Server to Client Notifications and Filtering

Expires: January 2006

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of Section 10 of RFC2026. By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

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

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

Lemonade server to client notifications and filtering provides some extensions to the IMAPv4 Rev1 protocol [RFC3501] for optimization in a mobile setting, aimed at delivering extended functionality for mobile devices with limited resources. These notifications support pushing changes actively to a client, rather than requiring the client to initiate contact to ask for state changes. Filtering allows setting and controlling which email and events are notified to or synchronized with the mobile client.

Conventions used in this document

In examples, "C:" and "S:" indicate lines sent by the client and server respectively.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocol(s) it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for a protocol is said to be "unconditionally compliant" to that protocol; one that satisfies all the MUST level requirements but not all the SHOULD level requirements is said to be "conditionally compliant." When describing the general syntax, some definitions are omitted as they are defined in [RFC3501].
3.2.1. State-Comparison-Based Synchronization...........5
3.2.2. Event-based Synchronization..........................7
3.2.3. Reconnecting in a same session.....................7
4. Server-Side Filtering........................................9
5. The Server to Client Notification and Filtering Design....10
   5.1. Implementing Filters..................................10
      5.1.1. The View Filter................................10
      5.1.2. The Notification Filter................................11
      5.1.3. The Event Filter................................11
   5.2. Connectivity Models...................................11
      5.2.1. In-Response Connectivity...........................12
      5.2.2. In-band Connectivity................................12
      5.2.3. Out-of-band Connectivity............................13
   5.3. Keeping the Client In Sync with the Mobile Repository...13
6. Events........................................................14
   6.1. Message Events Sent During In-band and In-response Mode..14
7. Interactions between the Client and Server....................15
   7.1. Revisions to IMAPv4 Rev1 Behavior.......................17
      7.1.1. Mobile Repository................................17
      7.1.2. The CAPABILITY Command............................17
      7.1.3. Session/Login......................................17
      7.1.4. IDLE.................................................19
      7.1.5. LPROVISION.........................................19
      7.1.6. LSETPREF & LGETPREFS................................20
      7.1.7. LFILTER............................................23
Security Considerations........................................24
References................................................................25
Normative Appendices..................................................26
   A. Event Payload................................................26
      A.1. Event Payload in Clear Text..............................26
      A.2. Out-of-band Channel Event Payload........................26
      A.3. Out-of-band SMS channel binding..........................29
Non-Normative Appendices...........................................29
   B. Use Cases..................................................29
      B.1. State Comparison-Based Sync............................29
      B.2. Event-Based Sync........................................30
   C. Other Issues...............................................31
      C.1. Client event filtering..................................31
Future Work..........................................................31
Version History....................................................31
Acknowledgments.....................................................32
Authors Addresses..................................................32
Intellectual Property Statement...................................34
Full Copyright Statement............................................34

1. Introduction
The Lemonade Server to Client Notifications and filtering extends IMAPv4 Rev1 [RFC3501]. The client devices in this document are assumed to be mobile with limited resources. The Lemonade Server to Client Notifications and filtering can be bound to any transport protocol for inband and outband connectivity.

The notifications inform the client of changes in an end user’s mailbox. This document will define what events and conditions generate notifications, as well as how the server will inform the client of these notifications. In addition, it covers how the client will process these notifications to maintain email synchrony.

The filtering allows selecting what e-mails and events that the client should be notified of.

These are important requirements for mobile e-mail as discussed in [MEMAIL] and [OMA-ME-RD].

2. Definitions

Inband notifications are notifications that can be transported via IMAP / SMTP and its direct Lemonade extensions.

Outband notifications are server to client event exchanges not transported via IMAP / SMTP and its direct Lemonade extensions.

3. Synchronization

3.1. The Poll Model vs. the Push Model

Today, most of the existing email clients implement a polling model, where the end user is notified of changes to an email account only after the email client polls the server for changes. How long it takes a client to learn of a change on the server is thus dependent on how often the client polls for changes. Many clients can poll at high rates so that the client can quickly learn of changes and reflect them on the client display to achieve a quasi-real time synchronization experience for the end user. The periodic poll model is used on conventional email clients. Because the client must continuously poll the server for changes, the bandwidth requirements can be quite high and the connection quality must be good in order to provide a quasi-real time experience to the user. This also generates additional load on the IMAP server. The periodic poll model is illustrated in Figure 1.
Another way to achieve synchronization is for the email server to tell the client when a crucial change to an email occurs, which is the push model. When important events happen to a user’s email account, the server informs the client device about the event, and then the client can respond to that event as necessary. In this case, the client device does not need to periodically poll the mail server, so the push model is particularly effective in the mobile computing environment when the cost of constant polling is high.

Server to Client Notifications and Filtering defines the semantics for pushing events to a client. The push model is seen in Figure 2.

---

### 3.2. Synchronization Techniques

After a client receives a notification that informs it that changes have occurred to a mailbox, it needs to employ a synchronization technique to reflect the server side changes onto the client device and the client device changes onto the server side. There are many techniques for determining what the changes between a server and client are. In this section, two techniques are presented that aim to keep a client device in sync with a given email account, meaning that the set of messages on the client device is the same as that in the given email account.

#### 3.2.1. State-Comparison-Based Synchronization

IMAPv4Rev1 clients use a state-comparison-based synchronization technique to be in sync with an email account. This technique is used when the client device connects to the server and establishes a new session. This technique requires the client to ask the server for information regarding all the folders and all the messages in each folder stored on the server. Client changes must be applied on the server first. The client must then compute the difference between the server state and the client device state, and make all necessary changes so that the client device state matches the server state. An example of the interaction between the client and server in the IMAPv4 Rev1 protocol for performing a state-comparison-based sync follows.
First, the client must retrieve the folders from the server. The client should issue LIST to figure out which folders have to be retrieved. It then uses LSUB to determine which folders are subscribed. For example:

C: A002 LIST "" "*"
S: * LIST (\NoInferiors) "/" "Drafts"
S: * LIST () "/" "Friends"
S: * LIST (\NoInferiors) "/" "INBOX"
S: A002 OK completed
C: A002 LSUB "" "*"
S: * LSUB () "/" "Drafts"
S: * LSUB () "/" "Friends"
S: * LSUB () "/" "INBOX"
S: A002 OK LSUB completed

Note, that the client should not use LIST "" *, as it might cause too much data to be returned.

After applying the changes from the client to the server, the client must compare its folders with the responses of the command above. If it does not have a folder, it must create that folder on the client device. If there is a folder on the device that is not in any of these responses, then the client may delete or keep that folder. In order to avoid losing changes performed on the client, the client should apply its changes first. In case when the client has changes to a folder that was deleted on the server, it should ask the user whether the changes should be uploaded to a different folder or be discarded (or be configured to automatically do one of the two).

