Changes Index

1. The structure of the report is reorganized to two parts, one for work flow and the other for components detail.
2. Add details of methods in each component.
3. Add description of TLRPC.java
4. Add description to each screenshot of codes.
5. Modified sequence diagram to fix mistakes.
6. Add screenshots of application to help understanding.
7. Separate screenshots of codes for better understanding.

Implementation of Sending Text Messages in Telegram

The Telegram is a cloud-based mobile messaging application, thus sending plain messages is an essential feature of the application. The Telegram can be used to send different types of messages, including images, voices, videos, locations, etc. However, talking about sending all those types messages is too much work, and sending text message is the basic of the sending message function, thus this report focus on sending text message. Multiple components of the application worked together to realize this feature. This report aims to assist readers to have a basic understanding of those components and their connections so that readers can makes some changes on them. The first part of the report describes the main components and the work flow of sending text messages, and the second part of the report discusses about the components and related methods in detail.

Part1 Main components and work flow

There are four main components work together to realize the sending text message function. They are written in three java classes, ChatActivityEnterView.java, SendMessageHelper.java, ConnectionManager.java and several C++ classes. The ChatActivityEnterView.java sets the input view and communicate with the user, the SendMessageHelper.java handles message types and information, the ConnectionManager.java and C++ classes help to connect the application and the internet.

When the user open the text message input view, the ChatActivityEnterView.java is invoked. The *constructor* ChatActivityEnterView() is called, and the input view as well as the buttons are set. The send button is set with a listener. When the user click the send button, the listener respond and call sendMessage(). In sendMessage(), the sendMessageInternal() method is called. In sendMessageInternal(), the processSendingText() is called. In processSendingText(), the text message is processed and sendMessage() in SendMessageHelper.java is called to send the message.

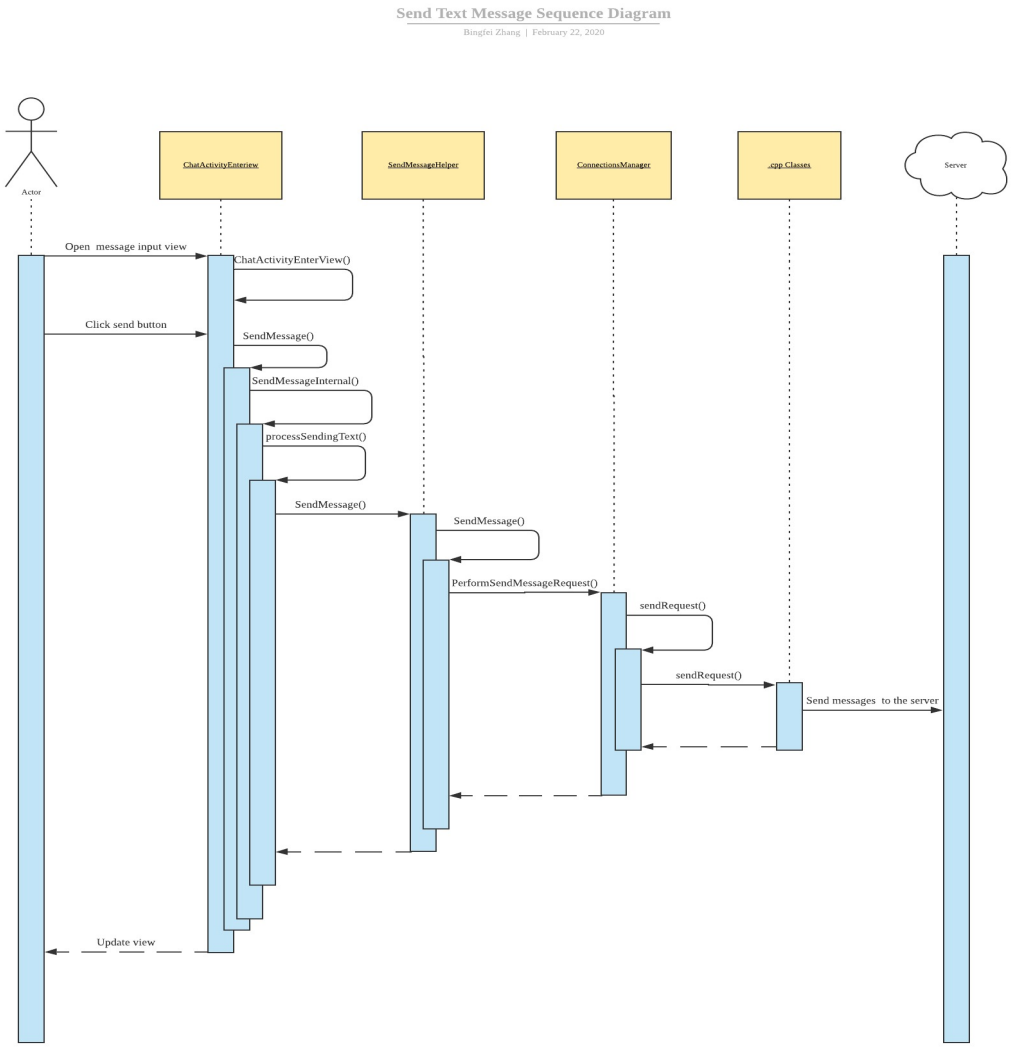
The sendMessage() called in SendMessageHelper.java focus on sending text message, it call a general sendMessage() method which handles all types of message including photos, videos, etc.

