Abstract

The Push-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 that unlike a standard IMAPv4 Rev1 server, which relies on the client to constantly initiate contact to ask for state changes, the P-IMAP server can push crucial changes to a client. In addition, P-IMAP contains extensions for email filter management, message delivery, and maintaining up-to-date personal information. Bindings to specific transport are explicitly defined.
Conventions used in this document

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

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

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocol(s) it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for a protocol is said to be "unconditionally compliant" to that protocol; one that satisfies all the MUST level requirements but not all the SHOULD level requirements is said to be "conditionally compliant."

When describing the general syntax, some definitions are omitted as they are defined in [RFC3501, Sec. 9].

Table of Contents

1. Introduction................................................... 3
   1.1. The Poll Model vs. the Push Model........................ 4
   1.2. Synchronization Techniques............................... 4
       1.2.1. State-Comparison-Based Synchronization............ 4
       1.2.2. Event-based Synchronization........................ 5
   1.3. The Server-Side Filtering in P-IMAP.................... 6
   1.4. Extra Functionality in P-IMAP........................... 7
2. The P-IMAP Design.............................................. 8
   2.1. Implementing Filters...................................... 8
   2.2. Connectivity Models..................................... 9
       2.2.1. In-Response Connectivity.......................... 10
       2.2.2. Inband Connectivity................................ 10
       2.2.3. Outband Connectivity............................... 10
   2.3. Keeping the Client In Sync with the Mobile Mailbox.... 11
3. Interactions between the P-IMAP Client and P-IMAP Server...... 11
   3.1. Revisions to IMAPv4 Rev1 Behavior....................... 13
       3.1.1. UID.............................................. 13
       3.1.2. Mobile Mailbox.................................... 13
       3.1.3. Forwarded Flag.................................... 14
       3.1.4. The CAPABILITY Command............................ 14
       3.1.5. P-IMAP Session/Login............................... 14
       3.1.6. IDLE............................................. 15
   3.2. P-IMAP Extension Commands and Responses.................. 15
       3.2.1. XPROVISION...................................... 15
       3.2.2. XSETPREF & XGETPREFS.............................. 16
       3.2.3. XFILTER.......................................... 18
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. 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 outband connectivity. Appendix A provides a normative binding to HTTP.

The organization of this document is as follows. The rest of this section introduces the core enhancements of P-IMAP so the reader can gain an understanding of the concepts that drive this design. Section 2 discusses actual design decisions for P-IMAP. Section 3 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 Appendix A, B, & C, cover some extra guidelines needed to support implementation level issues. The non-normative appendices, Appendix D & E, provide interesting use cases and examples.

1.1. The Poll Model vs. the Push Model

Today, most of the existing email clients have a polling model, where the end user is notified of changes to an email account only after his/her email client asks the server, called polling. 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, web-mail clients and WAP-based mobile clients.

Another way to achieve synchronization is for the email server to initiate conversation with the client when a crucial change to an email occurs, which is the push model. When important events happen to a user’s mailbox, 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.

1.2. Synchronization Techniques

In addition to how a client receives changes to an email account, there are many techniques for determining what those changes are. In this section, two techniques are presented that aim to keep a client device in sync with a given mailbox, meaning the set of folders and the set of emails in each folder on the client device is the same as that given mailbox.

1.2.1. State-Comparison-Based Synchronization

IMAPv4 Rev1 requires clients to use a state-comparison-based synchronization technique to be in sync with a mailbox. This technique requires the client to ask the server for information regarding all the folders and all the messages in each folder stored on the server. The client must then compute the difference between the state of the server and the client device state, and make all
necessary changes so that the device becomes in sync with the server. An example of the interaction between the client and server in the IMAPv4 Rev1 protocol for performing a state-comparison-based sync follows.

First, the client must retrieve the folders from the server.

- **C:** A002 LSUB "" ""
- **S:** * LSUB () "/" "Drafts"
- **S:** * LSUB () "/" "Friends"
- **S:** * LSUB () "/" "INBOX"
- **S:** A002 OK LSUB completed

The client must compare its folders with the responses of the command above. If it does not have a folder, it must create that folder on the client device. If there is a folder on the device that is not in any of these responses, then the client must delete that folder.