Next, the client needs to make sure that the emails in each of its folders match the server. It performs a SELECT and then a FETCH command for each folder. A sample of a SELECT and FETCH command for the inbox is as follows:

C: A003 SELECT "INBOX"
S: * 60 EXISTS
S: ... more untagged responses with information about the folder
S: A003 OK SELECT completed
C: A004 FETCH 1:* (FLAGS UID)
S: * 1 FETCH (FLAGS (\Answered) UID 120)
S: * 2 FETCH (FLAGS (\Seen) UID 121)
S: ... flags for messages with message sequence numbers 3-59
S: * 60 FETCH (FLAGS () UID 250)
S: A004 OK FETCH completed

The client must go through the full list of email messages in each folder. It must add an email in this list if it is not already on
the client. It must modify any email in this list on the client device to reflect any changes to the mutable flags of that message using IMAP STORE command. Also, it should remove any emails on the client device not in this list. After performing these operations, the client is in sync with the server.

3.2.2. Event-based Synchronization

Another technique is event-based synchronization. Event-based synchronization is used to keep the client device in sync with the server by communicating from the server to the client that a change has taken place on the server and what the change is and conversely from the client to the server that a change has taken place on the client and what the change is. This method requires that the client has been fully synchronized with the server at some earlier point. In the IMAPv4Rev1 protocol, the client must perform a state-comparison-based sync when it selects a folder, but then it can use event-based synchronization to keep itself in sync after that. Although event-based synchronization cannot totally replace state-comparison-based synchronization, it is a faster alternative for the client to maintain synchrony when the server is capable of change tracking for a client. Of course the client maintains tracks of its changes too.

In event-based synchronization, the server keeps track of what changes (called ‘events’) have occurred to the email account that are not yet reflected on the client device. When the client finishes processing all events since the last time it was in sync with the server, it is again in sync with the server. Event-based synchronization is particularly effective when the server can push events to the client for immediate processing. In this case, there are likely to be only a small number of events the client needs to process at one time.

Also, when a compliant client drops a connection or accidentally disconnects, the server can retain the associated session (to facilitate reconnection, authentication and to guarantee valid UIDs etc) and cache all events during the time the client is disconnected. When the client reconnects and is able to obtain the same session, it does not need to perform a state-comparison-based synchronization all over again, but rather, the server sends the list of pending events to the client. In order to avoid losing changes performed on the client during the time the client is disconnected, the client should apply its changes first.

3.2.3. Reconnecting in a same session

The IMAP protocol is predicated upon the assumption that the client establishes a session that is maintained during the client server
interaction. The IMAP protocol depends on the underlying transport protocol to provide the session. Attempts have been made to lower cost of establishing sessions via schemes like the quick reconnect technique being proposed for IMAP [CONNECT].

If the underlying transport is inherently unstable, such as over a wireless network, the transport protocol may drop the session frequently. Also if the e-mail protocols were to be implemented over session-less protocol such as SMS, or over asynchronous messaging system (e.g. MOM -- Message Oriented Middleware), then the session may not even be maintained by the underlying transport protocol. For this reason a future extension may allow IMAP and SMTP commands to optionally carry a session ID in them so that the server can relate any command to the right session if it exists, or respond with the LOGIN response if the session does not exist. If the session exists, the client can behave as if it never lost the session to the server. This technique is immune to the characteristics of the underlying transport protocol from the perspective of session reliability.

It is possible to include a session id in IMAP commands is to encode them as a prefix of the tags. For a definition of tagged and untagged exchanges, see section 7. In this scheme, when the client logs into the server with the device ID (defined later) appended to the login name, it will establish a session and associate a unique id (SID) with the session. For security reasons, the SID should be a random number generated over a very large range. The SID is sent back to the client (so that it be knowledgeable of it) as part of the response to this type of LOGIN response. The client will then construct IMAP command tags using the SID as a prefix. For any IMAP command, the client may receive an untagged LOGIN response. In this case, the client must assume that the session to the server is severed and must take the appropriate action. So with such a scheme, the client must always assume that the session is still alive unless the server informs it otherwise. The client therefore will behave like a connected client (i.e. logged in within a valid session) until such time as the server returns a LOGIN response. When a session is severed, the client may initiate the disconnected mode synchronization approach (i.e. start a state-comparison-based synchronization), unless if this can be avoided as discussed below.

Loss of session to the server does not necessarily mean the client has to resort to the state comparison based synchronization. It depends on the client and server capabilities. For example, if the server supports UID based operations and is able to return EXPUNGE untagged responses with UIDs instead of message sequence numbers, the client may do event based synchronization as long as the UIDs are still valid for the folder.
4. Server-Side Filtering

Server to Client Notifications and Filtering is meant to support mobile client devices with memory and connectivity constraints. Due to these constraints, an end user may want to specify filters to limit the number of notifications sent. These filters separate the user's messages into different sets that the server should handle differently. All end users have a complete repository, which is the place where a user's messages are stored on the server. The end user may want to receive a subset of these messages on their client device. The messages on the device are split further into two categories, lower priority messages that the user chooses to wait for until he can poll (i.e. pull from) the server and higher priority messages that the user would like to be notified of (i.e. pushed from the server) as soon as possible by the server. All three repositories have the same set of folders.

<table>
<thead>
<tr>
<th>COMPLETE</th>
<th>MOBILE</th>
<th>MOBILE</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPOSITORY</td>
<td>View</td>
<td>POL</td>
</tr>
<tr>
<td>all the emails</td>
<td></td>
<td>REPOSITORY Notification</td>
</tr>
<tr>
<td>in an end user's email account</td>
<td>=</td>
<td>on the mobile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>device</td>
</tr>
</tbody>
</table>

Figure 3: Filters and Repositories

Formally, a repository consists of a set of folders, and each folder has both a name and a set of messages associated with it. While the three repositories all have folders with the same name, there may be different messages in them. The complete repository consists of all folders of an end user and all the associated messages for each of those folders. Messages in the complete repository that pass the view filter make up the poll repository. An end user can specify exactly one view filter per folder per device. In addition, there is a second layer of filtering, called notification filter, and there is exactly one notification filter per folder per device. The mobile push repository is the set of all the messages in the complete repository that pass both the view and the notification filters. Note these repositories are only logical components.