In the general sendMessage() method, the peformSendMessageRequest() method in ConnectionsManager.java is called to send the request.

The peformSendMessageRequest() in ConnectionsManager.java send the request by calling a sendRequest() method focusing on message request. That sendRequest() then call a general sendRequst(), which is a native Java interface. The JNI sendRequest() connect to C++ classes where sending request realized.

The C++ classes at the bottom layer realize the function of sending request to the server. The details are not discussed in this report.

The sequence diagram is shown below, and the detail of each components and methods are discussed in the part two.



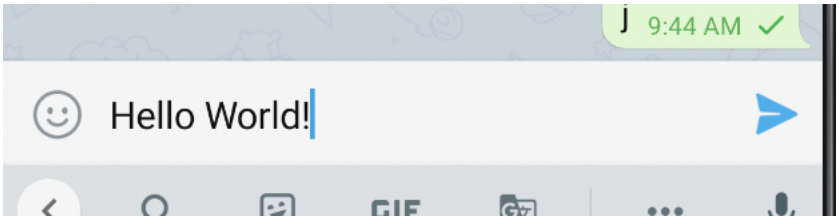
Part 2 Details of each components

2.1 ChatActivityEnterView.java

The ChatActivityEnterView.java is used to set the view of input message, including text, video and audio. The logics of user interaction with buttons are also write in this class. In this report, only view and logic of sending plain text in normal mode are discussed.

2.1.1 ChatActivityEnterView()

The constructor of ChatActivityEnterView.java initialize the message input view of the chat. As the figure shows:



First, the constructor set up the send button when not in schedule mode, which is shown as a blue paper plane:

```
1623        else {
1624            sendButtonDrawable = context.getResources().getDrawable(R.drawable.ic_send).mutate();
1625            sendButtonInverseDrawable = context.getResources().getDrawable(R.drawable.ic_send).mutate();
1626            inactinveSendButtonDrawable = context.getResources().getDrawable(R.drawable.ic_send).mutate();
1627        }
1628        sendButton = new View(context) {...};
1727        sendButton.setVisibility(INVISIBLE);
1728        int color = Theme.getColor(Theme.key_chat_messagePanelSend);
```

and then add a listener to it:

```
1738        sendButton.setOnClickListener(view -> {
1739            if (sendPopupWindow != null && sendPopupWindow.isShowing()) {
1740                return;
1741            }
1742            sendMessage();
1743        });
```

When click the send button, the listener call the sendMessage() method.