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:** Â more untagged responses with information about the folder
- **S:** A003 OK SELECT completed
- **C:** A004 FETCH 1:* FLAGS
- **S:** * 1 FETCH (FLAGS (\Answered) UID 120)
- **S:** * 2 FETCH (FLAGS (\Seen) UID 121)
- **S:** Â 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. Also, it should remove any emails on the client device not in this list. After performing these operations, the client is in sync with the server.

### 1.2.2. Event-based Synchronization

Another technique besides state-comparison-based synchronization is event-based synchronization for keeping the client device in sync with the server, but which requires that the client already be in sync with the server at some earlier point. In the P-IMAP protocol, the client must perform a state-comparison-based sync the first time it logs into a session, 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 while connected.

In event-based synchronization, the server keeps track of what changes have occurred to the mailbox that are not yet reflected on the client device. Such a change is called an event. When the client finishes processing all events since the last time it was in sync with the server, it is again in sync with the server. Event-based synchronization is particularly effective when the server can push events to the client for immediate processing. In this case, there are likely to only a small number of events the client needs to process at one time. Below are examples of events that can occur and how the client must respond to the event.

Now the client can receive the three following untagged responses from the server:

The client receives an EXISTS event from the server:

S: * 501 EXISTS

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

C: A02 FETCH 501 (ALL BODY[])
S: * 501 FETCH Â
S: A02 OK FETCH completed

The client receives an EXPUNGED event from the server.

S: * 25 EXPUNGED

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

The client receives an untagged FETCH event from the server, which can contain just FLAG information if the event is regarding an old message, or possibly other information if the event is regarding a new message.

S: * 101 FETCH (FLAGS (\Seen \Deleted))
The client saves the information contained in this response accurately in the client device.

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 separate their emails into different sets that the server should handle differently. All end users have a complete mailbox, which includes all their email messages that are stored on a server. The end user may only want some of these messages actually downloaded to their client device, which are all included in their mobile mailbox. Some of the messages in the mobile mailbox are of high importance, and the end user would
like to be notified immediately if there are crucial changes to them. Such messages are in the push mailbox. All three mailboxes have the same set of folders.

Formally, a mailbox consists of a set of folders, and each folder has both a name and a set of messages associated with it. While the three mailboxes all have folders with the same name, there may be different messages in them. The complete mailbox consists of all folders of an end user, and all the associated emails for each of those folders. To derive the mobile mailbox, P-IMAP allows the user to specify exactly one view filter for each folder. A set of email folders with all the same names as those of the complete mailbox and the resulting set of email messages in each folder that passes the view filters make up the mobile mailbox. In addition, there is a second layer of filtering, called notification filters, which are applied to folders of the mobile mailbox. Again, there is exactly one notification filter per folder. All the email folders and the resulting set of messages in each folder of the mobile mailbox that passes the notification filters are the push mailbox.

From this point forth, an event in this document refers to only and all changes to the mobile mailbox. These events can be further separated into message events and folder events, as well as poll events and push events. In this document, message events refer to changes to only messages in the mobile mailbox, while folder events refer to any change to any email folder. Poll events are events that the client must poll for, whereas push events are events that are sent immediately to the client. Every client device can define one event filter that informs the server what kinds of message events it would like to be notified of, like the arrival of new messages for example. The client uses the view filters and notification filters to specify which messages it thinks are important, and the event filter to specify which events regarding those messages it would like pushed onto the client device.

Whenever a change occurs to the server, it is first determined whether this change concerns a message or a folder. If it concerns a folder, it is a folder event and all folder events are push events. If the change concerns a message that passes the view filters, it is a message event. Otherwise, this change does not concern the mobile mailbox and thus is not considered an event for the purposes of P-IMAP. Next, if a message event concerns a message that passed the notification filters and that event passes the event filter, it is a pushed message event. Otherwise, if the message event concerns a message that does not pass the notification filters or does not pass the event filter, it is a polled message event.

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

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

[2] Sending emails - The P-IMAP server can be used to send email, thus eliminating the need to connect to a separate SMTP server for such functionality.

[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 mailbox while the client is disconnected, which it can then send to the client upon reconnection.

[4] Longer periods of inactivity tolerated - A P-IMAP server should wait at least 1 hour before logging out an inactive mobile client.

[5] Attachments conversion - The P-IMAP server can convert attachments to other formats to be viewed on a mobile device.

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

2. 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 mailbox. A P-IMAP client may fetch the contents of the mailbox 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.