From this point forth, it can be assumed that an event in this document refers to only and all changes to messages in the mobile repositories. When the client connects to the server and polls for messages, it can determine what changes have occurred to messages that passed the view filters. Whenever an event occurs to a message that passes the view and notification filters, that message becomes a candidate to be pushed.
Whenever a change occurs to the complete repository, it is first determined whether this change concerns a message or a folder. If it concerns a folder, it is a folder event and all folder events are push events. If the change concerns a message that passes the view filters, it is a message event. Otherwise, this change does not concern the mobile repository and thus is not considered an event for the purposes of this document. Next, if a message event concerns a message that passed the notification filter and that event passes the event filter, it is a pushed message event. Otherwise, if the message event concerns a message that does not pass the notification filters, it is a polled message event.

5. The Server to Client Notification and Filtering Design

Lemonade Server to Client Notification and Filtering extends IMAP and has the same basic model, where the client connects to the server to open a session to access its email account. A client may fetch the contents of the email account or make changes to it just as in IMAP.

5.1. Implementing Filters

A server should support multiple mobile devices for each email user, and should allow each device to have one unique event filter and a set of view filters and notification filters. A mobile client connects to the server by supplying its LOGIN information. The IMAP LOGIN information is extended by permitting the login name to be appended with a device ID. The device ID is a unique identifier, with respect to the server, for the client device issued by the server. If no device ID is given in the LOGIN information, then a regular IMAP session is initiated instead of the type of session described in this document session. The credentials in the LOGIN information is used to identify and authenticate the user, while the device ID is used to determine the user's profile (a set of filters) for the device as well as determining whether a valid session already exists for the user for the device. Associated with the user and device ID is exactly one view filter and exactly one notification filter for each folder. These filters are saved and thus persist across sessions. Filters can be modified when a session is open.

5.1.1. The View Filter

View filters are used to filter out email messages, which match certain criteria. If an email passes through the view filter, it is stored in the mobile repository. The syntax for defining a view filter includes any combination of most of the search criteria as defined for the SEARCH command of IMAP, in Section 6.4.4 and 7.2.5 of RFC 3501, or a ÂdaysÂ filter. The days filter filters messages
received starting a certain number of days before the current day. The ALL search criteria, when used alone, means that every email event satisfies the criteria. By default, view filters are set to ALL.

Whenever a view filter is modified, the UIDs become invalid for all the folders in the mobile repository and thus the client associated to the user must perform a state-comparison-based sync to keep in sync with the mobile repository.

5.1.2. The Notification Filter

Notification filters are used to form a subset of higher priority or special messages by logically copying messages, from the mobile repository into the mobile push repository, that match certain criteria. The syntax for defining a notification filter is the same as defining a view filter.

Because the view filter defaults to ALL and the notification filter to NONE, the mobile poll repository will mirror the complete repository, but none of the messages are added to the mobile push repository. This implies that the default behavior is equal to the IMAPv4 Rev1 model.

The client does not need to do anything after it resets a notification filter or event filter (i.e. sets all NONE and ALL to default values). The server should then only send out notifications that correspond to the most up-to-date filters.

5.1.3. The Event Filter

The event filter is used to filter out message events concerning messages in the push repository. The syntax for defining an event filter is ALL, NONE, or NEW. An event filter applies for all folders in a push repository.

ALL -- All message events concerning messages of the push repository will be sent to the client, such as if the message becomes seen or deleted.

NONE -- No events should be pushed to the client.

NEW -- Only events that concern new messages arriving to the push repository should be pushed to the client.

5.2. Connectivity Models

There are three connectivity models, depending on the capabilities of the server, the client, and the connection available between them. These models include in-response, in-band, and out-of-band. It is explicitly stated in what situations these three connectivity models can be used.
5.2.1. In-Response Connectivity

The in-response binding scenario is the most basic one and implements the poll model. In this case the client initiates the commands to the server and the server responds to client commands with events. In this case there is no need for a persistent connection between the client and the server. The client opens a connection only when it needs to send commands to the server, and that is the only time it is notified of new events.

![Diagram of In-Response connection]

Cases of in-response connection:
- Server Requires: IMAP
- Client Requires: IMAP + no IDLE

5.2.2. In-band Connectivity

The in-band binding scenario corresponds to a reliable push model. In this case the server pushes events to the client whenever they occur. To do so, it must have a reliable means of communication with the client, and the client should be ready to accept such notifications. In this case, there needs to be a persistent connection between the client and the server so that the server can push an event at any time. The client may optionally issue a request to retrieve more information concerning an event.

![Diagram of In-band Connection]

Cases of in-band connection: Always connected, IDLE
- Server Requires: IMAP + IDLE
- Client Requires: IMAP + IDLE, constant TCP connection
Both connectivity models above (In-response and in-band) involve a maintained data connection with notification exchanged within the IMAP ÂbandÂ (i.e. IMAP exchanges).

5.2.3. Out-of-band Connectivity

This case covers notification outside the IMAP ÂbandÂ:
- In a different connection
- Within the same data connection but outside the IMAP ÂbandÂ

The out-of-band binding scenario corresponds to a push model that may be unreliable. In this case the server pushes events to the client whenever they occur, to the best of its ability. To do so, it should be able to send messages to the client without necessarily the need for a persistent connection. However, the out-of-band channel can possibly lose and reorder messages. In addition, there are no timing guarantees.

Examples of out-band channels include SMS (and GSMSMS), JMS (Java Message Service), WAP Push (and WAPWDP), SIP notification and UDP. As in the in-band scenario, the client may optionally open a IMAP session over an in-band or in-response connection and send a command as a result of receiving an event.

![Figure 6: Out-of-band Connection](image)

Cases of out-of-band connectivity:

[1] A notification service from the server to the client
- Server Requires: A notification generator (defined by notification channel).
- Client Requires: A notification processor (defined by notification channel).

In-band or In-response exchanges are on:
- TCP/IP

5.3. Keeping the Client In Sync with the Mobile Repository

Whenever a client device opens a new session with a server, the client must perform a state-comparison-based sync with the mobile repository. Since the client has no way of directly detecting only
changes to the repository since the last login, it needs to retrieve information about every message in the mobile repository and calculate the changes itself. After that point, the client can use event-based synchronization to keep the device in sync.

The server tracks changes to all folders and returns them to the client for the duration of a session. Until the session is expired, the server must log all events that occur while a client is disconnected. This way, if the client temporarily loses a connection, it does not have to worry about missing any events and needing to perform another state-comparison-based sync. A client does have the option though to prematurely end a session by issuing a LOGOUT command. Additionally, clients can remain inactive for a certain period of time without being logged off the server and without the session expiring.

Events are only returned to the client for the currently selected folder, although they are still tracked for folders that aren’t currently selected. To support event-based synchronization for multiple folders, the client will have to issue a SELECT for each folder of interest to the user and receive the queued up events for that folder.

