VERSION>3.0</VERSION>
   <FN>Iulian Radu</FN>
   <N>Radu;Iulian;;Dipl.Ing.;</N>
   <ORG>Example Com\, Inc.;European Division</ORG>
   <EMAIL TYPE="internet,home">iulian.radu@gmx.at</EMAIL>
   <TZ>+01:00</TZ>
   <REV>2011-05-31T18:46:00Z</REV>
   <LOGO TYPE="image/jpeg" ENCODED="base64">ABCDEFGHIJ==</LOGO>
   <LABEL TYPE="work,parcel,intl">1. Operngasse
```

```
1010 Wien
Austria</LABEL>
</VCARD>
```

Figure 9

The BEGIN:VCARD is replaced with <VCARD> and END:VCARD is replaced with </VCARD>. Each type in VCARD has a corresponding tag. As in RFC all types are defined in uppercase so are also defined all tags. All attributes are lowercase. All values of attribute TYPE from a VCARD must to be gathered together in a single list and passed as attribute TYPE to the corresponding tag (they are delimited by commas). Any escaped semicolon or comma in VCARD is also passed prefixed with a backslash in XML. Any \n is replaced with a real end of line. The VCARD attributes LANGUAGE, VALUE, CONTEXT and CHARSET must not be present in XML. The attribute ENCODING="b" is replaced with ENCODED="base64", any other encoding scheme is silently dropped. The client can also send an attribute ENCODED="utf-8" as this is the default encoding for tags. The format for a date is: YYYY-MM-DD. The format for a time is: HH:MM:SS. The format for a timestamp is: yyyy-mm-ddThh:mm:ssZ. All times are UTC/GMT times according **Representation of dates and times** [ISO.8601.1988]. As VCARD attribute VALUE is missing then the format of the value is detected automatically based on the context. For example, a value starting with 'http://' corresponds to a 'VALUE=uri'; an 'ENCODED="base64"' corresponds to a 'VALUE=binary'. Were VCARD standar accepts a date or a date-time then we expect to receive a timestamp (date-time). It is not accepted a SOURCE VCARD type as the content of this VCARD must to be defined inside of the XML. Also are not accepted tags that corresponds to VCARD types having 'VALUE=uri:CID:', so referincing parts that cannot exists in this XML.

TOC

2.11.5. Response for File Folders

A response holding the content of a file has the following structure:

```
<FILE>
    <NAME>name</NAME>
    <CONTENT ENCODED="utf-8|base64">content</CONTENT>
</FILE>
```

Figure 10

The valid encodings type are: utf-8 and base64. The default encoding is base64.

Example:

```
<FILE>
    <NAME>Example.txt</NAME>
    <CONTENT ENCODED="base64">c3VyZS4=</CONTENT>
</FILE>
```

Figure 11

TOC

2.11.6. Response for Filter Folders

An entryTag can be: AND, OR, NOT, UID, TAG, IS, REGEX. The value associated to entryTag is specified as an XML text node. The IS and REGEX tags have two attributes: PATH and OP. Their values are set as for a filter command (see chapter "Syntax of a Filter" for more information). The tag RULES group all its rules in an AND group.

There must be assigned at least one folder and must be present at least a rule. Optionally can be gived a description using ABOUT tag. Cannot be assigned as folders for being searched folders of the following types: FILT, FOLD.

A response holding the content of a file has the following structure:

```
<FILTER>
    <ABOUT>...</ABOUT>?
    <FOLDERS>
        <FOLDER>...</FOLDER>...
    </FOLDERS>
    <RULES>
        <entryTag>...</entryTag>...
    </RULES>
</FILTER>
```

Figure 12

Example:

```
<FILTER>
    <ABOUT>A sample FILT filter.</ABOUT>
    <FOLDERS>
        <FOLDER>/INBOX</FOLDER>
        <FOLDER>/Spam</FOLDER>
    </FOLDERS>
    <RULES>
        <OR>
            <IS PATH="header/subject" OP="=">Viagra</IS>
            <AND>
                <UID>UIDx1234:UIDx4321</UID>
                <TAG>SPAM</TAG>
            </AND>
        </OR>
    </RULES>
</FILTER>
```

Figure 13

TOC

2.11.7. Response for Journal Folders

The format is derived from the one defined for VJOURNAL by the **iCalendar** [RFC5545] standard.

The following example corresponds to this VJOURNAL definition:

```
BEGIN:VJOURNAL
UID:20110531T184600Z-123456@agap.at
DTSTAMP:2011-06-07T13:52:38Z
DTSTART:2011-06-07T13:52:38Z
SUMMARY:Revise AGAP Internet-Draft
DESCRIPTION:1. The draft was revised\, saved and
 closed.\n2. It is ready to be published.\n
ATTACH;FMTTYPE=text/plain;VALUE=BINARY;ENCODING=BASE64:
 ABCDEFGHIJ==
END:VJOURNAL
```

Figure 14

Example:

```
<VJOURNAL>
    <UID>20110531T184600Z-123456@agap.at</UID>
    <DTSTAMP>2011-06-07T13:52:38Z</DTSTAMP>
    <DTSTART>2011-06-07T13:52:38Z</DTSTART>
    <SUMMARY>Revise AGAP Internet-Draft</SUMMARY>
    <DESCRIPTION>1. The draft was revised\, saved and closed.
2. It is ready to be published.
</DESCRIPTION>
    <ATTACH FMTTYPE="text/plain" ENCODED="base64">
        ABCDEFGHIJ==</ATTACH>
</VJOURNAL>
```

Figure 15

The BEGIN:VJOURNAL is replaced with <VJOURNAL> and END:VJOURNAL is replaced with </VJOURNAL>. Each type in VJOURNAL has a corresponding tag in uppercase. All attributes are lowercase. Any escaped semicolon or comma in VJOURNAL is also passed prefixed with a backslash in XML. Any \n is replaced with a real end of line. The VJOURNAL attributes LANGUAGE, VALUE and CHARSET must not be present in XML. The attribute ENCODING="BASE64" is replaced with ENCODED="base64", any other encoding scheme is silently dropped. The client can also send an attribute ENCODED="utf-8" as this is the default encoding for tags. The format for a timestamp is: yyyy-mm-ddThh:mm:ssZ. All times are UTC/GMT times according **Representation of dates and times** [ISO.8601.1988]. As VTODO attribute VALUE is missing then the format of the value is detected automatically based on the context. For example, a value starting with 'http://' corresponds to a 'VALUE=uri'; an 'ENCODED="base64"' corresponds to a 'VALUE=binary'. Were VJOURNAL standar accepts a date or a date-time then we expect to receive a timestamp (date-time). There are not accepted tags that corresponds to VJOURNAL types having 'VALUE=uri:CID:', so referincing parts that cannot exists in this XML.

TOC

2.11.8. Response for Message Folders

A response holding the content of a message has the following structure:

```
<MESSAGE>
    <HEADER>
        <header-entry-once>value</header-entry-once>...
        <header-entry-multi>value 1
 value 2
 ...
 value n...</header-entry-multi>...
    </HEADER>
    <TEXT? ENCODED="utf-8|base64">main text</TEXT>
    <HTML? ENCODED="utf-8|base64">main html</HTML>
    <ATTACHMENT-{id}*>
        <HEADER>
            ...
        </HEADER>
        <BODY ENCODED="utf-8|base64">
            ...
        </BODY>
    </ATTACHMENT-{id}>...
</MESSAGE>
```

Figure 16

The first attachment id has value 1.

The id of on item tag shows the order of the entries in the original message.

The default content encoding is utf-8. It is assumed that the content for TEXT and HTML is encoded in UTF-8 when the ENCODED attribut has the value base64.

The entries in the header of the main message and attachments are the same with the one from the e-mail message.

There can be at most 2,147,483,647 attachments defined and their numbers must be sequential starting with 1.

Example:

```
<MESSAGE>
    <HEADER>
        <from>example@no-spam.com</from>
        <to>example@example.com</to>
        <received>
            <item>
from mail.yahoo.com by example.com; Tue, 16 Mar 2010 12:14:24 +0100
            </item>
            <item>
from no-spam.com by mail.yahoo.com; Mon, 15 Mar 2010 11:13:23 +0100
            </item>
        </received>
        <content-type>multipart/mixed; boundary="XYZ"</content-type>
        <subject>A basic example</subject>
    </HEADER>
    <TEXT>Please see the attachments.</TEXT>
    <HTML>
&lt;b&gt;Please&lt;/b&gt; see the &lt;u&gt;attachments&lt;/u&gt;.
    </HTML>
    <ATTACHMENT-1/>
      <HEADER>
        <content-type>text/plain</content-type>
      </HEADER>
      <BODY ENCODED="utf-8">See the picture.</BODY>
    </ATTACHMENT-1>
    <ATTACHMENT-2>
      <HEADER>
        <content-type>image/jpeg</content-type>
        <content-transfer-encoding>base64</content-transfer-encoding>
      </HEADER>
      <BODY ENCODED="base64">c3VyZS4=</BODY>
    </ATTACHMENT-2>
</MESSAGE>
```

Figure 17

The previous example corresponds to a message with the following structure:

- multipart/mixed
    - multipart/alternative
        - text/plain
        - text/html
    - text/plain
    - image/jpeg

2.11.9. Response for Note Folders

A response holding the content of the note has the following structure:

```
<NOTE>
    <SUBJECT>...</SUBJECT>
    <CONTENT TYPE="text/..." ENCODED="utf-8|base64">...</CONTENT>
</NOTE>
```

Figure 18

Note: the subject can be any short text. The content can be encoded UTF-8 or BASE64. Implicit is content encoded in utf-8. The type can be any subtype of 'text/*'. Implicit is 'text/plain'. It is recommended to be used only 'text/plain' and 'text/html'.

Example:

```
<NOTE>
    <SUBJECT>Important!</SUBJECT>
    <CONTENT TYPE="text/plain" ENCODED="utf-8">
        To review the code.</CONTENT>
</NOTE>
```

Figure 19

2.11.10. Response for Task Folders

The format is derived from the one defined for VTODO by the **iCalendar** [RFC5545] standard.

The following example corresponds to this VTODO definition:

```
BEGIN:VTODO
UID:20110531T184600Z-123456@agap.at
DTSTAMP:2011-05-31T18:46:00Z
DTSTART:2011-06-07T12:00:00Z
DUE:2011-06-07T14:00:00Z
COMPLETED:2011-06-07T13:52:38Z
SUMMARY:Revise AGAP Internet-Draft
PRIORITY:1
STATUS:NEEDS-ACTION
ATTACH;FMTTYPE=text/plain;VALUE=BINARY;ENCODING=BASE64:
 ABCDEFGHIJ==
END:VTODO
```

Figure 20

Example:

```
<VTODO>
    <UID>20110531T184600Z-123456@agap.at</UID>
    <DTSTAMP>2011-05-31T18:46:00Z</DTSTAMP>
    <DTSTART>2011-06-07T12:00:00Z</DTSTART>
```

```
    <DUE>2011-06-07T14:00:00Z</DUE>
    <COMPLETED>2011-06-07T13:52:38Z</COMPLETED>
    <SUMMARY>Revise AGAP Internet-Draft</SUMMARY>
    <PRIORITY>1</PRIORITY>
    <STATUS>NEEDS-ACTION</STATUS>
    <ATTACH FMTTYPE="text/plain" ENCODED="base64">
        ABCDEFGHIJ==</ATTACH>
    <VALARM>20110607T170000Z-20110531T184600Z-123456@agap.at</VALARM>
</VTODO>
```

Figure 21

The following example corresponds to this alarm definition related to the previous task:

```
BEGIN:VALARM
TRIGGER;RELATED=START:2011-06-07T17:00:00Z
REPEAT:3
DURATION:PT15M
ACTION:AUDIO
ATTACH;FMTTYPE=audio/mpeg;VALUE=BINARY;ENCODING=BASE64:
 ABCDEFGHIJ==
END:VALARM
```

Figure 22

Example alarm:

```
<VALARM>
    <UID>20110607T170000Z-20110531T184600Z-123456@agap.at</UID>
    <RELATED-TO SOURCE="vtodo">
       20110531T184600Z-123456@agap.at</RELATED-TO>
    <TRIGGER RELATED="start">2011-06-07T17:00:00Z</TRIGGER>
    <REPEAT>3</REPEAT>
    <DURATION>PT15M</DURATION>
    <ACTION>AUDIO</ACTION>
    <ATTACH FMTTYPE="audio/mpeg" ENCODED="base64">
        ABCDEFGHIJ==</ATTACH>
</VALARM>
```

Figure 23

The BEGIN:VTODO is replaced with <VTODO> and END:VTODO is replaced with </VTODO>. The BEGIN:VALARM is replaced with <VALARM> and END:VALARM is replaced with </VALARM>. Each type in VTODO/VALARM has a corresponding tag in uppercase. All attributes are lowercase. Any escaped semicolon or comma in VTODO/VALARM is also passed prefixed with a backslash in XML. Any \n is replaced with a real end of line. The VTODO/VALARM attributes LANGUAGE, VALUE and CHARSET must not be present in XML. The attribute ENCODING="BASE64" is replaced with ENCODED="base64", any other encoding scheme is silently dropped. The client can also send an attribute ENCODED="utf-8" as this is the default encoding for tags. The format for a timestamp is: yyyy-mm-ddThh:mm:ssZ. All times are UTC/GMT times according **Representation of dates and times** [ISO.8601.1988]. As VTODO/VALARM attribute VALUE is missing then the format of the value is detected automatically based on the context. For example, a value starting with 'http://' corresponds to a 'VALUE=uri'; an 'ENCODED="base64"' corresponds to a 'VALUE=binary'. Were VTODO/VALARM standars accept a date or a date-time then we expect to receive a timestamp (date-time). There are not accepted tags that corresponds to VTODO/VALARM types having 'VALUE=uri:CID:', so referincing parts that cannot exists in this XML. The DURATION must be always a relative value (starts with a P or -P). As a VALARM is no more a child of a VTODO we need to link the two entities. A VTODO with an associated VALARM must include a VALARM tag having the UID of the associated VALARM. A VALARM

must have an UID defined and a RELATED-TO tag holding the UID of the associated VTODO. Optionally RELATED-TO can have a SOURCE attribute with a value 'vtodo'. An VTODO can have only one VALARM associated.

3. States

TOC

3.1. Not-authenticated State

TOC

This is the default state when a new connection is made to the server. The client becomes a welcome message.

From this state the client can use the command 'AUTH mechanism' to move in the 'Pre-authentication State'. This is the only other state in which the server can go.

The client can use the command 'STLS' for commuting in the encrypted mode of the channel. After STLS the server remains in the 'Not-authenticated State'. There is no command for switching back to clear-text communication.

The client can use the command 'SGZP' for commuting in the compressed mode of the channel. After SGZP the server remains in the 'Not-authenticated State'. There is no command for switching back to not-compressed communication.

A client can use at the same time the both modes (encrypted and compressed).

The client can use the command 'QUIT' for terminating the connection.

For finding what extensions are installed in server, the client can use the 'CAPA' command.

3.2. Pre-authentication State

TOC

This is the state where a client authenticate itself and move to the 'Authenticated State' or returns to the 'Not-authenticated State'.

This standard defines only one method for AUTH: PLAIN. Following is a description of the commands flow used by this authentication mechanism.

The client must send a 'USER account' followed by a 'PASS password' (if the server confirms the acceptance of the account name). If the pair account and password is accepted then the server move to the state 'Authenticated State' and the folder INBOX is selected by server. If this folder does not exist then the server moves in the 'Not-Selected State' and the client must to select an existing folder for operating with this account. If this pair is rejected then the server returns to the 'Not-authenticated State'. That means that the client must to send a new 'AUTH mechanism' for trying a new authentication.

The client can use the command 'QUIT' for terminating the connection.

A client can enter into this state only after a successful 'AUTH' command in 'Not-authenticated State'.

3.3. Authenticated (and Selected) State

TOC

This is the state from which a client operates with the content of an account.

From this state the client can use the command 'EXIT' to move in the 'Not-authenticated State'. After an unsuccessful SLCT, the server goes in 'Not-selected State'.

The client can use the command 'QUIT' for terminating the connection.

Check the following chapter for finding which commands can be performed from this state.

A client can enter into this state only after a successful authentication in the 'Pre-authenticated State' or after a successful 'SLCT' command in the 'Authenticated State' or 'Not-selected State'.

3.4. (Authenticated but) Not-selected State

TOC

This is the state from which a client must to select a folder for performing further operations.

From this state the client must use the command 'SLCT' to select a folder and to move in the 'Authenticated State'. This is the only other state in which the server can go.

The client CAN use the command 'LIST' for finding valid folder names that eventually CAN be selected with 'SLCT' command.

The client CAN use the command 'QUIT' for terminating the connection.

A client CAN enter into this state only after an unsuccessful 'SLCT' command or if the INBOX folder does not exists and it cannot be selected automatically after a successful authentication.

3.5. Presence State

TOC

This is the state in which a client can only ask information about the presence of an user/account.

In this state the client can use only the command 'PNFO USER' to ask for presence information of an account and the command 'QUIT' for terminating the connection.

3.6. Storing State

TOC

This is the state in which a client can only add items (for example: messages, events) in an account which it is not his/her.

In this state the client can use only the command 'FSTO' to find and store the item into a folder of specified type from specified user and the command 'QUIT' for terminating the connection.

---

4. Commands

TOC

4.1. Semantic and Syntax

TOC

Each command has its name from 4 letters and it is matched case-insensitive.

Each command is separated by its arguments by a 0x20 character. Also, each argument is separated from its adjacent arguments by a 0x20 character.

The minimal response has only the return code without any text.

A list of elements is enclosed between parentheses (round brackets).

4.2. Syntax of a Tag List

TOC

A tag list is used by the following commands: FTAG, GTAG and STAG.

A tag list defines what action to be done with its tags.

Syntax: ACTION TAG TAG=VALUE ...

ACTION:

- = - set only these tags;
- + - add this tags
- - - delete these tags.

Example:

```
C: STAG UIDx1234 = SEEN SPAM=YES
C: STAG UIDx1234 + SEEN FLAG=RED
C: STAG UIDx1234 - FLAG JUNK
```

Figure 24

Note: After this three commands we have only the following tags: SEEN SPAM=YES JUNK.

TOC

4.3. Syntax of a Filter

TOC

4.3.1. Syntax of a Filter for a Command

A filter of this type is used by the following commands: FCPY, FDEL, FTAG, FIND and FMOV.

A filter defines rules for matching entries. It is defined as lines with rules and it is ended by an empty line.

The keywords of the filter are case insensitive matched (ex.: UID and Uid are the same).

A rule must be completely defined in the same line (exception are grouping, AND, OR, and NOT rules).

Accepted rules:

- ( ) - grouping for AND and OR;
- AND - all following rules are with AND bonded (until the end of the current group). It is the implicit rule when the first rule is not an AND or an OR;
- OR - all following rules are with OR bonded (until the end of the current group);
- NOT - invert the result of the following rule;
- UID uid - one UID;
- UID uid_begin_range:uid_end_range - inclusive range;
- TAG tag_name - a tag of an entry;
- TAG tag_name=tag_value - an entry's tag with a value (tag_value is the complete value);
- IS field_path op string - a field from the content (as XML) with an exact matched text (string is written between ' and ' can be escaped with \'); op can be: <, <=, =, !=, >=, >;
- REGEX field_path op regex_string - a field from the content (as XML) with a regular expression matched text (regex_string is written between ' and ' can be escaped with \'); op can be: =, !=; the regex_string can match only a part of the content.

The field_path is a PATH in the response as it is returned by RETR and must point to an end leaf. It contains only tag names separated with /. Example: /MESSAGE/HEADER/subject, /MESSAGE/HEADER/received, /MESSAGE/BODY/HTML, /MESSAGE/BODY/ATTACHMENT-1/BODY.

Searching for a TAG without associating and a value to it will match all entries that have this

tag even if it have values set for it. For values which are lists with values delimited by comma or semi-colon it is possible to extract an item using square brackets ([]) for lists delimited by comma and curly brackets ({}) for lists delimited by semi-colons.

It can be searched only in the body of attachments that have a content type of type 'text/*'.

Example 1: These filters find all messages with the UID between UIDx0001:UIDx1000 and that were seen and marked as being spam or having a virus (the AND is redundant in the second case). Both filter definitions are equivalent.

