

IMAP Action Queue

February 1, 2006

1 About This Document

This document reflects the design for a feature that is in progress. A first pass at the IMAP transaction queue was implemented in Windows Eudora 7.0. As expected, this initial implementation was incomplete and the plan is to refine this feature based on feedback from the initial implementation. Some portions of this document have been implemented and some portions reflect possible future directions. Portions that reflect possible future directions are in [blue](#).

Note also that this document was to serve as a design document for Mac Eudora as well, though this feature was never implemented in Mac Eudora. There are sections of this document that address issues specific to Mac implementation but these sections are mostly incomplete.

2 Introduction

Under the new IMAP model, IMAP actions are performed locally, then queued and performed on the server at a later time, blurring the distinction between online and offline. In order to perform this we need to have a queue of transactions that are to be executed. Items might remain in the queue for only a short period (in the case of an action that is almost immediately performed on the server) or might be written to a file, read in later and then have the actions performed (in the case where Eudora is offline for a period of time). The transaction queue might potentially have items added to the end even as items are being taken from the beginning and performed on the server (in the case where Eudora is replaying logged actions and the user is continuing to perform other actions).

3 Queuing Actions

All IMAP user operations are placed in a queue and are performed on the server when that item from the queue is processed. If the action can be performed locally then it is done locally before it is placed on the queue. Actions that cannot be done locally are simply placed in the queue and performed.

3.1 Actions to Queue

A detailed discussion about which user actions should be queued is contained in the “Offline IMAP” design document and is not repeated here. Items in the below lists which appear in *italics* are actions that cannot be performed locally.

The following IMAP user operations are queued:

Mailbox Operations:

- Create
- Rename
- Delete
- *Refresh Mailbox List*
- *Resync a Mailbox*
- *Expunge a Mailbox*

Message Operations:

- Mark as read/unread (\Seen)
- Mark as replied (\Answered)
- Mark as deleted/undeleted (\Deleted)
- Mark as flagged (\Flagged) – For future use.
- Mark as draft (\Draft) – For future use.
- Copy (COPY or APPEND)
- Transfer (COPY or APPEND + STORE \Deleted)
- Delete (Fancy)
- Undelete (Fancy)
- Junk
- Unjunk
- *Download One or More Parts of a Message*
- *Copy Undownloaded Message Across Accounts*

For purposes of optimization it makes sense to use a single status change action to represent any possible change in status using a data field to indicate the changes to the specific flags.

It would also make sense to represent delete/undelete (in fancy trash mode) as transfer actions since that is essentially what those operations are. As of this writing, junking and unjunking are also represented as transfer actions. Junking and unjunking actually involve changing the message's junk score and unjunking a message should be followed up by running the incoming filters on the message, so a future revision should have separate actions to represent those operations.

3.2 Adding Actions to the Queue

For an action that can be done locally the action is performed locally first and if that succeeds the action is queued. For an action that cannot be performed locally the item is queued immediately. In this latter case, if Eudora is online at the time then the action needs to be expedited in the queue so that the user does not need to wait for those actions to be executed.

Each account maintains its own queue. Currently each queue contains all action items associated with that account in the order in which they should be executed.

As a possible future improvement we could be smarter about maintaining queues. One possibility would be for each account to keep a list of actions on a per mailbox basis, pointing to the first action on a given mailbox. Another possibility would be to be smarter about how items are added to the queue. When an action is added we could look for a previous action that should precede the new action. For example, if we are adding an action that acts on a specific mailbox then we could look for the previous action which acts on that mailbox. If such an action is found, that action will be set to point to the new action. The rationale for this is that the newly added action depends in some way on the previous action and failure of the previous action may have an impact on the newly added action. Actions on specific messages also depend on the mailboxes in which those messages are located. In this case, does a message action depend on just the previous action on that message or also on the previous action on the mailbox containing the message? In this model, perhaps the queue consists of the head items for each list and each item has a pointer to the next item in that list.

4 Storing Actions on Disk

In addition to the in-memory queue of actions we record actions to disk. Actions are stored to disk immediately after they are queued to prevent data loss.

4.1 Queue File

Each account stores its action queue in a single file stored in the top level of its data directory. The file is named "actionqueue.dat" and contains an XML representation of the in memory action queue that the account maintains. An action queue is marked as

needing written on any change – currently when an item is added or removed – and on each idle cycle Eudora will write the queue to disk if necessary.

4.2 Action Files

A second possible model would be to each action stored in an individual file. Files would be grouped in directories corresponding to the various IMAP accounts. If an action involves more than one account its file will be stored in the directory for the source account (e.g., for a cross account transfer, the account containing the source mailbox for the message to be transferred).

4.3 File Format

The IMAP queue file is a text file containing XML representing individual actions. The general file format is:

```
<IMAPQueueData>
  <DataFormatVersion>1</DataFormatVersion>
  <Action>
    ... Action Specific Data ...
  </Action>
  <Action>
    ... Action Specific Data ...
  </Action>
  ...
</IMAPQueueData>
```

The root element of the XML file is <IMAPQueueData>. This element has two child elements:

- <DataFormatVersion> indicates the version number of this document (not strictly needed for XML documents, but included anyway).
- <Action> indicates that the data contained within this element represents an IMAP action item.

The <Action> element has the following child elements:

- <ActionType> – the type of IMAP action. The following are valid types:
 - <status> – message status change.
 - <delete> – message deletion.
 - <undelete> –message undeletion.
 - <transfersame> – message transfer across mailboxes on the same account.
 - <copysame> – message copy across mailboxes on the same account.
 - <transfercross> – message transfer across mailboxes on different accounts.
 - <copycross> – message copy across mailboxes on different accounts.
 - <transfertolocal> – message transfer to a local mailbox
 - <copytolocal> – message copy to a local mailbox
 - <transferfromlocal> – message transfer from a local mailbox

- <copyfromlocal> – message copy from a local mailbox
- <creatembox> – mailbox creation.
- <renamembox> – mailbox renaming.
- <deletembox> – mailbox deletion.
- <refresh> – refresh of a mailbox tree.
- <expunge> – expunge of messages in a mailbox.
- <resync> – resync of a mailbox or mailbox tree.
- <download> – download of one or more parts of a message.
- <Mailbox> – name of mailbox being acted on (or mailbox the messages are in).
- <Delimiter> – delimiter used in the mailbox name.
- <Messages> – message ID's of messages being acted on.
- <Status> – the attribute to be set.
- <StatusSet> – 1 if the attribute is to be set, 0 if reset.
- <DestAccount> – for cross account transfer, account of the destination mailbox.
- <DestMailboxes> – for transfer or copy, name of destination mailbox.
- <DestDelimiter> – delimiter used in destination mailbox name.
- <NewName> – for mailbox rename, new name of mailbox.
- <Attach> – for download action, 1 if attachment should be download, 0 if not.
- <OnlyIfNot> – for download action, 1 if only what hasn't been downloaded
- <ClearCache> – for download action, 1 if message cache should be cleared
- <InvalidateCache> – for download action, 1 if cache should be invalidated
- <NormalCheck> – for resync action, 1 if normal (background) mail check
- <BitFlags> – for resync action, bit flags to be passed to mail check method
- <Notifier> – for download action, 1 if a notifier should be used on mail check.
- <DownloadOnly> – for download action, 1 if doing a forced manual resync

Here is a sample action queue file:

```
<IMAPQueueData>
  <DataFormatVersion>1</DataFormatVersion>
  <Action>
    <ActionType>status</ActionType>
    <Mailbox>INBOX</Mailbox>
    <Delimiter>/</Delimiter>
    <Messages>464709870</Messages>
    <Status>1</Status>
    <StatusSet>1</StatusSet>
  </Action>
  <Action>
    <ActionType>transfersame</ActionType>
    <Mailbox>INBOX</Mailbox>
    <Delimiter>/</Delimiter>
    <Messages>464709890,464709889</Messages>
    <DestAccount>993788650</DestAccount>
    <DestMailbox>A</DestMailbox>
    <DestDelimiter>/</DestDelimiter>
  </Action>
</IMAPQueueData>
```

5 Replaying Actions

At an appropriate time actions are taken from the queue in order and executed online.

5.1 Reading the Queue File

Eudora will only ever need to read the queue file when it is launched. Once the data has been read into the in memory copy of the queue Eudora acts on the data in the memory copy and modifies the queue file as necessary to reflect the data in the queue.

5.2 Replaying

While Eudora is online during idle time items are taken from the action queue one at a time and server commands are issued to perform those actions online. Actions are removed from the queue and replayed in the order they are found on the queue.