2.1. Implementing Filters

A P-IMAP server should support multiple mobile devices for each email user, and should allow each device to have its own event filter and set of view filters and notification filters. A mobile client
connects to the P-IMAP server by supplying its LOGIN information, and then must inform the server of this mobile client’s device ID, which is some unique identifier for the client device. The server and client should agree on what convention to use for this ID, and it could be a hash of IMEI. If no device ID is given, then a regular IMAP session is initiated instead of a P-IMAP session. The LOGIN information is used to specify a mailbox, while the device ID is needed to specify the mobile client. Associated with the mailbox and device ID is exactly one view filter and exactly one notification filter for each folder of the mailbox. These filters are saved and thus persist across P-IMAP sessions.

The syntax for defining an event filter is ALL, NONE, or NEW. ALL means that all message events concerning messages of the push mailbox will be sent to the client, such as if the message becomes seen or deleted. NONE means that no events should be pushed to the client. NEW means that only events that concern new messages arriving to the push mailbox should be pushed to the client. This one event filter applies for all folders.

View filters and notification filters are used to filter out email messages with only certain criteria. The syntax for defining a view filter or notification filter includes NONE or 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. The ALL search criteria, when used alone, means that every email event satisfies the criteria, while NONE means no email event does. By default, view filters are set to ALL, while notification filters are set to NONE. This means that the mobile mailbox includes all the messages in the complete mailbox, but none are pushed to the client, which degrades to the IMAPv4 Rev1 model. All and only IMAPv4 Rev1 filters on properties of an email message that are immutable can be supported by a P-IMAP server, such as search criteria concerning the text of a message. That way, whether an email passes a given filter or not always remains constant.

When a P-IMAP session is open, the client can set and change the filters. Whenever a view filter is modified, the client needs to perform a state-comparison-based sync to keep in sync with the mobile mailbox. The client does not need to do anything after it resets a notification filter or event filter, instead the server should then only send out notifications that correspond to the most up-to-date filters.

2.2. Connectivity Models

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

2.2.1. 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.

An in-response connection occurs in two situations:

1. HTTP/HTTPS binding
   - Server Requires: HTTP/HTTPS listener for IMAPv4
   - Client Requires: HTTP/HTTPS client with IMAPv4 processing
2. TCP Binding
   - Server Requires: IMAPv4
   - Client Requires: IMAPv4 + no IDLE

2.2.2. Inband Connectivity

The inband binding scenario corresponds to a reliable push model. In this case the server pushes events to the client whenever they occur. To do so, it must have a reliable means of communication with the client open, and the client should be ready to accept such notifications. The client may optionally issue a command and receive a response in response to an event. 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.

An inband connection occurs in the following situations:

1. TCP Binding, Always connected, IDLE
   - Server Requires: IMAPv4 + IDLE
   - Client Requires: IMAPv4 + IDLE, constant TCP connection
2. Any other persistent two-way connection
   - Server Requires: IMAPv4 + IDLE
   - Client Requires: IMAPv4 + IDLE, constant connection

2.2.3. Outband Connectivity

The outband binding scenario corresponds to an unreliable push model. In this case the server pushes events to the client whenever they occur, to the best of its ability. To do so, it should be able to send messages to the client. However, the outband channel can possibly lose and reorder messages, and there are no timing guarantees. Examples of out-band channels include SMS, JMS, WAP
Push, and UDP. As in the inband scenario, the client may optionally open a P-IMAP session and send a command as a result of receiving an event.

Outband connectivity occurs in the following situations:

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

2.3. Keeping the Client In Sync with the Mobile Mailbox

When a client device ever opens a new P-IMAP session, it must perform a state-comparison-based sync with the email server so that its state is the same as the mobile mailbox. Since the client has no way of directly detecting only changes to the mailbox since the last login, it needs to retrieve information about every message in the mobile mailbox and calculate the changes itself. After that point, the client can use event-based synchronization to keep the device in sync.

The P-IMAP server can issue a session to extend the life of a P-IMAP session. Until the session is expired, the server must log all events that occur while a client is offline. This way, if the client temporarily loses a connection, it does not have to worry about missing any events and needing to perform another state-comparison-based sync. A client does have the option though to prematurely end a session by issuing a LOGOUT command. Additionally, P-IMAP clients can remain inactive for at least an hour without being logged off the server.

