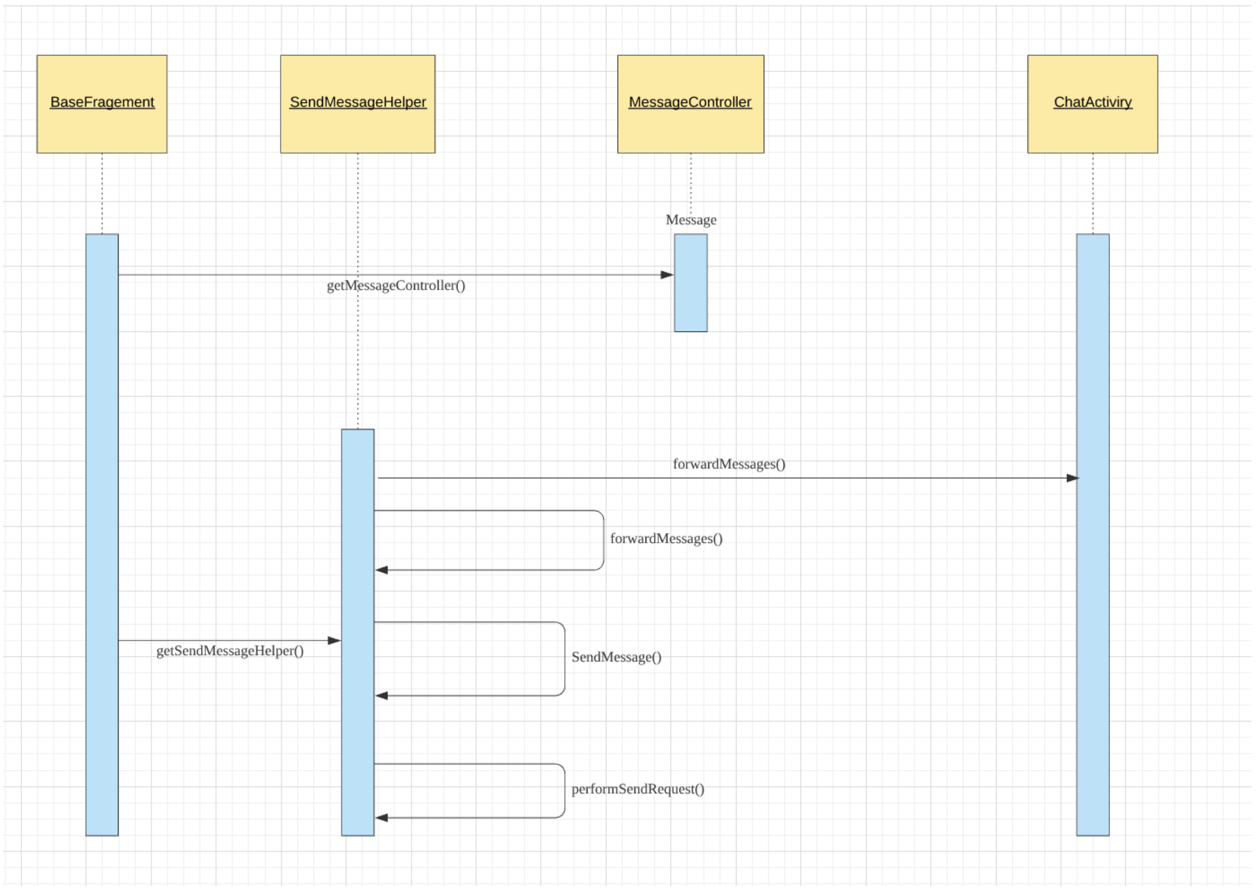
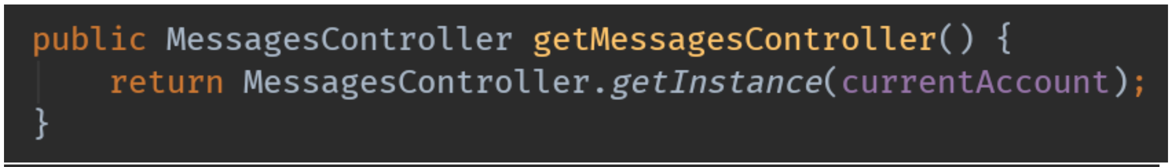
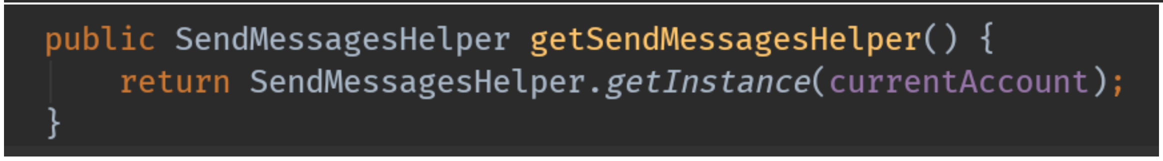
One of the essential features of Telegram is send text messages. Multiple classes worked together to realize this feature. Their relationships are roughly shown as follows:



When the app starting, the *BaseFragment* is called and two methods in it *getMessageController()* and *getSendMessageHelper()* initialized *SendMessageHelper* and *MessageController*.

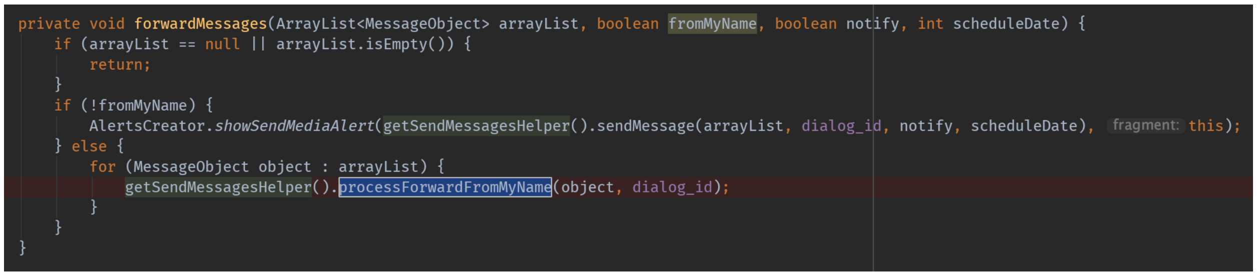


```
public MessagesController getMessagesController() {
    return MessagesController.getInstance(currentAccount);
}
```



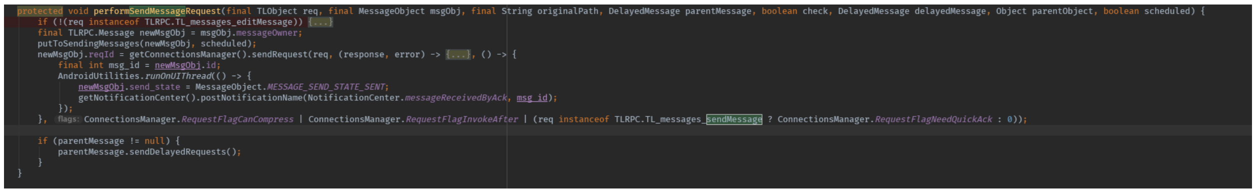
```
public SendMessagesHelper getSendMessagesHelper() {
    return SendMessagesHelper.getInstance(currentAccount);
}
```

When the user trying to send a text message, the *ChatActivity* is working. If the user input some text into the text field and press send, the ChatActivity called SendMessageHelper by method *forwardMessage()*.

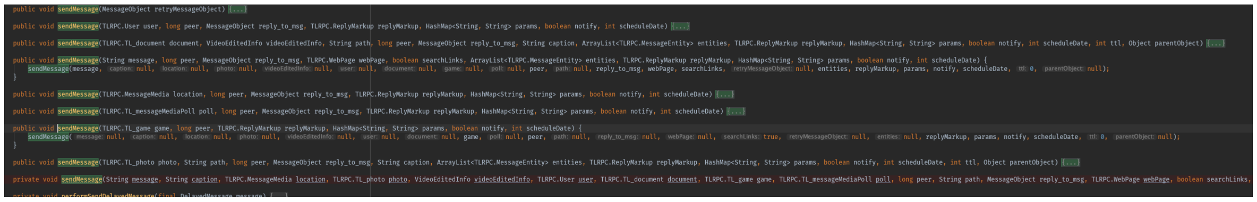


```java
private void forwardMessages(ArrayList<MessageObject> arrayList, boolean fromMyName, boolean notify, int scheduleDate) {
    if (arrayList == null || arrayList.isEmpty()) {
        return;
    }
    if (!fromMyName) {
        AlertsCreator.showSendMediaAlert(getSendMessagesHelper().sendMessage(arrayList, dialog_id, notify, scheduleDate),  fragment: this);
    } else {
        for (MessageObject object : arrayList) {
            getSendMessagesHelper().processForwardFromMyName(object, dialog_id);
        }
    }
}
```

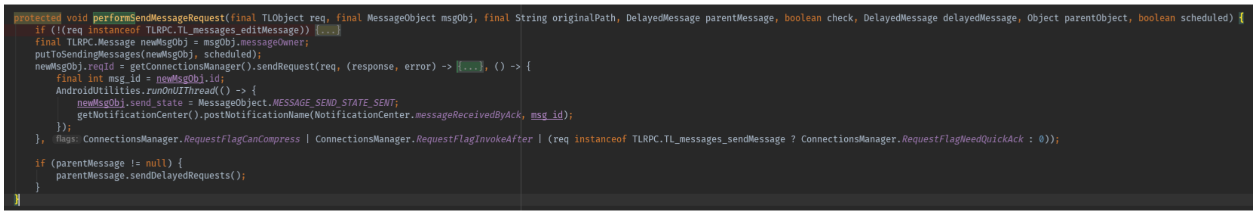
The *SendMessageHelper* have several methods working on sending message. The method performSendMessageRequest() is called in the ChatActivity. In the performSendMessageRequest(), method sendMessage() is called. It has several overloaded method *sendMessage()* to send different kind of messages, e.g. photo, lacation, etc. After message processed in *sendMessage(),* the method *performSendMessageRequest()* is called to send message out.



```java
protected void performSendMessageRequest(final TLObject req, final MessageObject msgObj, final String originalPath, DelayedMessage parentMessage, boolean check, DelayedMessage delayedMessage, Object parentObject, boolean scheduled) {
    if (!(req instanceof TLRPC.TL_messages_editMessage)) {...}
    final TLRPC.Message newMsgObj = msgObj.messageOwner;
    putToSendingMessages(newMsgObj, scheduled);
    newMsgObj.reqId = getConnectionsManager().sendRequest(req, (response, error) -> {...}, () -> {
        final int msg_id = newMsgObj.id;
        AndroidUtilities.runOnUIThread(() -> {
            newMsgObj.send_state = MessageObject.MESSAGE_SEND_STATE_SENT;
            getNotificationCenter().postNotificationName(NotificationCenter.messageReceivedByAck, msg_id);
        });
    },  flags: ConnectionsManager.RequestFlagCanCompress | ConnectionsManager.RequestFlagInvokeAfter | (req instanceof TLRPC.TL_messages_sendMessage ? ConnectionsManager.RequestFlagNeedQuickAck : 0));

    if (parentMessage != null) {
        parentMessage.sendDelayedRequests();
    }
}
```



```java
public void sendMessage(MessageObject retryMessageObject) {...}

public void sendMessage(TLRPC.User user, long peer, MessageObject reply_to_msg, TLRPC.ReplyMarkup replyMarkup, HashMap<String, String> params, boolean notify, int scheduleDate) {...}

public void sendMessage(TLRPC.TL_document document, VideoEditedInfo videoEditedInfo, String path, long peer, MessageObject reply_to_msg, String caption, ArrayList<TLRPC.MessageEntity> entities, TLRPC.ReplyMarkup replyMarkup, HashMap<String, String> params, boolean notify, int scheduleDate, int ttl, Object parentObject) {...}

public void sendMessage(String message, long peer, MessageObject reply_to_msg, TLRPC.WebPage webPage, boolean searchLinks, ArrayList<TLRPC.MessageEntity> entities, TLRPC.ReplyMarkup replyMarkup, HashMap<String, String> params, boolean notify, int scheduleDate) {