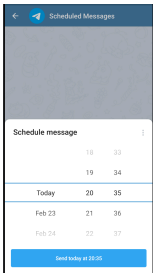
The constructor also set up several other layouts like buttons and keyboards, as well as their animation. Sending other types of messages are also initialized in the constructor. As they are not related to sending text message, they are not discussed here.

2.1.2 sendMessage()

The method sendMessage() check if the user is in a schedule mode.

```
2613        if (isInScheduleMode()) {
2614            AlertsCreator.createScheduleDatePickerDialog(parentActivity, parentFragment.getDialogId(), this::sendMessageInternal);
```

If the user is in the schedule mode, a date pick selector is shown. The schedule mode is a mode sending message on a scheduled time.



Otherwise method sendMessageInternal() is called.

```
2615            } else {
2616                sendMessageInternal( notify: true,  scheduleDate: 0);
2617            }
```

2.1.3 sendMessageInternal()

In sendMessageInternal(), the code first check if current mode is in slow mode. The slow mode is a mode limit the number of messages sending in a fixed time. A delegate which defined in the ChatActivityEnterView() is used to handle if the slow mode limit is reached. The report does not focus on slow mode and does not show detail of the slow mode here.

Then the method get notification settings from current chat fragment.

```
2627        if (parentFragment != null) {
2628            TLRPC.Chat chat = parentFragment.getCurrentChat();
2629            TLRPC.User user = parentFragment.getCurrentUser();
2630            if (user != null || ChatObject.isChannel(chat) && chat.megagroup || !ChatObject.isChannel(chat)) {
2631                MessagesController.getNotificationsSettings(currentAccount).edit().putBoolean("silent_" + dialog_id, !notify).commit();
2632            }
2633        }
```

After that, the code check if the message is sticker, audio or video, and process them differently. Which is not discussed in this report.

```
2634        if (stickersExpanded) {...}
2641        if (videoToSendMessageObject != null) {...} else if (audioToSend != null) {...}
```

If the message is text message, the code call processSendingText() method to send the text, and update the state to the initial state.

```
2672        if (processSendingText(message, notify, scheduleDate)) {
2673            messageEditText.setText("");
2674            lastTypingTimeSend = 0;
2675            if (delegate != null) {
2676                delegate.onMessageSend(message, notify, scheduleDate);
2677            }
2678        } else if (forceShowSendButton) {
2679            if (delegate != null) {
2680                delegate.onMessageSend( message: null, notify, scheduleDate);
2681            }
2682        }
```

2.1.4 processSendingText()

In the processSendingText() method, the String format message is transferred to ArrayList<TLRPC.MessageEntity>

```
2704        for (int a = 0; a < count; a++) {
2705            CharSequence[] message = new CharSequence[]{text.subSequence(a * maxLength, Math.min((a + 1) * maxLength, text.length()))};
2706            ArrayList<TLRPC.MessageEntity> entities = MediaDataController.getInstance(currentAccount).getEntities(message, supportsNewEntities);
```

and a SendMessageHelper instance call its sendMessage() method to send the transferred message.

```
2707            SendMessagesHelper.getInstance(currentAccount).sendMessage(message[0].toString(), dialog_id, replyingMessageObject, messageWebPage, messageWebPageSearch, entities,  replyMarkup: null,
```

2.2 SendMessageHelper.java

This class is a helper on sending all types of messages. Telegram can send text, image, audio, video, game, website, emoji, location etc. The report just focus on sending text message here.

2.1.1 sendMessage()

This method is a specific method to send text message.

```
2349    public void sendMessage(String message, long peer, MessageObject reply_to_msg, TLRPC.WebPage webPage, boolean searchLinks, ArrayList<TLRPC.MessageEntity> entities, TLRPC.ReplyMarkup
2350        sendMessage(message, caption: null, location: null, photo: null, videoEditedInfo: null, user: null, document: null, game: null, poll: null, peer, path: null, reply_to_msg, webPage, s
2351    }
```

It calls general sendMessage() method which handle all types of messages.

