Indexed Chat History and Infinite Conversation Scrollback

A proposal to improve the conversation history experience in Instantbird as part of GSoC 2014.

Personal Details

- Name: Nihanth Subramanya
- Email: nhnt11@gmail.com
- Telephone: +918095417331
- Other contact methods: IRC: nhnt11 | GTalk: nhnt11
- Country of residence: India
- Timezone: Indian Standard Time (UTC+05:30)
- Primary language: English

Project Proposal

Note: I'll be using the term "session" to refer to the messages exchanged from when a conversation is opened until the time it's closed.

Current Problems

This project aims to solve, broadly, two problems:

- Chat logs are ridiculously difficult to search.
- Conversation history is limited to the current session - all context is lost when a conversation is closed (you have to view the log).

The first problem is pretty self-explanatory. We need indexed logs.

The second problem, while appearing simple, has some points worth noting:

- An extremely long conversation results in every message in the session being stored - this takes memory, and makes searching, as well as restoring on-hold conversations, slow to the point of several seconds of unresponsiveness. (Ref. Bug 953744, Bug 955358)
- Loss of context means that you can no longer search the conversation without opening the log.
- Not completely related, but the mechanisms in which messages are appended to the conversation browser, and by which they can be selected, could use improvement – ref. Bug 953743, Bug 953754.

Desirable Behavior

- One should be able to search logs without knowing the date on which the message to be

found was sent/received.

- Conversations should be pre-loaded with some amount of context on opening. Scrolling up further should load further context from the log. Scrolling back down should result in messages being removed to save memory.
- Searching within a conversation should work even if the message/text to be found isn't already displayed – i.e. the log should be searched and then the search results should be shown (see point 1. h. under Implementation Details)
- A conversation when put on hold should discard excess context (keep only an arbitrary number of messages).

Implementation Details

Research I've done so far:

- I've looked at the convbrowser and message style code to get a decent understanding of how messages are handled in the conversation view.
- As an experiment to learn more about addition of messages to the convbrowser document, I wrote a small patch that adds copies of an arbitrary message to the top when you scroll up.

The following is a list of milestones with details of the work that needs to be done for each of them.

1. Indexed, searchable logs.
    a. The logging code will be extended to use an SQLite database indexed for full-text search. The database will be updated asynchronously as new log entries are created.
    b. The existing log sweeping code in the stats service will be moved to a separate module, so that it can be used for constructing the database as well as the stats cache from existing JSON logs. The database will work as an index for the JSON logs - querying it for a search string, date, etc. will return the paths of the files which contain the matching messages (and their positions in them).
    c. An API will be written to query this database. Methods will be implemented to obtain logs for a given date, buddy/chat room, and search string. There will be a way to tell if the returned message(s) is(are) the last one(s) available.
    d. The log viewer will be made to use this API to display and search logs. When a search string is entered, the date tree (left sidebar) will be filtered to show only dates whose logs match the string. When a date is clicked, the conversation displayed will highlight matches.
    e. The stats service code will be adapted to work with the database – both for log sweeping as well as for storing stats and getting the list of filtered conversations (assuming database lookup performance is satisfactory). This would mean decreasing memory usage and obsoleting the current JSON file.
2. Dynamic addition/removal of context in conversation's DOM.
    a. As an initial milestone before true infinite scrollback, the convbrowser will be made to remove excess context messages from the DOM when not visible (when the conversation is put on hold, or after the user scrolls down and stays at the bottom for an arbitrary amount of time – i.e. "garbage collection" of messages) and display them

only when necessary (when scrolling up). “Excess context” would be messages that are older than some arbitrary amount of time. I propose to never garbage collect unread messages, but if we decide against that, the position of the last unread message will need to be remembered in order to correctly display the unread ruler.

b. The convbrowser will listen for “scroll” events on its contentDocument, combined with a timer for garbage collection. Whether the user has scrolled up far enough that we need to prepend messages, as well as the number of messages to prepend, can be calculated from the document’s body’s scrollTop and scrollHeight properties. Prepending a fixed arbitrary number (n) of messages is also an option but may result in different amounts of scrollback (in pixels) being added depending on how long the messages were (this would be especially noticeable if there were several huge messages, compared to n single-word ones).
c. The convbrowser will store a buffer of messages to which garbage collected messages will be pushed, and from which messages to be prepended will be popped (when scrolling up). When restoring a conversation from hold that has a lot of unread messages, we will not want to display them all (in this case we’ll need to remember the last unread message, refer point a.).
d. The current code for inserting a message into a conversation in imThemes.jsm (get/insertHtmlForMessage()) and convbrowser.xml (displayMessage()) mostly relies on the assumption that the message is being appended at the bottom. These methods will be adapted (or new methods will be written) to easily insert messages at the top (“prepend”).
e. Messages will be prepended and removed in groups. This eliminates the problem of having to add/remove individual messages to/from an existing message group, since adding/removing a message from a group affects the markup of its neighboring messages. For the edge case where a conversation comprises entirely of one group, the group will be split into multiple groups each containing an arbitrary number of messages (maybe 25 or 50).
f. The bubbles theme may need to be updated to ensure features like time bubbles don’t break.
g. Searching as well as section scrolling (and other navigation mechanisms like Ctrl+Home/End) will require a way to prepend messages until a given message is displayed. This will be implemented.
h. Searching will need to be changed since all the messages to be searched may not already have been added to the browser. The buffer as well as the messages in the DOM will need to be filtered. To avoid traversing the DOM, we may want to keep an array of all messages as suggested in bug 953743, and redefine the buffer as a subset of this array (we’d store the index of the buffer’s last message in the main array rather than maintaining two arrays). Results will then be displayed incrementally (refer previous point).
i. Some search results may require prepending a huge number of messages in between to display them. This is expensive, and currently I propose showing a button or link to view results in the log viewer for such cases.