```
C: UID UIDx0001:UIDx1000 OR ( TAG SPAM TAG HAS=VIRUS ) TAG SEEN
C: UID UIDx0001:UIDx1000 AND( OR ( TAG SPAM TAG HAS=VIRUS ) TAG SEEN)
```

Figure 25

Example 2:

```
C: IS /MESSAGE/HEADER/SUBJECT = 'From University'
C: REGEX /MESSAGE/HEADER/FROM != '[^0-9]+@example\.com$'
C: IS /VCARD/FN = 'Anonymous'
C: REGEX /VCARD/ORG = '^[A-Za-z]+[0-9]$'
```

Figure 26

TOC

4.3.2. Syntax of a Filter for a FILT Folder

A filter of this type is used by the following command: STOR.

A filter defines rules for matching the different messages from different folders. It is defined as an XML with target folders and rules.

The keywords of the filter are case sensitive matched (ex.: UID and Uid are not the same). They are always lowercase.

Accepted rules:

- AND - all its entries must be matched;
- OR - at least one of its entries must be matched;
- NOT - invert the result of its child rule;
- UID uid - one UID;
- UID uid_begin_range:uid_end_range - inclusive range;
- TAG tag_name - a tag;
- TAG tag_name=tag_value - a tag with a value (tag_value is the complete value);
- IS field_path op string - a field from the content (as XML) with an exact matched text (string is written between ' and ' can be escaped with \'); op can be: <, <=, =, !=, >=, >;
- REGEX field_path op regex_string - a field from the content (as XML) with a regular expression matched text (regex_string is written between ' and ' can be escaped with \'); op can be: =, !=; the regex_string can match only a part of the content.

The field_path is a PATH in the response as it is returned by RETR and must point to an end leaf. It contains only tag names separated with /. Example: /MESSAGE/HEADER/subject, /MESSAGE/HEADER/received, /MESSAGE/BODY/HTML, /MESSAGE/BODY/ATTACHMENT-1/BODY.

Searching for a TAG without associating and a value to it will match all entries that have this tag even if it have values set for it (the empty string is also considered matched).

The following two examples corresponds to the two examples from the previous chapter:

```
<FILTER>
    <FOLDERS>
        <FOLDER>/INBOX</FOLDER>
    </FOLDERS>
    <RULES>
        <AND>
            <UID>UIDx0001:UIDx0010</UID>
            <OR>
                <TAG>SPAM</TAG>
                <TAG>HAS=VIRUS</TAG>
            </OR>
            <TAG>SEEN</TAG>
        </AND>
    </RULES>
</FILTER>
```

Figure 27

Example 2:

```
<FILTER>
  <FOLDERS>
    <FOLDER>/INBOX</FOLDER>
  </FOLDERS>
  <RULES>
    <OR>
      <IS PATH="/MESSAGE/HEADER/SUBJECT" OP="=">From University</IS>
      <REGEX PATH="/MESSAGE/HEADER/FROM" OP="!=">
         [^0-9]+@example\.com$</REGEX>
      <IS PATH="/VCARD/FN" OP="=">Anonymous</IS>
      <REGEX PATH="/VCARD/ORG" OP="=">^[A-Za-z]+[0-9]$</REGEX>
    </OR>
  </RULES>
</FILTER>
```

Figure 28

TOC

4.4. The Welcome Message - not-authenticated state

Results: 200 401 410 531

Result 200 - the client is accepted for sending commands;

Result 401 - there was an internal error;

Result 410 - too many connections;

Result 531 - the client is rejected permanently.

Description: When a client connects to the server it receives a welcome message. This message begins with a response code that shows if the client is accepted for sending commands.

Examples:

```
S: 200 Welcome localhost [127.0.0.1]
```

Figure 29

```
S: 401 Internal error, please contact our administrator
```

Figure 30

```
S: 410 Sorry, too many connections, please retry later
```

Figure 31

```
S: 531 Your hostname/IP (localhost:127.0.0.1) is blacklisted
```

Figure 32

TOC

4.5. Command QUIT - all states

Name: quit

Arguments: none

Result: 200

Description: The QUIT command close the connection between the client and server.

Example:

```
C: QUIT
S: 200 OK Bye
```

Figure 33

TOC

4.6. Command AUTH mechanism - not-authenticated state

Name: authenticate

Argument: mechanism

Results: 200 510 511

Result 200 - the mechanism is known and accepted.

Result 510 - unknown command.

Result 511 - the mechanism is unknown/unsupported.

Description: Choose an authentication method. The name of the mechanism can contain only latin letters (A-Z), digits (0-9), the signs minus (-) and underscore (_). It is case

insensitive. All supported mechanisms must to be advertised in CAPA's list of capabilities as AUTH-MechanismNameInUpperCase.

The PLAIN Authentication Mechanism: the client send the username and password in clear text using the commands USER and PASS.

The MD5 and SHA1 Authentication Mechanisms: the server send an additional line starting with a dot and providing a prefix that will gone be used by the client to send back to the server an MD5 or SHA-1 value computed on the string build from this prefix and user's password. This prefix can have between 1 and 256 characters. Allowed characters are any UTF-8 characters having a code bigger the decimal value 31 (first valid character is space). The initial dot is not part of the prefix. The client send the username and the computed hash using the commands USERR and HASH.

The PRESENCE Authentication Mechanism: this mechanism is used by a client to query the presence of an user having an account on the server. If the server knows to forward the request to other servers (in case that requested account in not local) then it can return the answer from the remote server. The client send the username and password in clear text using the commands USER and PASS. For an anonymous access the server can accept as username anonymous and as password the email address of the connecting user. Once the username and password are accepted, the server enters in the presence state and the client can execute only the commands PNFO USER and QUIT.

The PRESENCE-MD5 and PRESENCE-SHA1 Authentication Mechanisms: these mechanisms are working similar with MD5 and SHA1 authentication mechanisms, only that move the server in the same status as PRESENCE authentication mechanism.

The STORING Authentication Mechanism: this mechanism is used by a client to store items in accounts which are not his/hers. The client send the username and password in clear text using the commands USER and PASS. For an anonymous access the server can accept as username anonymous and as password the email address of the connecting user. Once the username and password are accepted, the server enters in the storing state and the client can execute only the commands FSTO and QUIT.

The STORING-MD5 and STORING-SHA1 Authentication Mechanisms: these mechanisms are working similar with MD5 and SHA1 authentication mechanisms, only that move the server in the same status as STORING authentication mechanism.

Examples:

```
C: AUTH PLAIN
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send PASS
C: PASS email@example.com
S: 200 OK User anonymous authenticated
```

Figure 34

```
C: AUTH MD5
S: .Hash prefix ... for user's password
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send HASH
C: HASH 79054025255fb1a26e4bc422aef54eb1
S: 200 OK Authenticated
```

Figure 35

```
C: AUTH SHA1
S: .Hash prefix ... for user's password
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send HASH
C: HASH de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b4
S: 200 OK Authenticated
```

Figure 36

```
C: AUTH PRESENCE
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send PASS
C: PASS email@example.com
S: 200 OK User anonymous authenticated
```

Figure 37

```
C: AUTH PRESENCE-MD5
S: .Hash prefix ... for user's password
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send HASH
C: HASH 79054025255fb1a26e4bc422aef54eb1
S: 200 OK User anonymous authenticated
```

Figure 38

```
C: AUTH PRESENCE-SHA1
S: .Hash prefix ... for user's password
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send HASH
C: HASH de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b4
S: 200 OK User anonymous authenticated
```

Figure 39

```
C: AUTH STORING
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send PASS
C: PASS email@example.com
S: 200 OK User anonymous authenticated
```

Figure 40

```
C: AUTH STORING-MD5
S: .Hash prefix ... for user's password
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send HASH
C: HASH 79054025255fb1a26e4bc422aef54eb1
S: 200 OK User anonymous authenticated
```

Figure 41

```
C: AUTH STORING-SHA1
S: .Hash prefix ... for user's password
S: 200 OK Send USER
C: USER anonymous
S: 200 OK Send HASH
C: HASH de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b4
S: 200 OK User anonymous authenticated
```

Figure 42

```
C: AUTH
S: 510 UNKNOWN command
```

Figure 43

```
C: AUTH unknown
S: 511 UNKNWON method
```

Figure 44

TOC

4.7. Command CAPA - not-authenticated state

Name: capabilities

Arguments: none

Result: 200

Description: Ask for the parts of this standards or extensions supported by the server.

Following is a list with all capabilities defined and covered by this document. If the server do no present an entry from the following list then the client must assume that the sever is unable to do the associated operations of the missing entry.

- ADDR - contact information;
- AUTH-PLAIN - suport plain authentication;
- AUTH-PRESENCE - suport authentication for asking about presence;
- AUTH-PRESENCE-MD5 - suport authentication for asking about presence;
- AUTH-PRESENCE-SHA1 - suport authentication for asking about presence;

- AUTH-MD5 - suport MD5 authentication;
- AUTH-SHA1 - suport SHA1 authentication;
- AUTH-STORING - suport authentication for receiving items;
- AUTH-STORING-MD5 - suport authentication for receiving items;
- AUTH-STORING-SHA1 - suport authentication for receiving items;
- CALE - events;
- CHNG - list the FCID of all folders;
- CONF - user accounts configuration;
- FILE - storing files;
- FILT - definition of a filter;
- JRNL - journal entries;
- MESG - e-mail messages;
- NOTE - short texts;
- PNFO - presence;
- SGZP - server accepts compression;
- STLS - server can encrypt the communication channel;
- TASK - tasks.

Example:

```
C: CAPA
S: .GZIP
S: .TLS
S: .Extension1
S: .Extension.2 argument1
S: .Extension-3 argument1 argument2
S: 200 OK CAPA completed
```

Figure 45

TOC

4.8. Command SGZP - not-authenticated state

Name: start using GZip

Arguments: none

Results: 200 510

Result 200 - the communication is now compressed.

Result 510 - unknown/unsupported command.

Description: Change the communication in compressed mode using **GZIP** [RFC1952] as compression method. If this command is executed from the compression mode then it simply returns a 200 response code. The response to this command is using still the not-compressed mode of the channel. The compression becomes effective only after a 200 response line was send by the server.

Note: With GZIP the data is compressed using the LZ77 algorithm and Huffman coding. Starting using this mode is like starting to write clear texts into a GZIP format archive and reading texts from a GZIP format archive. The compression is used both by the client and the server and they start to use it with the next line they send after the 200 response line received from the server.

Examples:

```
C: SGZP
S: 200 OK Using GZIP
```

Figure 46

```
C: SGZP
S: 510 UNKNOWN command
```

Figure 47

4.9. Command STLS - not-authenticated state

TOC

Name: start using TLS

Arguments: none

Results: 200 510

Result 200 - the communication is now encrypted.

Result 510 - unknown/unsupported command.

Description: Change the communication in mode TLS. If this command is executed from the encrypted mode then it simply returns a 200 response code. The response to this command is using still the not-encrypted mode of the channel. The encryption becomes effective only after a 200 response line was send by the server.

Examples:

```
C: STLS
S: 200 OK Using TLS
```

Figure 48

```
C: STLS
S: 510 UNKNOWN command
```

Figure 49

4.10. Command HASH - pre-authenticated state (MD5 and SHA1)

TOC

Name: hash

Argument: hash_code

Result: 200 510 511 512

Result 200 - the pair user/hash was successfully authenticated.

Result 510 - unknown/unsupported command.

Result 511 - invalid hash.

Result 512 - first send USER and then HASH.

Description: Send the hash code associated to the previous authentication method (MD5 or SHA1), previous USER and provided prefix.

Examples:

```
C: AUTH MD5
S: .prefix is-here!
S: 200 OK Send USER
C: USER account
S: 200 OK Send HASH
C: HASH 79054025255fb1a26e4bc422aef54eb1
S: 200 OK Authenticated
```

Figure 50

```
C: USER account
S: 200 OK Send HASH
C: HASH
S: 510 UNKNOWN command
```

Figure 51

```
C: USER account
S: 200 OK Send HASH
C: HASH 79054025255fb1a26e4bc422aef54eb1
S: 511 WRONG user/hash pair
```

Figure 52

```
C: AUTH SHA1
S: .prefix is-here!
S: 200 OK Send USER
C: HASH de9f2c7fd25e1b3afad3e85a0bd17d9b100db4b3
S: 512 EXPECTED USER
```

Figure 53

TOC

4.11. Command PASS - pre-authenticated state (PLAIN)

Name: password

Argument: password

Result: 200 510 511 512

Result 200 - the pair user/password was successfully authenticated.

Result 510 - unknown/unsupported command.

Result 511 - invalid password.

Result 512 - first send USER and then PASS.

Description: Send the password associated to the previous USER.

Examples:

```
C: USER account
S: 200 OK Send PASS
C: PASS password
S: 200 OK Authenticated
```

Figure 54

```
C: USER account
S: 200 OK Send PASS
C: PASS
S: 510 UNKNOWN command
```

Figure 55

```
C: USER account
S: 200 OK Send PASS
C: PASS password
S: 511 WRONG user/password pair
```

Figure 56

```
C: AUTH PLAIN
S: 200 OK AUTH completed
C: PASS password
S: 512 EXPECTED USER
```

Figure 57

TOC

4.12. Command USER - pre-authenticated state (PLAIN, MD5 and SHA1)

Name: user

Argument: account

Result: 200

Result: 200 510 511

Result 200 - the user is accepted and expecting the password.

Result 510 - unknown/unsupported command.

Result 511 - invalid account.

Description: Send an account name for authentication and authorization.

Examples:

```
C: AUTH PLAIN
S: 200 OK Send USER
C: USER account
S: 200 OK Send PASS
```

Figure 58

```
C: AUTH PLAIN
S: 200 OK Send USER
C: USER
S: 510 UNKNOWN command
```

Figure 59

```
C: AUTH PLAIN
S: 200 OK Send USER
C: USER account
S: 511 INVALID username
```

Figure 60

TOC

4.13. Command CHNG - authenticated and not-selected state

Name: report the FCID (Folder Change ID) for all folders

Arguments: path?

Result: 200 510 511

Result 200 - the command was successful.

Result 510 - unknown/unsupported command.

Description: Return a list with the FCID of all folders. In the list is included and the root folder for all other foldersa as slash ('/').

Examples:

```
C: CHNG
S: .0BIH /
S: .0009 /Temporary
S: .0001 /Temporary/1980
S: .0BIG /INBOX
S: .0123 /ARCHIVE
S: .0003 /ARCHIVE/2010
S: .0003 /ARCHIVE/2011
S: .00aA /ARCHIVE/2010/OLD
S: 200 OK CHNG completed
```

Figure 61

Note: A change in /ARCHIVE/2010 will change the FCID of /ARCHIVE/2010, but not the FCID of /ARCHIVE nor /.

```
C: CHNG
S: 510 UNKNOWN command
```

Figure 62

TOC

4.14. Command COPY - authenticated state

Name: copy entry

Arguments: UID_source path_destination_folder

Result: 200 510 511

Result 200 - the copy was successful.

Result 510 - unknown/unsupported command.

Result 511 - unknown uid, invalid destination folder or path not absolute.

Description: Copy a message/contact/event from the currently selected folder into another folder (by UID). You cannot copy from a folder having the tag NO-COPY or into a folder with the tag READ-ONLY.

Note: For copying a folder the client must use CPYF.

Examples:

```
C: COPY UIDx1234 /ARCHIVE_FOLDER/TODAY
S: 200 OK COPY completed
```

Figure 63

```
C: COPY
S: 510 UNKNOWN command
```

Figure 64

```
C: COPY UIDx1234 ARCHIVE_FOLDER/TODAY
S: 511 INVALID UID
C: COPY MSGx1234 ARCHIVE_FOLDER/1970
S: 511 INVALID Destination
```

Figure 65

4.15. Command CPYF - authenticated state

TOC

Name: copy folder

Arguments: path_destination_folder

Result: 200 510 511

Result 200 - the copy was successful.

Result 510 - unknown/unsupported command.

Result 511 - invalid destination folder, destination is not an absolute path or destination does not exists.

Description: Copy the content of a folder into another folder.

Note: In the destination folder are copied all non-folder entries found in the source. If the user needs to copy the content of the source folder in another folder then he must to create first a new folder with the MAKE command and then use the CPYF command. If the user wants to copy the folders found in the source then he must to do recursively MAKE and CPYF for each subfolder.

Examples (in TODAY are copied only the messages from INBOX):

```
C: SLCT /INBOX
S: 200 Selected /INBOX
C: CPYF /ARCHIVE_FOLDER/TODAY
S: 200 OK CPYF completed
```

Figure 66

```
C: CPYF
S: 510 UNKNOWN command
```

Figure 67

```
C: CPYF MISSING
S: 511 INVALID Destination
```

Figure 68

4.16. Command DATT - authenticated state (MESG folder type)

TOC

Name: delete attachment

Arguments: UID AttNum

Result: 200 510 511 521

Result 200 - the attachment was successfully deleted.

Result 510 - unknown/unsupported command.

Result 511 - unknown uid or uid is not for a message.

Result 521 - wrong attachment number.

Description: Delete from a message an attachment. The first attachment has number 1. If the deleted attachment is not the last one then the remaining attachments are renumbered. There is no guarantee that an attachment will keep its previous number had before the successful DATT command.

Note: It cannot be undone.

Examples:

```
C: DATT UIDx1234 1
S: 200 OK Attachment deleted
```

Figure 69

```
C: DATT
S: 510 UNKNOWN command
```

Figure 70

```
C: DATT UIDx1234 1
S: 511 INVALID UID
C: DATT UIDx1234 0
S: 511 INVALID Attachment number
```

Figure 71

```
C: DATT UIDx1234 10
S: 521 There are not so many attachments
```

Figure 72

TOC

4.17. Command DELE - authenticated state

Name: delete entry

Argument: UID

Result: 200 510 511

Result 200 - the entry was successfully deleted.

Result 510 - unknown/unsupported command.

Result 511 - unknown uid.

Description: Delete a message/contact/event by uid.

Note: It cannot be undone.

Examples:

```
C: DELE UIDx1234
S: 200 OK Message deleted
```

Figure 73

```
C: DELE
S: 510 UNKNOWN command
```

Figure 74

```
C: DELE 1234
S: 511 INVALID UID
```

Figure 75

TOC

4.18. Command DELF - authenticated state

Name: delete folder

Arguments: none

Result: 200 510 511

Result 200 - the folder was successfully deleted.

Result 510 - unknown/unsupported command.

Result 511 - no folder was selected or currently selected folder is a reserved folder.

Description: Delete currently selected folder and all its content and subfolders. A reserved folder cannot be deleted, but a read-only folder yes. If the operation is successful then after it no folder is selected.

Note: It cannot be undone.

Examples:

```
C: DELF
S: 200 OK Folder '/delete/me' was deleted
```

Figure 76

```
C: DELF
S: 510 UNKNOWN command
```

Figure 77

```
C: DELF
S: 511 Please select first a folder
C: DELF
S: 511 /INBOX cannot be deleted
```

Figure 78

TOC

4.19. Command EXIT - authenticated state

Name: exit

Arguments: none

Result: 200

Description: Return the server to the Not-authenticated State.

Example:

```
C: EXIT
S: 200 OK EXIT completed
```

Figure 79

TOC

4.20. Command FCPY - authenticated state

Name: find and copy entries

Arguments: path_destination_folder filter*

Result: 110 200 210 510 511

Result 110 - the client can send the filter.

Result 200 - the find and copy was successful for all found UIDs.

Result 210 - the find and copy was successful but not for all found UIDs.

Result 510 - unknown/unsupported command.

Result 511 - invalid destination folder, wrong filter, or no right to copy.

Description: Search for messages/contacts/events only from the currently selected folder (no subfolders) that correspond to a filter and copy them to a new folder. The tags are also copied. You cannot copy from a folder having the tag NO-COPY or into a folder with the tag READ-ONLY. If there is no match for the filter then it is returned a 200 code.

Note: The filter is delivered after the acceptance of the command (response code 110).

Examples:

```
C: FCPY /ARCHIVE/SPAM
S: 110 OK SEND filter definition (end it with an empty line)
C: UID 00000001:00001000 AND TAG SPAM
C:
S: 200 OK FCPY completed (10 matches)
```

Figure 80

```
C: FCPY /ARCHIVE/SPAM
S: 110 OK SEND filter definition (end it with an empty line)
C: UID 00000001:00001000 AND TAG SPAM
C:
S: 210 OK FCPY completed (8 from 10 were copied - out of space)
```

Figure 81

```
C: FCPY
S: 510 UNKNOWN command
```

Figure 82

```
C: FCPY MISSING
S: 511 INVALID folder or path not absolute
C: FCPY SEND
S: 110 OK SEND filter definition (end it with an empty line)
C: LATER
C:
S: 511 INVALID filter definition
```

Figure 83

TOC

4.21. Command FDEL - authenticated state

Name: find and delete entries

Argument: filter*

Result: 110 200 210 511

Result 110 - the client can send the filter.

Result 200 - the find and delete was successful for all found UIDs.

Result 210 - the find and delete was successful but not for all found UIDs.

Result 511 - wrong filter (inclusive empty filter) or no right to delete.

Description: Search for messages/contacts/events only from the currently selected folder (no subfolders) that correspond to a filter and delete them (no copy in TRASH). You cannot delete from a folder having the tag NO-DELETE. If there is no match for the filter then it is

returned a 200 code.

Note: The filter is delivered after the acceptance of the command (response code 110).

Examples:

```
C: FDEL
S: 110 OK SEND filter definition (end it with an empty line)
C: UID 00000001:00001000 AND TAG SPAM
C:
S: 200 OK FDEL completed (10 matches)
```

Figure 84

