Lemonade
Internet Draft: P-IMAP
Document: draft-maes-lemonade-p-imap-12

S. H. Maes
C. Kuang
R. Lima
R. Cromwell
E. Chiu
J. Day
R. Ahad
Oracle Corporation
Wook-Hyun Jeong
Samsung
Electronics Co.,
LTD
Gustaf Rosell
Sony Ericsson
J. Sini
-
Sung-Mu Son
LGE
Fan Xiaohui
Zhao Lijun
China Mobile
D. Bennett
Consilient

Expires: September 2006 March 2006

Push Extensions to the IMAP Protocol (P-IMAP)

Status of this Memo

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 changes actively to a client, rather then 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. Eventually P-IMAP aims at being neutral to the network transport neutrality.

P-IMAP is a recommendation for interoperable intermediate implementations awaiting [LEMONADEPROFILEBIS] or the realization of the OMA MEM enabler using it.

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

1. Introduction

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 in-band and out-of-band 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.

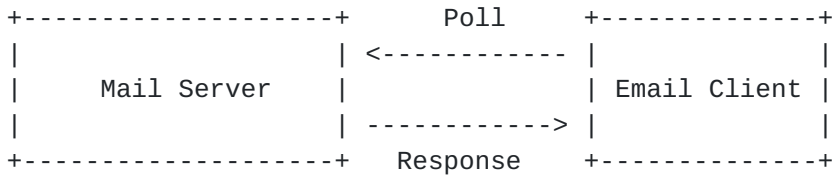

Figure 1: Periodic Poll Model

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. The P-IMAP protocol defines the semantics for pushing events to a client. The push model is seen in Figure 2.

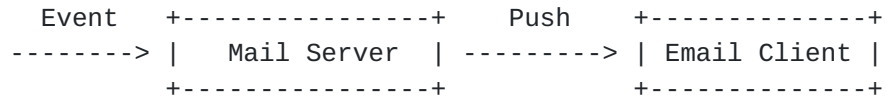

Figure 2: Push Model

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 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.

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 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 non-optimal but illustrative 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 a list of interesting folders from the server. The client should issue LIST to figure out which folders have to be retrieved. It could then use 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
```

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.

P-IMAP recommends that P-IMAP disconnected clients follow the

recommendations of [IMAP-DISC] and utilize the IMAP extensions that

are available as UIDPLUS [UIDPLUS], CONDSTORE [CONDSTORE}, LITERAL+ [LITERAL+], and MULTIAPPEND [RFC3502].

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 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 P-IMAP client drops a connection or accidentally disconnects, the P-IMAP 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.

1.2.3. 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 P-IMAP were to be implemented over session-less protocol such as SMS, or over an 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 P-IMAP commands to optionally carry a session ID in them so that the P-IMAP 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 P-IMAP 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 P-IMAP commands is to encode them as a prefix of the tags. For a definition of tagged and untagged exchanges, see later on. In this scheme, when the client logs in into the P-IMAP server with the device ID (defined later) appended to the user 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 P-IMAP client (so that it be knowledgeable of it) as part of the authentication. The P-IMAP client will then construct P-IMAP command tags using the SID as a prefix. For any P-IMAP command, the P-IMAP client may receive an untagged LOGIN response. In this case, the P-IMAP client must assume that the session to the server is severed and must take the appropriate action. So with such a scheme, the P-IMAP client must always assume that the session is still alive unless the P-IMAP server informs it otherwise. The client therefore will behave like a connected client (i.e. logged in within a valid P-IMAP 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 P-IMAP client has to resort to the state comparison based synchronization. It depends on the P-IMAP 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 P-IMAP client may do event based synchronization as long as the UIDs are still valid for the folder.

1.2.3. Clarification note on the term P-IMAP Session

P-IMAP session in this document differs from the definition of session (conventional IMAP session) in [RFC3501]. In RFC3501, the term session refers to time spent in a folder via SELECT/EXAMINE and the sequence of commands executed for that duration. SELECT or EXAMINE on another folder ends the session

The concept of P-IMAP session defined in this document pertains to all of the user associated state since LOGIN/AUTHENTICATE similar to [CONNECT]. This is significant, because event based synchronization adds significant amount of state to the session. The server is presumed to temporarily maintain for a limited duration, a list of changes made to every folder the user is subscribed to, which the client may receive when it SELECTs a folder. This is in addition to the normal IMAP state, such as remembering what the current selected mailbox is.

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 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 (ie pushed from the server) as soon as possible by the server. All three repositories have the same set of folders.

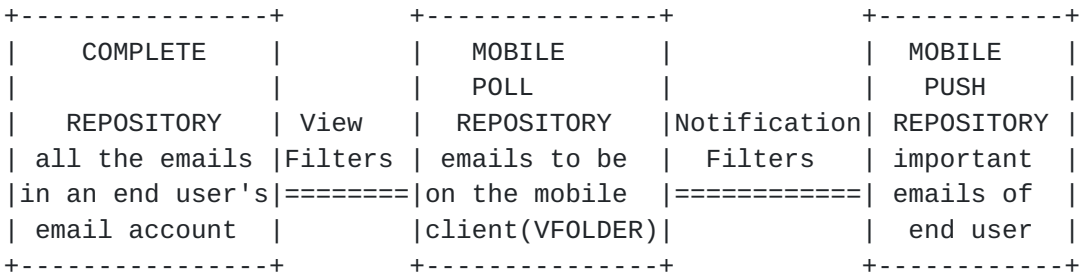

Figure 3: Filters and Repositories

Formally, a repository consists of a set of folders, and each folder has both a name and a set of messages associated with it. 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. 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 P-IMAP. 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.

Note UIDs are assumed the same in these repositories for the same message.

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 literal IMAP data in a command or response.

[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. This is not intended to replace SMTP but rather to provide a mechanism that can be easily and rapidly implemented by servers and that is especially well adapted to gateways / proxies used to enable e-mail and submission servers.

[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 can wait for certain period of time 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 edit portions of the original message and re-compose the edited body parts (addressees, body, attachments) using CATENATE [CATENATE] and IMAPURLs [RFC2192], [IMAPURLbis].

[6] Attachments conversion - The P-IMAP server can convert attachments to other formats to be viewed on a mobile device. The client can explicitly request a particular conversion. The server complies on a best effort basis. When not possible, the server determines based on its own strategy (e.g. based on knowledge of the client as discussed hereafter) how to convert, unless the client disables server fallback. If the server knows the characteristics of the device or can determine them (out of scope of P-IMAP), the attachments can also be optimized for the capabilities of the devices (e.g. form factor of pictures). See discussion in Appendix D. This is a recommended server behavior.

[7] PIM (personal information management) - The protocol can also provide 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.) Vendors may rely on P-IMAP to exchange calendar events and address book changes as attachments or messages. With the name reserved by [VFOLDER], it should be clear that folders starting with Calendar* and Contacts* can be used for this purpose. PIM (calendar and address book) MAY of course be separate services from email.

2. Relation with the Lemonade Profile and other E-mail specifications

P-IMAP is a recommendation for interoperable intermediate implementations awaiting [LEMONADEPROFILEBIS] or the realization of the OMA MEM enabler using it.

As the LEMONADE profile [LEMONADEPROFILE, LEMONADEPROFILEBIS] work progresses, the P-IMAP specifications are expected to further converge towards the profile.

Note that P-IMAP defines multiple bindings. When it relies on TCP bindings for P-IMAP requests and responses, P-IMAP can be viewed as a direct extension of IMAPv4 Rev1 (or IMAP4 profile for mobile) and therefore a good candidate for the Lemonade mobile optimization. With other bindings, it becomes a way to implement optimized mobile and push e-mail using IMAP semantics appropriately extended and transported on other bindings.

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 notification filters. A mobile client connects to the P-IMAP server by supplying its login information. P-IMAP extends the IMAP login information by permitting the username 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 during authentication, then a regular IMAP session is initiated instead of a P-IMAP session. The credentials sent during the LOGIN/AUTHENTICATE exchange are 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 one view or more view filters and exactly one notification filter for each view. These filters are saved and thus persist across P-IMAP sessions. Filters can be modified when a P-IMAP session is open, however modifications may impose a cost on the client (full state sync).

P-IMAP does not constrain how notification filters are defined. They

may be based on SIEVE [RFC3028] and following work. Creation of view filters are achieved via the P-IMAP extension VFOLDER.

3.1.1. 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. View filters are implemented using the VFOLDER CREATE extension which is used to create IMAP folders which are views of other folders. A P-IMAP server MAY choose to impose the restriction that vfolders are per-device and thus two vfolders with the same name can exist for the same P-IMAP user. The syntax for defining a vfolder follows the syntax for creating an IMAP folder combined with an IMAP search. The vfolder search syntax 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. Creating a filter of messages received starting a certain number of days before the current day can be achieved with the new SEARCH WITHIN [WITHIN] extension. The ALL search criteria, when used alone, means that every email event satisfies the criteria.

A view filter (vfolder) cannot be modified, it can only be deleted and recreated. Whenever this happens the client associated to the user must to perform a state-comparison-based sync to keep in sync with the mobile repository.

3.1.2. 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.

3.1.3. 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.

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, in-band, and out-of-band. It is explicitly stated in what situations these three connectivity models can be used.

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.

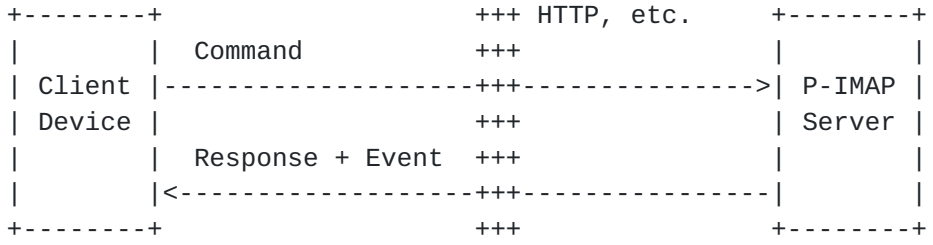

Figure 4: In-Response connection

Cases of in-response connection:
[1] HTTP/HTTPS binding
- Server Requires: HTTP/HTTPS listener for P-IMAP
- Client Requires: HTTP/HTTPS client with P-IMAP processing
[2] TCP Binding
- Server Requires: P-IMAP
- Client Requires: P-IMAP + no IDLE

3.2.2. 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.

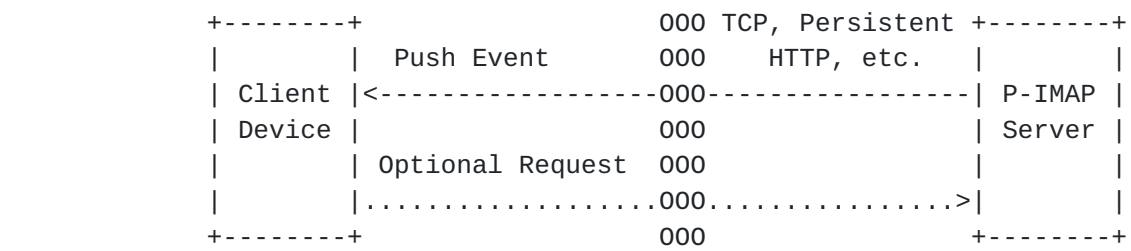

Figure 5: In-band Connection

Cases of in-band connection:

[1] TCP Binding, Always connected, IDLE

- Server Requires: P-IMAP + IDLE
- Client Requires: P-IMAP + IDLE, constant TCP connection

[2] Any other persistent two-way connection

- Server Requires: P-IMAP + IDLE on persistent connection (e.g. HTTP/HTTPS)
- Client Requires: P-IMAP + IDLE on persistent connection (e.g. HTTP/HTTPS), constant connection

Persistent HTTP/HTTPS may sometimes be difficult to achieve with today s intermediaries if the HTTP server does not support HTTP 1.1 correctly or has a very short timeout period for persistent connections.

Both connectivity models above (In-response and in-band) involve a maintained data connection with notification exchanged within the P-IMAP band (i.e. P-IMAP exchanges).

3.2.3. Out-of-band Connectivity

This case covers notification outside the P-IMAP band :

- In a different connection
- Within the same data connection but outside the P-IMAP band

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),WAP Push (and WAPWDP), SIP notification and UDP. As in the in-band scenario, the client may optionally open a P-IMAP session over an in-band or in-

response connection and send a command as a result of receiving an event.

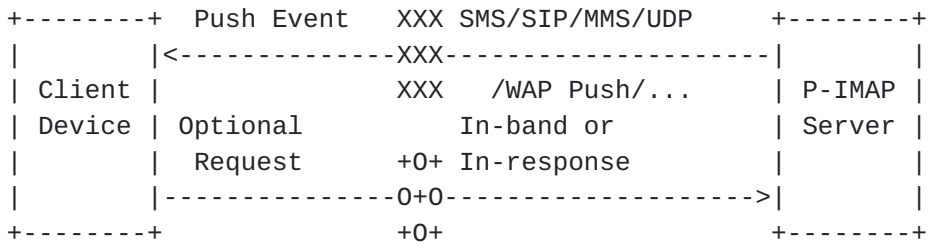

Figure 6: Out-of-band Connection

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:
- HTTP or HTTPS
- TCP
- Other transport

3.3. Recommended Connectivity Models

To address the problems discussed in [MEMAIL], it is a good idea to always support the out-of-band connectivity model with HTTP/HTTPS binding.

Support of HTTP/HTTPS binding is recommended as a minimum option.

Recommended out-of-band channels include SMS, UDP (if supported by target networks and deployment model) and SIP event notification all using the payload format discussed in appendix B.

3.4. Keeping the Client In Sync with the Mobile Repository

Whenever a client device opens a new P-IMAP session with a P-IMAP server and the P-IMAP server has to open a new session with the IMAP server for this client, 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 P-IMAP 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, P-IMAP 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, P-IMAP 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 P-IMAP server within a session, the client must SELECT each folder separately to receive the changes for that folder.

4. Events

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

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

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

[1] The client receives an EXISTS/RECENT event from the server indicating a new message.