3. Interactions between the P-IMAP Client and P-IMAP Server

A P-IMAP server must support all IMAPv4 Rev1 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 3.1. 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 notifications notify the
client when there are changes to messages that match an end user’s view filters and notification filters, as well as any changes to a client’s email folders. The syntax defined in this section is an abstract syntax, and payloads may vary according to the communication mechanism used. The normative appendix of this document describes some specific payloads.

The format for presenting commands is defined as follows:

<COMMAND NAME>

<Command Description - contains an explanation of the command>

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

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

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

Responses: <server responses for this command>

Result: <possible result that comes after the responses. This usually indicates the status of the execution of a particular command. Possible values are:

- OK if the execution was successful
- BAD for unknown commands, or when arguments syntax is incorrect
- NO when argument semantics are incorrect, or when command processing fails
- BYE when internal system or network error happens and processing cannot continue>

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

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

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

To connect to a P-IMAP server, the client must first follow the procedure for establishing an IMAP session. The client starts out in NOT AUTHENTICATED state and issues a LOGIN command with either a valid P-IMAP ID appended to the mailbox name. Firing this command enters the client into a P-IMAP session, where it can use all the P-IMAP extension commands, as opposed to a regular IMAP session, which will return errors to all P-IMAP defined extensions other than XTERSE, XZIP, XDELIVER, and XPROVISION.

The server responds to XPROVISION commands by returning any service specific parameters of the server, such as encryption keys or port numbers. XTERSE can be used to turn the terse mode of the server on or off. When in terse mode, the server sends an abbreviated untagged response. The XZIP command can be used to zip the response to another command. XDELIVER allows the client to send an email message through this server, instead of having to connect with an SMTP server.

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

### 3.1. 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 IMAPv4 Rev1 commands. A compliant P-IMAP server must implement all the commands in IMAPv4 Rev1, with these revisions. The IMAPv4 Rev1 syntax on commands and responses are found in sections 6 and 7 in [RFC3501]. This rest of this section defines any additional modifications to the IMAP commands that a P-IMAP server must implement to be compliant.

#### 3.1.1. UID

The UID of email messages MUST not change across sessions.

#### 3.1.2. Mobile Mailbox

In a P-IMAP session, the client can only access messages in the mobile mailbox. This affects the messages returned by FETCH, UID FETCH, etc. Message sequence numbers reflect the relative position of messages within the given folders of the mobile mailbox, so the
message sequence number of an email while logged in to P-IMAP may also differ from IMAP. When returning information about the email account, only messages in the mobile mailbox are taken into account.

3.1.3. Forwarded Flag

P-IMAP also supports a \Forwarded flag, to denote when a message has been forwarded. All IMAP commands and syntax should be modified to support this new flag, including STORE to allow users to set the \Forwarded flag, FETCH to include the status of the \Forwarded tag in the responses, XPSEARCH to allow a user to specify forwarded and unforwarded as a search criteria, etc.

3.1.4. The CAPABILITY Command

The CAPABILITY command is defined in RFC3501, section 6.1.1. The client sends a CAPABILITY command so it can query the server to find out what commands it supports. In RFC3501, the IMAP server is allowed to specify additional capabilities not included in that specification. A P-IMAP server conforms to that requirement, and must list what P-IMAP commands it supports. Minimally, this must include IDLE, XZIP, and XDELIVER.

capability_cmd =  tag SP "CAPABILITY"

Valid States: NOT AUTHENTICATED, AUTHENTICATED, SELECTED, or LOGOUT

Responses: REQUIRED untagged response: CAPABILITY

Result: OK - capability completed
BAD - command unknown or arguments invalid

Example: A P-IMAP server that implements all P-IMAP commands.
C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=LOGIN IDLE XCONVERT XFILTER XPSEARCH XTERSE XZIP XDELIVER XPROVISION X-PIMAP-PREF
S: a001 OK CAPABILITY completed

Example: A minimal P-IMAP server.
C: a001 CAPABILITY
S: * CAPABILITY IMAP4rev1 IDLE AUTH=LOGIN IDLE XZIP XDELIVER
S: a001 OK CAPABILITY completed

3.1.5. Forwarded Flag

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

The P-IMAP server will automatically try to resume a previous session for this client. If this is the case, the server informs the client
of the state of the server. If that state is SELECTED, the server also tells the client what the selected folder is. 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.

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

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

3.1.6. IDLE

The server should implement the IDLE command from RFC 2177. When the client issues this command, the server can push changes to a mailbox to the client. The server may replace the EXISTS/RECENT message with an untagged FETCH command as specified in Section 3.2.2.

3.2. P-IMAP Extension Commands and Responses

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

3.2.1. XPROVISION

The XPROVISION command is used to allow a device to obtain service specific parameters of the server. The server parameters XPROVISION can return include a base-64 encoded cryptographic key, "ENC_KEY", to encrypt notifications sent over an out-band channel (see Appendix B.2 for details.) 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
Example: The client issues an XPROVISION command. The server responds by returning the encryption key, modes, and channels supported by P-IMAP. Note the syntax for returning parameters.

C: A002 XPROVISION
S: * XPROVISION ENC_KEY Xsm9E07hYjlfvWtL7wg+ng==
S: * XPROVISION XNOTIFY UDP=3000 SMS
S: A002 OK XPROVISION completed

3.2.2. XSETPREF & XGETPREFS

The XSETPREF command allows a user to define certain configuration parameters, while the XGETPREF command allows a user to retrieve the configuration values. Any server that implements these commands must respond with X-PIMAP-PREF as one of the capabilities in response to a CAPABILITY command. These parameters affect how outband notifications are sent to the client, as well as the format for sending new event notifications. They are listed below and their names start with PIMAP_to identify them as P-IMAP parameters:

1. PIMAP_OUTBAND_TYPE_SMS - the number or email address to send SMS messages to the client
2. PIMAP_OUTBAND_TYPE_JMS - the number or email address to send JMS messages to the client
3. PIMAP_OUTBAND_CHANNEL - the channel to send outband notifications, either SMS, JMS, or NONE. When NONE, the P-IMAP server does not send the client any outband notifications.
4. PIMAP_INBAND_NEW_FORMAT - the FETCH parameters to automatically send to the client when there is a new message and there is a valid P-IMAP session, or NONE. If NONE, the server sends the client a traditional EXISTS message when a new message arrives in the mailbox. Otherwise, in place of the EXISTS message, the server sends a untagged FETCH response with the given information.
5. PIMAP_OUTBAND_NEW_FORMAT - the FETCH parameters to automatically send to the client when there is a new message and there is no valid P-IMAP session, or NONE. If NONE, the server sends the client a traditional EXISTS message when a new message arrives in the mailbox. Otherwise, in place of the EXISTS message, the server sends a untagged FETCH response with the given information.

\begin{verbatim}
xgetpref_cmd = tag SP "XGETPREF" SP "(" pimap_pref_list ")"
pimap_pref_list = pimap_pref [SP pimap_pref_list]
pimap_pref = (PIMAP_OUTBAND_TYPE_SMS / PIMAP_OUTBAND_TYPE_JMS / PIMAP_OUTBAND_CHANNEL / PIMAP_INBAND_NEW_FORMAT / PIMAP_OUTBAND_NEW_FORMAT)
\end{verbatim}
Valid States:  AUTHENTICATED or SELECTED
Responses:  REQUIRED untagged XGETPREF response with the value of the requested parameter.
Result:  OK - notify completed
         NO - notify failure: can’t alter notification (e.g. permission denied, not supported)
         BAD - command unknown or arguments invalid

Example: The client wished to know the current outband notification method it has set up. It sends an XGETPREF command.
         C: A003 XGETPREF (PIMAP_OUTBAND_CHANNEL)
         S: * XGETPREF (PIMAP_OUTBAND_CHANNEL SMS)
         S: A003 OK XGETPREF completed

xsetpref_cmd = tag SP "XSETPREF" ("PIMAP_OUTBAND_TYPE_SMS" SP sms_channel) / ("PIMAP_OUTBAND_TYPE_JMS" SP jms_channel) /
              ("PIMAP_OUTBAND_CHANNEL" SP ("SMS"/"JMS"/"NONE")) /
              ("PIMAP_INBAND_NEW_FORMAT" SP fetch_criteria) /
              ("PIMAP_OUTBAND_NEW_FORMAT" SP fetch_criteria)

Valid States:  AUTHENTICATED or SELECTED
Responses:  No specific responses.
Result:  OK - notify completed
         NO - notify failure: can’t alter notification (e.g. permission denied, not supported)
         BAD - command unknown or arguments invalid

Example: The client sets up its SMS notification channel and then selects that it wants SMS messages sent to the device.
         C: A002 XSETPREF PIMAP_OUTBAND_TYPE_SMS 13335559999
         S: A002 OK XSETPREF completed
         C: A003 XSETPREF PIMAP_OUTBAND_CHANNEL SMS
         S: A003 OK XSETPREF completed

Example: The client sets the inband NEW format to be ALL and BODY[], meaning it wants the server to automatically send it all the headers and body for any new message.
         C: A002 XSETPREF PIMAP_INBAND_NEW_FORMAT_(ALL BODY[])
         S: A002 OK XSETPREF PIMAP_INBAND_NEW_FORMAT completed

From now on, whenever a new message arrives in a mailbox 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 Â <headers and body part>

Example: The client asks to have all the headers of a new message delivered to it when there is no valid P-IMAP session.
When the client has closed a P-IMAP session, it will receive a notification with an untagged FETCH response with ALL header information for the message.

When the client has closed a P-IMAP session, it will receive a notification with an untagged FETCH response with ALL header information for the message.

S: * 60 FETCH <headers>

3.2.3. XFILTER

The XFILTER command allows users to set up view filters and notification filters. XFILTER can be fired as long when the state is AUTHENTICATED or SELECTED. The first argument to this command is the folder that that filter should be applied to, or "ALL" for all folders. Next the user specifies "V", "N", or "B" to set either a view filter, notification filter, or view filter and notification filter. The last argument is either "GET" to retrieve the filter, "NONE" to not allow any messages to pass through the filter, or a combination of search criteria as defined for the SEARCH command of IMAP, in Section 6.4.4 and 7.2.5 of RFC 3501. The ALL search criteria, when used alone, means that every email message satisfies the criteria.

By default, view filters are set to ALL, while notification filters are set to NONE. This means that the mobile mailbox includes all the messages in the complete mailbox, but none are pushed to the client, which is the IMAPv4 Rev1 model.

Exactly one view filter and one notification filter is associated with each folder. When a new view filter or notification filter is created, it replaces the previous filter for that folder. When a view filter is modified, the client needs to perform a state-comparison-based sync on the client in order for the device to be in sync with the mobile mailbox. The server always sends only notifications that correspond to the most up-to-date view filters and notification filters. All filters persist across P-IMAP sessions; once set, a filter on a folder applies until the user changes it.


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

Example: The client asks for the view filter for all the folders.
C: A001 XFILTER ALL V GET
S: * XFILTER ~/INBOX V ALL
S: * XFILTER ~/TRASH V NONE
S: A001 OK XFILTER completed

Example: Stop notifications on a particular device, fired while in AUTHENTICATED mode.
C: A001 XFILTER ALL N NONE
S: A001 OK XFILTER ALL N NONE completed

3.2.4. XTERSE

The XTERSE command puts the server in or out of terse mode. This command can be fired at any time, even if the client has not been identified as a P-IMAP client. In terse mode, the tagged responses do not contain any extra text, such as repeating the name of the command. Formally, this changes the following IMAPv4 definition for resp-text:

\[
\text{resp-text} = \[\text{"[" resp-text-code "]"}\]
\]

\[
\text{xterse\_cmd} = \text{tag SP "XTERSE" SP ("ON" / "OFF")}
\]

Valid States: NOT AUTHENTICATED, AUTHENTICATED, SELECTED, or LOGOUT
Responses: no specific responses
Result: OK - terse mode activated/deactivated
BAD - invalid arguments

Example: Turning terse mode ON
C: A001 XTERSE ON
S: A001 OK
C: A002 XFETCH BODY[HEADER.FIELDS (FROM SUBJECT)]
S: * 1 (57)
S: From: "John Doe" <John.Doe@foo.com>
S: Subject: test
S:
S:)
S: * 2 (58)
S: From: "John Doe" <John.Doe@foo.com>
S: Subject: test2
S:
S:)
S: A002 OK

Example: turning terse mode OFF
C: A001 XTERSE OFF
S: A001 OK XTERSE OFF
C: A002 XFETCH BODY[HEADER.FIELDS (FROM SUBJECT)]
S: * 1 XFETCH BODY[HEADER.FIELDS (FROM SUBJECT)] {57}
S: From: "John Doe" <John.Doe@foo.com>
S: Subject: test
S:
S: * 2 XFETCH BODY[HEADER.FIELDS (FROM SUBJECT)] {58}
S: From: "John Doe" <John.Doe@foo.com>
S: Subject: test2
S:
S:)
S:)
S: A002 OK XFETCH completed.

3.2.5. XZIP

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

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

Example: Zipping the response to a FETCH command.
C: A001 XZIP A002 FETCH 1:* ALL
S: * {10933843723} Â [zipped response to FETCH command]Â CRLF
S: A001 OK XZIP completed
When the client unzips the body of the response to the FETCH command it gets:
* 1 FETCH Â
Â
S: A002 OK FETCH completed

3.2.6. XDELIVER

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

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

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

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

Example: reply message
C: A001 XDELIVER R 203 Y {299}
  Date: Mon, 7 Feb 1994 21:52:25 -0800 (PST)
  From: Fred Foobar <foobar@Blurdybloop.COM>
  Subject: afternoon meeting
  To: mooch@owatagu.siam.edu
Message-Id: <B27397-0100000@Blurdybloop.COM> MIME-Version: 1.0
Content-Type: TEXT/PLAIN; CHARSET=US-ASCII

Hello Joe, do you think we can meet at 3:30 tomorrow?

A reply message for message 203 is prepared and includes all original attachments.
S: A001 OK XDELIVER completed

3.2.7. XCONVERT & XUIDCONVERT

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

xconvert_cmd = tag SP "XCONVERT" message-sequence-number SP part-id
   SP "as" SP mime-type "/" subtype
Valid States: SELECTED
Responses: untagged responses: XCONVERT
Result: OK - xconvert completed
       NO - xconvert error: can’t perform the command
       BAD - command unknown or arguments invalid

Example: The client fetches an attachment in the message with the message sequence number of 120 in the Inbox and asks to have that attachment converted to pdf format.
   C: a001 XCONVERT 120 BODY[3] as application/pdf
   S: * 2 XCONVERT <this part of a document in pdf format.>
   S: a001 OK XCONVERT COMPLETED

xuidconvert_cmd = tag SP "XUIDCONVERT" uid SP part-id SP "as"
   SP mime-type "/" subtype
Valid States: SELECTED
Responses: untagged responses: XUIDCONVERT
Result: OK - xuidconvert completed
       NO - xuidconvert error: can’t perform the command
       BAD - command unknown or arguments invalid

Example: The client fetches an attachment in the message with UID 120 (and message sequence number 2) in the Inbox and asks to have that attachment converted to pdf format.
   C: a001 XUIDCONVERT 120 BODY[3] as application/pdf
   S: * 2 XCONVERT <this part of a document in pdf format.>
   S: a001 OK XUIDCONVERT COMPLETED

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

xpsearch_cmd = tag SP "XPSEARCH" [SP "CHARSET" SP astring] 1*(SP search-key)
Valid States: SELECTED
Extension to: UID SEARCH command [RFC 3501, Sec. 6.4.4]
Responses: no specific responses
Result: OK - psearch created
NO - can't create the folder or incorrect query
BAD - invalid arguments

Example: create a persistent search for all messages from "John" since Jun, 1st 2003. The newly created folder name is called "from_john"
  C: A001 XPSEARCH from_john FLAGGED SINCE 1-Jun-2003 FROM "John"
  S: A001 OK XPSEARCH completed

4. Formal Syntax

The following syntax specification uses the augmented Backus-Naur Form (BNF) as described in RFC-2234 [i].

Security Considerations

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

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

References
http://www.3gpp2.org/Public_html/specs/X.S0016-311_MMS_Stage3.pdf

http://www.forum.nokia.com/smsforum/main/1,6566,1_2_5_1,00.html

[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

http://www.ietf.org/rfc/rfc2119.txt

http://www.ietf.org/rfc/rfc2234

http://www.ietf.org/rfc/rfc2420

http://www.ietf.org/rfc/rfc2616.txt

http://www.ietf.org/rfc/rfc2617.txt


http://www.isi.edu/in-notes/rfc2177.txt

http://www.ietf.org/rfc/rfc2818.txt


http://www.ietf.org/rfc/rfc3501
Normative Appendices

A. Implementation Guidelines for a P-IMAP Session

A.1. HTTP/HTTPS Request/Response Format

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

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

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

- The P-IMAP command should be plain text (7bit) and should follow what is specified in section 3 of this document.
- Multiple P-IMAP commands may be sent on the same request. Thus P-IMAP commands must be tagged.
- These are the only HTTP headers required to be sent to the P-IMAP servers.

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

```
HTTP/1.1 <HTTP Status Code> <CRLF>
Content-Type: application/vnd.pimap <CRLF>
Content-Length: <size of response string> <CRLF>
Content-Encoding: gzip <CRLF>
<tag> <P-IMAP Server response> <CRLF>
        [...] <P-IMAP Server response> <CRLF>
```  

Notes:
The first line is the HTTP status code of the command execution. This could be one of the following:
- 200 - all commands succeeded.
- 400 - at least one command syntax is not correct, or command syntax is correct but semantics is not correct, or the current state is not correct.
- 401 - client is not authenticated and needs to send authentication information to proceed.
- 500 - at least one command caused internal server error, meaning the P-IMAP Server failed to execute the command.

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

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

B. Event Payload

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

The event payload for a P-IMAP session follows the general format explained in Section 3, and is in clear text.

B.2. Outband Channel Event Payload

One suggested payload for notifications is that suggested by the OMA, see [OMA-EN].

If a notification payload contains sensitive information, it should be encrypted.

C. Attachments Handling

C.1. Retrieving Attachments
When the client device retrieves a message that contains attachments, the whole attachment should not be downloaded by default. If the end user requests a particular attachment it is then fetched and retrieved from the P-IMAP server. The IMAP Implementation Recommendation document [RFC2683, Sec. 3.2.1.4] also says that clients should not normally download the full contents of the message, but do so selectively.

IMAPv4 Rev1 has mechanisms on how to download specifics parts of a message and these must be used in order to retrieve the message attachments. For complete information, refer to [RFC3501, Sec 6.4.5].

Example: The client retrieves the message body structure of a two-part message consisting of a text and a BASE64-encoded text attachment:

```
C: A001 XFETCH 203 BODYSTRUCTURE
S: * 203 XFETCH (BODYSTRUCTURE ((("TEXT" "PLAIN" ("CHARSET" "US-ASCII") NIL NIL "7BIT" 1152 23) ("TEXT" "PLAIN" ("CHARSET" "US-ASCII" "NAME" "cc.diff") "<960723163407.20117h@cac.washington.edu>" "Compiler diff" "BASE64" 4554 73) "MIXED"))
S: A001 XFETCH Completed
```

Example: The client now asks for the attachment on the previous message, and the attachment is part 2.

```
C: A001 XFETCH 203 BODY[2]
S: * XFETCH 203 (BODY[2] {4554} BFIEhUTUwgUFVCTElDICItLy9XM0MvL0RURCBIVE1MI ...)
S: A001 XFETCH Completed
```

### C.2. Attachments Conversion

A client can ask to have one part of one message converted into a specified format. In the example below, a client asks for part 3 of the message with UID 120 of the Inbox to be converted into pdf format.

```
C: a001 XFETCH 120 BODY[3] as application/pdf
S: * 2 XFETCH <part of a document in pdf format.>
S: a001 OK XFETCH COMPLETED
```

When composing a reply/forward message, the P-IMAP server must attach the original message’s text body. If there are attachments in the original message, these are added only if specified in the XDELIVER command – see Section 3.2.6 for full syntax and examples.
D. Use Cases

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

D.1. State Comparison-Based Sync

Each time a client logs into a new P-IMAP session, it must perform a state comparison-based sync. To synchronize with the server, the client needs to fetch all the new messages, and all the flags of the old messages.

The client has N messages in its mailbox 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 Ä
- S: Ä <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 Ä ))
- S: Ä <info for 2 through n-1>
- S: * n FETCH (UID 3589 FLAGS (\Seen Ä ))
- S: A02 OK FETCH completed

D.2. Event-Based Sync

During a P-IMAP session, the client will receive events in the form of untagged EXISTS, RECENT, EXPUNGED, 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 below example 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 Ä
   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 EXPUNGED response from the server:
   S: * 25 EXPUNGED
The client deletes this message from the client device, as it has been removed permanently from the mailbox. The client can remain in IDLE mode.

When the client receives an untagged FETCH response from the server, either signal a flag change to an old message or a new message:
   S: * 101 FETCH (MODSEQ (357) FLAGS (\Seen \Deleted))
The client updates the information on the device for this message appropriately.

E. Other Issues

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

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

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.

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

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

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

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

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

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

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

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