I assume you are willing to deal with a completely unpolished interface, so I'll tell you what the current truth is about the Lafite on <LispCore>Library> and in the latest <LispCore>Next>Full.sysout. What follows might be viewed as the start of an implementor's manual. It is not, of course, anything I'd be willing to publish as a programmer's manual, but perhaps your experience will help us define what one might wish the interface really to be. I'm quite willing to add small things as you see the need, though there is little hope for any major projects in the near term. So let me know how this matches your needs.

My other reluctance, of course, is that internals, as these all are, are subject to change without notice, so I'd appreciate feedback on what parts you would like to see solidified into a supported interface.

There are almost no direct functional interfaces to what you want, but with some record definitions I think you can get most of it. You thus need to LOADCOMP(LAFITE). In general, you should probably avoid replacing record fields, as there are usually interactions; if you need to replace a field for which there is no programmatic interface, let me know. Fetching fields should be fine. The important records are two datatypes, `MAILFOLDER` and `LAFITEMSG`.

`MAILFOLDER` holds assorted facts and state about a single mail folder. The window property `MAILFOLDER` hangs off each Lafite browser window. Here are the most interesting fields:

`FULLFOLDERNAME`, `VERSIONLESSFOLDERNAME`, `SHORTFOLDERNAME` -- various forms of the folder's name.

`FOLDERSTREAM` -- a stream on the file behind the folder, if it's open.

`MESSAGEDESCRIPTORS` -- an array of `LAFITEMSG` objects, corresponding to messages 1 thru `#OFMESSAGES`. Thus `(ELT (fetch (MAILFOLDER MESSAGEDESCRIPTORS) of` folder`) n)` returns the nth `LAFITEMSG` in folder. This field is only valid while a browser is open on the folder.

`FOLDERLOCK` -- a monitor lock that should be acquired if you want to perform any operations on the folder.

`FIRSTSELECTEDMESSAGE`, `LASTSELECTEDMESSAGE` -- number of the first and last messages currently selected (equal if only one selected).

`BROWSERWINDOW`, `BROWSERMENU`, `BROWSERMENUWINDOW`, `BROWSERPROMPTWINDOW` -- pointers to parts of the browser window for this folder. `NIL` if no browser open on folder.

(`\LAFITE.GETMAILFOLDER` folderName) -- this function returns a `MAILFOLDER` object for folderName, which should be either a full file name or a full name sans version number. If the result has a non-`NIL` `BROWSERWINDOW` field, there is a browser currently open on the folder, else not.

(`\LAFITE.OPEN.FOLDER` folder access recog) -- returns a stream on the file behind folder. access and recog are as with `OPENSTREAM`. This is the standard way to get the stream (rather than the `FOLDERSTREAM` field), unless you're sure the file's open for the right access.

Each message in a browser is described by a `LAFITEMSG` record. Its most interesting fields:

`PARSED?` -- true if message has been parsed. Set by `LAFITE.PARSE.MESSAGE.FOR.TOC`, which also distributes the standard parse into the fields `FROM`, `SUBJECT`, `DATE`.

`DELETED?` -- true if message is marked deleted.

`SEEN?` -- true if message is marked seen (`NIL` => "?" in browser).

`MARKCHAR` -- character code of message's mark.

`MARKSCHANGED?` -- true if `MARKCHAR` has changed since last update.

`SELECTED?` -- true if message is currently selected.

`MSGFROMMEP` -- true if message is from the logged in user.

-- ordinal position of message in folder, from 1.

`BEGIN` -- byte position in file of start of message, which is always the internal header that starts "*start*".

`MESSAGELENGTH` -- length of message, including internal header.

STAMPLENGTH -- number of bytes between the BEGIN position and actual start of message.

START -- access field: byte position of the start of the actual message, viz., BEGIN+STAMPLENGTH.

END -- access field: byte position of the end of the message, viz., BEGIN+MESSAGELENGTH.

DATE -- date of message, actually just the 6 character string that appears in toc (so you can't give it to IDATE).

FROM, SUBJECT, TO -- From, Subject, To fields of the message. The TO field is usually only parsed out if MSGFROMMEP has been observed to be true.

Now, how to use all these:

The value of the variable LAFITE.AFTER.GETMAIL.FN, if non-NIL, is a function to call immediately after Lafite retrieves mail from a server. It is called with two arguments, the MAILFOLDER, and a list of LAFITEMSG objects describing the new messages (which have already been appended to MAILFOLDER:MESSAGEDESCRIPTORS, securely stored in the file, but not yet displayed in the browser window). The messages have not yet been parsed, I believe. The function is called while in control of MAILFOLDER's monitorlock.

Parsing messages. The function LAFITE.PARSE.MESSAGE.FOR.TOC sets all the fields in a LAFITEMSG that the toc needs, but you probably want more fields than that, so will need to do your own parsing. Lafite has a simple-minded but reasonably efficient parser that looks for header fields in a message (Field name followed by colon) and does something with them. It requires that you build a "parse table" describing what you want it to do. This is a list of lists, one for each field you want to parse. Each sublist is in the form ("FieldString" ResultFn . ResultArgs). Then "compile" the parse table by calling (LAFITE.MAKE.PARSE.TABLE table). The resulting compiled table can then be passed to this function:

(LAFITE.PARSE.HEADER stream parseTable start end onceOnly) -- Stream is the stream where the message lives -- the folder's stream for message in a file, or a tedit stream for a message you're sending. parseTable is the compiled parse table. Start and End are byte pointers in stream (normally the START and END fields of a LAFITEMSG). Parsing automatically stops at the end of the header. onceOnly is true if you want parsing to end as soon as any one field has been successfully parsed.

The parse proceeds as follows: the parser starts at byte position start and scans each line in the header, seeing if the field name matches (case-insensitively) a field in the parse table. If a match is found (the characters match up thru the colon), then the parser skips over any white space, then calls the ResultFn indicated in the parse table with two arguments, the stream (now positioned at the first non-blank character after the colon) and the ResultArgs from the table (this lets you multiplex one parsing function for several purposes). The function's job is to parse the rest of the field, leaving the stream positioned after the terminating <eol>, and to set freely the variable PARSERESULT as it wishes. PARSERESULT is what LAFITE.PARSE.HEADER ultimately returns.

You may not need to follow that too closely. Lafite's standard use of the parser is to make PARSERESULT an alist; the following functions are interesting as resultFn's, or to help them. You can look at the variable LA.FULLPARSEFIELDS and LA.TOCFIELDS for examples. The former is used for a "full parse", such as the Answer command needs; the latter parse three fields out for the toc.

(LAFITE.READ.TO.EOL stream) -- Reads the rest of the field, returning one long string. You should probably use this as the basis for all result fns, as it takes care of continuation lines and makes sure the stream is left properly at the end of the field.

(LAFITE.READ.LINE.FOR.TOC stream Args) -- Reads the field as one big string and stashes it on PARSERESULT indexed by (CAR Args). If all of your resultFn's are this one, then your parse result ends up something like ((field1 "contents1")(field2 "contents2")...). The string is limited in length to 255 characters (a toc restriction); if longer, it is truncated.

(`LAFITE.READ.NAME.FIELD` stream Args) -- This is almost the same, but no string length restriction, and it recognizes that some fields may have multiple occurrences (bad form, but allowable). If a field occurs twice, its alist entry has more than one string in its cdr: ((field1 "contents1a" "contents1b") ...).

(`\GV.PARSERECIPIENTS` field registry internalFlg editWindow) -- This function takes raw strings from the output of `LAFITE.PARSE.HEADER` and produces a list of individual recipient names. Field is a string or list of strings, such as `LAFITE.READ.NAME.FIELD` might produce during the header parse. registry is the grapevine registry default for names without registries (defaults to `DEFAULTREGISTRY`). If internalFlg is true, the names returned are in internal form (name . reg); if `NIL`, the names are strings of the form "name.reg". editWindow is for error messages (true when sending mail).

(`LA.SETDIFFERENCE` x y) -- like `LDIFFERENCE`, but treats x and y specifically as sets of strings, case-insensitively.

(`LA.REMOVEDUPLICATES` x) -- removes duplicates from x, treating x as a set of strings, case-insensitively.

Message manipulation. Most of these update the browser window appropriately.

(`SELECTMESSAGE` msgDescriptor mailFolder) -- "selects" msgDescriptor, a `LAFITEMSG` object, in mailFolder. Does not deselect anything.

(`UNSELECTALLMESSAGES` mailFolder) -- unselects all messages.

(`MARKMESSAGE` msgDescriptor mailFolder mark) -- changes msgDescriptor’s mark character to mark (a charcode).

(`SEENMESSAGE` msgDescriptor mailFolder) -- mark msgDescriptor "seen".

(`DELETEMESSAGE` msgDescriptor mailFolder) -- mark msgDescriptor "deleted".

(`UNDELETEMESSAGE` msgDescriptor mailFolder) -- undelete msgDescriptor.

Most of the browser commands consist of a function \LAFITE.nameOfCommand, some of which spawn processes named \LAFITE.nameOfCommand.PROC. Here’s the one for MoveTo:

(`\LAFITE.MOVETO.PROC` window mailFolder destinationFolder item menu shortOutput oldFileP) -- moves all selected messages from mailFolder to destinationFolder. Greys out item of menu while it’s working (the `BROWSERMENU` field of mailFolder gets you the menu). When finished, if exactly one message was moved, and it was the one currently displayed, does the "display after delete" action. shortOutput is the folder’s name (in short form, but it really doesn’t matter). oldFileP is true if the caller believes, but has not yet verified, that destinationFolder describes an existing folder (this is true if the folder was on the folder menu). window is mailFolder’s `BROWSERWINDOW`.

As I said, "unpolished". You probably really want the not yet defined subfunction of that: (\LAFITE.MOVE.MESSAGES mailFolder msgDescriptors destinationFolder).