3. Infinite scrollback
    a. The convbrowser will finally be adapted to use the database querying API to get messages to prepend. The buffer will be kept updated with an arbitrary number of messages to be prepended. As messages are popped from it, it will be "refilled" asynchronously from the logs. Messages getting pushed into it (when removed from the DOM) will cause messages to get shifted from the other end.
    b. Scrolling quickly may cause the buffer to deplete on slower machines (accessing the disk to query the database may not be as fast as desirable). After an arbitrary interval of time (say 100ms) a throbber will be shown until messages are retrieved (or we find out that there are none left).
    c. Section scroll will need to be adapted to work with this model.
4. Special cases/miscellaneous
    a. When a conversation is newly opened, an arbitrary amount of context will be loaded from the logs.
    b. For users who have logs disabled, a mechanism to fall back to the implementation in point 2 will be devised.
    c. There will be edge cases related to text selection, for example if the users does a select all and then copies. How many messages would be "selected"? An interesting possible solution to this problem is something suggested in bug 955349 - "[We] could bring up a preview window of the output, which allowed [the user] to [...] format the result."
    d. An interesting way to jump to arbitrary points in the conversation (when searching, or section scrolling) would be to display the required message with context, and dynamically add messages on scrolling *down*. To jump back to the end, the user could use section scroll (upon which the previous messages would be removed from DOM again). This may not be entirely intuitive, however - not all users may know about section scrolling. Another thing to consider is how to handle the resulting anomalous scrollbar behavior.
    e. Message actions will need to work for old messages loaded from the log.

Schedule of Deliverables

Rather than organizing deliverables by date, the following is a breakup of the implementation into individually-landable milestones/patches. Any patches that have the potential to blow things up can be pref'd off before landing if we deem it necessary at the time.
Parts 1. and 2. are separate projects and will be completed side by side. Part 3. will come afterwards since it depends on both 1. and 2.
I expect to make good progress on both 1. and 2. below by the midterm evaluation (they should be maybe one or two patches away from completion, not including bug fixes and unforeseen edge cases).

Each lettered subpoint is a separate patch unless mentioned otherwise.

1. Indexed, searchable logs
    a. Implementation of the database and log sweeping module (these can be separate patches but likely need to be landed together to be useful).
    b. API to query database
    c. Make log viewer use said API to search messages
    d. Update stats service to use database
2. Dynamic addition/removal of context in conversation's DOM
    a. Removal of messages from DOM and re-adding on scrolling (pref'd off). The methods required to prepend messages can be landed first in a separate patch.
    b. Update themes to work properly with the above if required.
    c. Make search work for messages that have been removed.
    d. Bug fixes, then pref on.
3. Infinite scrollback
    a. Query log database asynchronously to fill message buffer
    b. Update section scrolling and other conversation navigation methods
    c. Bug fixes and details like the throbber
4. Special cases/miscellaneous
    a. Load arbitrary amount of context into newly opened conversations - will be done during part 2.
    b. Fallback mechanism for users who have disabled logging - after part 3.
    c. Bug fixing and solutions to edge cases (in separate patches) - as and when required.

Before and during the community bonding period, I intend to do more research on the pathway a message follows from when it is received over the network and finally gets displayed in the UI. I will be looking at how the conversation binding, convbrowser and conversations service interact. This will enable me to make an informed decision on how/where to maintain the message buffer, how to avoid passing around excessive numbers of messages through XPCOM without breaking features like pings, and other further implementation details.

I have no other commitments during the coding period, but my final exams will be going on during

the community bonding period and will limit my free time.

Open Source Development Experience

I've been contributing to Instantbird ever since discovering it when I was looking for a project for GSoC 2013. I was accepted for the "Awesometab" project and completed it successfully over the summer. It was quite an experience.

I've been trying to contribute whenever I have time after GSoC as well. My pre-GSoC open source experience is overshadowed by the work I did last summer, but here it is anyway (quoted from last year's proposal):

"I was an active contributor to CyanogenMod - a community distribution of the Android operating system. I did quite a bit of work on bringing native resolution graphics to low pixel density (LDPI) devices, and added a few UI improvements. I also added quite a few modifications to help improve theme support, and fixed a couple of bugs. I was involved with OpenSEMC - an effort to bring the latest version of Android (through CyanogenMod 10.1) to the Sony Xperia S, which I own.
Apart from this, I usually upload the source code of any small project I do on my Github, mainly just to show my support for Open Source."

Work/Internship Experience

I've never been professionally employed or taken any internship apart from GSoC 2013, if that counts.

Academic Experience

I'm studying Electrical and Electronics Engineering (EEE) at BITS Pilani - KK Birla Goa Campus, an acclaimed institute of technology and science in India.

Why Me

As I said in my proposal last year, I think user experience is a fundamental part of any personal computer software and love working on things to improve it. Also from last year's proposal:

"I think it's very important to contribute to projects that you actually use yourself, because that gives you all the more motivation and inspiration to work on them. I personally use Instantbird and want this feature as much as anyone else, and this makes me itch to start coding away."

Why Mozilla

I've always respected Mozilla for its openness. I think it's a great organization with lots of brilliant people, bringing out excellent software that I personally use and enjoy contributing to. I had a great experience with Mozilla last summer and look forward to another one.