    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, searchLinks,  retryMessageObject: null, entities, replyMarkup, params, notify, scheduleDate,  ttl: 0,  parentObject: null);
}

public void sendMessage(TLRPC.MessageMedia location, long peer, MessageObject reply_to_msg, TLRPC.ReplyMarkup replyMarkup, HashMap<String, String> params, boolean notify, int scheduleDate) {...}

public void sendMessage(TLRPC.TL_messageMediaPoll poll, long peer, MessageObject reply_to_msg, TLRPC.ReplyMarkup replyMarkup, HashMap<String, String> params, boolean notify, int scheduleDate) {...}

public void sendMessage(TLRPC.TL_game game, long peer, TLRPC.ReplyMarkup replyMarkup, HashMap<String, String> params, boolean notify, int scheduleDate) {
    sendMessage( message: null,  caption: null,  location: null,  photo: null,  videoEditedInfo: null,  user: null,  document: null, game,  poll: null, peer,  path: null,  reply_to_msg: null,  webPage: null,  searchLinks: true,  retryMessageObject: null,  entities: null, replyMarkup, params, notify, scheduleDate,  ttl: 0,  parentObject: null);
}

public void sendMessage(TLRPC.TL_photo photo, String path, long peer, MessageObject reply_to_msg, String caption, ArrayList<TLRPC.MessageEntity> entities, TLRPC.ReplyMarkup replyMarkup, HashMap<String, String> params, boolean notify, int scheduleDate, int ttl, Object parentObject) {...}

private void sendMessage(String message, String caption, TLRPC.MessageMedia location, TLRPC.TL_photo photo, VideoEditedInfo videoEditedInfo, TLRPC.User user, TLRPC.TL_document document, TLRPC.TL_game game, TLRPC.TL_messageMediaPoll poll, long peer, String path, MessageObject reply_to_msg, TLRPC.WebPage webPage, boolean searchLinks,

private void performSendDelayedMessage(final DelayedMessage message) {...}
```



```java
protected void performSendMessageRequest(final TLObject req, final MessageObject msgObj, final String originalPath, DelayedMessage parentMessage, boolean check, DelayedMessage delayedMessage, Object parentObject, boolean scheduled) {
    if (!(req instanceof TLRPC.TL_messages_editMessage)) {...}
    final TLRPC.Message newMsgObj = msgObj.messageOwner;
    putToSendingMessages(newMsgObj, scheduled);
    newMsgObj.reqId = getConnectionsManager().sendRequest(req, (response, error) -> {...}, () -> {
        final int msg_id = newMsgObj.id;
        AndroidUtilities.runOnUIThread(() -> {
            newMsgObj.send_state = MessageObject.MESSAGE_SEND_STATE_SENT;
            getNotificationCenter().postNotificationName(NotificationCenter.messageReceivedByAck, msg_id);
        });
    },  flags: ConnectionsManager.RequestFlagCanCompress | ConnectionsManager.RequestFlagInvokeAfter | (req instanceof TLRPC.TL_messages_sendMessage ? ConnectionsManager.RequestFlagNeedQuickAck : 0));

    if (parentMessage != null) {
        parentMessage.sendDelayedRequests();
    }
}
```