```
C: FDEL
S: 110 OK SEND filter definition (end it with an empty line)
C: UID 00000001:00001000 AND TAG SPAM
C:
S: 210 OK FDEL completed (only 8 from 10 matches were deleted)
```

Figure 85

```
C: FDEL
S: 110 OK SEND filter definition (end it with an empty line)
C: LATER
C:
S: 511 INVALID filter definition
```

Figure 86

TOC

4.22. Command FIND - authenticated state

Name: find entry

Argument: filter*

Result: 110 200 511

Result 110 - the client can send the filter.

Result 200 - the find was successful.

Result 511 - wrong filter.

Description: Search for messages/contacts/events only from the currently selected folder (no subfolders) that correspond to a filter and return their UIDs. If the search is done for a filter folder then the server does not expect any filter and apply the current filter (if any). If there is no filter in the filter folder then it is returned only the return code. The answer consists of the UIDs and, for a filter folder, they are followed by a 0x20 character and the absolute path for which are the corresponding UID. If there is no match for the filter then it is returned a 200 code.

Note: For not FILT folders, the filter is delivered after the acceptance of the command. An empty filter matches all messages/contacts/events from that folder.

Examples:

```
C: SLCT /MESG-Folder
C: FIND
S: 110 OK SEND filter definition (end it with an empty line)
C: UID UIDx0001:UIDx9000 TAG SPAM
C:
S: .UIDx1234
S: .UIDx1235
S: .UIDx2340
S: 200 OK FIND completed (3 matches)
C: SLCT /FILT-Folder
C: FIND
S: .UIDx1234 /INBOX
S: .UIDx1234 /Trash
S: .UIDx1235 /Trash
S: 200 OK FIND completed (3 matches)
```

Figure 87

```
C: FIND
S: 110 OK SEND filter definition (end it with an empty line)
C: LATER
C:
S: 511 INVALID filter definition
```

Figure 88

TOC

4.23. Command FMOV - authenticated state

Name: find and move

Arguments: path_destination_folder filter*

Result: 110 200 210 510 511

Result 110 - the client can send the filter.

Result 200 - the find and move was successful for all found UIDs.

Result 210 - the find and move was successful but not for all found UIDs.

Result 510 - unknown/unsupported command.

Result 511 - invalid destination folder, wrong filter, or no right to move.

Description: Search for messages/contacts/events only from the currently selected folder (no subfolders) that correspond to a filter and move them to a new folder. The tags are also moved. You cannot move from a folder having the tag NO-MOVE or into a folder with the tag READ-ONLY. If there is no match for the filter then it is returned a 200 code.

Note: The filter is delivered after the acceptance of the command (response code 110).

Examples:

```
C: FMOV /ARCHIVE/SPAM
S: 110 OK SEND filter definition (end it with an empty line)
```

```
C: UID 00000001:00001000 AND TAG SPAM
C:
S: 200 OK FMOV completed (10 matches)
```

Figure 89

```
C: FMOV /ARCHIVE/SPAM
S: 110 OK SEND filter definition (end it with an empty line)
C: UID 00000001:00001000 AND TAG SPAM
C:
S: 210 OK FMOV completed (8 from 10 moved - out of space)
```

Figure 90

```
C: FMOV
S: 510 UNKNOWN command
```

Figure 91

```
C: FMOV MISSING
S: 511 INVALID folder or not absolute path
C: FMOV /SEND
S: 110 OK SEND filter definition (end it with an empty line)
C: LATER
C:
S: 511 INVALID filter definition
```

Figure 92

TOC

4.24. Command FSTO - storing state

Name: find and write an item into a folder with a specified type from a certain user

Arguments: FolderType Account

Result: 110 200 410 510 511

Result 110 - the requested folder was found and the client can send the item.

Result 200 - the item was successfully stored or there was no content sent by the client.

Result 410 - if the item cannot be stored.

Result 510 - unknown/unsupported command.

Result 511 - invalid folder type, unknown account, the data is not a valid XML or its schema does not correspond to the type of the destination folder.

Description: Locate a folder with a specified type in an user account for receiving items from other users and store there the item sent by the client. It behaves like STOR.

Note: Do not send a message content using CDATA as it can hold empty lines and an empty

line means for the server the end of the message to be stored.

Examples:

```
C: FSTO MESG kontakt@agap.at
S: 110 Send the message ended with an empty line
C: <MESSAGE><HEADER>...</HEADER><TEXT>...</TEXT></MESSAGE>
C:
S: 200 OK Message stored with UID UIDx1234 into INBOX
```

Figure 93

```
C: FSTO
S: 510 UNKNOWN command
C: FSTO MESG
S: 510 UNKNOWN command
```

Figure 94

```
C: FSTO -1 kontakt@agap.at
S: 511 INVALID folder type
C: FSTO MESG nouser@agap.at
S: 511 UNKNOWN account name
```

Figure 95

TOC

4.25. Command FTAG - authenticated state

Name: find and tag entries

Arguments: tag_list filter*

Result: 110 200 210 510 511

Result 110 - the client can send the filter.

Result 200 - the find and set of tag(s) was successful for all found UIDs.

Result 210 - the find and set of tag(s) was successful but not for all found UIDs.

Result 510 - unknown/unsupported command.

Result 511 - invalid tag list, wrong filter, or no right to tag.

Description: Search for messages/contacts/events only from the currently selected folder (no subfolders) that correspond to a filter and change their tags. You cannot tag in a folder with the tag READ-ONLY. If there is no match for the filter then it is returned a 200 code.

Note: The filter is delivered after the acceptance of the command (response code 110).

Examples:

```
C: FTAG + SEEN
```

```
S: 110 OK SEND filter definition (end it with an empty line)
C: UID 00000001:00001000 AND TAG NEW
C:
S: 200 OK FTAG completed (10 matches)
```

Figure 96

```
C: FTAG + SEEN
S: 110 OK SEND filter definition (end it with an empty line)
C: UID 00000001:00001000 AND TAG NEW
C:
S: 210 OK FTAG completed (only 8 from 10 matches taged)
```

Figure 97

```
C: FTAG
S: 510 UNKNOWN command
```

Figure 98

```
C: FTAG SEEN
S: 511 INVALID tag list
C: FTAG + SEEN
S: 110 OK SEND filter definition (end it with an empty line)
C: LATER
C:
S: 511 INVALID filter definition
```

Figure 99

TOC

4.26. Command GTAG - authenticated state

Name: get tag

Arguments: UID

Result: 200 510 511

Result 200 - the tags for UID were successful displayed.

Result 510 - unknown/unsupported command.

Result 511 - invalid UID.

Description: Return the tags associated to a message/contact/event.

Examples:

```
C: GTAG UIDx1000
S: .SEEN
```

```
S: .SPAM
S: 200 OK GTAG completed
C: GTAG UIDx1001
S: 200 OK GTAG completed
```

Figure 100

```
C: GTAG
S: 510 UNKNOWN command
```

Figure 101

```
C: GTAG -1
S: 511 INVALID UID
```

Figure 102

TOC

4.27. Command LIST - authenticated and not-selected state

Name: list folders

Arguments: path/filter?

Results: 200 511

Result 200 - the list was successful delivered (even if it is empty).

Result 511 - filter is invalid, the specified path (that has no wildcard) does not exist, or the specified path before last folder name (which has an wildcard) does not exist.

Description: List all folders that correspond to the filter (if it is provided), otherwise all direct children of currently selected folder together with their types. All returned folder names are prefixed with the type of the corresponding folder (as it is used by the MAKE command) followed by a white space and the absolute path to the folder.

Filter's path': It is a relative (does not begins with /) or an absolute (begins with /) path. The slash sign (/) is used to delimit folders in the hierarchy. There can be only a star (*) and must to be located in the name of the last folder. The server can return 511 if it founds '.' or '..' as folder names or '\' in the filter definition.

Examples:

```
C: LIST
S: .MESG YESTERDAY
S: .MESG YEAR-2000
S: 200 OK LIST completed (2 matches)
C: LIST /*
S: .MESG /INBOX
S: .MESG /TRASH
S: .CALE /CALENDAR
S: 200 OK LIST completed (3 matches)
C: LIST YEAR-2010/J*
S: .MESG /WORK/YEAR-2010/JUN
S: .MESG /WORK/YEAR-2010/JUL
```

```
S: 200 OK LIST completed (2 matches)
C: LIST /archive*
S: 200 OK LIST completed (0 matches)
```

Figure 103

```
C: LIST */*
S: 511 ERROR path filter can contain only one * in last folder name
C: LIST /ARCHIVE/2000
S: 511 ERROR The specified folder does not exist
C: LIST /ARCHIVE/2000/Documents *.doc
S: 511 ERROR The folder '/ARCHIVE/2000' does not exist
```

Figure 104

TOC

4.28. Command MAKE - authenticated and not-selected state

Name: make folder

Arguments: type path

Result: 200 510 511

Result 200 - the folder was successfully created.

Result 510 - unknown/unsupported command.

Result 511 - invalid path, unknown/unsupported type or the parent of the new folder does not accept to have subfolders.

Description: Create a folder of a certain type.

Note: A new folder has not any tag, except the tag PUBLIC if its parent has it.

Types: They are case insensitive

- ADDR - it holds contacts;
- CALE - it holds calendar events;
- CONF - it holds user's settings for roaming.
- FILE - it holds normal folders and files;
- FILT - it holds the results of a filter defined by the user (there can be only one filter per folder);
- FOLD - it contains only subfolders;
- JRNL - it holds a journal;
- NOTE - it holds user's notes;
- MESG - it holds messages;
- TASK - it holds tasks;

Note: If it requires parents that does not exist then the server will not create them for the client but it will return a 511 response code.

Examples:

```
C: MAKE MESG /ARCHIVE/2010
S: 200 OK Folder created
```

Figure 105

```
C: MAKE
S: 510 UNKNOWN command
```

Figure 106

```
C: MAKE 1234
S: 511 ERROR Missing folder name
C: MAKE new 1234
S: 511 ERROR Unknown folder type
C: MAKE MESG /INBOX/1234
S: 511 ERROR The parent folder does not accept subfolders.
```

Figure 107

TOC

4.29. Command MOVE - authenticated state

Name: move entry

Arguments: UID_source path_destination_folder

Result: 200 510 511

Result 200 - the move was successful.

Result 510 - unknown/unsupported command.

Result 511 - unknown uid or invalid destination folder.

Description: Move a message/contact/event into another folder (by UID). You cannot move from a folder having the tag NO-MOVE or into a folder with the tag READ-ONLY.

Note: For moving a folder the client must use MOVF.

Examples:

```
C: MOVE UIDx1234 ARCHIVE_FOLDER/TODAY
S: 200 OK MOVE completed
```

Figure 108

```
C: MOVE
S: 510 UNKNOWN command
```

Figure 109

```
C: MOVE UIDx1234 ARCHIVE_FOLDER/TODAY
```

```
S: 511 INVALID UID
C: MOVE MSGx1234 ARCHIVE_FOLDER/1970
S: 511 INVALID Destination
```

Figure 110

TOC

4.30. Command MOVF - authenticated state

Name: move folder

Arguments: path_destination_folder

Result: 200 510 511

Result 200 - the move was successful.

Result 510 - unknown/unsupported command.

Result 511 - invalid destination folder, destination is not an absolute path or destination does not exists.

Description: Move the content of a folder into another folder.

Note: In the destination folder are moved all non-folder entries found in the source. If the user needs to move the content of the source folder in another folder then he must to create first this new folder with the MAKE command and then use the MOVF command. If the user wants to move the folders found in the source then he must to do recursively MAKE, MOVF, and DELF for each subfolder.

Examples:

```
C: SLCT /INBOX
S: 200 Selected /INBOX
C: MOVF /ARCHIVE_FOLDER/TODAY
S: 200 OK MOVF completed
```

Figure 111

```
C: MOVF
S: 510 UNKNOWN command
```

Figure 112

```
C: MOVF MISSING
S: 511 INVALID destination
```

Figure 113

TOC

4.31. Command NAME - authenticated state

Name: rename folder

Arguments: new_name

Results: 200 510 511

Result 200 - the rename was successful.

Result 510 - unknown/unsupported command.

Result 511 - invalid new_name or trying to rename a reserved folder name.

Description: Rename a folder. The currently selected folder remains selected even if the name was changed. A reserved folder cannot be renamed.

Note: The new_name does not hold any path hierarchy.

Examples:

```
C: SLCT /ARCHIVE/2001
S: 200 OK
C: NAME OLD-2001
S: 200 OK NAME completed
```

Figure 114

```
C: NAME
S: 510 UNKNOWN command
```

Figure 115

```
C: SLCT /INBOX
S: 200 OK
C: NAME InBox
S: 511 ERROR The folder cannot be renamed (reserved name)
C: NAME /A/new-folder
S: 511 ERROR The argument must not be a path
```

Figure 116

TOC

4.32. Command NOOP - authenticated state

Name: noop

Arguments: none

Result: 200

Description: It does nothing (eventually announce what changes was done in current folder).

Example:

```
C: NOOP
S: 200 OK NOOP completed
```

Figure 117

TOC

4.33. Command PNFO - authenticated, not-selected and presence state

Name: announce presence

Arguments: HERE|AWAY|FOR number unit_of_time|IDLE number unit_of_time|STATUS text|AT text|USER user

Result: 200 510 511

Result 200 - the presence of the user was updated or requested information returned.

Result 510 - unknown/unsupported command.

Result 511 - unknown/unsupported/missing arguments.

Description: Announce that the logged user is still online, eventually since when is it idle. A QUIT command or disconnection means that the user is no longer online.

If there is present an argument HERE then that means the user is no longer idle. As there is no information for how long this status is valid, then it will remain valid until comes a new command which change it.

If there is present an argument AWAY then that means the user is no longer connected. As there is no information for how long this status is valid, then it will remain valid until comes a new command which change it.

If there is present an argument FOR then that means the user is suppose to be considered present for the given amount of time. The client is expected to send its updated status in this period. If comes no command to change the state in the specified time, then after this period the presence will be reported as unknown. The arguments are as by IDLE.

If there is present an argument IDLE then that means the user is idle and the arguments must be: IDLE number unit_of_time. The number must be a positive number and the unit_of_time must be one of the following: sec, min, hour, day, year. As there is no information for how long this status is valid, then it will remain valid until comes a new command which change it.

If there is present an argument STATUS then the user want to change the text that is associated with its presence online. After STATUS can follow any text. The text ends to the end of line. If there is no text then any previous text is deleted and no text is displayed as status text.

If there is present an argument AT then that user want to change the text that is associated with its present location. After AT can follow any text. The text ends to the end of line. If there is no text then any previous text is deleted and no text is displayed as location text.

If there is present an argument USER then the user wants to obtain information about the status of an other user or himself. If the server does not know how to obtain the information about this user then it returns an UNKNOWN as argument. Otherways can return a list with all set texts. After an HERE and AWAY is present when was this set, for an IDLE is the timestamp corresponding to the starting point of idle period. In answer can be present only one of these three. For an PNFO FOR is returned an HERE for a PNFO USER.

Examples:

```
C: PNFO HERE
S: 200 OK You are online and not idle
C: PNFO AWAY
S: 200 OK You are no more online
C: PNFO FOR 5 min
S: 200 OK You are now online. Expecting an update in 5 minutes.
```

```
C: PNFO IDLE 5 min
S: 200 OK You are idle since 5 minutes
C: PNFO STATUS Today I am doing HomeOffice
S: 200 OK You changed your status text
C: PNFO AT Headquarter, 1010 Vienna, Austria
S: 200 OK You changed your location text
C: PNFO USER user@example.com
S: .IDLE 2011-05-27 18:00:00 +1000
S: .STATUS Today I am doing HomeOffice
S: .AT Headquarter, 1010 Vienna, Austria
S: 200 OK USER found
C: PNFO USER user2@example.com
S: .HERE 2011-05-27 18:00:00 +1000
S: .STATUS Today I am in Austria
S: .AT Headquarter, 1010 Vienna, Austria
S: 200 OK USER found
C: PNFO USER user3@example.com
S: .AWAY 2011-05-27 18:00:00 +1000
S: 200 OK USER found
C: PNFO USER user@domain.com
S: .UNKNOWN
S: 200 OK USER not found
```

Figure 118

```
C: PNFO
S: 510 Presence is not supported
```

Figure 119

```
C: PNFO
S: 511 Missing argument
C: PNFO WRONG
S: 511 UNKNOWN argument WRONG
```

Figure 120

TOC

4.34. Command RETR - authenticated state

Name: retrieve

Arguments for a FILT folder: none

Arguments for other types: UID part?

Results: 200 510 511

Result 200 - the message/contact/event was found or filter content was delivered.

Result 510 - unknown/unsupported command.

Result 511 - invalid UID or part name.

Description: Fetch from server the message/contact/event with the given UID. For a filter folder, it must be called without an UID and it returns the content of the filter. Each line of answer is prefixed with a dot that it is not part of the returned object.

Part: It is a PATH in the response as it is returned by RETR and must point to an end leaf. It contains only tag names separated with /. Example: /MESSAGE/HEADER/subject, /MESSAGE/HEADER/received, /MESSAGE/HTML, /MESSAGE/ATTACHMENT-1/BODY. For an entry in the header with a multivalue are returned each value on its own line. For values which are lists with values delimited by comma or semi-colon it is possible to extract an item using square brackets ([]) for lists delimited by comma and curly brackets ({}) for lists delimited by semi-colons.

Examples:

```
C: RETR UIDx1234
S: .<MESSAGE><HEADER>...</HEADER><TEXT>...</TEXT></MESSAGE>
S: 200 OK RETR completed
C: RETR UIDx1234 /MESSAGE/HEADER/subject
S: .Message's subject
S: 200 OK RETR completed
C: RETR UIDx1234 /MESSAGE/HEADER/received
S: .from s0001.srv.example.com [10.11.12.13] by mx.example.com
S: . (Postfix) with ESMTP id 01234567890 for <user@example.com>;
S: . Thu, 19 Nov 2009 01\:02\:03 +0100 (CET)
S: . by userpc (192.168.192.168) id 20091119010204A;
S: . Thu, 19 Nov 2009 01\:02\:04 +0100 (CET)
S: 200 OK RETR completed
C: RETR
S: .<FILTER>
S: .<FOLDERS><FOLDER>/Spam</FOLDER></FOLDERS>
S: .<RULES></RULES>
S: .</FILTER>
S: 200 OK RETR completed
```

Figure 121

```
C: RETR
S: 510 UNKNOWN command (only FILT folders do not needs arguments)
```

Figure 122

```
C: RETR WrongUID
S: 511 INVALID UID
C: RETR UIDx1234 ABC
S: 511 UNKNOWN part name
C: RETR UIDx1234
S: 511 RETR with UID is not allowed for a FILT folder
```

Figure 123

TOC

4.35. Command SLCT - authenticated and not-selected state

Name: select a folder

Argument: path

Result: 200 510 511

Result 200 - the folder was successfully selected.

Result 510 - unknown/unsupported command.

Result 511 - unknown path or '/'.

Description: Select a folder. If the selection was not successful then no folder remains selected and the server switch in the 'Not-selected State'. The user cannot select the root (/) folder.

Examples:

```
C: SLCT /INBOX
S: 200 OK Folder selected
C: SLCT ARCHIVE/2000
S: 200 OK Folder selected
```

Figure 124

```
C: SLCT
S: 510 UNKNOWN command
```

Figure 125

```
C: SLCT 1234
S: 511 INVALID folder
C: SLCT /
S: 511 You cannot select /
```

Figure 126

TOC

4.36. Command SPAM - authenticated state (MESG folder type)

Name: spam

Argument: UID reason?

Result: 200 510 511

Result 200 - the entry was successfully marked.

Result 510 - unknown/unsupported command.

Result 511 - invalid or not found UID, or invalid reason.

Description: Mark a message as SPAM/MALWARE. The server can ignore this command or it can check the reason (if present) and improve its SPAM detection mechanism.

Reason: It can be the name of a field (ex.: header/subject) eventually followed by equal and the text from that field that identify it as a SPAM. If there is more than one criterion for marking a message as SPAM then the client must supply for each reason a SPAM command. If there is no reason then the whole indicated field's content is used.

Note: It cannot be undone and no tag is set for this message.

Examples:

```
C: SPAM UIDx1234
S: 200 OK Message added to the spam database
C: SPAM UIDx1234 header/subject
S: 200 OK Message added to the spam database
C: SPAM UIDx1234 body=V1AGRA
S: 200 OK Message added to the spam database
```

Figure 127

```
C: SPAM
S: 510 UNKNOWN command
```

Figure 128

```
C: SPAM 1234
S: 511 INVALID UID
```

Figure 129

TOC

4.37. Command STAG - authenticated state

Name: set tags of entries

Arguments: UID tag_list

Result: 200 510 511

Result 200 - the tags for UID were successful set.

Result 510 - unknown/unsupported command.

Result 511 - invalid UID.

Description: Set or delete tags associated to a message/contact/event.

Examples:

```
C: STAG UIDx1000 + SEEN
S: 200 OK STAG completed
```

Figure 130

```
C: STAG
S: 510 UNKNOWN command
```

Figure 131

```
C: STAG -1
S: 511 INVALID UID
C: STAG -1 + SEEN
S: 511 INVALID UID
C: STAG UIDx1234 SEEN
S: 511 INVALID tag list
```

Figure 132

TOC

4.38. Command STAT - authenticated state

Name: status

Arguments: none

Result: 200 512

Result 200 - the status of the folder was successfully delivered.

Result 512 - no folder is selected.

Description: Return the absolute path of the currently selected folder (PATH), its type (TYPE), its FCID, its UCID (only for folders with can hold items), the tags (TAGS) and eventually additional information associated with this type of folder. A change in a folder means that its structure was changed or there was a change of its items (like new messages, an event was canceled and automatically removed from a calender).

Additional information:

- ADDR - TOTAL;
- CALE - TOTAL;
- CONF - TOTAL.
- FILE - TOTAL;
- FILT - TOTAL;
- FOLD - none;
- JRNL - TOTAL;
- MESG - TOTAL and NEW;
- NOTE - TOTAL;
- TASK - TOTAL;

Examples:

```
C: STAT
S: .PATH /INBOX
S: .TYPE MESG
S: .FCID 1
S: .UCID 1
S: .TAGS RESERVED
S: .TOTAL 10
S: .NEW 2
S: 200 OK Folder status displayed
```

Figure 133

```
C: STAT
S: 512 ERROR First select a folder
```

Figure 134

TOC

4.39. Command STOR - authenticated and storing state

Name: store

Arguments: none

Result: 110 200 410 511

Result 110 - the client can send the item.

Result 200 - the item was successfully stored or there was no content sent by the client.

Result 410 - if the item cannot be stored.

Result 511 - if the data is not a valid XML or its schema does not correspond to the type of the destination folder.

Description: Store a new message/contact/event/filter into a folder. If it is written a new filter into a FILT folder, then the previous filter is deleted. If the new filter has an invalid XML structure or cannot be saved then the folder remains with the old filter (if any). The server can send a 410 or 511 respons before the empty line is send, so the client must check after each sended line of content if the server had rejected the content.

Note: Do not send a message content using CDATA as it can hold empty lines and an empty line means for the server the end of the message to be stored.

Examples:

```
C: STOR
S: 110 Send the message ended with an empty line
C: <MESSAGE><HEADER>...</HEADER><TEXT>...</TEXT></MESSAGE>
C:
S: 200 OK Message stored (UID is UIDx1234)
C: STOR
S: 110 Send the message ended with an empty line
C:
S: 200 OK Message not stored as it was empty
```

Figure 135

```
C: STOR
S: 110 Send the message ended with an empty line
C: <MESSAGE><HEADER>...</HEADER><HTML>...</HTML></MESSAGE>
C:
S: 410 Cannot store it, not enough space
```

Figure 136

```
C: STOR
```

```
S: 110 Send the message ended with an empty line
C: msg
C:
S: 511 Cannot store it, the message has an incorrect format
```

Figure 137

TOC

4.40. Command SUID - authenticated state

Name: last UID returned by STOR

Arguments: none

Results: 200 511

Result 200 - the command was accepted and eventually an UID was returned.

Result 511 - the command is not accepted in the actual state.

Description: This command returns the last UID generated by a STOR command in the currently selected folder since it was last time selected. Selecting an other folder or leaving the actual state makes to forget last generated UID. By selecting a folder, storing an item and then reselecting the same folder makes the UID to be forgot. If there is no UID stored then is returned a 200 without any line holding an UID. A failling STOR also makes no UID to be remembered.

Examples:

```
C: SLCT /Mbox
S: 200 Selected /Mbox
C: SUID
S: 200 OK There was no STOR since last SLCT
C: STOR
...
S: 200 OK STOR completed with UID UIDx1234
C: SUID
S: .UIDx1234
S: 200 OK SUID completed; found UIDx1234
C: SLCT /Mbox
S: 200 Selected /Mbox
C: SUID
S: 200 OK There was no STOR since last SLCT
C: STOR
...
S: 511 ERROR STOR ompleted with error
C: SUID
S: 200 OK There was no successfull STOR since last SLCT
```

Figure 138

```
C: SUID
S: 511 ERROR SUID is not accepted in this state
```

Figure 139

TOC

5. Responses

TOC

5.1. Semantic and Syntax

TOC

The Response-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. These codes are fully defined in the following section.

After the Response-Code, can follow a 0x20 character and then a Reason-Phrase intended to give a short textual description of the returned code. The Response-Code is intended for automatic use. The Reason-Phrase is intended for humane persons that debug the connection.

The first digit of the Response-Code defines the class of response. The last two digits do not have any categorization role. There are 4 values for the first digit:

- 1xx: Informational - Server waits for request continuation or send unrequested data;
- 2xx: Success - The action was successfully executed;
- 4xx: Server Error - The server failed to perform the request, retry later;
- 5xx: Server Error - The server failed to perform the request, permanent error;

There are commands that return a multi-line response. These are: CAPA, FIND, GTAG, LIST, RETR, and STAT. In this cases, the response code is at the beginning of the last line of the response. All other lines start with a dot (.).

5.2. 1xx Informational

TOC

5.2.1. 100 Reserved

TOC

Reserved.

5.2.2. 110 Continue

TOC

The client SHOULD continue sending the rest of this request. This response informs the client that the server accepted the initial part of the request and it is waiting for the next part of the request. The server sends a final response after the request has been completely received and processed.

5.3. 2xx Success

TOC

5.3.1. 200 OK

TOC

The request was successfully processed.

5.3.2. 210 Partial OK

TOC

The request was successfully applied for at least one item but not for all requested items. (see FCPY, FDEL, FMOV, and FTAG)

5.4. 4xx Temporary Server Error

TOC

5.4.1. 400 Reserved

TOC

Reserved.

5.4.2. 401 Internal Error

TOC

The request could not be processed because it was an internal error (ex.: something is wrong configured).

5.4.3. 410 Retry later

TOC

The operation must to be retried later. This return code is used when the data cannot be stored because there was an error (ex.: not enough space on disk).

5.5. 5xx Permanent Server Error

TOC

5.5.1. 500 Reserved

TOC

Reserved.

5.5.2. 510 Unknown Command

TOC

The request could not be processed because this command is unknown or its syntax is wrong.

5.5.3. 511 Invalid Parameter

TOC

The request could not be processed because the command has an invalid parameter.

This answer can be returned even if there was more than one 0x20 character between command and its arguments or between arguments.

5.5.4. 512 Out of order

TOC

This command has a valid syntax but must to be send after other command required by the logic of the server. (Ex.: PASS after USER in Pre-authenticated State.)

5.5.5. 521 Not found

TOC

This command has a valid syntax but the searched argument does not exist or cannot be accessed. (Ex.: LIST with a path does not have any wildcard and it does not exist.)

TOC

5.5.6. 531 Banned

The client is not allowed to interact with the server. (Ex.: the client's IP is blacklisted.)

TOC

6. All Possible Response Codes for All Commands

TOC

6.1. Not-authenticated State

The Welcome Message: 200 401 410 531

QUIT: 200

AUTH: 510 511

AUTH mechanism: 200 511

CAPA: 200

SGZP: 200 510

STLS: 200 510

other: 510

TOC

6.2. Pre-authenticating State (PLAIN method)

QUIT: 200

PASS: 510 511 512

PASS password: 200 511 512

USER: 510 511

USER account: 200 511

other: 510

TOC

6.3. Pre-authenticating State (MD5 and SHA1 methods)

QUIT: 200

HASH: 510 511 512

HASH hashcode: 200 511 512

USER: 510 511

USER account: 200 511

other: 510

6.4. Authenticated State

QUIT: 200

CHNG: 200 510

COPY: 510 511

COPY arguments: 200 511

CPYF: 510 511

CPYF arguments: 200 511

DATT: 510 511

DATT arguments: 200 511 521

DELE: 510 511

DELE arguments: 200 511

DELF: 510 511

DELF arguments: 200 511

EXIT: 200

FCPY: 110

FCPY arguments: 200 210 511

FDEL: 110

FDEL arguments: 200 210 511

FIND: 110

FIND arguments: 200 511

FMOV: 110

FMOV arguments: 200 210 511

FSTO: 510

FSTO arguments: 110 200 410 510 511

FTAG: 510

FTAG arguments: 110 200 210 511

GTAG: 510

GTAG arguments: 200 511

LIST: 200

LIST arguments: 200 511

MAKE: 510 511

MAKE arguments: 200 511

MOVE: 510 511

MOVE arguments: 200 511

MOVF: 510 511

MOVF arguments: 200 511

NAME: 510 511

NAME arguments: 200 511

NOOP: 200

PNFO: 510 511

PNFO arguments: 200 511

RETR: 510 511

RETR arguments: 199 200 511

SLCT: 510 511

SLCT arguments: 200 511

SPAM: 510 511

SPAM arguments: 200 511 512

STAG: 510 511

STAG arguments: 200 511

STAT: 200 512

STOR: 110 200 410 511

SUID: 200 511

other: 510

6.5. Not-selected State

TOC

LIST: 200

LIST arguments: 200 511

SLCT: 510 511

SLCT arguments: 200 511

other: 510

---

7. Example of Conversations

TOC

7.1. Successful connection and authentication

TOC

```
S: 200 Welcome
C: AUTH PLAIN
S: 200 OK Send USER
C: USER account
S: 200 OK Send PASS
C: PASS password
S: 200 OK Authenticated
C: STAT
```

```
S: .PATH /INBOX
S: .TYPE MESG
S: .TAGS RESERVED
S: .TOTAL 10
S: .NEW 2
S: 200 OK Folder status displayed
```

Figure 140

```
S: 200 Welcome
C: AUTH MD5
S: .Use this as prefix, please!
S: 200 OK Send USER
C: USER account
S: 200 OK Send HASH
C: HASH 79054025255fb1a26e4bc422aef54eb1
S: 200 OK Authenticated
C: STAT
S: .PATH /INBOX
S: .TYPE MESG
S: .TAGS RESERVED
S: .TOTAL 10
S: .NEW 2
S: 200 OK Folder status displayed
```

Figure 141

TOC

7.2. Successful connection but unsuccessful authentication

```
S: 200 Welcome
C: AUTH PLAIN
S: 200 OK Send USER
C: USER account
S: 200 OK Send PASS
C: PASS password
S: 511 WRONG user/password pair
```

Figure 142

TOC

7.3. Connection refused

```
S: 531 Your IP is blacklisted
```

Figure 143

```
S: 410 Too many connections, please retry later
```

Figure 144

```
S: 401 Internal error, the server has an error in its configuration
```

Figure 145

7.4. Find what folders are available with messages

TOC

```
C: LIST /*
S: .MESG /INBOX
S: .MESG /TRASH
S: .CALE /CALENDAR
S: 200 OK LIST completed (3 matches)
```

Figure 146

7.5. Find all entries available in a folder

TOC

```
C: SLCT /INBOX
S: 200 OK Folder selected
C: FIND
S: 110 OK SEND filter definition (end it with an empty line)
C:
S: .UIDx1230
S: .UIDx1231
S: .UIDx1234
S: .UIDx1235
S: .UIDx2340
S: 200 OK FIND completed (5 matches)
```

Figure 147

7.6. Retrieve a message

TOC

```
C: SLCT /INBOX
S: 200 OK Folder selected
```

```
C: FIND
S: 110 OK SEND filter definition (end it with an empty line)
C: TAG NEW IS /HEADER/subject = 'Newsletter from Example.com'
C:
S: .UIDx1234
S: .UIDx1235
S: .UIDx2340
S: 200 OK FIND completed (3 matches)
C: RETR UIDx1234
S: .<MESSAGE><HEADER>
S: .<from>HCCP&lt;news@example.com&gt;</from>
S: .<to>newsletter@localhost.localdomain</to>
S: .<subject>Newsletter from Example.com</subject>
S: .</HEADER>
S: .<TEXT>This is your weekly newsletter.</TEXT>
S: .</MESSAGE>
S: 200 OK RETR completed
```

Figure 148

7.7. Retrieve a contact

TOC

```
C: SLCT /CONTACT
S: 200 OK Folder selected
C: FIND
S: 110 OK SEND filter definition (end it with an empty line)
C: OR REGEX /VCARD/FN = 'RADU.*?'
C:    REGEX /VCARD/FN = '.*? Iulian'
C:
S: .CONx0001
S: 200 OK FIND completed (1 match)
C: RETR CONx0001
S: .<VCARD>
S: .   <VERSION>3.0</VERSION>
S: .   <FN>Iulian Radu</FN>
S: .   <N>Radu;Iulian;;Dipl.Ing.;</N>
S: .   <ORG>Example Com;European Division</ORG>
S: .   <EMAIL TYPE="internet,home">iulian.radu@gmx.at</EMAIL>
S: .   <TZ>+01:00</TZ>
S: .   <REV>2011-05-31T18:46:00Z</REV>
S: .</VCARD>
S: 200 OK RETR completed
```

Figure 149

7.8. Retrieve an event

TOC

```
C: SLCT /CALENDAR
S: 200 OK Folder selected
C: FIND
S: 110 OK SEND filter definition (end it with an empty line)
C: IS /VEVENT/SUMMARY = '*RFC*'
```

```
C:
S: .EVNx0001
S: 200 OK FIND completed (1 match)
C: RETR EVNx0001
S: .<VEVENT>
S: .    <UID>20110531T114600Z-123456@agap.at</UID>
S: .    <DTSTAMP>2011-05-31T12:10:00Z</DTSTAMP>
S: .    <DTSTART>2011-06-07T18:00:00Z</DTSTART>
S: .    <DTEND>2011-06-07T24:00:00Z</DTEND>
S: .    <SUMMARY>AGAP RFC Party</SUMMARY>
S: .    <DESCRIPTION>Celebration of a new revision!
S: .0.4</DESCRIPTION>
S: .    <VALARM>
S: .       20110607T170000Z-20110531T114600Z-123456@agap.at</VALARM>
S: .</VEVENT>
S: 200 OK RETR completed
```

Figure 150

7.9. Store a message

TOC

```
C: SLCT /OUTBOX
S: 200 OK Folder selected
C: STOR
S: 110 Send the message ended with an empty line
C: <MESSAGE><HEADER>
C: <from>HCCP&lt;news@example.com&gt;</from>
C: <to>newsletter@localhost.localdomain</to>
C: <subject>HCCP Newsletter</subject>
C: </HEADER>
C: <TEXT>This is your weekly newsletter.</TEXT>
C: </MESSAGE>
C:
S: 200 OK Message stored (UID is UIDx1234)
```

Figure 151

7.10. Store a contact

TOC

```
C: SLCT /CONTACT
S: 200 OK Folder selected
C: STOR
S: 110 Send the contact info ended with an empty line
C: <VCARD>
C:    <VERSION>3.0</VERSION>
C:    <FN>Iulian Radu</FN>
C:    <N>Radu;Iulian;;Dipl.Ing.;</N>
C:    <ORG>Example Com;European Division</ORG>
C:    <EMAIl TYPE="internet,home">iulian.radu@gmx.at</EMAIL>
C:    <TZ>+01:00</TZ>
C:    <REV>2011-05-31T18:46:00Z</REV>
C: </VCARD>
```

```
C:
S: 200 OK Contact stored (UID is UIDx1234)
```

Figure 152

TOC

7.11. Store an event

```
C: SLCT /CALENDAR
S: 200 OK Folder selected
C: STOR
S: 110 Send the contact info ended with an empty line
C: <VEVENT>
C:     <UID>20110531T114600Z-123456@agap.at</UID>
C:     <DTSTAMP>2011-05-31T12:10:00Z</DTSTAMP>
C:     <DTSTART>2011-06-07T18:00:00Z</DTSTART>
C:     <DTEND>2011-06-07T24:00:00Z</DTEND>
C:     <SUMMARY>AGAP RFC Party</SUMMARY>
C:     <DESCRIPTION>Celebration of a new revision!
C: 0.4</DESCRIPTION>
C:     <VALARM>
C:        20110607T170000Z-20110531T114600Z-123456@agap.at</VALARM>
C: </VEVENT>
C:
S: 200 OK Event stored (UID is UIDx1234)
```

Figure 153

TOC

7.12. Mark messages as SPAM an move them in a new folder

```
C: STAG UIDx1000 + SPAM
S: 200 OK STAG completed
C: SPAM UIDx1000 header/subject
S: 200 OK Message added to the spam database
C: MAKE MESG /Archive-SPAM
S: 200 OK Folder created
C: FMOV /Archive-SPAM
S: 110 OK SEND filter definition (end it with an empty line)
C: TAG SPAM
C:
S: 200 OK FMOV completed (19 matches)
```

Figure 154

TOC

7.13. Create a filter folder, find the matching entries of the filter and read its filter definition

```
C: MAKE FILT /New-messages
S: 200 OK Folder created
C: STOR
S: 110 Send the filter content ended with an empty line
C: <FILTER>
C: <FOLDERS><FOLDER>/INBOX</FOLDER></FOLDERS>
C: <RULES>
C: <AND><NOT><TAG>SEEN</TAG></NOT></AND>
C: </RULES>
C: </FILTER>
C:
S: 200 OK Filter stored
C: SLCT /New-messages
S: 200 OK Folder selected
C: FIND
S: .UIDx1234 /INBOX
S: .UIDx1234 /Trash
S: .UIDx1235 /Trash
S: 200 OK FIND completed (3 matches)
C: RETR
S: .<FILTER>
S: .<FOLDERS><FOLDER>/INBOX</FOLDER></FOLDERS>
S: .<RULES><NOT><TAG>SEEN</TAG></NOT></RULES>
S: .</FILTER>
S: 200 OK RETR completed
```

Figure 155

TOC

7.14. Create a folder and rename it

```
C: MAKE MESG /My/NewFolder
S: 200 OK Folder created
C: NOOP
S: 200 NOOP OK
C: SLCT /My/NewFolder
S: 200 OK Selected /My/NewFolder
C: NAME AFolder
S: 200 OK /My/NewFolder --> /My/AFolder
```

Figure 156

TOC

7.15. Find the status for a folder

```
C: LIST /*
S: .MESG /INBOX
S: .MESG /TRASH
S: .CALE /CALENDAR
S: 200 OK LIST completed (3 matches)
C: SLCT /INBOX
S: 200 OK SELECT completed
C: STAT
```

```
S: .PATH /INBOX
S: .TYPE MESG
S: .TAGS RESERVED
S: .TOTAL 10
S: .NEW 5
S: 200 OK Folder status displayed
```

Figure 157

7.16. Set and check the tags of a message

TOC

```
C: STAG UIDx1000 + SEEN
S: 200 OK STAG completed
C: GTAG UIDx1000
S: .SPAM
S: .FLAG=RED
S: .SEEN
S: 200 OK GTAG completed
```

Figure 158

7.17. Find messages that can be SPAM and delete them

TOC

```
C: FTAG + SPAM
S: 110 OK SEND filter definition (end it with an empty line)
C: REGEX header/subject = '[Vv][i1]agra'
C:
S: 200 OK FTAG completed (10 matches)
C: FDEL
S: 110 OK SEND filter definition (end it with an empty line)
C: UID 00000001:00001000 AND TAG SPAM
C:
S: 200 OK FDEL completed (10 matches)
```

Figure 159

7.18. Connect for a short period

TOC

```
C: PNFO FOR 15 min
S: 200 Nice to se you back
C: PNFO AT At Home
S: 200 So you are there
C: PNFO STATUS Today I am doing HomeOffice
S: 200 So kind to share your thoughts with us
C: PNFO USER coworker
```

```
S: .AWAY 2011-05-27 18:00:00 +1000
S: 200 Sorry, your buddy is not here
C: PNFO AWAY
S: 200 Oh, so soon
```

Figure 160

8. References

TOC

8.1. Normative References

TOC

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

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML," RFC 2629, June 1999 (TXT, HTML, XML).

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations," BCP 72, RFC 3552, July 2003 (TXT).

8.2. Informative References

TOC

[ISO.8601.1988] International Organization for Standardization, "Data elements and interchange formats - Information interchange - Representation of dates and times," ISO Standard 8601, June 1988.

[RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-Pehrson, "GZIP file format specification version 4.3," RFC 1952, May 1996 (TXT, PS, PDF).

[RFC2426] Dawson, F. and T. Howes, "vCard MIME Directory Profile," RFC 2426, September 1998 (TXT, HTML, XML).

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol," RFC 2821, April 2001 (TXT).

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646," STD 63, RFC 3629, November 2003 (TXT).

[RFC3921] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," RFC 3921, October 2004 (TXT, HTML, XML).

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings," RFC 4648, October 2006 (TXT).

[RFC5545] Desruisseaux, B., "Internet Calendaring and Scheduling Core Object Specification (iCalendar)," RFC 5545, September 2009 (TXT).

Author's Address

TOC

Iulian Radu (editor)
Email: iulian.radu@gmx.at