2.1.2 sendMessage()

In this method, the code first check if the user, peer, message is empty.

```
2370        if (user != null && user.phone == null) {...}
2373        if (peer == 0) {...}
2376        if (message == null && caption == null) {...}
```

If is empty, the code return.

Then the method check if the message is a resend message. If true, resend it.

```
2411            if (retryMessageObject != null) {
2412                newMsg = retryMessageObject.messageOwner;
2413                if (parentObject == null && params != null && params.containsKey("parentObject")) {...}
2416                if (retryMessageObject.isForwarded()) {...} else {...}
```

If it is a new message, the method check the type of the message and set flags and attributes related to the type of the message.

```
2465            if (message != null) {...} else if (poll != null) {...} else if (location != null) {...} else if (photo != null) {
```

```
2539            } else if (game != null) {...} else if (user != null) {...} else if (document != null) {...}
```

The method also set attributes like ids, reply or not, medias for the message.

```
2661            if (params != null && params.containsKey("bot")) {...}
2672            newMsg.params = params;
2673            if (retryMessageObject == null || !retryMessageObject.resendAsIs) {...}
2696            newMsg.flags |= TLRPC.MESSAGE_FLAG_HAS_MEDIA;
2697            newMsg.dialog_id = peer;
2698            if (reply_to_msg != null) {...}
2707            if (replyMarkup != null && encryptedChat == null) {...}
2711            if (lower_id != 0) {...} else {...}
2763            if (high_id != 1 && (MessageObject.isVoiceMessage(newMsg) || MessageObject.isRoundVideoMessage(newMsg))) {...}
```

Then the method write a log is permitted.

```
2820        if (BuildVars.LOGS_ENABLED) {
2821            if (sendToPeer != null) {
2822                FileLog.d("send message user_id = " + sendToPeer.user_id + " chat_id = " + sendToPeer.chat_id + " channel_id = " + sendToPeer.channel_id + " access_hash = " + sendTo
2823            }
2824        }
2825
```

After the message is set, the method set the send request attributes based on the type of the message.

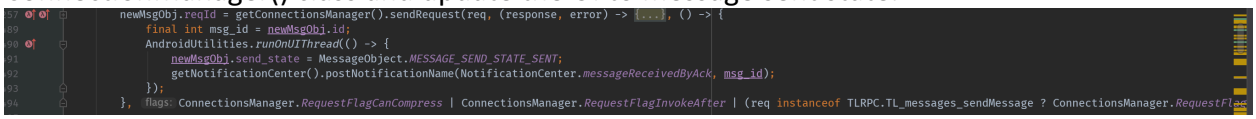
```
2828        if (type == 0 || type == 9 && message != null && encryptedChat != null) {...} else if (type >= 1 && type <= 3 || type >= 5 && type <= 8 || type == 9 && encryptedChat != null || type
```

And then call performSendMessageRequest() method to send the request.

```
3497                performSendMessageRequest(reqSend, newMsgObj, originalPath: null, delayedMessage: null, parentObject, scheduled: scheduleDate != 0);
```

2.2.3 performSendMessageRequest()

The performSendMessageRequest initialized a new thread and call sendRequest() method in ConnectionManager() class and update the UI to message sent state.



3 ConnectionsManager.java

This class is used to work on internet connection. Not only sending message but also other functions like account, documents functions are worked with this class. In this report, only methods related to sending message is discussed.

3.1 sendRequest()

This class is a method for sending message request. It call a general sendRequest() method.

3.2 sendRequest()

This class process sending all types of request. It uses a native Java interface to call classes written in C++ to perform the real work of sending request.

4 C++ classes

The C++ classes are not discussed in this report.

5 TLRPC.java

One of the most important class in the whole project is the TLRPC.java. It defines multiple classes used in the project. It also writes methods to serialize and deserialize objects, so that the objects information can be stored or send to other places.

For example: