Matrix protocol support

Abstract:

The project aims to make Matrix as a usable protocol for Instantbird.

Content:

Personal Details

- Name: Pavan Karthik Boddeda
- Email: pavankarthikboddeda@gmail.com
- IRC nick: matrixisreal
- Telephone: 7086691664, 9248667796
- Country of residence: India
- Timezone: UTC +5:30

Project Proposal:

Current Support for Matrix:

Initial support for Matrix was added in bug 1199855 which handles :

1. Joining Rooms.
2. Fetching Participants.
3. Sending/Receiving Text Messages.

I have filed some bugs regarding existing implementation and new features. I have already started working on some of them but still everything is WIP.

So, a lot of work has to be done yet.

Support to be added in the project:

Here's a list of features I'd like to add and their implementation details :

(All of the Major features were well discussed with the Matrix and Instantbird teams before coming up with the design, however needless to say, some changes might be made accordingly when I start working on them.)

1. **Support for Direct Messages** :
   The current implementation doesn't have the concept of Private Chats and

Group Chats, So we should add that by logically isolating Private Chats from other Rooms. This has been briefly discussed here.
The major tasks are :

a) Differentiate Private Chats from Group Chats, by flagging Private chats in the prpls..

b) Prevent to open multiple conversations with the same interlocutor . This can be done using the Private Chat flag, i.e checking if a Private conversation with the given interlocutor already exists or not. If it does, bring that conversation to the UI,and If it doesn't create one.

c) Make sure others can't join a Private Conversation, this is just by setting the Room's "is_direct" flag to true while creating it.

d) Write tests for the same.

2. **A bunch of other features**, all of these are necessary and I felt are relatively easier to implement than the others in the list.

a) ***Typing Notifications*** :

Incoming Typing Events : The server fires "RoomMember.typing" whenever it knows that someone is typing. After catching the "RoomMember.typing" event we just need to update the "typing" status of the buddy here. And we need to set a timer to check and update the member's typing status. A similar implementation exists for xmpp, we can use that as a reference.

Outgoing Typing Events : The UI calls sendTyping whenever the user types something in the text box, then we should use the API call, sendTyping on the Client to notify the server. Even this needs a timer to check and update for the latest typing status.

b) ***Support Powers*** : Matrix grants permissions to users using "Powers", More about powers is talked here and the permissions in different parts of the same page. As of now I plan to just show the Moderators and Admin Level Users by probably mapping them to Op and Founder respectively after a small discussion about this on IRC with Patrick.

c) ***Leave Rooms*** : This would be accessed just by the "Close Conversation" in the UI (This would be just leaving the room while we call close() in Conversation prototype) and also a /leave command.

d) ***Handle Invites*** : A "invite" is sent whenever the user is invited to other rooms. Riot does this by notifying the users and giving them options "accept" (join room) and "decline" (leave room) in the UI. I plan to do a similar thing using a custom tab.

- In the first iteration we accept all the invites, and letting the user leave using /leave command. This will be a good first step for the handle.

- Then on the second iteration I plan to Implement UI for the invites tab, with accept and decline options. A new conversation is created when user accepts. This would be done in a way similar to the about Panel.

e) ***Contacts*** : To add support to Matrix Contacts we shall implement the GenericAccountBuddyPrototype Interface and necessary functions like addBuddy in Matrix Account Prototype.

f) ***Proper Tooltip Info*** of participants : The properties of a user which can be shown in the tooltip include displayName, UserId, presence, presence Status Message and Last Activity as of now. Doing this should be just implementing requestBuddyInfo function in Account Prototype which notifies the observers with nsISimpleEnumerator of prplITooltipInfo as the Subject.

g) ***Presence*** :

User : Letting the server know about the user presence(status type and message) by calling setPresence() on the client, this should probably be done in observe() of the Account Object.

Contacts : The event "User.presence" is fired whenever any user's presence in the room changes, So if the presence of the user in contact list is changed we call setStatus to update the status type and message.

Participants : A Participant presence is shown only in the tooltip info, so we just have to fetch the user object of the Participant and fetch the presence type, message and timestamps and push it into the tooltip info.

h) ***Handling Notices*** : Currently notices are just dumped as some json objects. So notices shall be caught and displayed as proper notices.

i) ***Auto join Rooms*** in which the user is already in, I have written a test patch for this (attached on a different bug), This should be just finishing it off.

j) ***Display Proper Usernames*** : A matrix participant has a display name(optional) and a userId. A display name has to be used whenever it is available instead of using the userId every time.

k) ***Display Proper Room names*** : This is related to this bug. This also includes coming up with a proper naming convention to display for Private Conversations, instead of showing the Room ID.

A few other things might be added If they come to my notice and they fit in the available time frame.

3. **Add support for Commands** :
   The aim is to implement all the Riot commands in the first iteration. I have implemented a /join command in this test patch. The Commands of Riot namely

/me, /ban, /kick, /deop, /invite, /leave, /nick, /dgg (or an alternative.) are to be added with proper Help Strings and error messages. This list of commands is taken from Riot's source code[link]. I have added /leave to the list though, and some other commands must be appended to this, if they are found to be necessary or useful in the second iteration.
And writing necessary tests to them.

4. **Seen Receipt for Messages** :

I am planning to add this feature only to the Private Conversations, However the backend is planned to support Group Conversations too, so that it would easy adding it, once the UI for Group Convos is ready.

Here's some context before the implementation details :

Matrix Events include everything that can happen in a room like messages, notices, calls, change of room name, topic etc. Matrix uses the concept of "receipts" which are a form of acknowledgement of an event. *m.read* is the only acknowledgment that matrix supports which indicates that the user has read up to a given event.

i) Sending Receipts :

If we want to notify the servers of some event, we need to send a proper receipt that our user has seen the event after doing the client side certainty check.

UI : The ImConversations have an implementation of tracking the unread messages[link]. A markAsRead is called when the tab is on focus or is onSelect which clears all the variables tracking different unread message counts.

Backend : We check for the unreadTargetedMessageCount to become zero, which notifies the backend that the user has seen an unseen message. Then we send a receipt for the latest message event which is detected to be last read from the UI using sendReadReceipt() API call of Client. We can detect the latest message's event by keeping track of the latest event which is not emitted by our user.

ii) Receiving Receipts :

(Some Context before the implementation details)

This part includes updating our UI to show the read notifications.

First: Updating the messages which are written to the Conversation UI, isn't possible yet, so I plan to implement this as follows :

- By adding an array or a map of messages containing the prpl object and the DOM object of the message which have been written to the conversation, as a field in convbrowser.xml**.** See this bug**.**

- This array has to be updated whenever the observer in conversation.xml is notified of "add-text" event or whenever we call addMsg.

- We shall add a new event "update-seen-receipts" to the observer in conversation.xml**.**

*Second***:** Matrix uses "up to" markers to mark the read events. This marker indicates that the acknowledgement applies to all events "up to and including"

the event specified. For example, marking an event as "read" would indicate that the user had read all events *up to* the referenced event. So if that event is a message, we should recognize all the messages up to that as read.

Backend :

a) A concept of "isSeen" and “bywhom” and some other necessary metadata like timestamps should be introduced to the messages.

b) Update the “isSeen” status and other meta data of the messages, after receiving the notifications from the server. This would be done something similar to this.

c) Now we have to notify the observer of the event “update-seen-receipts” with the message prpl object as Data.

UI :

Now we have to handle the event “update-seen-receipts” in the observer in conversation.xml, by updating the UI element of that message (with seen timestamps) fetched from the message array in convbrowser.xml.

The UI for showing the seen Receipts has two options,1) showing receipts on the Tooltip and 2) Adding seen timestamps to the message theme (just for private Conversations). However (2) is the target and (1) would be considered as the first step. Some notes about the two options :

(As said earlier this is planned only for DM’s.)

1) *Tooltip* :

a) The Sent and Seen timestamps should be just shown in the tooltip info of messages which have been seen and may be

b) An “unseen” tag for unseen messages.

The will be something similar to this[a] and this[b].

(nhnt11 suggests to make this Priority 1 as this provides a basic UI to test the backend for the feature.)

Pros : Relatively easy to implement, easy to test.

Cons : Not very intuitive.

2) *Adding receipts to Message Theme* :

- A proper UI similar to Facebook’s seen receipts[image] is to be added to the current message theme i.e a receipt just to the last message sent by the user.

- On the second iteration this has to be designed and implemented.

Pros : Intuitive UI, This is also the final objective of this feature.

Cons : Might take time to design and implement.

5. **Support for Video Calls** :

a) Matrix has VoIP support using WebRTC implementation of Calls. I have tested this out using this example. The WebRTC part is already handled, So the work is to properly handling the call/accept/reject actions and placing the video stream in the UI.

b) The UI to support Video Calls and an API design for the same is already implemented in this bug which is still WIP, but we can use most of its work and make changes if necessary.

Schedule of Deliverables :

I am already familiar with the Instantbird Code base, so I can dive right into the Coding process from the first week itself.

Before Phase 1 :

Part 0 : In the Community Bonding period of one month, I would take up a mini project of adding a very basic support for some well known protocol like IRC or Telegram and write a blog post Mozilla Docs giving a walk through for the same. This can be used by new Instantbird Developers as a framework for adding a new protocol.

(Each of the mentioned parts and their subparts below shall be dealt in separate patches)

During Phase 1 :

Part 1 : Support for Direct Messages (1) including the tests. **[ET : 2 weeks]**

Part 2 : Implement the features mentioned in (2) [possibly each subpart in a different patch] and Commands (3) [one patch for all commands].

[ET : 2 weeks]

During Phase 2 :

Part 2 [Contd.] : Complete the unfinished works of part 2 of Phase 1 and write tests.

[ET : 2 weeks]

Part 3 : Complete Seen Receipt for Messages (4) . Most of the time would be used to design, implement the UI adding a neat API for other protocols to use the same.

[ET : 2-3 weeks]

(*By this time Matrix must be ready with most of its core features, without the edge cases and unforeseen errors unhandled.)*

During Phase 3 :

-**Part 4** : Support for Video Calls (5). Major tasks include :

- Setting up UI and plugin the API of UI to protocol. **[ET : 1 week]**
- Implement the backend and testing. **[ET : 1-2 weeks]**

-**Part 5** : Buffer time for handling unforeseen things, bug fixes and writing necessary tests.

Cleaning up the entire code and making it available for a final review process and final evaluation. **[ET : 2 weeks]**

Along with these, these are the constant things which had to be done throughout the coding period :

- Write a Blog post every week about my learnings at Instantbird and Matrix and improving the existing Docs about adding a new protocol for Instantbird.
- Try to **Fix bugs in Matrix SDK** that I have come across which might improve the implementation either by changing things in our repo or sending them a PR.
- Hanging out in #matrix and #matrix-dev to seek suggestions.
- Use matrix on daily basis to dog food the code.

Open Source Development Experience :

I have started contributing to a big Organization after reporting and fixing a bug in Firefox Since then I have been actively contributing to Mozilla and have fixed/reported some bugs in Instantbird, Firefox and Thunderbird. However most of my open source experience is while contributing to Instantbird which includes the components IRC and Matrix, which makes me already well known to the community. During this project I hope to contribute to Matrix also whenever possible, I have already send them few simple PR's, but they are not pushed in yet though.

My InstantBird Bugs. A lot of which are still in the review cycle and some are WIP.

My Firefox and other Mozilla Bugs.

Besides these, I use a lot of Open Source Code and try to improve it for my pet projects.

Academic and Internship Experiences :

I am currently doing my Bachelor's Degree in Computer Science in Indian Institute of Technology Guwahati (IIT-G) which is an acclaimed institute in India. I have done Courses in Algorithms and currently doing a course in Networks which I am very much interested in and also this has been one of my motivation of choosing to work with an Instant Messaging Client like Instantbird.

I have done an Internship in Cloud Kinetics (A manager of Amazon Cloud Services), where I did a project to automate the process of Migration of the AWS resources within regions, accounts etc. I learn quite a few things about the Computer Networks in the process.