Push Extensions to the IMAP Protocol (P-IMAP)

Status of this Memo

This document is an Internet-Draft and is subject to all provisions of Section 10 of RFC2026.

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

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

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

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

Abstract

Push Extensions to the IMAP protocol (P-IMAP) defines extensions to the IMAPv4 Rev1 protocol [RFC3501] for optimization in a mobile setting, aimed at delivering extended functionality for mobile devices with limited resources. The first enhancement of P-IMAP is extended support to push crucial changes actively to a client, rather than requiring the client to initiate contact to ask for state changes. In addition, P-IMAP contains extensions for email filter management, message delivery, and maintaining up-to-date personal information. Bindings to specific transport are explicitly defined.

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].

Table of Contents

Status of this Memo...............................................
Abstract..........................................................
Conventions used in this document................................
Table of Contents.................................................
1. Introduction...................................................
    1.1. The Poll Model vs. the Push Model....................
    1.2. Synchronization Techniques...........................
        1.2.1. State-Comparison-Based Synchronization........
        1.2.2. Event-based Synchronization....................
    1.3. The Server-Side Filtering in P-IMAP................
    1.4. Extra Functionality in P-IMAP......................
2. Relation with the Lemonade Profile.........................
3. The P-IMAP Design............................................
    3.1. Implementing Filters...................................
        3.1.1. The View Filter...................................
        3.1.2. The Priority/Notification Filter................
        3.1.3. The Syntax to define Priority/Notification Filters
    3.2. Connectivity Models...................................
        3.2.1. In-Response Connectivity........................
        3.2.2. Inband Connectivity.............................
        3.2.3. Outband Connectivity............................
3.3. Keeping the Client In Sync with the Mobile Repository...
4. Events........................................................
    4.1. Message Events Sent During Inband and Inresponse Mode
    4.2. Folder Events.........................................
    4.3. PIM Events.............................................
5. Interactions between the P-IMAP Client and P-IMAP Server...
    5.1. Revisions to IMAPv4 Rev1 Behavior....................
    5.1.1. UID................................................

Maes                    Expires - January 2005                [Page 2]
The Push-IMAP protocol (P-IMAP) is based on IMAPv4 Rev1 [RFC3501], but contains additional enhancements for optimization in a mobile setting. Thus, the client devices in this document are assumed to be mobile with limited resources. P-IMAP takes into account the limited resources of mobile devices, as well as extra functionality desired. This document covers key P-IMAP concepts, defines the syntax and functionality of the server and client, as well as provides examples of interactions within the protocol. P-IMAP can be bound to any transport protocol for inband and outband connectivity. Appendix A provides a normative binding to HTTP.
The organization of this document is as follows. The rest of this section introduces the core enhancements of P-IMAP so the reader can gain an understanding of the concepts that drive this design. Section 2 positions P-IMAP and the Lemonade Pull Model described in [LEMONADEPROFILE]. Section 3 discusses actual design decisions for P-IMAP. Section 4 defines the bindings for expressing events, while Section 5 is the main body of the protocol, which describes the interactions between the P-IMAP server and client. Next are sections concerning the formal syntax, security considerations, and references. Finally, there are normative and non-normative appendices, which provide useful information for those who wish to implement the P-IMAP protocol. The normative appendices, including Appendices A, B, and C cover some extra guidelines needed to support implementation level issues. The non-normative appendices, D and E, provide interesting use cases and examples.

1.1. 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.

```
+------------------------+      Poll      +------------------------+
|                        | <------------- |                        |
| Mail Server            |               | Email Client            |
|                        | ------------> |                        |
+------------------------+   Response    +------------------------+
```

Figure 1: Periodic Poll Model

Another way to achieve synchronization is for the email server to initiate a session with 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. The P-IMAP protocol defines the semantics for pushing events to a client. The push model is seen in Figure 2.

![Figure 2: Push Model](image)

1.2. 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. 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.

1.2.1. State-Comparison-Based Synchronization

IMAPv4Rev1 clients use a state-comparison-based synchronization technique to be in sync with an email account. This technique requires the client to ask the server for information regarding all the folders and all the messages in each folder stored on the server. 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 has to be retrieved. It than uses LSUB to determine which folders are subscribed. For example:

```
C: A002 LIST "" "*
S: * LIST () "INBOX"
S: A003 OK completed
C: A002 LSUB "" "*
S: * LSUB () "INBOX"
```

Maes                    Expires - January 2005                [Page 5]
S: A002 OK LSUB completed

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

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 must delete that folder. In order to avoid loosing 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 mailbox 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 ÂINBOXÄ
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.

1.2.2. Event-based Synchronization

Another technique is event-based synchronization. Event-based synchronization is used to keep the client device in sync with the server. 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.

In event-based synchronization, the server keeps track of what changes have occurred to the email account that are not yet reflected on the client device. Such a change is called an event. 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 P-IMAP client drops a connection or accidentally disconnects the server can retain the session and cache all events during the time the client is disconnected. When the client reconnects it does not need to perform a state-comparison-based synchronization all over again, and the server sends the list of pending events to the client.

1.3. The Server-Side Filtering in P-IMAP

The P-IMAP protocol 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 their emails into different sets that the server should handle differently. All end users have a complete repository, which includes all their email messages that are stored on a server. The end user may want to receive a small subset of these messages on their client device, which are to be included on the mobile device. The messages on the device are split further into two categories, lower priority messages that the user chooses to wait for until it can poll the server and higher priority messages that the user would like to be notified of as soon as possible by the server. All three repositories have the same set of folders.

Maes

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 emails 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 priority or notification filters, and there is exactly one priority filter per folder per device. The push repository is the set of all the messages in the complete repository that pass both the view and the priority filters.

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 priority filters, the server actively pushes a notification to the client.

Whenever a change occurs to the server, 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 P-IMAP. Next, if a message event concerns a message that passed the notification filters 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 or does not pass the event filter, it is a polled message event.

1.4. Extra Functionality in P-IMAP

The P-IMAP server supports a rich set of extra functionality over the IMAP server to support extra features for a mobile client, and these features are presented:

[1] Compression - The P-IMAP protocol allows for compression of responses to a command. Preliminary testing results show significant performance results when the response to FETCH FLAGS or header information are compressed.

[2] Sending emails - The P-IMAP server can be used to send email, thus eliminating the need for the P-IMAP client to connect to a separate SMTP server.
[3] Support for unstable mobile connections - After a client drops a connection, the P-IMAP server can temporarily maintain the session for the mobile client. During this time, the server caches any events concerning the mobile repository while the client is disconnected, which it can then send to the client upon reconnection.

[4] Longer periods of inactivity tolerated - A P-IMAP server should wait at least 24 hours before logging out an inactive mobile client and ending its session.

[5] Attachments forward/reply behavior - When forwarding/replying to a message from the P-IMAP client, the end user may choose to reattach the original’s message attachments by just specifying the UID of the original message and specifiers for the required bodyparts. The client need not download the attachments of the original message itself. This is an expected server behavior.

[6] Attachments conversion - The P-IMAP server can convert attachments to other formats to be viewed on a mobile device. This is an expected server behavior.

[7] PIM - The protocol also provides support for updating personal information on a client device, even when these changes are initiated from another client (i.e. a personal assistant connects to an end user’s account from a desktop and changes contact information.) These additional uses are especially useful for mobile devices, where end users need up-to-date information on the fly.

2. Relation with the Lemonade Profile

P-IMAP optimizes IMAP for mobile clients. It governs exchanges between mobile clients and servers.

The Lemonade Profile [LEMONADEPROFILE] specifies:
- The Lemonade Pull Model that governs the exchanges among mail servers or between desktop mail client and mail servers
- Mobile optimizations

P-IMAP should be seen as mobile profile for Lemonade that addresses the issues mobile optimization.

This document assumes that clients MUST be compliant to P-IMAP. The Lemonade server MUST be compliant to the P-IMAP for its exchanges with the mobile client.

The Lemonade server MAY follow the Lemonade Pull Model described in [LEMONADEPROFILE].
3. The P-IMAP Design

P-IMAP 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 P-IMAP client may fetch the contents of the email account or make changes to it just as in IMAP. P-IMAP does, however, have many enhancements to IMAP, and this section introduces the core design changes. There are many requirements given in this section, as well as concepts that are essential to understanding the protocol.

3.1. Implementing Filters

A P-IMAP 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 priority/notification filters. The server only needs to support one connection per mobile device for each email user. A mobile client connects to the P-IMAP server by supplying its LOGIN information, and then must inform the server of this mobile client's device ID, which is some unique identifier for the client device. The server and client should agree on what convention to use for this ID, and it could be a hash of IMEI. If no device ID is given, then a regular IMAP session is initiated instead of a P-IMAP session. The LOGIN information is used to specify a user, while the device ID is needed to specify the mobile client. Associated with the user and device ID is exactly one view filter and exactly one priority/notification filter for each folder. These filters are saved and thus persist across P-IMAP sessions. Filters can be modified when a P-IMAP session is open.

3.1.1. The View Filter

View filters and priority/notification 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 or notification 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 days 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 client needs to perform a state-comparison-based sync to keep in sync with the mobile repository since the messages in the mobile repository may have changed.
3.1.2. The Priority/Notification Filter

Priority/Notification filters are used to select emails in the mobile repository which match certain criteria. If an email passes through the notification filter, it is stored in the push repository. The syntax for defining a priority/notification filter is discussed below. By default, priority/notification filters are set to NOT ALL to reduce default traffic at the cost of some delays.

Because the view filter defaults to ALL and the priority/notification filter to NOT ALL, the mobile repository will mirror the complete repository, but none of the messages are added to the 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 priority/notification filter or event filter, instead the server should then only send out notifications that correspond to the most up-to-date filters.

3.1.3. The Syntax to define Priority/Notification Filters

The syntax for defining a priority/notification filter is ALL, NONE, or NEW. A priority/notification 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.

This one event filter applies for all folders.

3.2. Connectivity Models

There are three connectivity models for P-IMAP, depending on the capabilities of the P-IMAP server, the client, and the connection available between them. These models include in-response, inband, and outband. It is explicitly stated in what situations these three connectivity models arise.

3.2.1. 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 P-IMAP 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 P-IMAP server, and that is the only
time it is notified of new events.

![Diagram of In-Response connection]

An in-response connection occurs in two situations:

1. HTTP/HTTPS binding
   - Server Requires: HTTP/HTTPS listener for IMAPv4
   - Client Requires: HTTP/HTTPS client with IMAPv4 processing

2. TCP Binding
   - Server Requires: IMAPv4
   - Client Requires: IMAPv4 + no IDLE

3.2.2. Inband Connectivity

The inband 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 Inband connection]

An inband connection occurs in the following situations:

1. TCP Binding, Always connected, IDLE
   - Server Requires: IMAPv4 + IDLE
   - Client Requires: IMAPv4 + IDLE, constant TCP connection

2. Any other persistent two-way connection
   - Server Requires: IMAPv4 + IDLE
   - Client Requires: IMAPv4 + IDLE, constant connection

3.2.3. Outband Connectivity
The outband binding scenario corresponds to an unreliable push model. 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 the need for a persistent connection. However, the outband channel can possibly lose and reorder messages, and there are no timing guarantees. Examples of out-band channels include SMS, JMS, WAP Push, and UDP. As in the inband scenario, the client may optionally open a P-IMAP session over an inband or in-response connection and send a command as a result of receiving an event.

Outband connectivity occurs in the following situations:

- A notification service from the server to the client
  - Server Requires: A notification generator.
  - Client Requires: A notification processor.

3.3. Keeping the Client In Sync with the Mobile Repository

Whenever a client device opens a new P-IMAP session, it must perform a state-comparison-based sync with the email server so that its state is the same as 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 P-IMAP server can issue a session and track changes to a selected folder for the duration of a session. Until the session is expired, the server must log all events that occur while a client is offline. 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, P-IMAP clients can remain inactive for at least twenty four hours without being logged off the server and without the session expiring.
4. Events

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

4.1. Message Events Sent During Inband and Inresponse 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 possibly other information if the event is regarding a new message. This event is received if a message’s flags are changed, or in response to 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.

4.2. Folder Events

This section will contain syntax for indicating folder events.

4.3. PIM Events

This section will contain syntax for indicating PIM events.
5. Interactions between the P-IMAP Client and P-IMAP Server

A P-IMAP server must support all IMAPv4Rev1 commands from client devices following the syntax defined in [RFC3501]. Thus, a P-IMAP client may issue any existing IMAP commands to the P-IMAP server, and both the server and client must behave as specified in RFC3501 except for the changes specified in Section 5.1. In addition, P-IMAP defines extension commands for IMAPv4 Rev1 using the Experimental/Expansion mechanism defined in [RFC3501, Sec 6.5] and, as per RFC definition, P-IMAP command names must start with X. P-IMAP commands are tagged and asynchronous following the same rules as in IMAPv4 Rev1.

Client commands, as well as the server responses to them, are included in this section. The P-IMAP protocol 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:

<COMMAND NAME>

<Command Description - contains an explanation of the command>

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

Valid States: <states of the P-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. These commands include extensions to the IMAP protocol that have been created in order to better support mobile devices, and these extensions are all prefixed with X. 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 P-IMAP server, the client must first follow the procedure for establishing an IMAP session. The client starts out in NOT AUTHENTICATED state and issues a LOGIN command with a valid P-IMAP device ID appended to the username. Firing this command enters the client into a P-IMAP session, where it can use all the P-IMAP extension commands, as opposed to a regular IMAP session, which will return errors to all P-IMAP defined extensions other than XZIP, XDELIVER, and XPROVISION. To establish a regular IMAP session, the client may also login in the usual fashion with their username and password.

The server responds to XPROVISION commands by returning any service specific parameters of the server, such as which outband channels are supported. The XZIP command can be used to zip the response to another command. XDELIVER allows the client to send an email message through this server, instead of having to connect with an SMTP server.

Once entered into the P-IMAP session, the client can issue XFILTER, XCONVERT, XSETPIMAPPREF, XGETPIMAPPREFS, and XPSEARCH as needed. XFILTER is used to set the view filters and notification filters. XCONVert is used for attachments conversion and XPSEARCH is an enhanced version of SEARCH in IMAPv4 Rev1.

5.1. Revisions to IMAPv4 Rev1 Behavior

The section describes all the differences between how an IMAPv4 Rev1 server vs. a P-IMAP server responds to all IMAPv4Rev1 commands for
implementing the custom mobile features. A compliant P-IMAP server must implement all the commands in IMAPv4 Rev1, with these revisions. The IMAPv4Rev1 syntax on commands and responses are found in sections 6 and 7 in [RFC3501]. The rest of this section defines any additional modifications to the IMAP commands that a P-IMAP server must implement to be compliant.

5.1.1. UID

The UID of email messages MUST not change across sessions. Changing the UID of email messages requires a heavy computational burden on the mobile client, so the server should avoid doing so.

5.1.2. Mobile Repository

In a P-IMAP 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, so the message sequence number of an email while logged in to P-IMAP may also differ from IMAP. When returning information about the email account, only messages in the mobile repository are taken into account.

5.1.3. 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 P-IMAP server conforms to that requirement, and must list what P-IMAP commands it supports. Minimally, this must include XZIP, XDELIVER, and either IDLE or outband notification. XZIP capability is also returned independently of the binding.

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 P-IMAP server that implements all P-IMAP commands.

C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=LOGIN IDLE XCONVERT XFILTER XPSEARCH XZIP XDELIVER XPIMAPPREF
S: a001 OK CAPABILITY completed

Example: A minimal P-IMAP server over TCP binding.

C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=LOGIN IDLE XZIP XDELIVER
S: a001 OK CAPABILITY completed

5.1.4. P-IMAP Session/Login

An email user's LOGIN name for a P-IMAP session is its regular username + "#" + its P-IMAP device ID + optionally, the email domain. P-IMAP device IDs might be "P" + the client's 10 digit telephone number. To enter a P-IMAP session, the client uses a LOGIN command with this new LOGIN name.

The P-IMAP server will automatically try to resume a previous session for this client. If this is the case, the server informs 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.

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

When there is no active P-IMAP 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 only 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: A01 OK LOGIN completed

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

Example: A successful P-IMAP login resuming an old session in SELECTED state with the INBOX selected.
C: A02 LOGIN joe#P6505551234 password
S: * SESSION SELECTED
Example: A successful P-IMAP 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: A02 OK LOGIN completed

5.1.5. IDLE

The server should 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 5.2.2. The client should fire this command while in-session to enter inband mode, where the server will actively push notifications to the client.

5.1.6. XENCRYPTED

For certain proxy-based implementation of P-IMAP (see Security Considerations and Appendix C), it may be necessary to have only encrypted responses for retrieving email content. In that case in place of any untagged FETCH response, the P-IMAP server will return an untagged XENCRYPTED response with message content. The server should return XENCRYPTED in response to the CAPABILITY command if it implements this security mechanism and must announce the encryption methods specified (see the example following).

untagged XENCRYPTED Response = "**" SP "XENCRYPTED" SP encrypted_message_data

Server’s response to the CAPABILITY command announcing XENCRYPTED methods.
C: A02 CAPABILITY
S: * CAPABILITY IMAP4rev1 XENCRYPTED=3DES,RC40,AES
S: A02 CAPABILITY completed

5.2. P-IMAP Extension Commands and Responses
The following subsections define P-IMAP extension commands and as per RFC 3501, their names start with X.

5.2.1. XPROVISION

The XPROVISION command is used to allow a device to obtain service specific parameters of the server. This includes what XFILTERS are supported, since a server may not actually be able to support all IMAPv4Rev1 Search criteria. Also, it will supply a list of all P-IMAP preferences and the values they can be set to. A P-IMAP server can return other parameters as long as its syntax is agreed upon with the P-IMAP client.

\[
xprovision\_cmd = \text{tag SP } "XPROVISION" \text{ SP } \text{device-id [notif-id]}
\]

Valid States: AUTHENTICATED or SELECTED

Responses: REQUIRED untagged responses XPROVISION

Result: OK - provision completed
NO - can't provision this device
BAD - command unknown, invalid argument

untagged XPROVISION XFILTER response = "*" SP "XPROVISION" SP "XFILTER" SP "(" filter_criteria_list ")"
untagged XPROVISION XPIMAPPREF response = "*" SP "XPROVISION" SP "XPIMAPPREF" SP prev-name SP "(" pref_val_list ")"

Example: The client issues an XPROVISION command. The server responds by returning the encryption key, modes, and channels supported by P-IMAP. Note the syntax for returning parameters.

C: A002 XPROVISION
S: * XPROVISION XFILTER (AND OR DAYSBEFORETODAY HEADER FROM TO CC)
S: * XPROVISION XPIMAPPREF PIMAP_OUTBAND_CHANNEL (SMS NONE)
S: * XPROVISION XPIMAPPREF PIMAP_INBAND_NEW_FORMAT (NONE)
S: * XPROVISION XPIMAPPREF PIMAP_INBAND_PUSH (ON OFF)
S: A002 OK XPROVISION completed

5.2.2. XSETPIMAPPREF & XGETPIMAPPREFS

The XSETPIMAPPREF command allows a user to define certain configuration parameters, while the XGETPIMAPPREFS command allows a user to retrieve the configuration values. Any server that implements these commands must respond with XPIMAPPREF 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 XPROVISION command as specified as follows. These parameters affect how outband notifications are sent to the client, as well as the format for sending new event notifications. If the server supports XPIMAPPREF they are required to support all of the following preferences with at
least one value to set each preference to. They are listed following
and their names start with PIMAP to identify them as P-IMAP
parameters:

[1] PIMAP_OUTBAND_ADDRESS - the number or email address to send
SMS/JMS notification messages to the client. This must be a valid
number or email according to the outband channel requirements.
This will not be returned in the XPROVISION command.

[2] PIMAP_OUTBAND_CHANNEL - the channel to send outband
notifications, either SMS, JMS, WAP_PUSH, MMS, or NONE. When NONE,
the P-IMAP server does not send the client any outband
notifications. The list of values may be extended when different
outband channels are available. The valid values for this
preference that the server supports will be given in response to
the XPROVISION command.

[3] PIMAP_INBAND_NEW_FORMAT - the FETCH parameters to automatically
send to the client when there is a new message and there is a valid
P-IMAP 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 XPROVISION command.

[4] PIMAP_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.

C: A003 XGETPIMAPPREFS (PIMAP_OUTBAND_CHANNEL)
S: * XGETPIMAPPREFS (PIMAP_OUTBAND_CHANNEL SMS)
S: A003 0K XGETPIMAPPREFS completed

xsetpimappref_cmd = _tag_sp "XSETPIMAPPREF"_ sp
  ("PIMAP_OUTBAND_ADDRESS" SP device_address) /
  ("PIMAP_OUTBAND_CHANNEL" SP ("SMS"/"JMS"/"WAP_PUSH"/
    "MMS"/"NONE")) /
  ("PIMAP_INBAND_NEW_FORMAT" SP fetch_criteria) /
  ("PIMAP_INBAND_PUSH" SP ("ON" / "OFF"))

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.
  C: A002 XSETPIMAPPREF PIMAP_OUTBAND_ADDRESS 13335559999
  S: A002 0K XSETPIMAPPREF completed
  C: A003 XSETPIMAPPREF PIMAP_OUTBAND_CHANNEL SMS
  S: A003 0K XSETPIMAPPREF completed

Example: The client sets the inband NEW format to be ALL, meaning it
wants the server to automatically send it all the headers for any new
message.
  C: A002 XSETPIMAPPREF PIMAP_INBAND_NEW_FORMAT ALL
  S: A002 0K XSETPIMAPPREF PIMAP_INBAND_NEW_FORMAT completed

From now on, whenever a new message arrives in a folder during a
valid P-IMAP 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>

5.2.3. XFILTER

The XFILTER command allows users to set up view filters and
priority/notification filters. XFILTER can be fired as long when the
state is AUTHENTICATED or SELECTED. The first argument to this
command is the folder that that filter should be applied to, or "ALL"
for all folders. Next the user specifies "V", "N", or "B" to set
either a view filter or a priority/notification filter, or both.
Following this, it must specify the filter criteria using a
combination of search criteria as defined for the SEARCH command of
IMAP, in Section 6.4.4 and 7.2.5 of RFC 3501, or the days filter.
The ALL search criteria, when used alone, means that every email
message satisfies the criteria. Or it can specify "V" or "N" to get a view filter or get a priority/notification filter. In this case, the last argument is "GET" to retrieve the filter.

By default, view filters are set to ALL, while priority/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.

Exactly one view filter and one priority/notification filter is associated with each folder for each device. When a new view filter or priority/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 priority/notification filters. All filters persist across P-IMAP sessions; once set, a filter on a folder applies until the user changes it.

P-IMAP 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.

```plaintext
xfilter_cmd = tag SP "XFILTER" SP ("ALL" / folder) SP
    ( ("V" / "N" / "B") SP xfilter_criteria ) /
    ( ("V" / "N") "GET")
xfilter_criteria = (IMAPv4Rev1_searching_criteria / days_filter)
[SP xfilter_criteria]
days_filter = "DAYSBEFORETODAY" SP int
Valid States: AUTHENTICATED or SELECTED
Responses: untagged responses: xfilterGet_resp
xfilterGet_resp = "*" SP "XFILTER" SP folder SP ("V"/"N")
    xfilter_criteria
Result: OK - filter created
       NO - can’t create the filter
       BAD - invalid arguments
```


C: A001 XFILTER INBOX P SINCE 1-Jun-2003 FROM "John"
S: A001 OK XFILTER completed

Example: The client asks for the view filter for all the folders.

C: A001 XFILTER ALL V GET
S: * XFILTER ~/INBOX V ALL
S: * XFILTER ~/TRASH V NOT ALL
S: A001 OK XFILTER completed
Example: Stop notifications on a particular device, fired while in AUTHENTICATED mode.
   C: A001 XFILTER ALL P NOT ALL
   S: A001 OK XFILTER ALL P NOT ALL completed

5.2.4. XZIP

The XZIP command is used for zipping the response of a command and can be used while the server is in any state. The XZIP command takes in a complete second command (including a tag for that command). In an untagged response to XZIP, the server gives the number of bytes in the zipped response to the second command, as well as the response to that command in g-zip format.

XZIP is optional when HTTP/HTTPS binding is used as discussed in Appendix A, as the P-IMAP server may rely on the HTTP/HTTPS compression mechanism. For the other bindings XZIP is mandatory.

xzip_cmd =  tag SP "XZIP" SP command
Valid States:  NOT AUTHENTICATED, AUTHENTICATED, SELECTED, or LOGOUT
Responses:  "{" num "}" zipped-response-to-command
Result:  OK - the command given was g-zipped correctly and sent
BAD - invalid arguments, i.e. command given is in the wrong format.

Example: Zipping the response to a FETCH command.
   C: A001 XZIP A002 FETCH 1:* ALL
   S: * {1093843723} ...[zipped response to FETCH command]... CRLF
   S: A001 OK XZIP completed
When the client unzips the body of the response to the FETCH command it gets:
   * 1 FETCH ...
   ...
   A002 OK FETCH completed

5.2.5. XDELIVER

The XDELIVER command can be used for creating new messages, or replying to/forwarding an existing message. The first argument after the command name indicates whether this is a new message "N", a reply "R" or a forward "F" of an existing message. When replying/forwarding a message, the client must specify the UID of the message being replied to or forwarded and whether or not to include the attachments of the original message in the reply/forward, by indicating either "Y" or "N" after the UID parameter. The text of the message being replied to/forwarded is automatically appended to the end of the new message regardless. If the user wishes to save a copy of this message to some folder, it can specify that next by using "SAVETO" followed by the name of the folder. If and only if SAVETO is
specified, the server will return an APPENDUID response code with the
UID validity and then the UID of that saved message in that folder.
If the message cannot be saved to the server, an okay response will
still be returned, but without a UID. The last argument of the
XDELIVER command is a number in braces that denotes the number of
bytes in the Internet message (conforming to RFC 2822) that is to
follow. A "+" before the closing braces means the client will send a
CRLF and then the Internet message immediately, without waiting for a
continuance response from the server. The server continues to wait
until it receives the number of bytes specified, and then waits for
an additional CRLF. If more bytes were input before this additional
CRLF than was specified, the server returns an error. Thus, the
client should input exactly the number of bytes specified for the
Internet Address, and then one final CRLF to terminate the XDELIVER.

xdeliver_cmd = tag SP "XDELIVER" SP
   ("N" / ("R"/"F") SP folder SP uid SP ("Y" / "N"))
   [SP "SAVETO=" folder]
   SP "{" number ["+" ] "}"
   internet_msg

Valid States: NOT AUTHENTICATED, AUTHENTICATED, SELECTED, or LOGOUT
Responses: no specific responses
Result: OK - mail delivered successfully by the SMTP server,
        XDELIVERUID response code included if the SAVETO is
        included in the command.
        BAD - invalid arguments, for example missing parameter.
        NO - when the envelope information is invalid

Example: new message
  C: A001 XDELIVER N SAVETO=~~/Sent (299)
  Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
  From: Fred Foobar <foobar@Blurdybloop.COM>
  Subject: afternoon meeting
  To: mooch@owatagu.siam.edu
  Message-Id: <B27397-0100000@Blurdybloop.COM> MIME-Version: 1.0
  Content-Type: TEXT/PLAIN; CHARSET=US-ASCII
                <a blank line>
  Hello Joe, do you think we can meet at 3:30 tomorrow?
  <a blank line>
A new message is prepared and sent.
  S: A001 OK XDELIVER [APPENDUID 1 140] completed

Example: reply message
  C: A001 XDELIVER R Inbox 203 Y {299}
  Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
  From: Fred Foobar <foobar@Blurdybloop.COM>
  Subject: afternoon meeting
  To: mooch@owatagu.siam.edu
Hello Joe, do you think we can meet at 3:30 tomorrow?

A reply message for message 203 is prepared and includes all original attachments.

S: A001 OK XDELIVER completed

5.2.6. XCONVERT & UID XCONVERT

XCONVERT and XUIDCONVERT is used for attachments conversion. In this case, the client sends one message sequence number or UID, a body part number, and gives the mime-type and subtype to convert the attachment to.

xconvert_cmd = tag SP "XCONVERT" message-sequence-number SP part-id SP "as" SP mime-type "/" subtype

Valid States: SELECTED

Responses: untagged responses: XCONVERT

Untagged Xconvert response = "" SP message-sequence-number SP ""XCONVERT" SP document_in_converted_format

Result: OK - xconvert completed

NO - xconvert error: can’t perform the command

BAD - command unknown or arguments invalid

Example: The client fetches an attachment in the message with the message sequence number of 120 in the Inbox and asks to have that attachment converted to pdf format.

C: a001 XCONVERT 120 BODY[3] as application/pdf

S: * 2 XCONVERT <this part of a document in pdf format.>

S: a001 OK XCONVERT COMPLETED

xuidconvert_cmd = tag SP "UID" SP "XCONVERT" uid SP part-id SP "as" SP mime-type "/" subtype

Valid States: SELECTED

Responses: untagged responses: XCONVERT

Result: OK - xuidconvert completed

NO - xuidconvert error: can’t perform the command

BAD - command unknown or arguments invalid

Example: The client fetches an attachment in the message with UID 120 (and message sequence number 2) in the Inbox and asks to have that attachment converted to pdf format.

C: a001 UID XCONVERT 120 BODY[3] as application/pdf

S: * 2 XCONVERT <this part of a document in pdf format.>

S: a001 OK UID XCONVERT COMPLETED
5.2.7. XPSEARCH

The XPSEARCH command and response syntax follows the same rules as the ones defined for the SEARCH command in RFC3501, Sec. 6.4.4 and 7.2.5 respectively. The XPSEARCH command extension allows the search to be made persistent on the server and to appear as a virtual folder. Following the successful execution of an XPSEARCH command, a new folder appears when using the LIST command under the root folder with the specific folder name requested. This new folder needs to be created on the client device. Clients operating on this folder see a view of the underlying folder with only messages matching the search criteria displayed. Operations on messages in this folder do not affect that message.

```
xpsearch_cmd = tag SP "XPSEARCH" [SP "CHARSET" SP astring] 1*(SP search-key)
```

**Valid States:** SELECTED

**Extension to:** UID SEARCH command [RFC 3501, Sec. 6.4.4]

**Responses:** no specific responses

**Result:**
- OK - xpsearch created
- NO - can’t create the folder or incorrect query
- BAD - invalid arguments

Example: create a persistent search for all messages from "John" since Jun, 1st 2003. The newly created folder name is called "from_john"

```
C: A001 XPSEARCH from_john FLAGGED SINCE 1-Jun-2003 FROM "John"
S: A001 OK XPSEARCH completed
```

Security Considerations

The protocol calls for the same security requirements for an in-response and inband connectivity mode as IMAP.

For the outband connectivity mode, servers should use encryption methods for notifications if sensitive information is included in the payload of that notification.

When an implementation of P-IMAP is proxy-based, this may create new security issues. These issues are discussed in detail in Appendix C, because the issues are dependent on the implementation of this protocol rather than inherent to the protocol itself.

The use of HTTPS as described in appendix A can provide end-to-end security.
References


Normative Appendices

A. Implementation Guidelines for a P-IMAP Session

A.1. HTTP/HTTPS Request/Response Format

It is possible to use HTTP/HTTPS as transport protocol for commands between the client and server. In this case, the client device embeds P-IMAP commands in the body of a request and POSTs it to the P-IMAP server. Multiple P-IMAP commands may be included in the same POST request. The P-IMAP server sends HTTP responses back to the device client with the result of the execution of the P-IMAP commands and pending events. If the client indicates that it understands gzip-compressed response by setting "Accept-Encoding: gzip" in the request header, server will compress the response, regardless of the current IMAP commands or session state.

The content-type is defined as "application/vnd.pimap". The general format for a client device to send commands to a P-IMAP server is:

```
POST /pimap HTTP/1.1 <CRLF>
Content-Type: application/vnd.pimap <CRLF>
Content-Length: <size of command string(s)> <CRLF>
Accept-Encoding: gzip <CRLF>
<CRLF>
<tag> <P-IMAP command> <CRLF>
[<tag> <P-IMAP command> <CRLF>]
```

- The P-IMAP command should be plain text (7bit) and should follow what is specified in section 4 of this document.
- Multiple P-IMAP commands may be sent on the same request. Thus P-IMAP commands must be tagged.
- These are the only HTTP headers required to be sent to the P-IMAP servers.

When the P-IMAP server sends back a response it must be in the following format:

```
HTTP/1.1 <HTTP Status Code> <CRLF>
Content-Type: application/vnd.pimap <CRLF>
Content-Length: <size of response string> <CRLF>
Content-Encoding: gzip <CRLF>
```
Notes:
The first line is the HTTP status code of the command execution. This could be one of the following:
- 200
  - One of the following 4 cases: all commands succeeded, or at least one command syntax is not correct, or at least command syntax is correct but semantics is not correct, or the current state is not correct. The Lemonade client needs to further parse response body to see what is the case. It should not depend on HTTP status code.
- 500
  - at least one command caused internal server error, meaning the Lemonade Server failed to execute the command.

A.2. Using Persistent HTTP/HTTPS for In-band Mode

It is possible to use persistent HTTP or persistent HTTPS so that the server can instantly send notifications to the client while a P-IMAP session is open. The client needs to open a persistent connection and keep it active. In this case, the HTTP headers must be sent the first time the client device opens the connection to the P-IMAP Server. These headers define a huge content-length and set the transfer coding to be chunked [RFC2616, Sec. 3.6.1]. All subsequent client-server requests are written to the open connection. Thus, the server can use this open channel to push events to the client device at any time.

B. Event Payload

B.1. Event Payload in Clear Text for P-IMAP Sessions

The event payload for a P-IMAP session follows the general format explained in Section 1.2.2, and is in clear text.

B.2. Outband 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.
When the client finally connects, the P-IMAP server has opportunity to send other pending events for this client.

Example: new message arrives on the server and this is notified via outband.
S: pushes SMS with the following text:
<emn
    mailbox="mailat:joe@foo.com"
    timestamp="2004-02-20T06:40:00Z">
</emn>
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.

C. Security Issues for Proxy-Based Implementations of P-IMAP

In some implementations of P-IMAP, the client may connect to a proxy that sits in an operator network, but the backend email storage server sits in a separate enterprise network. The enterprise network is assumed to be secure, but the operator network may not be trusted. If unencrypted information lies in the operator network, that information is vulnerable to attacks.

If the P-IMAP extensions are all implemented in the enterprise network, then the proxy on the carrier should be an encrypted SSL pass-through proxy. The proxy is unaware of the encryption keys and thus cannot encrypt any data. Without the encryption key, this proxy cannot see any of the information sent from the client, nor can it send any bogus commands to the backend enterprise email server to corrupt the user’s mailbox. The additional cost for this design is that the backend enterprise email server and the client devices must have additional processing to handle this encryption.

If the P-IMAP server is implemented as a backend IMAP server with additional command processing done on the proxy, there are more complex security issues. This proxy must be able to send commands to the backend server to accomplish its tasks, as well as read information coming from the backend server. An attacker thus can
send commands to the backend to change the state of the mail storage, possibly corrupting it. In addition, it can read responses from the mail server that might contain confidential email information. This proxy may also send bogus responses back to the client. Clearly, this setup is not an ideal issue and many complications that make this problem complex to solve. The suggestion recommended is to remedy the problem of unencrypted, untagged FETCH responses that may contain confidential information. Untagged XENCRIPTED responses (see Section 5.1.6) should be used in place of any untagged FETCH responses, which contain encrypted message information to be passed through the P-IMAP proxy on the operator network. The key exchange for encryption should not occur through the proxy. It has to be done through another channel: manually entered by user (e.g. password), or via an HTTP SSL request to the enterprise server. Any other additional server responses containing sensitive information (passwords, etc.) should be XENCRIPTED. The server should implement 3DES encryption and use the client’s password as the key.

Non-Normative Appendices

D. Use Cases

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

D.1. State Comparison-Based Sync

Each time a client logs into a new P-IMAP 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 P-IMAP server. It connects to the server and performs the following command:

First, it retrieves all the new messages.

C: A01 UID FETCH X+1:* ALL
S: * m FETCH ...
S: ... <more new messages if they exist>
S: A01 OK FETCH completed

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

C: A02 FETCH 1:m-1 (UID FLAGS)
S: * 1 FETCH (UID 3242 FLAGS (\Seen ...))
S: ... <info for 2 through n-1>
S: * n FETCH (UID 3589 FLAGS (\Seen ...))
S: A02 OK FETCH completed

D.2. Event-Based Sync

During a P-IMAP 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 P-IMAP 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:
C: A02 IDLE
S: + Ready for argument

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:
S: * 501 EXISTS
First, the client must exit from this IDLE command.
C: DONE
S: A02 OK IDLE completed
Next, the client retrieves this new message using a FETCH command.
C: A02 FETCH 501 ALL
S: * 501 FETCH ...
S: A02 OK FETCH completed
The client returns to IDLE mode by issuing another IDLE command.
C: A03 IDLE
S: + Ready for argument

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.
E. Other Issues

E.1. Using a Side Channel for a P-IMAP session

In some cases, it may be more efficient for a mobile client to connect to a P-IMAP session through a side channel rather than directly. This side channel opens a P-IMAP session, acting as the client device and must conform to all requires of the client in this document. The requirement is that the side channel must ensure that the client is in sync with the mobile repository.

An example would be if a mobile client connected to a desktop on a cradle, and then that desktop opens a P-IMAP session as the mobile client via a fast connection. The desktop should then retrieve the state of the client device and modify it using event-based or state-comparison-based synchronization over the cradle. The connection from the client to the server over the cradle and then the desktop to server connection might be much faster or easier than any connection the client could maintain itself. The desktop might also perform most of the computation needed for a state-comparison-based synchronization, easing up the burden on the mobile client.

If the client uses some other kind of side channel that does not connect to the P-IMAP server when checking email, it is the client’s responsibility to make sure to ignore pending events as appropriate.

Future Work

[1] Allow support for a client device to track changes in multiple folders at once.
[2] Enhance XZIP so that a client device can zip requests to the server.
[3] Have an N most recent messages filter.
[4] Allow support in outband notifications to contain message events.

Version History

Updates for Release 03

[2] Section 1.1: Additional positioning of pull / poll model versus push model.
[3] Clarification in section 1.2 of the reaction of P-IMAP clients to events.
[4] Clarifications of sections 1.2.1, 1.2.2 and 1.3.
[6] Section 2 has been added to position P-IMAP and the Lemonade Pull Model described in [LEMONADEPROFILE].
[8] Section 3.1 Â Reorganization of the text for clarification.
[9] Section 3.2.3 Â Additional motivation for using outband notification
[10] Change of title fpr section 4.1
[11] Section 5.1.1 Â Change of normative statement from SHOULD to MUST.
[12] Section 5.2.3 Â Extension of the type of outbound notification channels.
[13] Section 5.2.3 Â Fixes of examples: Changes of N to P.
[14] Section 5.2.4 Â Clarification of XZIP normative statements depending on the selected binding for P-IMAP.
[15] Mention of HTTPS under security considerations
[17] Reference updates to reflect [LEMONADEPROFILE].
[19] Updates to release history (Release 03)
[20] Updates of authors
[21] Additions of sections on Intellectual Property Statement and Full Copyright Statement

Updates for Release 02
[1] Throughout this document Â took out references to mailbox since its definition was ambiguous. Now, the terms folder, email account, and repository are used instead.
[2] Section 1.2.2 Â took out message events, which is now described in new section 3.
[3] Section 1.4 Â removed attachments behavior
[4] Section 3 Â new section containing event payloads
[5] Old section 3.1.3 Â removed this section on forwarded flags
[6] Old section 3.1.4 Â added resync, folder, and session untagged response syntax
[7] Old section 3.1.5 Â UID becomes should instead of must requirement
[8] Old section 3.1.7 Â took out resync, which is now in login section
[9] New section 4.1.6 Â a new section concerning untagged XENCRYPTED responses in place of untagged FETCH responses.
[10] Old 3.2.1 Â XPROVISION now just returns what XFILTERS are supported and what values some PIMAP Prefs can take on
[11] Old 3.2.2
   [a] Took out PIMAP_OUTBAND_NEW_FORMAT
   [b] Added in PIMAP_INBAND_PUSH format
valid values for some preferences are given in XPROVISION

XGETPIMAPPREFS untagged response

Old 3.2.3 - defined XFILTER untagged response

Old 3.2.4 - dropped this section on XTERSE

Old 3.2.6 - changed syntax so only V & N can be given for get.

Old 3.2.7

XUIDCONVERT -> UID CONVERT
[b] added untagged response syntax

Security Considerations section - added in that there are additional security considerations when the server is implemented through a proxy on a distrusted operator network.

Appendix B.2 - changed example where client gets events in response to a login command (instead of noop)

Appendix C - new appendix to cover security issues for proxy-based deployments of P-IMAP.

Appendix E.2 on further considerations, which are things to add in the upcoming releases.

Updates for Release pre-01

1. Sections 1.1, 1.3, 2.2.1, 2.2.2, and 2.2.3
   Added diagrams to better explain P-IMAP concepts

2. Section 1.4
   [a] Point 1 - changed term definition to Compression
   [b] Added points 5 and 6 regarding Attachment Handling

3. Section 3.1.4
   Updated minimal P-IMAP server requirements

4. Section 3.1.5
   [a] Fixed the title Â– P-IMAP Session/Login
   [b] Added examples for Â– First LoginÂ– and Â–Login after LogoutÂ–
   [c] Added Section 3.1.7
   [d] RESYNC untagged response when missed notifications occur

5. Section 3.2.2
   [a] XSETPREF and XGETPREF -> XSETPIMAPPREF and XGETPIMAPPREF
   [b] Reduced the number of preference parameters

6. Section 3.2.3
   Added a Days Before Today filter

7. Removed section 4

8. References
   [a] Added references to IMAP-DISC and RFC 2180
   [b] Removed references to MIMAP, NSMS

9. Appendix B
   [a] added example of outband notification
   [b] explained client behavior in response to notifications

10. Old Appendix C
    Removed completely, as attachment conversion is described in XCONVERT command and ways of retrieving it are discussed in RFC 2683

Appendix C now features security considerations for proxy-based implementations of P-IMAP.

Release 00
Initial release published on Feb. 8th 2004

Acknowledgments
The authors want to thank their colleagues from Oracle and colleagues from the numerous other companies who have contributed key insight and extensively reviewed several versions of the P-IMAP concepts and early P-IMAP specifications.

A special thanks is addressed to several employees of Nokia and Openwave and original co-author Jean Sini.

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

Jean Sini
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
USA

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

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

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

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

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 implementors 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 2003. All Rights Reserved.

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.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS 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.