In other words, this approach supports multi-folder semantics, including separate notification and event filters for each folder, as well as tracking changes to each folder, with the caveat that during event retrieval from the server within a session, the client must SELECT each folder separately to receive the changes for that folder.

6. Events

This section contains the syntax that the server uses to send events to the client.

6.1. Message Events Sent During In-band and In-response Mode

The client can receive the following untagged responses from the server:

   S: * 501 EXISTS
   S: * 1 RECENT
Next, the client retrieves this new message using a FETCH command.
   C: A02 FETCH 501 (ALL BODY[])  
   S: * 501 FETCH ...
   S: A02 OK FETCH completed
[2] The client receives an EXPUNGE event from the server from a message has been permanently removed from a folder.

S: * 25 EXPUNGE

The client deletes this message from the client device, as it has been removed permanently from the folder. The client does not need to send any command back to the server.

[3] The client receives an untagged FETCH event from the server, which can contain just FLAG information if the event is regarding an old message or FLAG plus possibly other information if the event is regarding a new message. This event is received if a message’s flags are changed, or for a new message if the user’s preferences are set to do so.

S: * 101 FETCH (FLAGS (\Seen \Deleted))

The client saves the information contained in this response accurately in the client device.

7. Interactions between the Client and Server

A server must support all IMAPv4Rev1 commands from client devices following the syntax defined in [RFC3501]. Thus, a client may issue any existing IMAP commands to the server, and both the server and client must behave as specified in RFC3501 except for the changes specified in Section 7.1.

Client commands, as well as the server responses to them, are included in this section. The Server to Client Notification and Filtering also defines events to be sent by the server to the client. These events notify the client when there are changes to messages that match an end user’s view filters and notification filters, as well as any changes to a client’s email folders. The syntax defined in this section is an abstract syntax, and payloads may vary according to the communication mechanism used. The normative appendix of this document describes some specific payloads.

The format for presenting commands is defined as follows (SEE RFC3501):

<COMMAND NAME>

(Command Description - contains an explanation of the command)

Formal Syntax: <command syntax described in ABNF [RFC2234]>

Valid States: <states of the IMAP session in which this command can be used>

[Extension to: <states what IMAP command this command should be used in place of>]
Responses: <server responses for this command>

Result: <possible result that comes after the responses. This usually indicates the status of the execution of a particular command. Possible values are:
- OK if the execution was successful
- BAD for unknown commands, or when arguments syntax is incorrect
- NO when argument semantics are incorrect, or when command processing fails
- BYE when internal system or network error happens and processing cannot continue>

Example: <Description of what this example is meant to illustrate>

C: <client issued commands>
S: <server returned results>

This section describes commands where the client initiates contact with the server, like all the commands in the IMAPv4 Rev1 protocol. They are used to perform actions on messages: retrieve, delete, search, etc., as well as set up the filters and notification methods of a mobile client. These commands are sent over a reliable connection as required for IMAP, see [RFC3501, Sec. 2.1] for more details. Client devices can send several commands at one time and, thus, these commands must be tagged. The server can send tagged and untagged responses to the client. Untagged responses contain information requested by a command. Tagged responses give the status of the command execution and its tag identifies the command it corresponds to.

To connect to a server, the client must first follow the procedure for establishing a session. The client starts out in NOT AUTHENTICATED state and issues a LOGIN command with a valid device ID appended to the username. Once this command is accepted, the session is started and the client enters the AUTHENTICATED state.

To establish a regular IMAP session, the client may also login in the usual fashion with their username and password.

The server responds to LPROVISION commands by returning any service specific parameters of the server, such as which out-of-band channels are supported.

Once entered into the session, the client can issue LFILTER, LSETPREF and LGETPREFS as needed. LFILTER is used to set the view filters and notification filters.
7.1. Revisions to IMAPv4 Rev1 Behavior

7.1.1. Mobile Repository

In a session, the client can only access messages in the mobile repository. This affects the messages returned by FETCH, UID FETCH, etc. Message sequence numbers reflect the relative position of messages within the given folders of the mobile repository. When returning information about the email account, only messages in the mobile repository are taken into account.

7.1.2. The CAPABILITY Command

The CAPABILITY command is defined in RFC3501, section 6.1.1. The client sends a CAPABILITY command so it can query the server to find out what commands it supports. In RFC3501, the IMAP server is allowed to specify additional capabilities not included in that specification. A server MUST list what commands it supports.

A server can also enumerate individually the other commands that it supports.

capability_cmd =  tag SP "CAPABILITY"
Valid States:  NOT AUTHENTICATED, AUTHENTICATED, SELECTED, or LOGOUT
Responses:  REQUIRED untagged response: CAPABILITY
Result:  OK - capability completed
        BAD - command unknown or arguments invalid

Example: A server that implements Server to Client Notification and Filtering.

C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=LOGIN IDLE LPROVISION, LSETPREF, LGETPREF, LFILTER
S: a001 OK CAPABILITY completed

7.1.3. Session/Login

An email user’s LOGIN name for a session is its regular username + "#" + its device ID + the email domain. Device IDs might be "P" + the device ID issued by the server (e.g. it may be the client’s digit telephone number. Note the length of the phone number should not be limited to a specific value as it may change from country to country). To initiate a session, the client uses a LOGIN command with this new LOGIN name.

The server will automatically try to resume a previous session for this client. It can check the device ID to see if the session exists
(which will work for connection-based transport such as TCP), or it can rely on the new mechanisms described in section 1.2.3 otherwise the server can inform the client of the state of the server by sending an untagged SESSION response. If that state is SELECTED, the server also tells the client what the selected folder is by sending an untagged FOLDER response. Next, the server sends the client any pending events that have occurred in this folder while the client has been disconnected. Thus, the client can just service these pending events and need not perform a full sync. If these events could not be cached for some reason or the server senses the client may have not received some events, the RESYNC Response is returned, and the client should perform a state-comparison based sync. A SESSIONID Response is returned whenever a session is initiated/resumed.

untagged SESSION Response = "**" SP "SESSION" SP ("AUTHENTICATED" / "SELECTED") untagged SESSIONID Response = "***" SP "SESSION" SP
untagged FOLDER Response = "**" SP "FOLDER" SP folder
untagged RESYNC Response = "**" SP "RESYNC"

When there is no active session - either because this is the very first time client logins, or because the client explicitly sent a LOGOUT command to close a previous session - then the server returns a new session ID response and the tagged response to the LOGIN command, and the client needs to perform state-comparison-sync to synchronize its contents.