```
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.

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 (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 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/AUTHENTICATE command with a valid P-IMAP device ID appended to the username. Once this is accepted, the P-IMAP session is started and the client enters the AUTHENTICATED state 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 out-of-band channels are supported. The XZIP command can be used to fetch body parts and zip returned literal data. 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 XCONVERT, XSETPIMAPPREF, XGETPIMAPPREFS, and XVFOLDER related commands as needed. XCONVERT is used for body parts.

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. Mobile Repository

In a P-IMAP session, the client can access messages in the mobile repository (e.g a VFOLDER defined on INBOX) or it may choose to access messages in the complete repository.

5.1.2. 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 P-IMAP server conforms to that requirement, and must list what P-IMAP version it supports.

XPIMAPv1r2 means that the server supports PIMAP version 1.2, which includes the commands, XPROVISION, XSETPIMAPPREF, XGETPIMAPPREF, XDELIVER, XZIP, XCONVERT, XVFOLDER, XENCRYPTED and BINARY. It also implies support for LITERAL+ [LITERAL+].

A server can also enumerate individually the P-IMAP commands and additional 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 P-IMAP server that implements P-IMAP Version 1rev2.
C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=LOGIN IDLE xPIMAPv1r2
S: a001 OK CAPABILITY completed

A P-IMAP server can declare the draft revision that it complies to via: xPIMAPcomplyrev12 for revision 12 etc...

5.1.3. P-IMAP Session/Login

An email user's username for a P-IMAP session is its regular username + "#" + its P-IMAP device ID + the email domain. P-IMAP device IDs might be "P" + the device ID issued by the P-IMAP 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 P-IMAP session, the client uses a LOGIN/AUTHENTICATE command with this new username.

The P-IMAP 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 PIMAP 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 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 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 P-IMAP 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 P-IMAP 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 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: * SESSIONID 123456
S: A02 OK LOGIN completed
```

5.1.4. 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 5.3.2. The client should send this command while in-session to enter in-band mode, where the server will actively push notifications to the client.

5.1.5. XENCRYPTED

For certain proxy-based implementation of P-IMAP (see Security Considerations and Appendix C), it may be necessary to object level encryption for FETCH responses of email content. In that case, the server uses a new encrypted literal syntax for any FETCH data that it s security policy wants unreadable by a proxy. 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).

Server's response to the CAPABILITY command announcing XENCRYPTED methods.

```
C: A02 CAPABILITY
S: * CAPABILITY IMAP4rev1 XENCRYTPED=3DES,RC40,AES
S: A02 CAPABILITY completed
```

The new encrypted literal extends the IMAP BINARY literal8 syntax, and has the following syntax:

```
literalx =  ~{X  number [ + ]  }  CRLF *BINCHAR /
        {X  number [ + ]  }  CRLF base64

string =/ literalx

BINCHAR = <0x00   0xFF>
       ; encrypted data payload
       ; subject to RFC2630 Section 6.3 padding before encrypting
```

The key used to encrypt data using algorithms such as 3DES or AES, must be computed or provisioned into the device. If the client uses SASL [RFC2222] to authenticate, then a session key should be computed according to Section 2.4 of [RFC2831]. Otherwise, a key must be provisioned via a mechanism which is out-of-band with respect to the proxy server, either through an alternate TCP or HTTP mechanism, such as a TLS or HTTPS connection, a secure SMS, or manually by user entry of a shared secret.

The user s password or shared secret with the server may also be used to derive this key.

Keys and key updates can be provided via XPROVISION only in the case that the connection is secured against the proxy, such as using SSL through the proxy to provision keys, or via an out-of-band (with respect to the proxy) method such as XPROVISION executed over an HTTPS binding. Otherwise, a session key may be derived in the same way that SASL derives session keys when confidentiality is requested. See also the analysis presented in Appendix C.

5.2.
Registering with the server

When the client registers itself with the server, it sends a HELLO message with the device ID in plain text and a payload, which is the device ID, encrypted using the encryption key associated with the server, to the Notification Delivery Service. The format of this message is:

HELLO sp deviceID sp encrypted-deviceID and network-characteristics

Network-characteristics may be the device IP address or any other information the device wants to send. The server is expected to use what it understands and disregard the rest.

The server will look up the encryption key associated with the device. If the encryption key does not exists, INVALID ENCRYPTION KEY response is sent to the Notification Processor in plain text. If the encryption key exists the Notification Delivery Service will use it to decrypt the payload using 64-bit Advanced Encryption Standard or 64-bit Triple-DES algorithms and compare it to the device ID. If they match, it will retrieve information that it has on the device. It will then send the OK response to the caller (client). When UDP notifications are used it will send with the encrypted server IP Address and port number of the Notification Delivery Service as described in XPROVISION.

Whenever the server must send a notification to the client, the server generates a unique sequence number and content for the notification, encrypts it using the encryption key, and sends it to the device. The mechanism to send it may be a UDP/IP session if one is available to the device or any other out-of-band message otherwise.

When XENCRYPTED is used, all in-band messages from the server are similarly encrypted.

The client can use the same key to encrypt its messages to the server.

Note that if proxies are not an issue (see Appendix C and section on Security considerations), STARTTLS may be used by the client. In such cases, XENCRYPTED does not present any advantages and should not be used.

5.3. P-IMAP Extension Commands and Responses

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

5.3.1. XPROVISION

The XPROVISION 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 P-IMAP preferences and the values they can be set to. In addition, UDP information may be given when UDP notification is supported, such as the host name and port. A P-IMAP server can return other parameters as long as its syntax is agreed upon with the P-IMAP client.

```
xprovision_cmd =  tag SP "XPROVISION" SP 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 XPIMAPPREF response = "*" SP "XPROVISION" SP
   "XPIMAPPREF" SP prev-name SP "(" pref_val_list ")"
untagged XPROVISION UDP_HOST response = "*" SP "XPROVISION" SP
   "PIMAP_UDP_HOST" SP "(" udp_hostname ")"
untagged XPROVISION UDP_PORT response = "*" SP "XPROVISION" SP
   "PIMAP_UDP_PORT" SP "(" udp_portnum ")"
untagged XPROVISION ENC_KEY response = "*" SP "XPROVISION" SP
   "PIMAP_ENC_KEY " SP "(" encryptionkey ")"
```

If LPROVISION is returning keys, the server should only do so in the following circumstances:

a) TLS/SSL is being used
b) SASL [RFC2222] confidentiality protection has been negotiated and enabled.

Example: The client issues an XPROVISION command. The server returns the values of various PIMAPPREF's and the values they can be set to. 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 XPIMAPPREF PIMAP_OUTBAND_CHANNEL (SMS NONE)
S: * XPROVISION XPIMAPPREF PIMAP_INBAND_NEW_FORMAT (NONE)
S: * XPROVISION XPIMAPPREF PIMAP_INBAND_PUSH (ON OFF)
S: * XPROVISION XPIMAPPREF PIMAP_EVENT_FILTER (NEW)
S: * XPROVISION XPIMAPPREF PIMAP_OUTBAND_FORMAT (EMN EXTENDED)
S: * XPROVISION PIMAP_NOTIFICATION ADDRESS (address)
S: * XPROVISION PIMAP_NOTIFICATION PORT (portnum)
```

```
S: * XPROVISION PIMAP_ENC_KEY (enc_key)
S: A002 OK XPROVISION completed
```

The following two instructions should be deprecated but are currently maintained for backward compatibility to earlier versions of P-IMAP:

```
S: * XPROVISION PIMAP_UDP_HOST (udp_hostname)
S: * XPROVISION PIMAP_UDP_PORT (udp_portnum)
```

UDP HOST and UDP PORT define where the client initiates a UDP session for UDP notification.

Event payloads are discussed in Appendix B.

5.3.2. 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 out-of-band 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.

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

[1] PIMAP_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 XPROVISION command. This is not applicable to out-of-band UDP notifications.

[2] PIMAP_OUTBAND_CHANNEL - the channel to send out-of-band notifications, either SMS, GSMSMS, WAP_PUSH, WAPWDP, MMS, SIP, UDP or NONE. When NONE, the P-IMAP 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 XPROVISION command.

[3] PIMAP_IN-BAND_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.

[5] PIMAP_OUTBAND_FORMAT - the format to send the out-of-band notifications, i.e. EMN or EXTENDED.

[6] PIMAP_EVENT_FILTER - The event filter for this user. Possible values are ALL or NONE or NEW, depending on the server's capabilities.

```
xgetpimappref_cmd =  tag SP "XGETPIMAPPREFS" SP "("
                     pimap_pref_list ")"
pimap_pref_list = pimap_pref [SP pimap_pref_list]
pimap_pref = (PIMAP_OUTBAND_ADDRESS /
              PIMAP_OUTBAND_CHANNEL / PIMAP_INBAND_NEW_FORMAT /
              PIMAP_INBAND_PUSH / PIMAP OUTBAND FORMAT /
              PIMAP_EVENT_FILTER)
Valid States:  AUTHENTICATED or SELECTED
Responses:  REQUIRED untagged XGETPIMAPPREFS response with the value
of the requested parameter.
   untagged XGETPIMAPPREFS response - "*" XGETPIMAPPREFS pref-pair
   pref-pair = "(" pimap-pref SP pimap-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 XGETPIMAPPREFS
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"/"GSMSMS"/"WAP_PUSH"/"WAPWDP"/"MMS"/"UDP"/"SIP"/ "NONE"))
      /
       ("PIMAP_INBAND_NEW_FORMAT" SP fetch_criteria) /
```

```
       ("PIMAP_INBAND_PUSH" SP ("ON" / "OFF")) /
       ("PIMAP_OUTBAND_FORMAT SP ("EMN" / "EXTENDED"))
```

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.

```
   C: A002 XSETPIMAPPREF PIMAP_OUTBAND_ADDRESS 13335559999
   S: A002 OK XSETPIMAPPREF completed
   C: A003 XSETPIMAPPREF PIMAP_OUTBAND_CHANNEL SMS
   S: A003 OK XSETPIMAPPREF completed
   C: A004 XSETPIMAPPREF PIMAP_OUTBAND_FORMAT EXTENDED
   S: A004 OK XSETPIMAPPREF 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 XSETPIMAPPREF PIMAP_INBAND_NEW_FORMAT ALL
   S: A002 OK 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>
```

5.3.3. XZIP

XZIP should be seen as a way to address the issues of bandwidth optimization.

The XZIP command is an extension of [RFC3516] IMAP BINARY, which introduces three new commands XZIP , XZIP.PEEK , XZIP.SIZE that parallel the syntax and semantics of BINARY , BINARY.PEEK , and BINARY.SIZE in [RFC3516]. In general, XZIP inherits all of the requirements and semantics of [RFC3516] s BINARY and BINARY.PEEK , except that the content transfer encoding being requested is understood to be the result of what would be returned from BINARY decoding, followed by the application of the DEFLATE algorithm.

Example: Zipping a body part fetch
C: A1 FETCH 123 XZIP.PEEK[1.2]
S: * XZIP[1.2]~{1234}
S: .binary decoded and deflated data .
S: A1 OK FETCH completed

As mentioned in RFC3516, XZIP.SIZE is a potentially expensive operation, as in XLZIP, so clients should be aware that making successive requests for the same part may be expensive.

5.3.4. XDELIVER

XDELIVER enables the efficient composition and transmission of email using IMAP commands. This provides simple ways to provide reply and forward without download complete messages utilizing a gateway to the email and submit servers.

XDELIVER is not intended to replace SMTP [RFC2821]. Instead it is envisaged as a simple way to implement gateways that support features like reply and forward without downloading complete messages when the email and submit servers may not support the commands described in [LEMONADEPROFILE] to support such capabilities.

XDELIVER may allow some clients to reduce the amount of protocols supported ports in use, parameters to set or provisioned, or network protocols required.

All these are important features required in particular to support mobile email use cases [MEMAIL],[OMA-ME-RD]:
- Forward and reply without download
- Ease of provisioning over the air
- Ease of manual provisioning
- Reduction of resources on the client
- Ease of implementation and deployment with existing email and submit servers

The XDELIVER specification proposes a new command and also requires the CATENATE [CATENATE], LITERAL+ [LITERAL+] and RFC2192BIS IMAPURL extensions {IMAPURLbis].

5.3.5. IMAPURL extensions

IMAPURL is hereby modified according to RFC2192BIS [IMAPURLbis] to support the partial specifier in IMAPURLs which allows byte ranges of messages to be addressed. The format is
/;PARTIAL=<offset>[.<length>]

5.3.6. The XDELIVER command

After a message has been composed, it can be handed off to a submit server. The mechanism by which it does this is by proxying a batched set of SMTP commands to an SMTP server. XDELIVER is not an active SMTP tunnel, but instead works similarly to Batch SMTP [RFC2442], by allowing the client to compose a set of SMTP commands to be executed. The major difference is that those commands are not delivered via a special MIME message, but rather XDELIVER is the batch SMTP processor. Moreover, since XDELIVER exposes SMTP extensions that are available, the client need not make any assumptions about which SMTP extensions are available.

Finally, XDELIVER reuses the CATENATE and IMAPURL extensions when building the batch in order to allow inclusion of pre-composed messages or editing of envelope parameters.

Formal Syntax

```
xdeliver-cmd =  XDELIVER  SP ( CAPABILITY  / text-literal )
```

Examples

The following example will pick up the message that has been previously composed (via APPEND/CATENATE)

Example:

```
C: a004 XDELIVER CAPABILITY
S: * XDELIVER CAPABILITY ( 8BITMIME   EXPN   HELP )
C: a005 XDELIVER TEXT {123+}
C: EHLO
C: MAIL FROM: john@smith.com
C: RCPT TO: mooch@owatagu.siam.edu
C: DATA
C: URL  /Inbox;UIDVALIDITY=9999/;UID=33;Section=BODY
.
S: * XDELIVER {321}
S: 220 mail.metastructure.net ESMTP
S: 250-mail.metastructure.net
S: 250-AUTH LOGIN CRAM-MD5 PLAIN
S: 250-AUTH=LOGIN CRAM-MD5 PLAIN
S: 250-PIPELINING
S: 250 8BITMIME
S: 250 ok
S: 250 ok
S: 354 go ahead
S: 250 ok 1126337586 qp 28229
```

5.3.7. Note on XDELIVER, SMTP and Lemonade Profile

A P-IMAP server MAY advertise support for SMTP. A P-IMAP client MAY then select to rely on SMTP instead of XDELIVER. This of course may reduce the forward / reply without download capabilities that may be available.

A server MAY also advertise via capability support for Lemonade Profile [LEMONADEPROFILE] or the subset of commands (see [LEMONADEPROFILE] needed to support forward without download. In such case, the client MAY rely on the Lemonade profile forward without download mechanisms.

It is generally not expected that mobile clients will run mailing list services from mobile devices, utilize large distribution lists, or run automated mail notification services. Therefore, XDELIVER is not designed to support SMTP functions that take advantage of full control of the SMTP envelope, or SMTP extensions like NOTARY.

In general, because of mobile device resource constraints, and corporate firewall and security policies, XDELIVER is easier to deploy for mobile devices, than exposing and restricting SMTP services to mobile devices, especially those devices without VPN functionality.

5.3.8. XCONVERT BODY and BINARY data item extension

The client and server SHOULD support the IMAP Binary specification [RFC3516] and declare it via CAPABILITY.

XCONVERT is a FETCH extension used to transcode the media type of a leaf MIME part into another media type, and/or the same media type, with different encoding parameters. It adds new options to the section-spec part of the BODY data item, a new FETCH response data item BODYPARTSTRUCTURE, and new response codes. It is also expected to work with IMAP BINARY data item extension, whose grammar is modified as well.

XCONVERT s syntax is modeled after the HEADER.FIELDS syntax in RFC3501, and is generally structured as:

BODY[section-part.XCONVERT[.STRICT] (media type/subtype (parameters))]

BODY.PEEK[section-part.XCONVERT[.STRICT] (media type/subtype (parameters))]<partial>

BINARY[section-part.XCONVERT[.STRICT] (media type/subtype
(parameters))]<partial>

BINARY.PEEK[section-part.XCONVERT[.STRICT] (media type/subtype
(parameters))]<partial>

BINARY.SIZE[section-part.XCONVERT[.STRICT] (media type/subtype
(parameters))]<partial>

Example: The client fetches body part section 3 in the message with the message sequence number of 2 and asks to have that attachment converted to pdf format.

```
   C: a001 FETCH 2 BODY[3.XCONVERT ( APPLICATION/PDF )]
   S: * 2 FETCH (BODYPARTSTRUCTURE[3] ("APPLICATION   PDF" () NIL
      NIL "Base64" 2135 NIL NIL NIL) BODY[3] {2135}
      <the document in .pdf format>
      )
   S: a001 OK FETCH COMPLETED
```

Example: The client requests for conversion of a text/html section as text/plain and asks for a charset of us-ascii. The server cannot respect the charset request because there are non-us-ascii characters in the html code. Thus, in the untagged response, the server returns the charset of UTF-8 and utilizes a content transfer encoding capable of representing the full 8-bit range, along with the converted text.

```
   C: a001 FETCH 2 BODY[3.XCONVERT ( text/plain  ( charset   us-
ascii ))]
   S: * 2 FETCH (BODYPARTSTRUCTURE[3] ("TEXT" "PLAIN" () NIL
      NIL "Base64" 2135 181 NIL NIL NIL) BODY[3] {2135}
        the document in text/plain format
        )
   S: a001 OK FETCH COMPLETED
```

Example: The client requests for conversion of a text/html section as text/plain, but only wants 1000 bytes, starting from byte 2001.

```
   C: a001 FETCH 2 BODY[3.XCONVERT ( TEXT/PLAIN  ( CHARSET   us-
ascii ))]<2001.1000>
   S: * 2 FETCH (BODYPARTSTRUCTURE[3] ("TEXT" "PLAIN" () NIL
      NIL "7bit" 2135 181 NIL NIL NIL) BODY[3]<2001> {135}
        bytes 2001 - 2135 of the document in text/plain format
        )
   S: a001 OK FETCH COMPLETED
```

The server is not required to respect a particular transcoding request or its request parameters, although it MAY try to make a best effort to fulfill that request. Indeed, the server may know a priori information about the client obtained through a different mechanism outside the scope of P-IMAP (e.g. dynamically through device description mechanisms or when the device was associated to the account). These preferences may be used to predefine what conversions are possible. Ideally the client should request the same conversions. In addition, this information may also allow attachment adaptation (e.g. picture form factor) instead of solely format conversion.

5.3.9. FETCH response extensions

The BODYPARTSTRUCTURE data item is introduced when using the XCONVERT extension. It follows the exact syntax specified in the [RFC3501] BODYSTRUCTURE data item, but contains information for only the converted part. All information contained in BODYPARTSTRUCTURE pertains to the state of the part after it is converted, such as the converted mime-type, sub-type, size, or charset. The client must respect the return values and not assume the conversion request succeeds.

5.3.10. Status responses, Response code extensions

Some transcodings may require parameters. If a transcoding request is sent for a format which requires parameters, the server can reply with a BAD response. Likewise, malformed mime types may also generate BAD responses.

If the server is unable to perform the requested conversion because a resource is unavailable (internal error, transcoding service down) than a BAD response should be returned.

If a request is denied because of an operational error, such as lack of disk space, or because the requested conversion for some reason cannot be performed, and there is no fallback for this particular device (such as the case where a proprietary document format has no existing transcoding implementation, and the server recognizes that the client has no default viewer for it), the server SHOULD return a NO response.

Otherwise, the server should return an OK response. The client in general can tell from the BODYPARTSTRUCTURE response whether or not its request was honored exactly, but may not know exactly why it different.

The following extension response codes are provided for OK and NO responses to disambiguate those situations, or warn about possible important data loss.

INFORMATIONLOSS the conversion was satisfied for conversion request, but it may have resulted in the loss of important data (primarily of use for loss of text data, since richmedia is often compressed with loss)

BADPARAMETERS (xconvert-params) the listed parameters were not understood, or could not be honored for the reasons noted in section-text

SERVEROVERRIDE the server overrode the request because it determined it could substitute a better one based on preferences, device capability knowledge, or server policy.

5.3.11. Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (ABNF) notation as used in [ABNF], and incorporates by reference the Core Rules defined in that document.

This syntax augments the grammar specified in [RFC3501] and [RFC3516].

In the ABNF syntax section-binary of [RFC3516], is amended to:

```
section-binary = "[" [section-part [ .XCONVERT [.STRICT] SP
convert-params] "]"
```

In the ABNF syntax msg-att-static of [RFC3501], is amended to:

```
msg-att-static =/ BODYPARTSTRUCTURE ( body-type-1part )
```

In the ABNF syntax resp-text-code of [RFC3501], is amended to:

```
Resp-text-code =/ INFORMATIONLOSS / SERVEROVERRIDE /
                  BADPARAMETERS SP ( bad-param-list )

bad-param-list = transcoding-params
```

In addition, the following ABNF describes the syntax of the GETANNOTATION entries in Section 4.2

```
convert-entry-req = available-conversions / available-
transcoding-parameters

available-conversions = /convert/ from-mime-type
```

```
      from-mime-type =  *  /(astring [ /  (astring /  * )]
                      ; i.e.  *  or  type/*  or  type/subtype

      from-concrete-mime-type = astring  /  astring
                      ; i.e.   type/subtype

      to-mime-type = astring  /  astring

      available-transcoding-parameters =  /convert/  from-concrete-
mime-type  /  to-mime-type
                     ;i.e.
/convert/fromtype/fromsubtype/totype/tosubtype
```

Finally, two standard annotation attributes are defined. Under available-conversions entry, there will be an attribute named types.shared with the following ABNF:

```
      types-shared-value = from-concrete-mime-type( ;  from-concrete-
mime-type)*
```

And under an available-transcoding-parameters entry, there will be an attribute named params.shared with the following ABNF:

```
      params-shared-value = transcoding-param-name ( ;  transcoding-
param-name)*

params = "(" (media-basic / default-conversion)  [SP "("
   transcoding-params ")"] ")"

   transcoding-params  = transcoding-param-name SP transcoding-param-
value
   *(SP transcoding-param-name SP transcoding-param-value)

   transcoding-param-name = string

   transcoding-param-value = string

   default-conversion = "NIL" "NIL"
```

In the ABNF syntax section-binary of [RFC3516], is amended to:

```
       section-binary =   "[" [section-part [ . SP convert-params]
"]"
```

5.3.12. XVFOLDER

The XVFOLDER extension is present in any IMAP4 implementation which returns XVFOLDER as one of the supported capabilities in the CAPABILITY command.

A virtual folder is an IMAP4 folder with attached search criteria. The search criteria specify the backing mailbox, as well as a subset IMAP SEARCH grammar which may be applied to the immutable properties of messages in the backing mailbox. Once created, all operations applied to the virtual mailbox, such as APPEND and STORE, are actually applied to the backing mailbox. For all intents and purposes, the virtual folder looks and behaves like a real IMAP4 folder.

Any changes made to the underlying folder must pass the search criteria for the virtual folder before being visible. UIDs are preserved, and as well as the UIDVALIDITY value. In general, most mailbox state and metadata present on the backing folder should be identical on the virtual folder, except where it doesn t make sense. (e.g. EXISTS, RECENT, in general, values which are based on then number of messages which have/do not have a certain property in the mailbox)

Message sequence numbers will be different, but the order of the messages in the sequence, and the ordering of UIDs, MUST be preserved.

From the client s perspective, whether or not a mailbox is a vfolder is not visible, and for all intents and purposes, it appears as any other mailbox name. This includes the ability for a new virtual folder to be created by using another virtual folder as a backing mailbox.

For the purposes of this draft, immutability refers to message flags and non-immutable messages annotations.

Format Syntax:

The following syntax specification uses the Augmented Backus-Naur Form (ABNF) notation. Elements not defined here can be found in the formal syntax of the [ABNF], [RFC3501], and [ABNFEXTEND].

The create ABNF grammar in [RFC3501] is hereby modified to the grammar defined in [ABNFEXTEND]. An additional CREATE param LPSEARCH is introduced whose value is a list containing the backing store mailbox and the search parameters.

create_param =/ XPSEARCH SP (backing-mailbox psearch)

```
                    ;; conforms to generic "create-param" syntax as
   defined in [ABNFEXTEND]

   backing-mailbox = mailbox

   psearch          = search-program
                       ; defined in [ABNFEXTEND], amended by [WITHIN]
```

6. Considerations beyond the P-IMAP protocol

6.1. P-IMAP client security

It is recommended that P-IMAP clients SHOULD encrypt the email stored on the client and relies on password or other authentication to access the e-mail client.

To ensure revocation of the client when it is lost or compromised, it is recommended that clients SHOULD support the notification extension LOCK_DOWN described in Appendix B.2 to lock the client and delete all available e-mails.

6.2. P-IMAP client updates

It is recommended that P-IMAP client SHOULD be designed and deployed in ways that allow easy updates as the protocol evolves. Until standardization is completed, it is expected that P-IMAP will evolve from release to release.

Although servers MAY seek backward compatibility from release to release; it is rather encouraged to provides ways to update the client when required by the server.

Recommended approaches include:
- server being knowledgeable of the client revision support
- server able to provision over the air (e.g. OMA Device Management and OMA Client Provisioning) the new client or able to notify (e.g. via email) for update over cradle, or other means of the client.

6.3. P-IMAP client-side behavior

P-IMAP clients MAY allow additional user preferences like not reflecting to the server changes that have taken place on the client (e.g. email deleted on the client) or some changes on the server (e.g. flag changes or deleted email on the server). In such cases, the client is responsible for maintaining its own state and it MUST

make sure that it behaves with respect to the server as if it had reflected all the changes as expected by a P-IMAP server. This is further discussed in Appendix G.2.

6.4. Minimum binding interoperability requirements

For now, it is recommended to always support at the minimum HTTP/HTTPS binding for P-IMAP with EMN (SMS, GSM SMS or WAP WDP) for out-of-band notifications and IDLE over HTTP/HTTPS for in-band. The server SHOULD then also support other bindings to offer interoperability of preferred by the client.

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 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.

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 P-IMAP protocol.

Section 6.1 discusses encryption and passwords on the client.

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

References

[ABNF] D. Crocker, et al. "Augmented BNF for Syntax Specifications: ABNF , RFC 2234, November 1997. http://www.ietf.org/rfc/rfc2234

[BURL] Newman, C., "Message Composition", draft-ietf-lemonade-burl-xx (work in progress).

[CATENATE] Resnick, P. IMAP CATENATE Extension , draft-ietf-

lemonade-catenate-xx.txt, (work in progress).

[CONDSTORE] Melnikov, A. and S. Hole, "IMAP Extension for Conditional STORE", work in progress.

[CONNECT] Melnikov, A. et al. "IMAP4 extension for quick reconnect", draft-ietf-lemonade-reconnect-XX.txt, (work in progress)

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

[IMAP-DISC] Melnikov, A. "Synchronization operations for disconnected IMAP4 clients , draft-melnikov-imap-disc-xx, (work in progress).

[IMAPURLbis] Newman, C, Melnikov, A. and Maes, S. H. "IMAP URL Scheme , draft-ietf-lemonade-rfc2192bis-xx (work in progress).

[LEMONADEPROFILE] Maes, S.H. and Melnikov A., "Lemonade Profile", draft-ietf-lemonade-profile-xx.txt, (work in progress).

[LEMONADEPROFILEBIS] Maes, S.H., Melnikov A. and D. Cridland, " LEMONADE profile bis", draft-ietf-lemonade-profile-bis-xx.txt, (work in progress).

[LITERAL+] Myers, J., "IMAP4 non-synchronizing literals", RFC 2088, January 1997.

[MEMAIL] Maes, S.H., Lemonade and Mobile e-mail", draft-maes-lemonade-mobile-email-xx.txt, (work in progress).

[OMA-EN] Open Mobile Alliance Email Notification Version 1.0, August 2002. http://www.openmobilealliance.org/tech/docs/EmailNot/OMA-Push-EMN-V1_0-20020830-C.pdf

[OMA-ME-AD] Open Mobile Alliance Mobile Email Architecture Document, (Work in progress). http://www.openmobilealliance.org/

[OMA-ME-RD] Open Mobile Alliance Mobile Email Requirement Document, (Work in progress). http://www.openmobilealliance.org/

[OMA-DS] Open Mobile Alliance Data Synchronization, versions 1.1.2 and 1.2, http://www.openmobilealliance.org/release_program/ds_v112.html, http://www.openmobilealliance.org/release_program/ds_v12.html.

[OMA-STI] Open Mobile Alliance, Standard Transcoding Interface Specification, version 1.0, [Work in progress]

(http://member.openmobilealliance.org/ftp/Public_documents/BAC/STI/Permanent_documents/OMA-STI-V1_0-20050209-D.zip).

[OMA-vObject] Open Mobile Alliance, vObject Minimum Interoperability Profile, v 1.0, http://www.openmobilealliance.org/release_program/docs/CopyrightClick.asp?pck=vObject&file=v1_0-20050118-C/OMA-TS-vObjectOMAProfile-V1_0-20050118-C.pdf

[RFC1951] Deutsch, P. DEFLATE Compressed Data Format Specification version 1.3 , RFC1951, May 1996. http://www.ietf.org/rfc/rfc1951

[RFC2088] Myers, J. IMAP non-synchronizing literals , RFC2088, January 1997. http://www.ietf.org/rfc/rfc2088

[RFC2119] Brader, S. "Keywords for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119

[RFC2180] Gahrns, M. "IMAP4 Multi-Accessed Mailbox Practice", RFC 2180, July 1997. http://www.ietf.org/rfc/rfc2180

[RFC2192] Newman, C. IMAP URL Scheme , RFC 2192, September 1997. http://www.faqs.org/rfcs/rfc2192.html

[RFC2222] Myers, J. "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997. http://www.ietf.org/rfc/rfc2222

[RFC2234] Crocker, D. and Overell, P. "Augmented BNF for Syntax Specifications", RFC 2234, Nov 1997. http://www.ietf.org/rfc/rfc2234

[RFC2420] Kummert, H. "The PPP Triple-DES Encryption Protocol (3DESE)", RFC 2420, September 1998. http://www.ietf.org/rfc/rfc2420

[RFC2442] Freed, N., Newman, C., Belissent, J. and Hoy, M., "The Batch SMTP Media Type", RFC 2442, November 1998. http://www.ietf.org/rfc/rfc2442.txt?number=2442.

[RFC2616] Fielding, R. et al. "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999. http://www.ietf.org/rfc/rfc2616

[RFC2617] Franks, J. et al. "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999. http://www.ietf.org/rfc/rfc2617

[RFC2683] Leiba, B. "IMAP4 Implementation Recommendations", RFC 2683 Sep 1999. http://www.ietf.org/rfc/rfc2683

[RFC2177] Leiba, B. "IMAP4 IDLE Command", RFC 2177, June 1997. http://www.ietf.org/rfc/rfc2177

[RFC2818] Rescorla, E. "HTTP over TLS", RFC 2818, May 2000. http://www.ietf.org/rfc/rfc2818

[RFC2822] Resnick, P. "Internet Message Format", RFC 2822, April 2001. http://www.ietf.org/rfc/rfc2822

[RFC2831] Leach, P., and Newman, C. "Using Digest Authentication as a SASL Mechanism", RFC 2831, May 2000. http://www.ietf.org/rfc/rfc2831

[RFC3028] Showalter, T. "Sieve: A Mail Filtering Language", RFC 3028, January 2001. http://www.ietf.org/rfc/rfc3028.txt?number=3028

[RFC3501] Crispin, M. "IMAP4, Internet Message Access Protocol Version 4 rev1", RFC 3501, March 2003. http://www.ietf.org/rfc/rfc3501

[RFC3502] Crispin, M. " Internet Message Access Protocol (IMAP) - MULTIAPPEND Extension", RFC 3502, March 2003. http://www.ietf.org/rfc/rfc3502

[RFC3516] Nerenberg, L. IMAP4 Binary Content Extension , RFC3516, April 2003. http://www.ietf.org/rfc/rfc3516

[UIDPLUS] Crispin, M., "Internet Message Access Protocol (IMAP) - UIDPLUS extension", work in progress, draft-crispin-imap-rfc2359bis-XX.txt.

[WAPWDP] Wireless Datagram Protocol, Version 14-Jun-2001, Wireless Application Protocol WAP-259-WDP- 20010614-aWAP (WDP)

Normative Appendices

A. Implementation Guidelines for Using HTTP with P-IMAP

This appendix describes how HTTP can optionally be used to transfer P-IMAP commands and responses (as an optional HTTP binding). This binding is intended to facilitate the use of P-IMAP in deployments involving a variety of intermediaries, and offers a standardized alternative to de facto proprietary implementations of such a feature.

The need for an optional HTTP binding is driven by the needs of the mobile network operator community (see [MEMAIL], [OMA-ME-RD]), where the reuse of an existing and well-understood technology will allow operators to apply their experience in solving practical deployment issues. Specifically, HTTP allow operators to reuse a similar setup and model that is already used for many other similar and related services, such as certain proprietary push e-mail and synchronization offerings, OMA Data Synchronization, Web services and Web access.

Using HTTP/HTTPS can simplify deployment in a corporate network through the potential use of a reverse proxy to achieve end-to-end encryption. This also has the advantage of not requiring changes to any firewall configurations and reduces the concerns that this often presents to corporation. In general the solution is compatible with any existing firewall. A reverse proxy can also support deployment models that offer roles to other service providers in the value chains, as discussed in [OMA-ME-AD].

The security, encryption and compression capabilities used with HTTP and already implemented in a wide range of existing mobile device, which be also be reused.

Studies have also shown that a persistent HTTP session has usually proven more resilient than an IMAP IDLE over TCP connection over an unreliable bearer such as a GPRS-based mobile network.

The use of HTTP as an underlying protocol for other application protocols has received much attention (see [RFC3205]). In particular, the concern exists that this circumvents firewall security policies. Another concern is the potential misuse or neglect of HTTP semantics by the application protocol that uses HTTP as a substrate.

Note that if the suppression of P-IMAP (or indeed any other application) traffic on HTTP/HTTPS is an issue, firewall administrators can still prevent such passage and this can provide incentives to re-configure firewalls to allow solutions on other transports (e.g. TLS) or offer the HTTP-based solution using another provisioned port (via the P-IMAP XGETPIMAPPREFS (L_HTTP_TUNNEL) instruction). The aim, therefore, is to allow for the use of this solution in the widest possible set of circumstances by codifying a standard way to do so that works with existing, deployed (i.e., HTTP

only) firewalls, while explicitly allowing the possibility of

detecting and filtering such traffic in deployments using the HTTP Content-Type in deployments where this is not permitted.

To use HTTP/HTTPS as the transfer protocol for IMAP commands and responses between the IMAP client and server, the client MUST send an HTTP POST request to the server, and embed IMAP commands (commands to an IMAPv4 Rev1 server or IMAP servers supporting Lemonade extensions) in the body of the request. A server MUST reject a HTTP GET request from the client. The content-type header of the POST request MUST be set to "application/vnd.lemonade". Multiple IMAP commands may be included in one POST request. In general, the HTTP server is expected to preserve session state between HTTP commands to the best of its ability, therefore the client does not need to reauthenticate and reissue a SELECT until it receives an (IMAP) error response showing that it is not authenticated.

In what follows, the term Lemonade client/server is used to refer to a client/server that supports both IMAPv4 Rev1 as well as any LEMONADE extensions.

When the HTTP binding is used, the Lemonade server listens on whatever port has been configured for this.

The following is an example of a possible Lemonade HTTP request:

```
POST /lemonadeServletPPath HTTP/1.1 <CRLF>
Content-Type: application/vnd.lemonade <CRLF>
[other headers]
<CRLF>
(<tag> SP <Lemonade command> <CRLF> | literal )
[(<tag> SP <Lemonade command> <CRLF> | literal )]
```

The Lemonade command MUST be plain text (7bit).

Multiple Lemonade commands MAY be sent on the same request. Thus Lemonade commands must be tagged. The client must be able to deal with recovering from errors when commands are batched. See RFC2442 Batch SMTP for a further discussion. In general, if a command is expected to produce a synchronized literal or continuation request, it MUST be the last command in the batch.

The Content-Type header is the only HTTP headers that MUST be sent to a Lemonade server. Other headers such as Cache-Control MAY be included.

When the Lemonade server sends back a response it is in following format:

```
HTTP/1.1 <HTTP Status Code> <CRLF>
```

```
Content-Type: text/plain <CRLF>
<CRLF>
[<untagged responses>]
<tag> SP <Lemonade Server response> <CRLF>
[<untagged responses>]
<tag> SP <Lemonade Server response> <CRLF>
```

Notes:
The Lemonade Server uses the following HTTP status codes, and what each code indicates is given below:

- 100
  - This indicates the presence of a synchronizing literal or continuation request. The server is waiting for more data from the client (another HTTP request) before continuing. If the HTTP request includes batched commands after the command which generates a continuation request or synchronized literal, the server MUST generate a 5xx request.

- 200
  - This indicates normal execution of the Lemonade commands from an IMAP perspective. The client should further parse the response body to get the tagged responses to the commands and process those accordingly.
- 401
  - This indicates that the execution of the IMAP commands might have been successful, but the session is no longer authenticated. The client should try to reauthenticate to the IMAP server, and then resend the commands.

- 5xx
  - This indicates that at least one command was malformed/protocol level error, or, a command could not complete due to a problem in the IMAP server. In conforming to HTTP semantics, this means the IMAP server responses such as BAD or NO on a tagged response generate a HTTP 500 response code.

When using HTTP to transfer IMAP commands and responses, the client SHOULD utilize built-in features of HTTP to their advantage. For example, the client SHOULD use HTTPS instead of HTTP whenever possible, since HTTPS has built in encryption and MAY have compression capabilities. STARTTLS should not be needed in this case, as it just requires additional overhead without any additional benefit.

HTTP can be used in both in-response and in-band modes. Details about these transport modes are given in the following two subsections.

A.1. Non-Persistent HTTP for In-response Connectivity Mode

If the client uses a traditional HTTP connection (either by establishing a different socket for each HTTP request to the Lemonade server, or by reusing the same socket for all HTTP requests, but sending each request under its own header), it has in-response connectivity to the server. The client can issue as many commands as it would like in one HTTP request to the server, and the server responds by sending back one HTTP response with all the responses to all the commands in the HTTP request. With this connectivity mode, the IDLE command cannot be issued. Other commands that use a continuation response or synchronized literal cannot be issued unless they are the last command in the batch. [LITERAL+] SHOULD be used to eliminate synchronized literals when using APPEND.

In order for the server to identify separate HTTP requests as belonging to the same session, an in-response HTTP client needs to accept cookies. A session-id is passed in the cookie to identify the session.

Example: the headers for a HTTP In-response Response after the client has issued its first HTTP request to the server.

```
   HTTP/1.1 <HTTP Status Code> <CRLF>
   Content-Type: text/plain <CRLF>
   Set-Cookie:JSESSIONID=94571a8530d91e1913bfydafa;
path=/lemonade<CRLF>
   <CRLF>
   [<untagged responses>]
   <tag> SP <Lemnade Server response> <CRLF>
   [[<untagged responses>]
   <tag> SP <Lemonade Server response> <CRLF>]
```

Example: the headers for a HTTP In-response Response after the client has issued its first HTTP request to the server, with the final command generating a continuation request.

```
   HTTP/1.1 100 Continue <CRLF>
   Content-Type: text/plain <CRLF>
   Set-Cookie:JSESSIONID=94571a8530d91e1913bfydafa;
path=/lemonade<CRLF>
   <CRLF>
   [<untagged responses>]
   <tag> SP <Lemnade Server response> <CRLF>
   +continuation-request
```

The client must then save this cookie and send it back to the server with the next request in order for the server to reattach these commands to the same session as the previous commands.

```
POST /lemonadeServletPath HTTP/1.1 <CRLF>
Content-Type: application/vnd.lemonade <CRLF>
Cookie: JSESSIONID=94571a8530d91e1913bfydafa
[other headers]
<CRLF>
<tag> SP <Lemonade command> <CRLF>
[<tag> SP <Lemonade command> <CRLF>]
```

A.2. Using Persistent HTTP/HTTPS + Chunked Transfer Encoding for In-band Connectivity Mode

It is possible to use persistent HTTP or persistent HTTPS plus chunked- transfer-encoding so that the server can instantly send notifications to the client while a 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 Lemonade Server and these headers MUST set the transfer coding to be chunk-encoded [RFC2616, Sec. 3.6.1]. All subsequent client-server requests are written to the open connection, without needing any additional headers negotiations. The server can use this open channel to push events to the client device at any time. In this case, the client SHOULD NOT accept cookies.

The client must send the HTTP headers one time only:

```
POST /lemonadeServletPath HTTP/1.1 <CRLF>
Content-Type: application/vnd.lemonade <CRLF>
Connection: keep-alive <CRLF>
Pragma: no-cache <CRLF>
Transfer-Encoding: chunked <CRLF>
```

The server responds with the following header:

```
HTTP/1.1 <HTTP Status Code> <CRLF>
Cache-Control: private
Keep-Alive: timeout=15, max=100 (or other suitable setting)
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/plain
```

Then the client can send a command anytime it wants with the

following format:

```
   <length of Lemonade command, including bytes in CRLF> <CRLF>
   <tag> SP <Lemonade command> <CRLF>
   <CRLF>
```

And example of an actual client command is:

```
   e <CRLF>
   2 CAPABILITY<CRLF>
   <CRLF>
```

The server responds to each command with as many untagged responses as needed, and one tagged response, where each response is in the format that follows:

```
   <length of a single response, including bytes in CRLF> <CRLF>
   <tagged or untagged response> <CRLF>
   <CRLF>
```

An actual Server response might be:

```
   d5 <CRLF>
   * CAPABILITY IMAP4REV1 AUTH=LOGIN NAMESPACE SORT MULTIAPPEND
LITERAL+ UIDPLUS IDLE XORACLE X-ORACLE-LIST X-ORACLE-COMMENT X-
ORACLE-QUOTA X-ORACLE-PREF X-ORACLE-MOVE X-ORACLE-DELETE ACL X-
ORACLE-PASSWORD LDELIVER LZIP LCONVERT LFILTER LSETPREF LGETPREF
<CRLF>   <CRLF>
   1b <CRLF>
   2 OK CAPABILITY completed <CRLF>
   <CRLF>
```

Note however that the HTTP protocol is in general not meant to be used in such a way. To maintain such an open channel might be a practical challenge to proxies/firewalls, which might not forward the requests chunk by chunk to the server, and meanwhile route responses back to the client chunk by chunk. Consequently the session closes. Chunked transfer encoding requests MAY not be honored by an HTTP server. In cases where such requests are denied, the client should be prepared to use the non-chunked encoding technique from section 2.1

The same challenges exist for TCP session.

In any case, the session can be automatically started again by the client after a lost connection or by the server through out-of-band; after some defined time-out.

A.3. Using HTTP CONNECT

If a HTTP proxy server is available to the client which supports the HTTP CONNECT method, and the IMAP server the user wishes to reach

allows external connections outside the destination network s

firewall, the client may wish to tunnel a regular TCP connection through the HTTP proxy.

See [LUOTONEN] or section 5.2 of [RFC2817] for a detailed description of the technique. Note that HTTP Proxy servers may not honor all CONNECT requests, and may in fact, limit CONNECT requests to a small number of common ports, such as 80, 443, 8080, etc. It is advised that networks wishing to allow their users to use this feature allow clients within their network to CONNECT to ports 25, 143, 587, and 993.

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 4, and is in clear text. P-IMAP 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.

B.2. 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.

P-IMAP 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 the P-IMAP specifications.

Wake-up events consists of the following payload: <emn mailbox="mailat:john.doe@somewhere.com" timestamp="date_format_as_specified_in_[EMN]"></emn>

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 out-of-band.

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.

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 <tag> is  _%$P-IMAP$%_ ,
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
XPROVISION
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 be 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.

B.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 a P-IMAP 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 for P-IMAP 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 P-IMAP port, the message will be routed to the appropriate P-IMAP client application for processing.

Because such mechanisms are network specific, a P-IMAP 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.

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. SSL ensures confidentiality and integrity of the proxied datastream, ensuring that the proxy cannot monitor the content of messages, nor inject commands to modify or corrupt the enterprise email server to corrupt the user's mailbox.

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 IMAP response syntax information coming from the backend server. An attacker who compromises the proxy 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 one 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. Sensitive data may be encrypted and sent via an encrypted literal to the client as detailed in the XENCRYPTED extension. XENCRYPTED (see Section 5.1.5) should be used in 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 unless the key can be derived during a SASL authentication exchange. Otherwise, it must 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 XENCRYPTED. The server should implement 3DES encryption and use the client's password as the key.

It is beyond the scope of this document to define the implementation of transcoding services. In general, it is recommended that they reside within the same domain as the IMAP server, and are not performed by third party services, which may compromise the privacy of the data being transcoded.

D. XCONVERT transcoding parameters

P-IMAP servers MAY support additional transcoding parameters for each media type. All P-IMAP compliant servers MUST minimally support recognition of charset and encoding parameters for text/* mime types, although they may decline to honor some requests. For media types other than text, it is beyond the scope of this document to define conversion parameters. In general however, P-IMAP compliant servers MAY choose to support additional parameters, and if so, they SHOULD follow the OMA STI 1.0 spec [OMA-STI] adopting the same parameter names as defined in second 5.2.4 and above for the most popular image/*, video/*, and audio/* codecs

As an example, in section 5.2.6.2 of [OMA-STI], the parameters "width" and "height" are defined. The following example illustrates how these OMA STI parameters can be used with XCONVERT.

```
      C: a001 UID FETCH 100 BINARY[2.XCONVERT ( IMAGE   JPG  ( WIDTH
 128   HEIGHT   96 ))]
      S: * 2 FETCH (UID 100 BODYPARTSTRUCTURE[2] ("IMAGE" "JPG"
         () NIL NIL "8bit" 4182 NIL NIL NIL) BINARY[2] {4182}
         <this part of a document is a rescaled image in JPG format
with width=128, height=96.>
         )
      S: a001 OK UID FETCH COMPLETED
```

E. Note on XDELIVER, SMTP and Lemonade Profile

A server MAY always rather support SMTP instead of XDELIVER. A client MAY then select to rely on SMTP instead of XDELIVER. This of course may reduce the forward / reply without download capabilities that may be available.

A server MAY also advertise via capability support for Lemonade Profile [LEMONADEPROFILE] or the subset of commands (see [LEMONADEPROFILE] needed to support forward without download. In such case, the client MAY rely on the Lemonade profile forward without download mechanisms.

In general, because of mobile device resource constraints, and corporate firewall and security policies, XDELIVER is easier to deploy for mobile devices, than exposing and restricting SMTP services to mobile devices, especially those devices without VPN functionality.

Non-Normative Appendices

F. 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.

F.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
```

F.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 signally 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.

G. Other Issues

G.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.

G.2. Client event filtering

It is recommended that a P-IMAP 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 the P-IMAP specifications.

A client may keep track of such changes and:
- not transmit them to the server via P-IMAP
- 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] Investigate adding a client to server command to ask the server to stop pushing notifications.
[2] Investigate the use of P-IMAP to trigger / notify other applications.
[3] Integrate/relate more in detail with SIEVE [RFC3028] and related work
[4] Cleanup and/or generalize the effect that the login deviceID has on namespace collisions (e.g. can two VFOLDERs with the same name exist)

In addition, as P-IMAP has now evolved significantly as part of the [LEMONADEPROFILE] and [LEMONADEPROFILEBIS] work as well as OMA MEM activities, we plan to release an update that re-builds the document from the ground up to summarize in a cleaner manner the current normative and informative statements.

Version History

Updates for Release 12

- Fix quick reconnect session terminology that did not mention AUTHENTICATE
- Remove XCOMPOSE in favor of CATENATE+IMAPURL (RFC2192bis extension)
- Incorporate [IMAP-DISC] by reference for state-sync best practices
- Incorporate changes in latest XZIP, XENCRYPTED, XCONVERT, XVFOLDER, and HTTP Binding drafts
- Add [WITHIN] reference
- More text discussing key exchange mechanisms for XENCRYPT and notification encryption
- Remove XFILTER
- Updated References

Updates for Release 11

- Mobile Repository concept altered to be a VFOLDER
- Updates to XCONVERT syntax to track lemonade
- Removal of XFILTER

Updates for Release 10
- Correction in section 5.3.5 regarding the message literals introduced
- Correction of the description of the first example.

Updates for Release 09
- Section 1.3: Clarification that UIDs are the same across repositories.
- Section 3.1: Addition of mention of SIEVE and outband filter management
- Remove section 5.1.1. on MONOINCUID as it tunr sout not to be needed.
- Section 5.3.4: Updates to add support for block encryption to follow the changes in draft-maes-lemonade-lzip-02 with respect to draft-maes-lemonade-lzip-01.
- Section 5.3.14: Addition of details on folder manipulation.
- Appendix A: Updates to follow the changes in draft-maes-lemonade-http-binding-02 with respect to draft-maes-lemonade-http-binding-01:
  - Clarification of binding and motivations
  - Editorial updates and corrections
- Section 5.3.5
  - Updated based on comments received on Lemonade mailing list and from Sun and to follow the changes in draft-maes-lemonade-ldeliver-01 with respect to draft-maes-lemonade-ldeliver-00.
    - New command LCOMPOSE
    - Updated command LDELIVER
- Updated references
- Updates future work

Updates for Release 08
- Updates to follow the changes in draft-maes-lemonade-lconvert-01 with respect to draft-maes-lemonade-lconvert-00.
- Updates to follow the changes in draft-maes-lemonade-lzip-01 with respect to draft-maes-lemonade-lzip-00.
- Updates to follow the changes in draft-maes-lemonade-monoincuid-01 with respect to draft-maes-lemonade-monoincuid-00.
- Editorial fixes
- Author updates
- Clarification of P-IMAP session in section 1.2.3.

Updates for Release 07
- Section 1.2.3: Editorial updates and qualification of SID as a random number.

- Section 1.3: Editorial updates.

- Section 1.4:
  - Editorial updates
  - Addition of edits of messages parts and IMAP URL
  - Additional motivation of XDELIVER
  - Additional details on server driven and client request conversions and adaptation
  - Re-introducing PIM as supported data objects.
- Section 2: Updates of the relationship between P-IMAP and [LEMONADEPROFILE].
- Section 3.1: Update of login details.
- Section 3.1.1: Update on session validity.
- Section 3.2.2: Editorial updates
- Section 3.2.3: Addition of [GSMSMS] and [WAPWDP].
- Section 3.4: Update of the explanation on opening a new session and support of multiple folders
- Section 5.1.1: Addition of monotonically increasing UID and MONOINCUID CAPABILITY feature.
- Section 5.1.3: Correction of client versus server and addition of the declaration of compliance to a P-IMAP revision.
- Section 5.1.4: Update / clarification of the login details consistent with updates in section 3.1 and SID consistent with updates in section 1.2.3.
- New section 5.2 on registering with the server by splitting pas section 5.1.6.
- Section 5.3.1: deprecation of explicit UDP port num and host num address and introduction of NOTIFICATION ADDRESS and PORT.
- Section 5.3.2: Editorial updates and addition of support for [GSMSMS] and [WAPWDP].
- Section 5.3.3: Editorial updates.
- Section 5.3.4: Support for XZIP with XDELIVER.
- Section 5.3.5:
  - Clarification of text / attachment append
  - Additional support of IMAP-URL.
  - Manipulation of address field.
  - Update XDELIVER to address issues with respect to what would be expected based on SMTP (add ENVELOP parameter).
- New section 5.3.6 on IMAP-URL.
- New section 5.3.7 on SMTP and [LEMONADEPROFILE] forward without download mechanisms and add details on XDELIVER.
- Section 5.3.8:
  - Addition of mechanisms to support of document based on [OMA-STI].
  - Mechanism to request DEFAULT conversion.
- New section 6 on P-IMAP:
  - Client security
  - Client updates
  - Client-side behavior
  - Minimum binding interoperability requirements

- Update of Security section.

- Updates of Reference section.
- Appendix A: updates of usage of HTTP / HTTPS binding.
- Appendix B.2: editorial updates
- New Appendix B.3 on usage of [GSMSMS] and [WAPWDP].
- New appendix E.3 on using [OMA-STI] for transcoding with XCONVERT and XUIDCONVERT.
- Clarification of future work item [2] and addition of item [8].
- Corrections of author s names.

Updates for Release 06

[1] Update of the author list

[2] Section 1.4: Update of the details on attachment conversion and PIM

[3] Section 2: Clarification of positioning with respect to other e-mail specifications

[4] Editorial improvement of section 3.1.2.

[5] Improvement of explanations in section 3.2.

[6] New section with recommendations on the connectivity model

[7] Removal of sections 4.2 and 4.3 on Folder events and PIM events.

[2] Editorial improvement in section 5.

[3] Section 5.1.3. The Capability command can now return XPIMAPv1.

[4] Section 5.1.4: Supplying an Email Domain is no longer optional during login to a PIMAP session. Addition of the notion of SESSIONID. Removal of the constrain on 10 digits for phone numbers.

[5] Section 5.1.6: Additional details on how keys are selected, exchanged, updated and used for encryption of out-of-band notifications and in-band messages.

[6] Section 5.2.1: Additional details on XPROVISION of encryption key and UDP notification details when supported. Other information included also such as XFILTER's available.

[7] Section 5.2.2: Addition of support for richer out-of-band notification formats than simply [EMN]. Also, allows user to set the active view and notification filters, as well as the active event filter. Add explicitly UDP as an out-of-band notification mechanism.

[8] Section 5.2.3: Changes in XFilter usage and syntax. Now XFilter is used to name and describe a set of criteria for a filter. The active view and notification filters are now set with XSETPIMAPPREF.

[9] Section 5.2.5: Now, there is only an XDELIVER command, but no UID XDELIVER command. XDELIVER requires both a uid validity and

uid for a message to be forwarded or replied.

[10] Section 5.2.8: Extension of XCONVERT. XCONVERT has been extended to allow the client to alter the server's character set encoding, as well as the transfer encoding (compression). Namely, XCONVERT now provides the ability to request a character set conversion, which may or may not be honored. If it is not honored, default is either the original encoding, or UTF-8. Principally, if the message part the client is requesting conversion of is text, it may attempt to convert it from US-ASCII, ISO-8859, UTF-8, UCS-2/UCS-4, etc to compatible encodings. Also, the client may request a transfer encoding from base64, quoted-printable, or 8-bit clean. Converting from say, base-64 to 8-bit, may result is a savings of up to 33% before compression. Addition also of the details on how device information obtained outside P-IMAP is expected to be used.
[11] Section 5.2.7: Correction of some typos
[12] Security Considerations: Indications that server tools are out of scope of P-IMAP.
[13] Update of references
[14] Update of section A.1 according to section 3.3.
[15] Update of section A.3 to qualify problem raised by intermediaries.
[16] Section B.2 extensions of the out-of-band notification format to beyond [EMN].
[17] Update of Future Work.
[18] Update of version history.
[19] Update of Authors Addresses.

Updates for Release 05
[1] Abstract update to explicitly call out the objective of network transport neutrality
[2] Section 1.2: Add explicitly that the clients changes are transmitted to the server.
[3] Section 1.2.1: Clarifies when new session and State-based-comparison synchronization is used.
[4] Added section 1.2.3.
[5] Clarification by renaming in section 1.3 and after notification / priority filter as notification filter only.
[6] Section 1.4: removed explicit duration before logging out the client + editorial improvements
[7] Section 2: removed explicit assumption that P-IMAP is the mobile profile of Lemonade. This is still to be determined.
[8] Section 3.1: Editorial improvement by removing unnecessary implementation specific sentence on amount of session supported per user and device.
[9] Section 3.1.2: Clarification of the explanation of notification filter.
[10] Section 3.1.3: Clarification of the explanation of event

filter.

[11] Section 3.2.3: Added SIP notification as a possible out-of-band notification mechanism.
[12] Section 3.3: Editorial changes and removal of exact time amount before session expiration.
[13] Section 4.3: Added a clarification on how PIM events can be supported.
[14] Section 5.1.6: Added detail on key exchange via XPROVISION and recommendation not to use XENCRYPTED when STARTTLS is used (and when proxies are not used or an issue).
[15] Section 5.2.2: Added support for SIP notifications.
[16] Section 5.2.4: Remove mandate to use gzip if STARTTLS is used.
[17] Section 5.2.6: Add consideration on using XCONVERT to compress or encrypt.
[18] Security considerations: Add spam as an issue.
[19] Added [CONNECT] to references
[20] Update example syntax for chunked encoding versus long live.
[21] Appendix A.3: Add caveat when using HTTP long live sessions.
[22] Appendix B.1: Clarification of the explanations
[23] Appendix B.2: Clarification of the explanations
[24] Added Appendix E.2.
[25] Additional future work items ([5] and [6])
[26] Updates for Release 05
[27] Update of authors

Updates for Release 04
[1] Section 5.1.1. - Made the UID change condition SHOULD to be consistent with IMAP.
{2} Appendix A.2 added to discuss choosing between HTTPS and HTTP.

Updates for Release 03
[1] Throughout this document - editorial fixes.
[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.
[5] Addition of details about the "attachments forward/reply behavior".
[6] Section 2 has been added to position P-IMAP and the Lemonade Pull Model described in [LEMONADEPROFILE].
[7] Throughout the document - Terminology change to prioritization/notification filter.
[8] Section 3.1 - Reorganization of the text for clarification.
[9] Section 3.2.3 - Additional motivation for using out-of-band notification
[10] Change of title for section 4.1
[11] Section 5.1.1 - Change of normative statement from SHOULD to

MUST, back to SHOULD

[12] Clarifications in section 5.1.3 and 5.1.5.
[13] Section 5.2.3 - Extension of the type of out-of-band notification channels.
[14] Section 5.2.3 - Fixes of examples: Changes of N to P.
[15] Section 5.2.4 - Clarification of XZIP normative statements depending on the selected binding for P-IMAP.
[16] Mention of HTTPS under security considerations
[17] Reference updates to reflect [LEMONADEPROFILE].
[18] Appendix A.1 - Fixes of some HTTP/HTTPS Request/Response Formats.
[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 - 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 - took out message events, which is now described in new section 3.
[3] Section 1.4 - removed attachments behavior
[4] Section 3 - new section containing event payloads
[5] Old section 3.1.3 - removed this section on forwarded flags
[6] Old section 3.1.4 - added resync, folder, and session untagged response syntax
[7] Old section 3.1.5 - UID becomes should instead of must requirement
[8] Old section 3.1.7 - took out resync, which is now in login section
[9] New section 4.1.6 - a new section concerning untagged XENCRYPTED responses in place of untagged FETCH responses.
[10] Old 3.2.1 - 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
  [c] valid values for some preferences are given in XPROVISION
  [d] XGETPIMAPPREF -> XGETPIMAPPREFS
  [e] defined XGETPIMAPPREFS untagged response
[12] Old 3.2.3 - defined XFILTER untagged response
[13] Old 3.2.4 - dropped this section on XTERSE
[14] Old 3.2.6 - changed syntax so only V & N can be given for get.
[15] Old 3.2.7
  [a] XUIDCONVERT -> UID CONVERT
  [b] added untagged response syntax

[16] Security Considerations section - added in that there are additional security considerations when the server is implemented through a proxy on a distrusted operator network.
[17] Appendix B.2 - changed example where client gets events in response to a login command (instead of noop)
[18] Appendix C - new appendix to cover security issues for proxy-based deployments of P-IMAP.
[19] 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 - P-IMAP Session/Login
[b] Added examples for "First Login" and "Login after Logout"
[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 out-of-band 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
[11] New Appendix C

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 all who have contributed key insight and extensively reviewed several versions of the P-IMAP concepts and early P-IMAP specifications.

Special thanks to authors of past versions including Vida Ha.

A special thanks is addressed to several employees of Nokia and Openwave.

A special thanks to editors and reviewers of some of the derived drafts whose input was reflected back in the P-IMAP draft including especially D. Cridland, N. Mitra (Ericsson), A. Melnikov (Isode), C. Newman (Sun), M. Pozefsky (IBM), A. Srivastava (Sun).

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

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
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 66006688 EXT 3137

Zhao Lijun

CMCC R&D
ADD: 53A, Xibianmennei Ave.,Xuanwu District,
Beijing,100053
China
TEL:.8610
.66006688.3041

Dwayne Bennett
Consilient
P.O. Box 2172
St. John's, NL A1C 6E6
Canada
Tel: +1 709 576 1706
E-mail: bennett@consilient.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgement

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