5.3 Threading

Actions are replayed in threads. Since the actions consist of communication between the client and server and do not involve changing any local data there is no problem performing any kind of cached action in a thread. If an action fails we have to change local data in order to bring the client and server back to the same state. This kind of post processing happens in the main thread after the secondary thread completes.

5.3.1 Windows Eudora Threading

Windows Eudora creates one network socket per IMAP mailbox and that socket can be shared by multiple threads acting on that mailbox. The code is written to provide appropriate locking of the connection object to prevent potential conflict. Many IMAP operations are coded with the assumption that they will be performed in their own thread. It was decided for our first pass to work with this assumption so each action being replayed in given its own thread. Once a thread for a given action completes we are free to launch the next thread to perform the next action.

If two or more consecutive actions have nothing in common (they operate on completely different mailboxes and messages) then there is no reason why these actions could not be performed by multiple threads acting at the same time. This is an optimization that could be considered for later.

Since the connection object and the socket will be shared the only overhead in giving each action its own thread will be the creation and launching of a new thread. Still, there might be reason to consider breaking some of these assumptions to allow for the creation

of fewer multiple-purpose threads. This is another optimization that could be considered for later.

5.3.2 Mac Eudora Threading

I presently don't know enough about this topic to attempt any meaningful comment.

5.4 Poor Connectivity

If Eudora is able to detect that it is online but has limited bandwidth we could adjust the order in which items are replayed to allow quicker replaying of actions which might have a higher priority to the user. For example, a user might want mailbox resyncs and message downloads to have priority over actions like expunges, or perhaps even over message transfers. This is a feature for future consideration.

6 Optimizations

We could certainly replay each action in the log one at a time in order and get the desired results. However, with some extra analysis of the actions to be reconciled we might be able to optimize the replay by reducing the number of actions to be reconciled and by reducing the number of commands to be issued to the server. We must bear in mind that opportunities to optimize are likely to be the exception and not the rule so we must be careful that the cost of calculating the optimizations does not outweigh the benefit.

In all likelihood little or no attempt at optimization will be made unless the unoptimized replaying proves to be in need of optimization.

6.1 Performing Commands on Ranges

The primary (and perhaps only) way we would optimize replay would be to combine actions that perform identical commands on different messages into a single action that operates on a range of messages. For example, if a user individually transfers multiple messages from one mailbox to another, Eudora might be able to combine multiple transfers into a single command that operates on a range of messages.

The following operations could potentially be performed in ranges:

- Copying of multiple messages. If multiple sequential messages in a mailbox are copied from one mailbox into the same destination mailbox then a single COPY command could be issued to copy those messages.
- Changing message data for multiple messages. If multiple sequential messages in a mailbox have the same data changes performed then a single STORE command could be issued to change the data for those messages.
- Transferring of multiple messages. Since transfer is simply a COPY followed by a STORE \Deleted we could also perform transfers on ranges.
- Junking/Unjunking of multiple messages. Since junking requires a separate action item this is a separate optimization than transferring.

6.2 Removing Unnecessary Actions

Since a user could potentially perform multiple actions on the same mailbox or message, one potential optimization would be to eliminate unnecessary actions and to coalesce multiple actions on the same object into one action. For example, if a message is transferred from mailbox A to mailbox B then later transferred from mailbox B to mailbox C we could then just transfer the message from mailbox A to mailbox C.

There are five kinds of actions that might potentially be minimized:

- Multiple transfers of the same message. If the same message is transferred multiple times we can skip the intermediate transfers and simply perform a single transfer from the original source mailbox to the final destination mailbox.

- A round trip transfer of a message. If the same message is transferred multiple times and ends up in the same mailbox as it began in then none of the transfers need to be performed.
- Multiple status changes for the same message. If the same message has its status changed multiple times we can perform a single status change call containing all of the status changes.
- Creation and renaming of a mailbox. If a mailbox is created then renamed one or more times we can simply create the mailbox with the name it was given by the final rename.
- Creation and deletion of a mailbox with no intermediate actions. If a mailbox is created and then deleted before any messages are added to that mailbox then we can simply skip creating the mailbox in the first place. If the mailbox is renamed one or more times between creation and deletion then we can still skip the creation. If any messages are transferred or copied into the mailbox between the creation and deletion then we cannot skip any of these steps.

It is worth noting that these cases are likely to be the exception, not the normal case. Multiple status changes are the most likely to occur: a message is marked as read, then replied, then possibly deleted.

Implementing this form of optimization would almost certainly be a list ditch effort if performance is a considerable problem.

6.3 Timing of Optimizing

We should be most concerned about optimizing in the case where Eudora is offline and a potentially large number of actions are being added to the queue. If Eudora is online and actions are being temporarily queued we would not expect to accumulate as many actions in the queue.

For the case where Eudora is offline, the optimal time to be performing any optimizations would be while Eudora is still offline as items are being added to the queue. If we attempt to perform the optimizations when Eudora goes online and the actions are being replayed then we potentially run into the case where we spend more time looking for optimizations than we save by optimizing replay.

6.4 Performing the Optimizing

After discussion it was decided that the preferred optimization strategy would be to optimize items as they are added to the list and continue optimizing any actions that are combined via the optimization.

In this model when an item is added to the list it is marked as not being optimized. During idle time we could process the queue. We would find the first item that is marked

as not optimized. Starting at the beginning of the queue we would look for an item which could be combined with the unoptimized item. For example, a transfer action could be combined with another transfer action that has the same source and destination mailboxes. In this case, the two items would be combined and the resulting action would be marked as unoptimized. On the next optimization pass, the process would be repeated with the newly combined item being found as unoptimized and would attempt to optimize it. If no items are marked as not optimized then there are no optimizations left to perform.

7 Failure Handling

We need to be dynamic in our handling of failures that occur while reconciling local changes on the server.

7.1 Temporary Failures

Some failures should be considered temporary and should result in that action (and any subsequent actions that depend on that action) remaining in the queue for a future attempt at reconciliation.

The following failures should be considered temporary:

- Action fails because mailbox is locked.
- Action fails because of authentication failure.

For example, if a message transfer action fails because the destination mailbox is locked then that transfer action (and any subsequent actions that depend on that transfer) will remain in the queue to be retried at a future time.

7.2 Permanent Failures

Some failures by nature will be permanent and should result in that action (and any subsequent actions that depend on that action) being removed from the queue.

The following failures should be considered permanent:

- Action fails because source or destination mailbox cannot be found.
- Action fails because message cannot be found.

For example, if a message transfer action fails because the destination mailbox cannot be found then any actions on the destination mailbox or on messages within that mailbox must fail on a permanent basis. At this point those actions can be removed from the queue.

7.3 Correctable Failures

Some failures are not fatal but would require Eudora to make adjustments.

The following failures should be dealt with by making adjustments to the actions:

- User creates mailbox offline while an identical mailbox is created online via a different client. There are a couple ways we could handle this. We could simply use the existing mailbox with the same name or we could generate a new unique name for the newly created mailbox.

7.4 Failures and Future Actions

If an action cannot be performed we will need to remember data about why the action failed so we can avoid attempting other actions that will fail for the same reason. If an action fails because of a problem with the mailbox (it cannot be found or is locked) then we need to remember that mailbox. If a message action cannot be performed because the message cannot be found we need to remember that message. If a subsequent action is found that refers to a problem mailbox or message no attempt will be made to perform that action at that time.

If the failure results in an orphaned message then that message is moved to the orphaned message mailbox.

If the failure is a temporary one, the action will remain in the queue in its original location and will be retried at a future time. If the failure is a permanent one, the action is removed from the queue entirely and will never be retried.

8 Lost Log File

Eudora will provide a setting to indicate whether or not Eudora thinks a log file exists and if the file is missing when Eudora goes online we will warn the user of the missing log file. The user could be presented with two options: go online and synchronize back to the state on the server or stay offline for the moment.

If the user chooses to remain offline then no reconciliation will be attempted. This is a dubious state to be in because we would not want the user to continue performing actions offline since their state is already out of sync with the server. However, this state could allow the user to perform actions to safeguard against losing any data. They might copy messages to local mailboxes or back up portions of their hard disk. If this state is considered too risky we could simply not present this option to the user. If we choose to allow a user to enter this state and the user attempts any actions that involve changing the local data we should warn the user of the potential danger.

When the user chooses to go online, Eudora should immediately refresh the mailbox list (assuming offline mailbox actions are allowed) so we know the local mailbox list matches that on the server. After that, as each mailbox is synched Eudora will update the local data to match the data on the server.