Example: First login, the client needs to perform a state-comparison-sync to get in sync.
   C: A01 LOGIN joe#P6505551234 password
   S: * SESSIONID 123456
   S: A01 OK LOGIN completed

Example: A successful login resuming an old session
   C: A02 LOGIN joe#P6505551234@foo.com password
   S: * SESSION AUTHENTICATED
   S: * SESSIONID 123456
   S: A02 OK LOGIN completed

Example: A successful login resuming an old session in SELECTED state with the INBOX selected.
   C: A02 LOGIN joe#P6505551234 password
   S: * SESSION SELECTED
   S: * FOLDER INBOX
   S: * 14 EXISTS
   S: * 49 FETCH (....
   S: * SESSIONID 123456
   S: A02 OK LOGIN completed

Example: A successful login resuming an old session in
SELECTED state with the INBOX selected, but where the server could not cache all the events since the last disconnect.

C: A02 LOGIN Joe#P6505551234 password
S: * SESSION SELECTED
S: * FOLDER INBOX
S: * RESYNC
S: * SESSIONID 123456
S: A02 OK LOGIN completed

7.1.4. IDLE

The server must implement the IDLE command from RFC 2177. When the client issues this command, the server can push changes to a folder to the client. The server may replace the EXISTS/RECENT message with an untagged FETCH command as specified in Section 7.1.6. The client should send this command while in-session to enter in-band mode, where the server will actively push notifications to the client.

7.1.5. LPROVISION

The LPROVISION command is used to allow a device to obtain service specific parameters of the server. This includes what filters have been defined. Also, it will supply a list of all preferences and the values they can be set to. For the special LFILTER preference, there are three things returned, the folders, the filter types, and the names of the lfilters supported. In addition, UDP information may be given when UDP notification is supported, such as the host name and port. A server can return other parameters as long as its syntax is agreed upon with the client.

lprovision_cmd = tag SP "LPROVISION" SP device-id [notif-id]
Valid States: AUTHENTICATED or SELECTED
Responses: REQUIRED untagged responses LPROVISION
Result: OK - provision completed
          NO - can’t provision this device
          BAD - command unknown, invalid argument

untagged LPROVISION LFILTER response = "**" SP "LPROVISION" SP "LFILTER_GET" SP "(" ["DESC"] [SP "CRITERIA"]")"
untagged LPROVISION LFILTER response = "**" SP "LPROVISION" SP "LFILTER_SET" SP "(" filter_criteria supported ")"
untagged LPROVISION LPREF response = "**" SP "LPROVISION" SP "LPPREF" SP prev-name SP "(" pref_val_list ")"
untagged LPROVISION UDP_HOST response = "**" SP "LPROVISION" SP "L_UDP_HOST" SP "(" udp_hostname ")"
untagged LPROVISION UDP_PORT response = "**" SP "LPROVISION" SP "L_UDP_PORT" SP "(" udp_portnum ")"
untagged LPROVISION ENC_KEY response = "*" SP "LPROVISION" SP "L_ENC_KEY" SP "(" encryptionkey ")"

Example: The client issues an LPROVISION command. The server returns that the client may get the description of a filter, cannot create any named lfilters (since the search criteria supported is empty), and also the values of various LPREF’s and the values they can be set to. The server responds by returning the encryption key, modes, and channels supported by it. Note the syntax for returning parameters.

C: A002 LPROVISION
S: * LPROVISION LFILTER_GET (DESC)
S: * LPROVISION LFILTER_SET ()
S: * LPROVISION LPREF L_OUTBAND_CHANNEL (SMS NONE)
S: * LPROVISION LPREF L_INBAND_NEW_FORMAT (NONE)
S: * LPROVISION XPREF L_INBAND_PUSH (ON OFF)
S: * LPROVISION XPREF XFILTER_FOLDER (INBOX)
S: * LPROVISION XPREF XFILTER_TYPE (B)
S: * LPROVISION XPREF XFILTER_NAME (ALL URGENT PROFILE1)
S: * LPROVISION XPREF L_EVENT_FILTER (NEW)
S: * LPROVISION XPREF L_OUTBAND_FORMAT (EMN EXTENDED)
S: * LPROVISION L_NOTIFICATION ADDRESS (address)
S: * LPROVISION L_NOTIFICATION PORT (portnum)
S: * LPROVISION L_ENC_KEY (enc_key)
S: A002 OK LPROVISION completed

Event payloads are discussed in Appendix B.

7.1.6. LSETPREF & LGETPREFS

The LSETPREF command allows a user to define certain configuration parameters, while the LGETPREFS command allows a user to retrieve the configuration values. Any server that implements these commands must respond with LPREF as one of the capabilities in response to a CAPABILITY command. It must also announce the values these parameters can be set to in the LPROVISION command as specified as follows. These parameters affect how out-of-band notifications are sent to the client, as well as the format for sending new event notifications. If the server supports LPREF they are required to support all of the following preferences.

This command also allows the user to set active filters. By default, view filters are set to ALL, while notification filters are set to NOT ALL. This means that the mobile repository includes all the messages in the complete repository, but none are pushed to the client, which is the IMAPv4 Rev1 model. To set a filter, first the folder that the filter in question should be applied to, or "ALL" for all folders should be specified. Next the user specifies "V", "N", or "B" to set either a view filter or a notification filter, or both.
Following this, it must specify the name of the filter for that folder. This filter may have been created by the LFILTER command, or be a system defined filter. Exactly one view filter and one notification filter is associated with each folder for each device. When a new view filter or notification filter is created, it replaces the previous filter for that folder. When a view filter is modified, the client needs to perform a state-comparison-based sync on the client in order for the device to be in sync with the mobile repository. The server always sends only notifications that correspond to the most up-to-date view filters and notification filters. All filters persist across sessions; once set, a filter on a folder applies until the user changes it.

The preferences that can set with this command are as follows and their names start with L to identify them as the parameters introduced in this draft. (They may not apply in some configuration (e.g. no L OUTBAND ADDRESS when using UDP notifications)):

1. **L_OUTBAND_ADDRESS** - the number or email address to send out-of-band notification messages to the client. This must be a valid address according to the out-of-band channel requirements. This will not be returned in the LPROVISION command. This is not applicable to out-of-band UDP notifications.

2. **L_OUTBAND_CHANNEL** - the channel to send out-of-band notifications, either SMS, GSMSMS, JMS, WAP_PUSH, WAPWDP, MMS, SIP, UDP or NONE. When NONE, the server does not send the client any out-of-band notifications. The list of values may be extended with new values when different out-of-band channels are available. The valid values for this preference that the server supports will be given in response to the LPROVISION command.

3. **L_IN-BAND_NEW_FORMAT** - the FETCH parameters to automatically send to the client when there is a new message and there is a valid session, or NONE. If NONE, the server sends the client a traditional EXISTS message when a new message arrives in the folder. Otherwise, in place of the EXISTS message, the server sends an untagged FETCH response with the given information. The valid values for this preference that the server supports will be given in response to the LPROVISION command.

4. **L_INBAND_PUSH** - whether or not the server should automatically IDLE the server when a folder is selected. The valid values for this preference that the server supports will be given in response to the XPROVISION command.

5. **L_OUTBAND_FORMAT** - the format to send the out-of-band notifications, i.e. EMN or EXTENDED.
[6] L_EVENT_FILTER - The event filter for this user. Possible values are ALL or NONE or NEW, depending on the server’s capabilities.

[7] L_LFILTER - Sets a named filter as the active filter for a given folder. The value of this parameter includes the folder, filter type (possibly VIEW, NOTIF, or BOTH depending on server capabilities), and the name of the LFILTER.

lgetpref_cmd = tag SP "LGETPREFS" SP "("  
    l_pref_list ")"
l_pref_list = l_pref [SP ;_l_pref_list]
l_pref = (L_OUTBAND_ADDRESS /  
    L_OUTBAND_CHANNEL / L_INBAND_NEW_FORMAT /  
    L_INBAND_PUSH / L_OUTBAND_FORMAT /  
    L_EVENT_FILTER / L_LFILTER)

Valid States: AUTHENTICATED or SELECTED
Responses: REQUIRED untagged LGETPREFS response with the value of the requested parameter.

untagged LGETPREFS response - "*" LGETPREFS pref-pair
pref-pair = "(" l-pref SP l-pref-val [pref-pair] ")"

Result: OK - command completed
        NO - command failure: can’t alter preference
        BAD - command unknown or arguments invalid

Example: The client wishes to know the current out-of-band notification method it has set up. It sends an LGETPREFS command.
C: A003 LGETPREFS (L_OUTBAND_CHANNEL)  
S: * LGETPREFS (L_OUTBAND_CHANNEL SMS)  
S: A003 OK LGETPREFS completed

Example: The client wishes to know the current active lfilters.
C: A003 LGETPREFS (L_LFILTER)  
S: * LGETPREFS (L_LFILTER (INBOX V ALL)  
    L_LFILTER (INBOX N PROFILE1)  
    L_LFILTER (FOLDER2 B ALL))  
S: A003 OK LGETPREFS completed

lsetpref_cmd = tag SP "LSETPREF" SP (("L_OUTBAND_ADDRESS" SP  
    device_address) /  
    ("L_OUTBAND_CHANNEL" SP  
    ("SMS"/"GSM_SMS"/"JMS"/"WAP_PUSH"/"WAPWDP"/"MMS"/"UDP"/"SIP"/  
    "NONE")) /  
    ("L_INBAND_NEW_FORMAT" SP fetch_criteria) /  
    ("L_INBAND_PUSH" SP ("ON" / "OFF")) /  
    ("L_OUTBAND_FORMAT" SP ("EMN" / "EXTENDED")) /  
    ("L_LFILTER" SP lfilter_value)

lfilter_value = ("mailbox SP ("V"/"N"/"B") SP
lfilter_name)

Valid States: AUTHENTICATED or SELECTED
Responses: No specific responses.
Result: OK - command completed
         NO - command failure: can’t get a preference
         BAD - command unknown or arguments invalid

Example: The client sets up its SMS device address and then selects
that it wants SMS messages sent to the device, and the format of the
SMS it wants. It also sets the view and notification filters for the
inbox to an LFILTER named PROFILE1.
   C: A002 LSETPREF L_OUTBAND_ADDRESS 13335559999
   S: A002 OK LSETPREF completed
   C: A003 LSETPREF L_OUTBAND_CHANNEL SMS
   S: A003 OK XSETPREF completed
   C: A004 LSETPREF L_OUTBAND_FORMAT EXTENDED
   S: A004 OK XSETPREF completed
   C: A005 LSETPREF LFILTER (INBOX B PROFILE1)
   S: A005 OK XSETPREF completed.

Example: The client sets the in-band NEW format to be ALL, meaning it
wants the server to automatically send it all the headers for any new
message.
   C: A002 LSETPREF L_INBAND_NEW_FORMAT ALL
   S: A002 OK LSETPREF L_INBAND_NEW_FORMAT completed

From now on, whenever a new message arrives in a folder during a
valid session, the server will try to send an untagged FETCH response
of the new message with the specified information to the client at
the earliest opportunity. This untagged FETCH response replaces the
untagged EXISTS response that IMAP sends regarding a new message.
   S: * 60 FETCH ...<headers>

7.1.7. LFILTER

The LFILTER command allows users to name a set of IMAP search terms
to be used as a view filters or notification filters, or to get the
description or search terms associated with a named filter. LFILTER
can be sent in a session when the state is AUTHENTICATED or SELECTED.
The first argument to this command is whether to "GET" or "SET" an
lfilter.

To retrieve the value of filter, "GET" is used. Optionally, the
charset can be specified next. After that is the name of the lfilter
to get. After that is a parenthesized list of parameters to retrieve
regarding this filter. "DESC" for a description of the lfilter, and
"CRITERIA" for the combination of IMAPv4 search criteria (defined in
Section 6.4.4. and 7.2.5. of RFC 3501) used to create this filter (including the optional the days filter).

To set a filter, "SET" is used. Optionally, the charset can be specified next. After that is the name of the lfilter that is being created, followed by an informal description of the lfilter, and last a parenthesized list of the IMAPv4 search criteria (and an optional days filter) for this lfilter.

Server to Client Notification and Filtering introduces a filter, the days filter, which allows a user to specify from how many days before today it would like to see emails. To see only today’s email, a 0 should be used for the int.

lfilter_cmd =  tag SP "LFILTER" SP
  ("GET" SP ["CHARSET" SP charset] filter_name SP
   "(" ["DESC"] [SP CRITERIA] ")") / ("SET" SP ["CHARSET" SP charset] filter_name SP
desc SP "(" criteria ")")
criteria = (IMAPv4Rev1_searching_criteria / days_filter)
[SP lfilter_criteria]
days_filter = "DAYSBEFORETODAY" SP int
Valid States: AUTHENTICATED or SELECTED
Responses: untagged responses: ;filterGet_resp
lfilterGet_resp = "*" SP "LFILTER" SP filtername SP "("
  ["DESC" SP desc] [SP] ["CRITERIA" "(" criteria ")") ] ")"
Result: OK - filter created
NO - can't create the filter
BAD - invalid arguments

Example: The client creates an lfilter for all messages from "John" since Jun. 1st, 2003.
C: A001 LFILTER SET FROM_JOHN "EMAILS FROM JOHN SINCE JUN-1-2003" (SINCE 1-Jun-2003 FROM "John")
S: A001 OK LFILTER completed

Example: The client asks for the description and search criteria of PROFILE1.
C: A001 LFILTER GET PROFILE1 (DESC CRITERIA)
S: * LFILTER PROFILE1 (DESC "Today’s messages only" CRITERIA (DAYSBEFORETODAY 0))
S: A001 OK LFILTER completed

Security Considerations

The protocol calls for the same security requirements for an in-response and in-band connectivity mode as IMAP.
For the out-of-band connectivity mode, servers should use encryption methods for notifications if sensitive information is included in the payload of that notification.

When an implementation is proxy-based, this may create new security issues.

On bandwidth limited mobile networks where users pay per data volumes and/or notifications, spam may become an important issue. It can be mitigated with appropriate filters and server-side spam prevention tools. These are of course outside the scope of the current draft.

It is also recommended that clients be explicitly registered with the server through separate channels / application. Exchanges should then be paired.

References


[GSM03.40] GSM 03.40 v7.4.0 Digital cellular telecommunications system (Phase 2+); Technical realization of the Short Message Service (SMS). ETSI 2000


Event Payload in Clear Text

The event payload follows the general format explained in Section 6, and is in clear text. Server to Client Notification and Filtering treats the event as a signal to the client to fetch the information on the server that awaits it.

In-band anything sent from the server is treated as an wake-up signal.

Out-of-band Channel Event Payload

One suggested payload for notifications is that suggested by the OMA, see [OMA-EN]. This notification basically informs the client that some push event has happened on the server, so it must connect to fetch the information.

Server to client notifications and filters treats the event as a client wake up event to fetch the information on the server that awaits it. The client may present other behaviors that exploit additional information provided in the notification. However this is out of scope of these specifications.
Wake-up events consists of the following payload: 

```xml
<emn
mailbox="mailat:john.doe@somewhere.com"
timestamp="date_format_as_specified_in_[OMA-EMN]"/>
```

When the client finally connects, the server has opportunity to send other pending events for this client.

Example: new message arrives on the server and this is notified via out-of-band.

S: pushes SMS with the following text:

```xml
<emn
mailbox="mailat:joe@foo.com"
timestamp="2004-02-20T06:40:00Z"/>
```

C: needs to connect and send any command to get the pending events and act upon them.
C: A00 Login joe password
S: * SESSION SELECTED
S: * FOLDER INBOX
S: * 100 EXITS
S: * 87 EXPUNGE
S: * 90 FETCH (FLAGS \Seen)
S: A00 OK LOGIN completed
C: must now act on the events on the order they are received, meaning, first perform a FETCH to get new message, then expunge message 87 and change flags of message 90.

If EXTENDED notification format is supported by the client, the following notification may be send instead of the wake-up event as:

The notification message is of the form:

```
<tag> <notification seq no> <client-email-account -name> <event> [<uid>, <sender>, <date>, <time>, <subject>, [<body.]]
```

where `<tag>` is `Â_%$L$%_Â`, and `<event>` is one of

- NEW_MESSAGE
- DELETED_MESSAGE
- CHANGED_MESSAGE
- SYNC
- FULL_SYNC
- STATE_COMPARISON_SYNC
- NEW_ENC_KEY
- LOCK_DOWN

Except for the `<tag>`, the notification message is encrypted using the encryption key.

The different tags are:
NEW_MESSAGE: a new message has arrived on the server
DELETED_MESSAGE: a message has been deleted on the server
CHANGED_MESSAGE: a message has changed on the server
SYNC: Initiate an incremental synchronization
FULL_SYNC: Initiate a full synchronization
STATE_COMPARISON_SYNC: Compare state
NEW_ENC_KEY: New encryption key is available to be obtained by
LPROVISION
LOCK_DOWN: Lock the client (in case of lost device).

The latter assumes that the client is able to support client lock to
prevent usage / access to data of lost devices, or in general when
desired by the server administrator.

In the case of new encryption (NEW_ENC_KEY) and to cater for the
unreliable nature of the notification channel, messages encrypted
using old encryption key from a device MUST be accepted by the server
until the server receives a message encrypted using the new key. From
that point onward it MUST only accept the messages encrypted using
the new key.

In the case of SYNC requests (incremental synchronization), the
client sends its messages that are to be sent, describes the delete
or change status operations to do on the server or and sending a NOOP
message to the server and processing the responses. New messages are
fetched using UID FETCH command with the range (lastUID + 1):*. Where
lastUID is that lastUID received so far. This typically happens when
the server determines that the session is valid and the UID VALIDITY
(See [IMAP-DISC]) is the same in client and server.

In the case of FULL_SYNC requests (full synchronization), the client
sends its messages that are to be sent, discards delete or change
status operations to do on the server, discard its local emails (e.g.
in INBOX) and populating the Inbox with messages using the FETCH 1:*
command. It also rebuilds the UID-Sequence map. Full synchronization
also takes care of the new client whose UID_VALIDITY is initially set
to -1. This typically happens when the server determines that the
session is invalid and the UID VALIDITY is different in client and server.

In the case of STATE COMPARISON_SYNC requests (state comparison
synchronization), the client sends its messages that are to be sent,
describes the delete or change status operations to do on the server,
requests for and updates the flag values for each of the messages in
the Inbox folder of the client message store, removes message from
the Client Message Store that are no longer in Server Message Store
and requests for new messages. This typically happens when the server
determines that the session is valid and the UID VALIDITY is
different in client and server.
A.3. Out-of-band SMS channel binding

One method for delivering wake-up notifications is by pushing the notification payload as a binary SMS message. Upon receiving an SMS, a client would then parse the payload, determine if it is an email notification or some other SMS message, and process the message appropriately.

This has the unfortunate side effect of forcing the client to parse every message trying to sense what kind of message it is. The proposed mechanism to fix this is to utilize the binary SMS User Data Header (UDH) to specify a destination port, according to the Application Port Addressing Scheme in [GSM03.40] or alternatively, on CDMA networks, to use the WAP WDP mapping to GSM SMS [WAPWDP].

Although any port number is usable, it might make sense to use port 143 for consistency, which is the IANA IMAP port. Thus, OMA EMN or extended format notifications should be sent to port 143 via GSM SMS or WAP WDP. The client upon receiving the SMS will check the port number, and if the port is the right port, the message will be routed to the appropriate client application for processing.

Because such mechanisms are network specific, a server should determine if a port specific SMS or WAP WDP mapping can be used based on knowledge of the device / network or on strategies that determine if the device reacts to such notifications. However, a client may also declare it / selecting the out-of-band notification channel as GSMSMS or WAPWDP as for any other notification channel.

Non-Normative Appendices

B. Use Cases

In this section some use cases are presented so that it is possible to correctly understand concepts and message flow.

B.1. State Comparison-Based Sync

Each time a client logs into a new session, it must perform a state comparison-based sync. To synchronize with the server, the client needs to fetch all the new messages, and all the flags of the old messages.
The client has \( N \) messages in a given folder with highest UID = \( X \) and is disconnected from the server. It connects to the server and performs the following command:

First, it retrieves all the new messages.

\[
\begin{align*}
C: & \quad A01 \text{ UID FETCH } X+1:* \text{ ALL} \\
S: & \quad * \text{ m FETCH ...} \\
S: & \quad \ldots \text{ <more new messages if they exist> } \\
S: & \quad A01 \text{ OK FETCH completed}
\end{align*}
\]

The client stores all this information on the device and displays it. Next, it wishes to sync up the old messages.

\[
\begin{align*}
C: & \quad A02 \text{ FETCH } 1: m-1 \text{ (UID FLAGS)} \\
S: & \quad * 1 \text{ FETCH (UID 3242 FLAGS (\Seen ...))} \\
S: & \quad \ldots \text{ <info for 2 through n-1> } \\
S: & \quad * n \text{ FETCH (UID 3589 FLAGS (\Seen ...))} \\
S: & \quad A02 \text{ OK FETCH completed}
\end{align*}
\]

### B.2. Event-Based Sync

During a session, the client will receive events in the form of untagged EXISTS, RECENT, EXPUNGE, or FETCH responses. The client must respond to these events. Sometimes, it will receive these events by polling, by issuing a command, such as NOOP. It can also use IDLE so that the server can push events to the client. The example following shows how the client acts during an IDLE command, but it should also take the same actions (minus firing and exiting IDLE mode) when it receives these events through polling.

A client can choose to issue an IDLE command to get events pushed to it, or it can receive events from polling using NOOP or any other IMAP command. First the client issues the IDLE command:

\[
\begin{align*}
C: & \quad A02 \text{ IDLE} \\
S: & \quad + \text{ Ready for argument}
\end{align*}
\]

Now the client can receive any of the three following untagged responses from the server.

When the client receives an EXISTS/RECENT response from the server:

\[
\begin{align*}
S: & \quad * 501 \text{ EXISTS} \\
\end{align*}
\]

First, the client must exit from this IDLE command.

\[
\begin{align*}
C: & \quad \text{DONE} \\
S: & \quad A02 \text{ OK IDLE completed}
\end{align*}
\]

Next, the client retrieves this new message using a FETCH command.

\[
\begin{align*}
C: & \quad A02 \text{ FETCH 501 ALL} \\
S: & \quad * 501 \text{ FETCH ...} \\
S: & \quad A02 \text{ OK FETCH completed}
\end{align*}
\]

The client returns to IDLE mode by issuing another IDLE command.
When the client receives an EXPUNGE response from the server:
   S: * 25 EXPUNGE
The client deletes this message from the client device, as it has been removed permanently from the folder. The client can remain in IDLE mode.

When the client receives an untagged FETCH response from the server, either signal a flag change to an old message or a new message:
   S: * 101 FETCH (FLAGS (\Seen \Deleted))
The client updates the information on the device for this message appropriately.

C. Other Issues

C.1. Client event filtering

It is recommended that a client allows the user to select what client-side events are to be propagated to the server (e.g. are messages read or deleted on the client to be read or deleted on the server).

This is out-of-scope of these specifications.

A client may keep track of such changes and:
   - not transmit them to the server
   - selectively present to the user status changes later received from the server (e.g. not re-display a message locally deleted).

This is considered as client implementation specific behavior, out of scope but recommended.

Future Work

[1] Have an N most recent messages filter.
[2] Investigate adding a client to server command to ask the server to stop pushing notifications.
[3] Investigate the use of Server to Client Notifications and Filtering to trigger / notify other applications.

Version History

Release 00
Detail updates of the text throughout the document following lessons learned so far in P-IMAP 07 [P-IMAP].

When released as draft-smaes-lemonade-server-to-client-notifications-00 Initial release published in June 2004 as draft-ietf-lemonade-notification-00 then republished as draft-maes-lemonade-notification.

Acknowledgments

The authors want to thank all who have contributed key insight and extensively reviewed and discussed the concepts of Client to server Notifications and Filtering and its early introduction in P-IMAP [P-IMAP].

Authors Addresses

Stephane H. Maes
Oracle Corporation
500 Oracle Parkway
M/S 4op634
Redwood Shores, CA 94065
USA
Phone: +1-650-607-6296
Email: stephane.maes@oracle.com

Rafiul Ahad
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Eugene Chiu
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Ray Cromwell
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Jia-der Day
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Vida Ha
Wook-Hyun Jeong
Samsung Electronics, CO., LTD
416, Maetan-3dong, Yeongtong-gu,
Suwon-city, Gyeonggi-do,
Korea 442-600
Tel: +82-31-279-8289
E-mail: wh75.jeong@samsung.com

Chang Kuang
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Rodrigo Lima
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA

Gustaf Rosell
Sony Ericsson
P.O. Box 64
SE-164 94 Kista,
Sweden
Tel: +46 8 508 780 00

Jean Sini
Symbol Technologies
6480 Via Del Oro
San Jose, CA 95119
USA

Sung-Mu Son
LG Electronics
Mobile Communication Technology Research Lab.
Tel: +82-31-450-1910
E-Mail: sungmus@lge.com

Fan Xiaohui
Product Development Division
R&D CENTER
CHINA MOBILE COMMUNICATIONS CORPORATION (CMCC)
ADD: 53A, Xibianmennei Ave., Xuanwu District,
Beijing, 100053
China
TEL: +86 10 6600 6688 EXT 3137

Zhao Lijun
CMCC R&D
ADD: 53A, Xibianmennei Ave., Xuanwu District,
Beijing, 100053
China
TEL: 8610.66006688.3041

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF’s procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights, which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

Full Copyright Statement

Copyright (C) The Internet Society (2005). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE
INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED
WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

This document and translations of it may be copied and furnished to
others, and derivative works that comment on or otherwise explain it
or assist in its implementation may be prepared, copied, published
and distributed, in whole or in part, without restriction of any
kind, provided that the above copyright notice and this paragraph are
included on all such copies and derivative works. However, this
document itself may not be modified in any way, such as by removing
the copyright notice or references to the Internet Society or other
Internet organizations, except as needed for the purpose of
developing Internet standards in which case the procedures for
copyrights defined in the Internet Standards process must be
followed, or as required to translate it into languages other than
English.

The limited permissions granted above are perpetual and will not be
revoked by the Internet Society or its successors or assigns.