University of the Philippines Manila
College of Arts and Sciences
Department of Physical Sciences and Mathematics

Reliable and Robust File Transfer System over Short Messaging Service

A special problem in partial fulfillment
of the requirements for the degree of
Bachelor of Science in Computer Science

Submitted by:

Axel Philip Advento

May 2018

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "Reliable and Robust File Transfer System over Short Messaging Service" prepared and submitted by Axel Philip Advento in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Gregorio B. Baes, Ph.D. (*candidate*)
Adviser

EXAMINERS:

		Approved	Disapproved
1.	Avegail D. Carpio, M.S.	________	________
2.	Richard Bryann L. Chua, Ph.D. (*candidate*)	________	________
3.	Perlita E. Gasmen, M.S. (*candidate*)	________	________
4.	Marvin John C. Ignacio, M.S. (*candidate*)	________	________
5.	Ma. Shiela A. Magboo, M.S.	________	________
6.	Vincent Peter C. Magboo, M.D., M.S.	________	________
7.	Geoffrey A. Solano, Ph.D. (*candidate*)	________	________

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Shiela A. Magboo, M.S.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Leonardo R. Estacio, Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

This study presents a reliable and robust method for transferring files over Short Messaging Service. By testing the system against a set of repeated rigid test cases and fault injection methods, it has proven itself reliable and robust enough to survive problems found on an unreliable SMS service. Protection against adversaries are also implemented as a feature using security measures such as encryption and authentication.

Results show an effective alternative for sending and receiving files but show noticeable drawbacks in terms of speed. The work presented here has profound implications for future studies of utilizing low bandwidth but widely accessible means of communications such as Short Messaging Service and may help solve the problem of data transmission in places where Internet connectivity is unavailable or where other means of transmission of data is hazardous or impossible.

Keywords: File transfer, Short Messaging Service, Reliability, Robustness

Contents

I. Introduction

A. Background of the Study

File Sharing Methods and Technologies

File sharing can be defined as "a method of distributing computer files, ... among a large number of users".[24] Such practice enabled users to share various digital data such as documents, multimedia, and computer programs across the world.

Files can be shared in a number of ways. They can be distributed using removable storage media, over wired connections, or wirelessly such as wireless Ethernet and mobile data.

Files were first exchanged using non-volatile physical media. In 1725, Basil Bouchon used punched paper rolls to store instructions for machines. In 1846, punch tapes became viable for use in recording telegrams. Data stored on these storage devices were more of instructions for machines rather than actual recorded data, however. In 1880, Hermal Hollerith invented recording data on punch cards which ultimately became a format that IBM used in the following decades.

Afterwards, various magnetic media such as magnetic tapes (used in 1951 as data storage medium), drum memory (1946), hard disk drives (1956), and floppy disks (1971) were used as storage media. Magnetic tapes and floppy disks saw widespread use as removable media. Around mid-1980s, specifications for storing digital signals on optical discs are completed which led to the development of Compact Discs (CDs), Digital Video Discs (DVDs), and Blu-ray Discs (BD).

Electronic flash memory were introduced for storing digital data later on. Examples include CompactFlash cards, Secure Digital (SD) cards, Memory Sticks, and USB Flash Drives.

Concurrently, file sharing has also been a practice through wired connections. During 1970s up to late 1980s, file sharing was mostly done over landline telephones and modems using systems such as Usenet, BBS, and XModem. Internet Service Providers (ISPs) emerged on 1990s and the Internet became fully commercialized by 1995. This gave way to modern methods of distributing files such as the File Transfer Protocol (FTP), the World Wide Web, Internet Relay Chat (IRC) programs, Instant Messaging, and later on, the BitTorrent protocol.[13]

Wireless connectivity made the Internet accessible to people who don't have access to wired connections and people living on locations where setting up infrastructure for wired services would be difficult or impossible. In 1996, access to mobile web was made possible with the introduction of the Nokia 9000 Communicator phone. It was done over 2G cellular networks, which initially do not offer data services. During the early 2000s, Wireless Access Protocol (WAP) and i-mode achieved popularity as means of accessing the mobile web and Internet services. In 2000, General Packet Radio Service (GPRS) was introduced as a packet-switched data service, which extends Global System for Mobile (GSM) Packet circuit-switched data capabilities. While in 2003, Enhanced Data rates for GSM Evolution (EDGE) was deployed on GSM networks which allows improved data transmission rates over GPRS.

The first Internet-ready satellite was made available for consumers in 2003. Satellite Internet was targeted mainly to rural areas as an alternative to traditional Internet connections such as dial-up and ADSL. The first commercial Third Generation (3G) network was also finally launched in the same year.

With the advent of 3G cellular networks and its extensions, higher data speeds and better security compared to 2G networks are observed on 3G-capable devices. However, adoption in some countries are slowed down due to the large cost of upgrades and spectrum licensing fees. In 2013, Fourth Generation technologies (4G) started to roll out. Technologies can only be considered 4G if "a substantial level of improvement in performance and capabilities with respect to the initial third generation systems now deployed".[8]

Short Messaging Service (SMS) and Its State in the Philippines

Short Messaging Service is a component of web, telephone, and mobile device networks that enables users to send and receive text messages. SMS is part of standardized communication protocols that were defined in 1985 as part of the GSM standards. [25]

SMS messages are sent to Short Message Service Centers (SMSC), which then attempts to forward the message to the recipient. This delivery however is "best-effort". In case the SMSC cannot reach the intended recipient, it keeps the message for a duration set by the mobile operator and retries until the recipient becomes available again, the message expires, or the retry limit set by the operator has been reached. This also means that there are no guarantees that the message will be delivered on time or be delivered to the recipient at all. Despite being best-effort, SMS is an almost reliable service with an observed failure rate as high as 5.1% during normal operation conditions.[12]

The Philippine Statistics Authority reported in 2017 that an estimated 8,957,952 people are subscribed on an Internet Service Provider which includes wired DSL, cable Internet, and wireless Internet. The reported number of active mobile telephone service subscriptions, on the other hand, is 130,319,459, which is more than the country's total population.[15] Various studies and news sources also consistently place the Philippines as the "text messaging capital of the world".[5]

B. Statement of the Problem

With the current technologies available to consumers today, ideally anyone should be able to send and receive digital data conveniently anytime and anywhere. Unfortunately, these methods and technologies are not as accessible or widely available in the real world.

First, transferring files over wired connections is limited only to areas where wired telephone networks and ISPs had laid out infrastructure for their services. These areas exclude remote rural communities and areas where building infrastructure would be challenging or just impossible.

Wireless Internet, on the other hand, has its own challenges as well. Satellite Internet typically requires a completely clear line of sight and is sensitive to even minor obstructions. 3G and 4G networks have limited coverage and availability compared to traditional 2G networks. An assessment of 2G and 3G mobile phone networks found that UMTS networks perform worse compared to GSM networks in terms of coverage.[20] These 3G networks will have problems covering an area with an otherwise good GSM coverage unless aided by a significantly increased power consumption and additional equipment. Signals attenuate more and higher noise figures can be observed on higher frequency channels that are used on 3G and 4G networks (~5MHz and 40MHz respectively). 3G and 4G technologies also demands for stricter signal-to-noise ratio requirements.[22]

While network carriers would like to advertise that they have "100% 3G and 4G coverage", that would not be exactly true.[7] As mentioned earlier, theoretically 3G and 4G networks would perform worse than 2G networks in terms of coverage. Additionally, in the Philippines, 3G and 4G are not as widespread as real-world data shows. Over 63 percent of Filipinos are still not connected to the Internet. This is mainly attributed to lack of infrastructure and lack of mobile Internet coverage and network access.[11] On a later report, it was found out that in the Philippines mobile Internet users get 3G or 4G connections only 68.63% of the time.[13] Compared with the rest of the world, the Philippines severely lags behind on modern mobile Internet both in terms of speed and availability.[14]

Transporting files over physical non-volatile removable media has its share of its own problems too. There is still a possibility of the data being corrupted in the removable storage being used. There are also chances of the removable media being damaged or lost during transit. Weather, terrain, and other hazards will heavily affect the success and speed of this method. In case of failure, the whole process of copying to and transporting the physical media needs to be repeated again.

C. Objectives of the Study

With the limited availability and accessibility of technologies that enable people to connect and share digital data, there is a need to produce a method that makes sharing digital data more accessible. For this reason, it appeared reasonable to attempt to develop a system with the following criteria:

1. The system should be able to send and receive files over SMS. In terms of availability and coverage, 2G is a suitable candidate despite being an older technology. SMS is also a viable as data carrier due to its simplicity, low cost, and its ubiquity eliminating the need for specialized equipment and technologies.

2. The protocol to be used should be reliable and robust enough to address the following issues inherent on Short Messaging Service:

    a. SMS is still just a best-effort service.[12] The system should be able to request and retransmit lost messages during the file transfer.

    b. There is no guarantee that SMS messages will arrive instantaneously nor arrive in order.[12] The system should have a mechanism to sort the messages received in order.

    c. There is also a possibility that an SMS message intended for a previous state or session has been received. The system should be able to handle such cases.

   d. In case the received SMS messages are incomplete or erratic, there should be a verification mechanism to validate each message received by both sending and receiving ends.

3. The protocol must allow the users to transfer files securely barring any adversary to be able to read or manipulate the files and keys being transferred. This can be done by a combination of the use of the Advanced Encryption Standard (AES) and Rivest-Shamir-Adleman (RSA) algorithms, which is used by majority of technologies used today.

4. The system must give the users a convenient way to to pair with another user's system, a requirement for transferring files securely.

5. The system should be able to transfer files over SMS regardless of their type or content.

D. Significance of the Study

The results of this study will benefit people who have the need to share digital data but are unable to due to other methods being unavailable, inconvenient, or hazardous. The system that will be developed will serve as an alternative to other methods of transferring files.

This system can be used by private organizations and government agencies doing data collection that requires information gathered be collected as soon as possible, such as beneficiaries of government welfare programs that require immediate assistance, disaster recovery assistance data, and earthquake/flood mapping data.

This can also serve rural government offices that do not have stable Internet connection but still require constant information exchange such as police, local government, etc.

People in the scientific field may utilize this to communicate important metrics such as volcanic activity and weather data when wired or wireless Internet is unavailable and conditions for manual transporting data are unfavorable.

In case 3G and 4G connectivity fails but the location has good reception for sending and receiving SMS messages, in automated election systems this is viable as a last resort for wireless communications before defaulting to manual transport of storage media. This can alleviate concerns over safety of data and personnel when terrain, weather, and other factors increase hazard if transporting the data physically.

Finally, the protocol, the software, and the system developed can be a framework for development of future technologies that can take advantage of SMS and its successors in the future beyond third and fourth generation networks.

E. Scope, Limitations, and Assumptions

This study will be limited to the bounds specified below:

1. While the protocol to be formulated is intended to be platform and hardware independent, the software to be developed for this study will be limited only to the phones and modems supported by the Gammu project and most Android devices.

2. There will be a huge drawback in terms of transmission speed which is caused by the non-instantaneous delivery and slow nature of SMS.[6][12][18][19]

3. The software will only allow transmission of files of sizes less than or equal to 10 megabytes at a time due to the limit of 160 7-bit characters for each SMS message, which will constrict the number of characters reserved for both the segment numbers and the payload. This should be enough for common file sizes of 2-12 kilobytes for images, 4-20 kilobytes for documents, and 3-4 megabytes for a full mp3 song.[17]

4. There should be enough signal strength and network coverage for the hardware components to be able to send and receive SMS messages properly.

5. This study assumes that the sender and receiver Subscriber Identification Modules (SIMs) and their respective network subscriptions are allowed to send and receive unlimited SMS messages.

6. Man-in-the-middle (MiTM) attacks would not be entirely impossible. The only security provided by the protocol to be formulated would be authentication and encryption for the keys and files exchanged using the combination of RSA and AES.

7. Denial-of-Service Attacks by an adversary will not be addressed in this study.

II. Review of Related Literature

People who have limited or no access to wired and wireless technologies such as ADSL, 3G networks, and 4G networks will benefit from having a more accessible alternative method of sending data. Sending files over SMS is one possible way to achieve this. Despite the topic being given limited attention, sending binary data over SMS has been explored in various ways.

As early as 2008, an SMS-based transport layer has been designed and developed by Rao et. al.[18] Having GPRS and other technologies being unavailable in some areas has been a motivation for the study with the intention of completely bypassing GPRS networks and mobile ISPs. It was implemented by converting TCP packets and UDP datagrams into payloads and headers carried by SMS messages, sending significantly smaller packets or datagrams in batches, if possible. It was found out that such scheme is efficient for infrequent and non-bulky data transfers such as micro-browsing or Yellow Page searches. Its use for frequent and bulky data transfers is considered expensive and unoptimized however. Additionally, using the scheme for time critical applications is highly discouraged.

An attempt to provide a free Internet access has been developed by using an Internet-connected server and a specialized client browser that communicates with each other via SMS.[19] This method however is intended only for basic web browsing, stripping off images and other unnecessary elements leaving only the HTML source. It was concluded that while web browsing is possible, the service was deemed too slow for practical use despite the attempts of stripping out elements to further speed up the browsing experience.

A method of sending images over SMS as an alternative to Multimedia Messaging Service (MMS) has been presented on a separate study.[21] Motivated by the unavailability of MMS in some areas, the study attempts to provide a more accessible alternative. Unlike the previously discussed study by Rao et. al., the method used in this study simply converts the images into a text file using Base 64 encoding and sends the encoded data through several SMS messages. This eliminates the overhead introduced by the TCP or UDP protocols. However, SMS's default 7-bit encoding seems to be underutilized by using Base 64 as the encoding scheme for the data.

Another study tested the viability of data transmission using SMS as an alternative over GPRS. It was found out that SMS in general works in a slow but consistent pace while GPRS using UDP is faster but in an inconsistent manner.[6] Additionally, using the data gathered in the study, it was concluded that UDP will always be faster than SMS when sending large amounts of data even if UDP retries to send the correct data and if SMS had perfect transmission rates. Similar to the method of sending images over SMS previously discussed earlier, the whole file is encoded and split into different SMS messages. This time, a header is included together with a CRC32 checksum. The CRC32 checksum is used to verify the integrity of each message while the header value contains the queue number, which is requested by the receiver in case of timeouts or missing parts.

From the existing studies previously presented and discussed, it can be inferred that SMS as a data bearer will be slow. Additionally, using SMS for time critical applications should be strongly discouraged. An SMS message's size would be too small to fit a single file so splitting a file into smaller parts would be unavoidable. Using an efficient encoding scheme would also be needed due to SMS messages' textual nature. SMS messages aren't guaranteed to be delivered at all times so a method to retransmit missing parts would also be necessary.

III. Theoretical Framework

A. Reliability

Reliability can be defined as a quality of a protocol that ensures the accuracy and integrity of transmitted data. Reliable protocols guarantee that the data will be delivered to the recipient. This ensures that at least one copy of the message will be always delivered to the recipient. A reliable protocol will always verify that all data transmitted is received in the correct order and is intact.

On the other hand, reliability can be defined in terms of failure. If a protocol can handle a certain set of well-defined and understood failures, then it can be considered reliable with respect to those failures.[27]

For the purpose of this study, we will use both approaches but with more emphasis on guaranteed delivery, integrity, and correct order of delivered data. During the development and implementation of the protocol, certain set of tests will be done to ensure that the protocol will be reliable against a limited set of well-defined failure scenarios.

B. Robustness

Robustness can be defined as "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions".[26] Fuzz testing is a common tool for demonstrating a system's robustness. Alternatively, fault injection can be used for testing robustness.

Fault injection techniques will be used in testing the protocol to be developed, given its effectiveness in testing for error handling in code paths as well as its predictability so that bugs can be easier to track and fix. These will be implemented in test cases where the input is mutated in a controlled manner and the use of custom components in place of the ones expected by the system, which produces repeatable and well-defined faulty scenarios.

C. Data Compression

Data compression is defined as encoding information using fewer bits than the original representation. Compression can be classified as either lossless or lossy compression. Lossy compression minimizes the space needed by removing parts of data that are deemed unnecessary or less important. Lossless compression uses various methods in reducing size without loss of data.

The common factors to be considered when choosing a compression algorithm are compression/decompression speed, memory usage, and compression ratio. Compression ratio is defined as the ratio between the uncompressed size and the compressed size of the data:

$$CompressionRatio = \frac{UncompressedSize}{CompressedSize}$$

Since this project deals with SMS, a slow data bearer, compression is a necessary tool to decrease the time needed for a transmission to be completed. This project will also not be using lossy compression.

The compression algorithm that will be used for this project will be selected based on compression ratio alone. Compression/decompression speeds and memory usage do not matter since modern computers can handle these without problems and delays produced by a slow compression algorithm is insignificant if compared to the total time needed to transmit the compressed data. With this, the LZMA algorithm is chosen based on benchmark results.[2][10] The XZ utility program will be used for handling LZMA and LZMA2 compression algorithms.[3]

D. Binary-to-Text Encoding

Binary-to-Text encoding is simply the action of encoding binary data into plain text. This is necessary if the transmission method does not support sending of binary data. Common examples include Base16, Base64, and MIME.

For the purposes of this study, a slightly modified ASCII85 encoding is used due to its high efficiency of 80%, which is more efficient than common encoding schemes like uuencode or Base64. The encoding scheme is also optimal for SMS's default 7-bit character encoding reducing the size of the encoded data while at the same time, taking advantage of a slightly larger character set.

IV. Design and Implementation

A. Protocol

Overview

The TransmiSMS file transfer protocol* is designed to be a simple, connection-oriented, reliable, and robust protocol. The protocol aims to facilitate sending and receiving of files over SMS using the least amount of SMS messages sent as possible while providing essential features that provides reliability, robustness, security, privacy, and convenience.

The most basic unit of data used by the TransmiSMS file transfer protocol is called a Protocol Data Unit (PDU). Each PDU is contained in a single SMS message and contains identifiers for its type, Session ID, its segment number, checksum (if already connected), human-readable instructions for force disconnecting (when not yet connected), and user data.

Transferring files and pairing with a peer starts with establishing a connection. The connection process is a three-way handshake, initiated by the sender/initiator. This prevents the sender from flooding the receiver if the receiver is not yet ready to receive data and to prevent unsolicited file transfers. During the handshake, each PDU contains human-readable instructions on how to reject and force disconnect the connection.

* Not to be confused with FTP; this protocol is not related to the File Transfer Protocol specified by RFC 959.[16]

Terminating the connection is also done via a three-way handshake normally initiated by the sender/initiator. This ensures that all parties are notified about each other's disconnection state. After sending the last PDU, the initiator waits for a while to retransmit missing PDUs in case the other end has failed to receive them.

File transfer and key exchange operations are managed via sessions. Each session is assigned a Session ID which is a 128-bit Universally Unique Identifier (UUID) negotiated during the establishment of the connection. UUIDs have an almost zero percent chance of collisions so no two Session IDs will be the same. This is done so that PDUs not belonging to the current Session can be safely ignored and prevent them from interrupting the current operation.

The protocol will also ignore duplicate PDUs and PDUs not appropriate for the current connection state. Invalid and malformed PDUs will also be ignored.

Pairing

Before the sender and the receiver can use security features such as encryption and authentication, they must be paired first prior the file transfer. Pairing means exchanging necessary RSA public keys for encryption and authentication. This can be done over the air or manually by importing the appropriate keys.

Pairing the keys over the air is a three-step process. An initiator party must send a pairing request to the other end in order to establish the connection. After the responder has accepted the request, they both exchange Base85-encoded encryption and authentication keys split into multiple parts. After confirming that both parties have received the keys correctly, the initiator then requests disconnection.

Optionally, after receiving the encryption and authentication keys, the parties are allowed to further authenticate they keys that they have received. They are presented with an easy to read set of words which they communicate and compare over a secure channel to confirm that no man-in-the-middle attack has taken place.

File Transfer

File transfer starts with the sender requesting a file transfer request to the receiver. After the connection is established and the receiver has agreed to receive the file, both ends exchange information for the file transfer.

If security features are enabled, the receiver first generates a new AES key for file encryption and encrypts it using the sender's RSA public key. The receiver then sends the encrypted key split into multiple parts via multiple INIT PDUs. The sender then decrypts the file encryption key using its own RSA private key. The sender can compress the file first using LZMA. The sender also needs to sign the unencrypted file using the receiver's authentication key. These can be done concurrently. The file can then be processed for encryption via the AES key afterwards. Optionally, the sender can further compress the encrypted file, if its size can be further reduced. It can then be Base85-encoded and split into segments. The sender then sends the number of segments, if compression is used, and the encoded RSA signature to the receiver via a series of META PDUs.

After receiving the META PDU, confirming the information provided, and allocated the necessary resources for receiving the file, the receiver will send a READY PDU to the sender to start the file transfer. The sender then proceeds to send the processed and encoded data using a series of DATA PDUs and sending and END PDU to signify that the sender has completed sending all segments.

After the receiver has confirmed receiving the all the DATA PDU segments, reordered the DATA PDUs received, assembled and processed the encoded data, and authenticated the signature sent by the sender, it sends its own END PDU to confirm that file transfer is successful. Upon receiving the END PDU, the sender then proceeds with the disconnection procedure.

Without the security features, the receiver just sends an empty INIT PDU to the sender to signify acceptance of the request. The sender first calculates the SHA-512 checksum of the file. The sender then skips the encryption and proceeds with the compression. Using the same encoding, the data is then encoded and split into parts. Using a META PDU, the sender sends the number of segments, the checksum, if compression is used, and the file's encoded checksum to the receiver. The whole process of sending and receiving the DATA PDUs proceeds along with the verification of the received data using SHA-512 instead of an RSA signature.

Retransmission and Timeouts

The protocol assumes that all previously sent PDUs are successfully received by the receiver unless a NACK request for retransmission is received. This is done to reduce the number of PDUs the receiver needs to transmit such as ACK for every

PDU received and to take advantage of SMS's low failure rate on normal operational conditions. Retransmission requests are sent in batches inside the least amount of NACK PDU necessary. On some states during the transmission, resending a PDU signals a request for retransmission of the previously sent PDU. PDUs can also retransmitted in case of timeouts.

The maximum PDU lifetime is set as 60 seconds. Not receiving any PDU for the duration of the maximum PDU lifetime x 2 will result in a timeout. The user will then be prompted if he/she wants to continue or if the connection must be terminated immediately.

Security

For file encryption, a 128-bit symmetric AES key will be used. Cipher Block Chaining (CBC) mode of operation for AES will be used as well. Additionally, Public Key Cryptography Standards #7 (PKCS#7) will be used for padding for added security. For file authentication, a 2048-bit asymmetric RSA key will be used together with PKCS#1 with SHA-512 hashing. The AES key that will be transmitted will be further encrypted using a 2048-bit asymmetric RSA key with Optimal Asymmetric Encryption Padding (OAEP) with SHA-256 for Mask Generation Function 1 (MGF1) padding.

To reduce predictability, for every file transfer session, a new AES key is generated. Additionally, only public keys are exposed during the pairing process. The user is also warned if the already paired peer requests to redo the pairing process.

During the file transfer, when receiving the encrypted AES key, the sender can immediately verify the receiver's authenticity. The receiver, however, can only verify the sender's authenticity if the sender failed to decrypt the encryption key sent or after all the parts of the file is sent successfully but the receiver cannot decrypt the file using its own keys. The receiver must be able to verify the authenticity of the decrypted file using the stored keys the receiver and the sender has exchanged prior the transmission.

Because the protocol allows a kill switch for the connection, an adversary with enough resources to spoof the other end can always terminate the connection prematurely. However, the adversary won't be able to fake a file transfer successfully nor view the contents of the keys and the files being exchanged.

B. Hardware Setup

The Hardware to be used for this project will consist of a sender and a receiver host. Each host will be a computer connected to either a compatible mobile phone or an Internet dongle which will function as a GSM modem. There should also be ample signal coverage for these modems to operate normally. The sender and receiver hosts can also switch roles depending on the tests needed to be performed.

C. Software and Libraries Used

In this study, Java is used as the programming language due to its cross-platform nature and the number of libraries available to be used. It has a native support for classes, events, and threading which is useful for the development of the system and its components. The JavaFX platform is used for the Graphical User Interface (GUI) due to its consistent look and feel across different operating systems and its logical separation between program logic, GUI layout, and design.

As mentioned earlier, LZMA is used as the compression algorithm and a modified implementation of ASCII85 is used as the binary-to-text encoder due to their excellent compression ratio and encoding efficiency.

Gammu SMSD will be used as a service layer to interface with various mobile phones, GSM modems, and Internet dongles from different vendors removing the need to send AT commands directly.[4] A custom Android companion app was also developed to communicate with Android devices.

A combination of RSA and AES is used for encrypting data to be transfered over SMS. AES is used for encrypting the raw data to be sent while RSA is for encrypting the AES key to be shared between the sender and receiver. RSA is also used for authentication.

For verifying each PDU's integrity, CRC8 is used for its simplicity and small size. RSA authentication will also serve as verification for the file received. SHA-512 will be used in place of RSA for verifying files if RSA authentication is disabled.

Pretty Good Privacy (PGP) word list is used when manually authenticating key exchange over the air to detect man-in-the-middle attacks.

D. Hardware and Software Requirements

In order for the system to be fully operational, it requires the sender and receiver computers to meet the following requirements:

Hardware

- GSM modem or mobile phone capable of sending and receiving SMS messages with Gammu SMS support or an Android device capable of sending and receiving SMS (running at least Android 4.4)
- Bluetooth interface or USB/Serial Cable that comes with the GSM modem or mobile phone
- An available USB/Serial port when using USB/Serial interfaces
- 200 MB of available RAM or higher
- 20 MB of available hard disk space or higher

Software

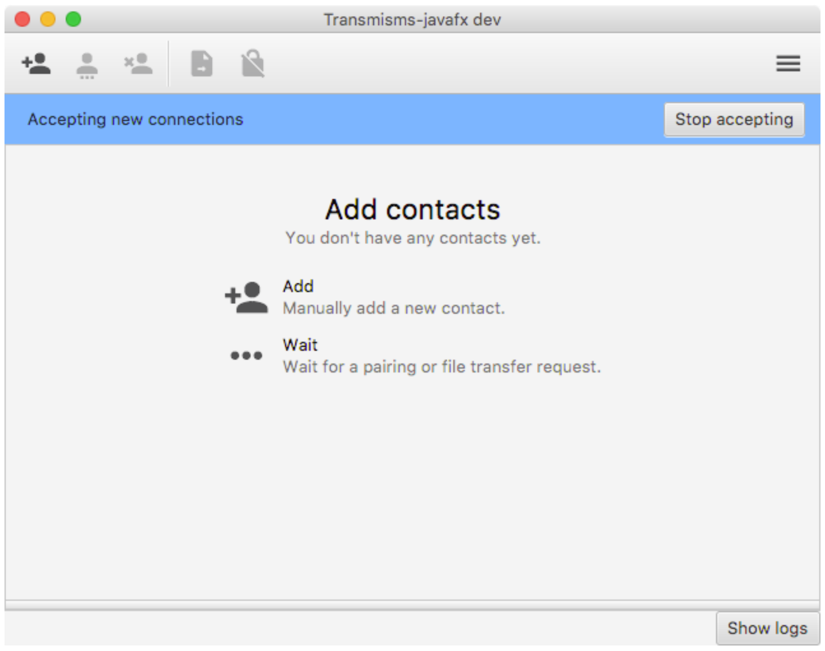
- GNU+Linux, BSD, Mac OS, or Windows operating systems
- Java Runtime Environment 8 or newer
- Gammu 1.3 or newer (and a complete setup for the chosen Gammu backend)

- Drivers for the devices to be used

V. Results

The TransmiSMS File Transfer program is a JavaFX-based GUI frontend that enables users to send and receive files reliably over SMS. This file transfer system uses the TransmiSMS protocol designed for the purposes of this study.

The program features contact management, which allows the user to add, edit, and remove contacts. The user can then send and receive files, as well as pair and exchange security keys from these contacts. The program's toolbar allow the user to do the previously mentioned actions as well as the option to send files either via a secure or an unsecure connection.


Transmisms-javafx dev

Accepting new connections

Stop accepting

Add contacts

You don't have any contacts yet.

Add

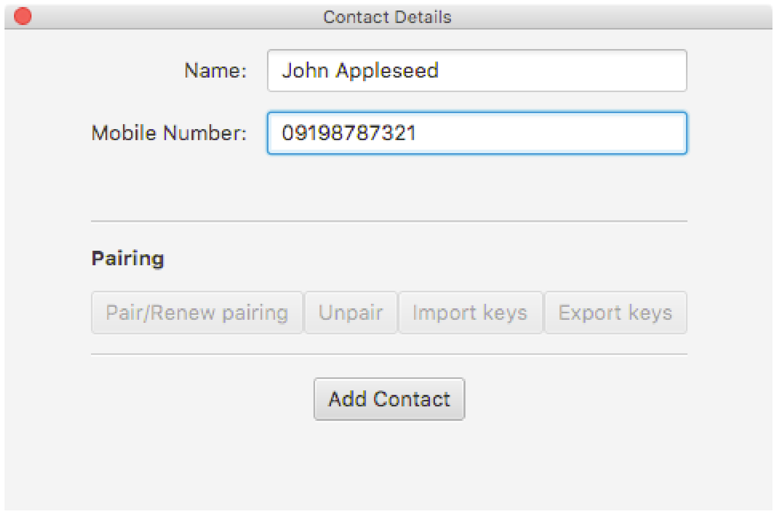
Manually add a new contact.

Wait

Wait for a pairing or file transfer request.

Show logs

Figure 1. Main Window with no Contacts Added yet

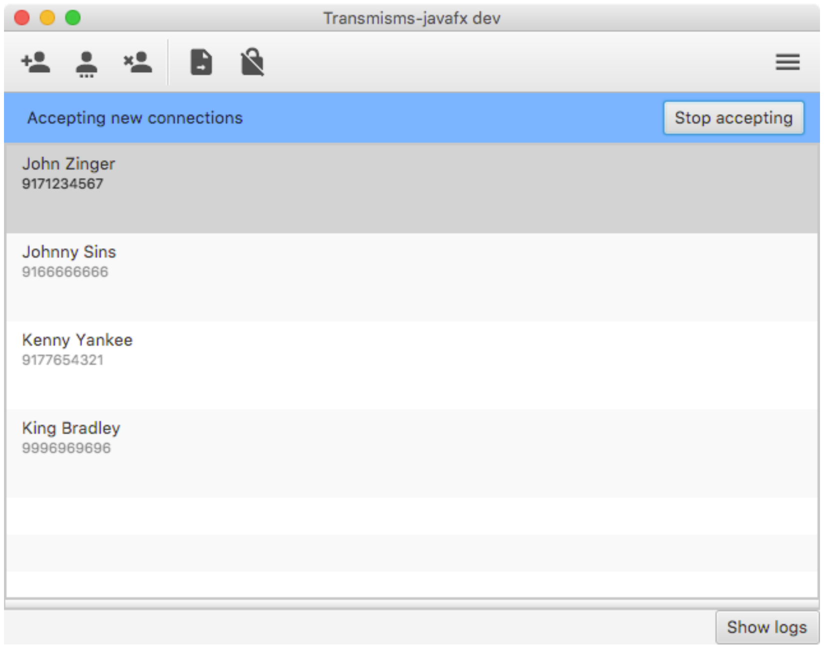

Contact Details

Name: John Appleseed

Mobile Number: 09198787321

Figure 2. Add/Edit Contact Dialog

As soon as the program starts, it starts listening for connection requests from both registered contacts and non-registered users. It can be disabled by pressing the "Stop accepting" button on the infobar.



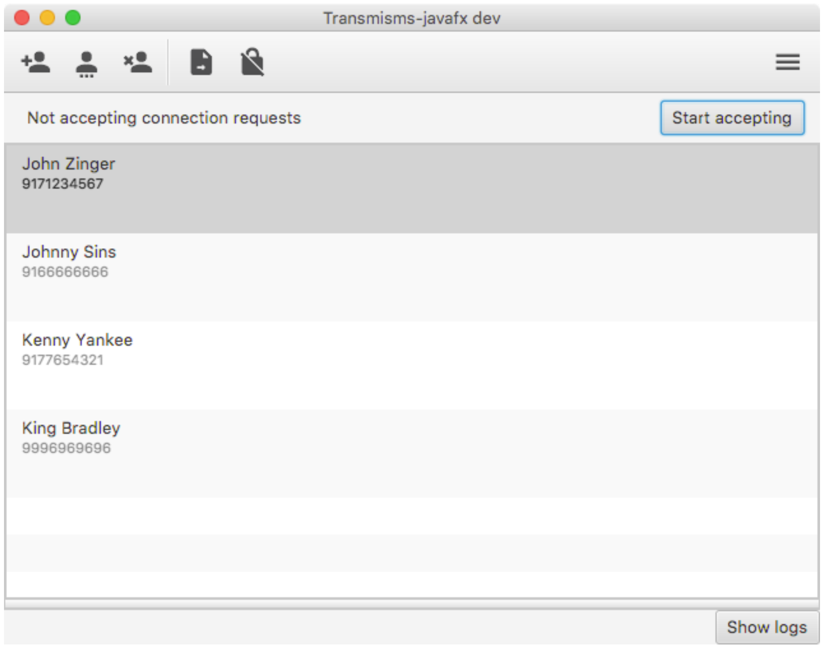
John Zinger
9171234567

Johnny Sins
9166666666

Kenny Yankee
9177654321

King Bradley
9996969696

Figure 3. Main Window, Listening for New Connections



John Zinger
9171234567

Johnny Sins
9166666666

Kenny Yankee
9177654321

King Bradley
9996969696

Figure 4. Main Window, Not Listening for New Connections

The program also detects if there are problems encountered from the SMS service being used. If such problems exist, the user must resolve these issues first before proceeding. Supported SMS services are Gammu SMSD via PostgreSQL and the Android Companion app developed mainly to enable the program to interact with Android devices.

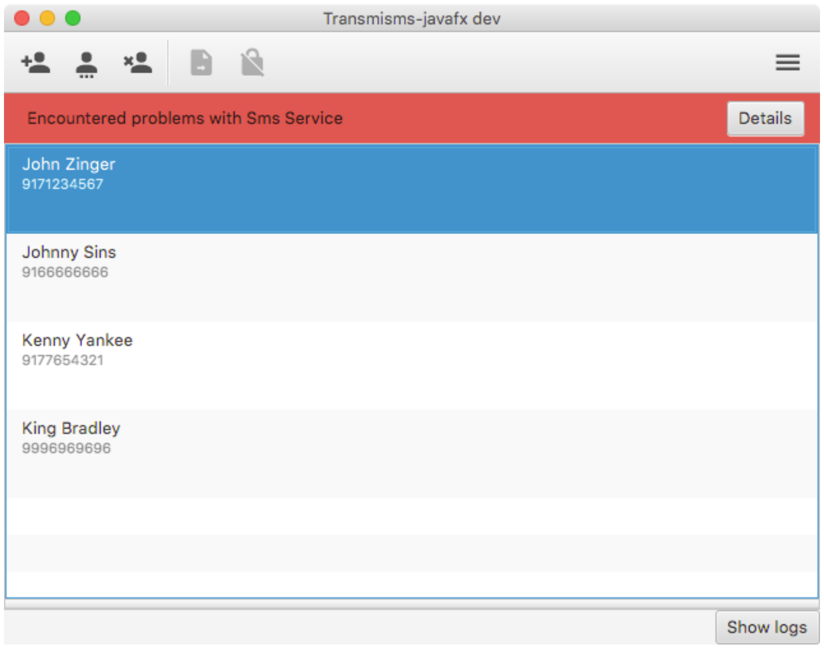
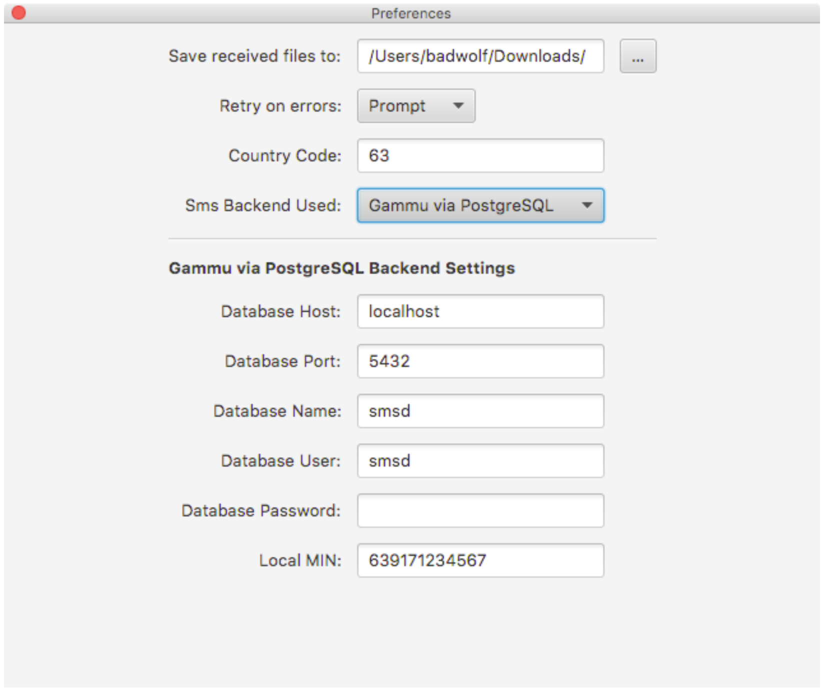


Figure 5. Detected Problems with SMS Service

The program automatically saves the files received on a predetermined location. The user can change this on the preferences window, along with SMS backend-specific settings.


Preferences

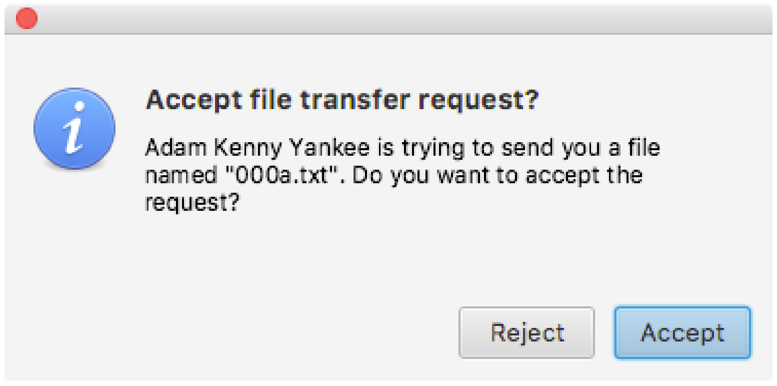
Save received files to:	/Users/badwolf/Downloads/
Retry on errors:	Prompt
Country Code:	63
Sms Backend Used:	Gammu via PostgreSQL

Gammu via PostgreSQL Backend Settings

Database Host:	localhost
Database Port:	5432
Database Name:	smsd
Database User:	smsd
Database Password:	
Local MIN:	639171234567

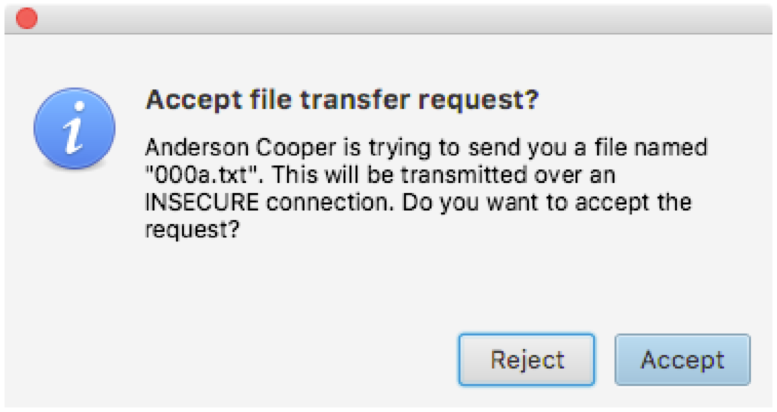
Figure 6. Preferences Window

The user will be prompted if he/she receives a file transfer request. A detailed prompt containing the senders' name, the sender's number if the sender is not yet in the user's contact list, the file name, and if the transfer will use a secure connection will be displayed.


Accept file transfer request?

Adam Kenny Yankee is trying to send you a file named "000a.txt". Do you want to accept the request?

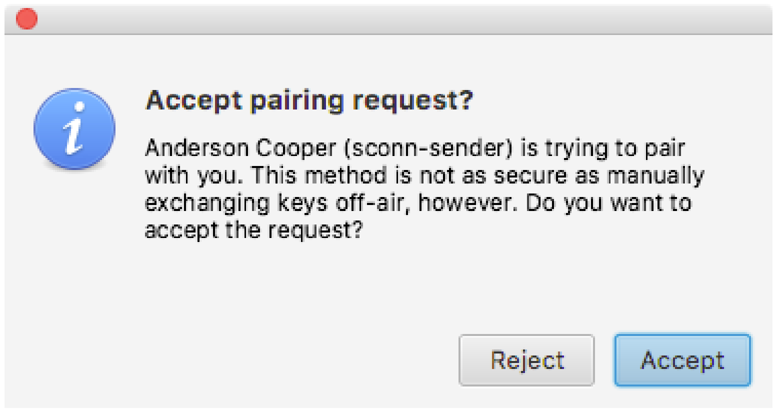
Figure 7. Secure File Transfer Request Prompt


Accept file transfer request?

Anderson Cooper is trying to send you a file named "000a.txt". This will be transmitted over an INSECURE connection. Do you want to accept the request?

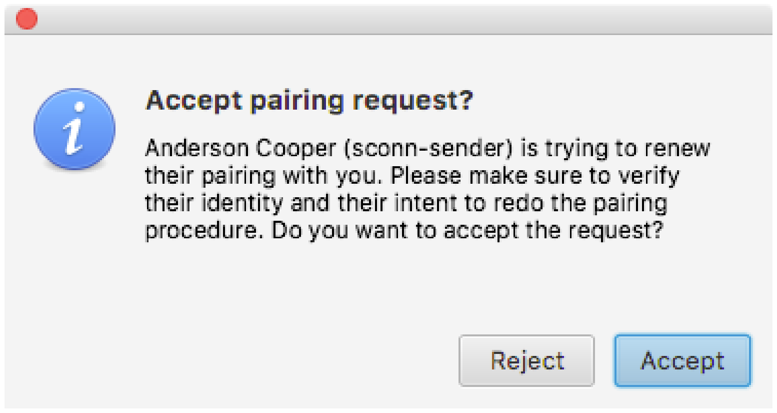
Figure 8. Unsecure File Transfer Request Prompt

The user will also be prompted in case he/she receives a pairing request. The sender's details are displayed on the prompt and if the pairing request has already been done before.


Accept pairing request?

Anderson Cooper (sconn-sender) is trying to pair with you. This method is not as secure as manually exchanging keys off-air, however. Do you want to accept the request?

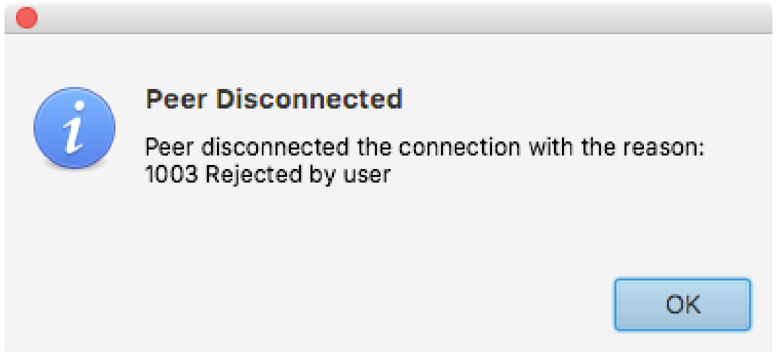
Figure 9. Pairing Request Prompt


Accept pairing request?

Anderson Cooper (sconn-sender) is trying to renew their pairing with you. Please make sure to verify their identity and their intent to redo the pairing procedure. Do you want to accept the request?

Figure 10. Renew Pairing Request Prompt

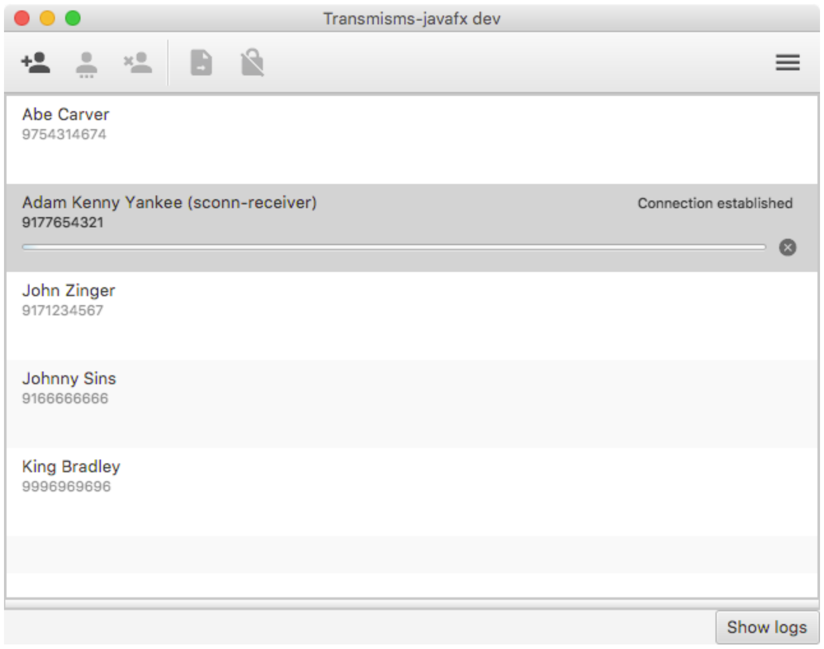
If the receiver rejects the file transfer or pairing request, the sender is prompted immediately that the receiver has rejected the request.


Peer Disconnected

Peer disconnected the connection with the reason:
1003 Rejected by user

Figure 11. Peer Reject/Disconnect Notification

Upon accepting the request, the connection will be established and the sender and receiver will then begin exchanging information. At this point, the progress of the key exchange or the file transfer will not yet be displayed.


Transmisms-javafx dev

Abe Carver
9754314674

Adam Kenny Yankee (sconn-receiver)
9177654321
Connection established

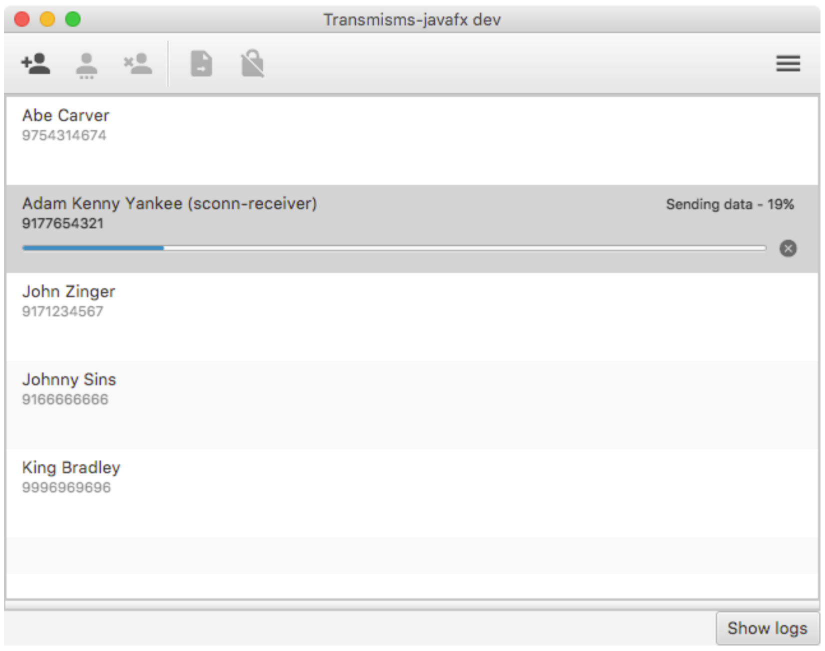
John Zinger
9171234567

Johnny Sins
9166666666

King Bradley
9996969696

Figure 12. Connection Established

After successfully exchanging information with each other, the sender and receiver will begin the transmission. The transmission progress will be both displayed on the sender's and receiver's GUI. The users will be then notified if the file transfer has been successful.


Transmisms-javafx dev

Abe Carver
9754314674

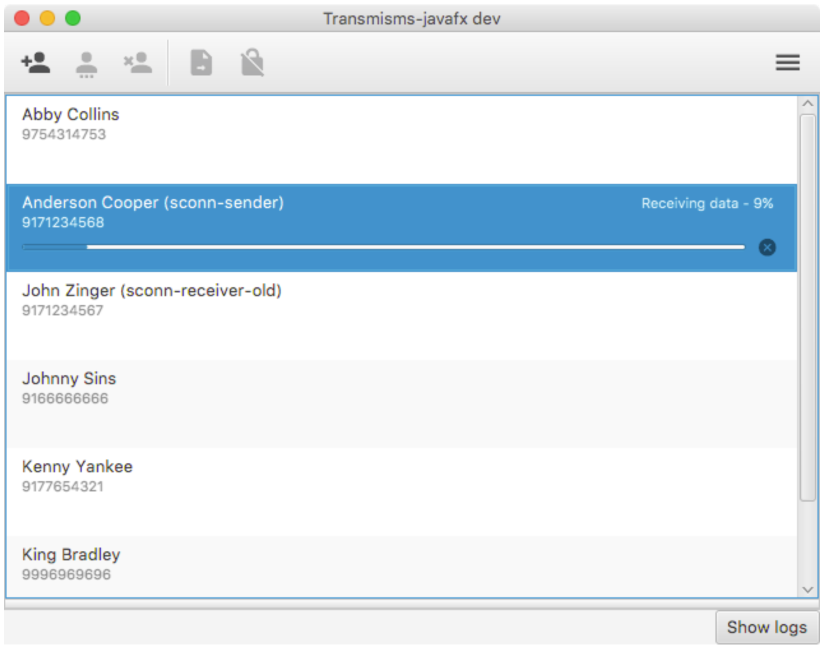
Adam Kenny Yankee (sconn-receiver)
9177654321
Sending data - 19%

John Zinger
9171234567

Johnny Sins
9166666666

King Bradley
9996969696

Figure 13. Sending File in Progress



Abby Collins
9754314753

Anderson Cooper (sconn-sender)
9171234568
Receiving data - 9%

John Zinger (sconn-receiver-old)
9171234567

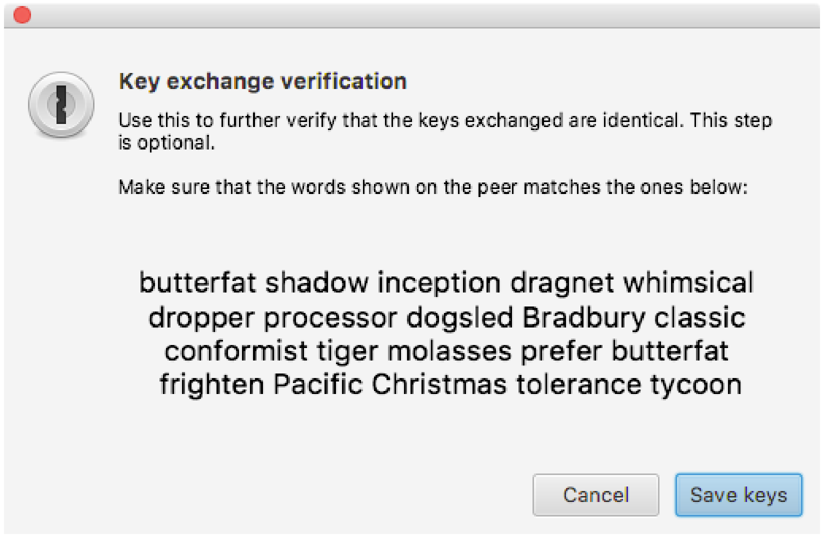
Johnny Sins
9166666666

Kenny Yankee
9177654321

King Bradley
9996969696

Figure 14. Receiving File in Progress

Another feature implemented on this program is enabling users to manually verify keys exchanged by comparing the displayed PGP words on their screen. This can be done over a secure channel or at least on a trusted voice channel. The words are designed to be distinct from each other even when dictated over a phone call. This prevents man-in-the-middle attacks by an adversary.


Key exchange verification

Use this to further verify that the keys exchanged are identical. This step is optional.

Make sure that the words shown on the peer matches the ones below:

butterfat shadow inception dragnet whimsical dropper processor dogsled Bradbury classic conformist tiger molasses prefer butterfat frighten Pacific Christmas tolerance tycoon

Figure 15. Key Exchange Verification using PGP Word List

VI. Discussions

The conceived TransmiSMS program and protocol aims to provide users a more accessible alternative for sharing digital data. It uses SMS as a data bearer to make transmission of files possible without the need for an Internet connection provided by a wired or wireless provider. It can even take advantage of 2G's larger coverage that wired, 3G, and 4G connections are not able to achieve.

Using a set of repeated rigid test cases and fault injection during the development process and testing, both simulated and real world tests have all been successful. The following scenarios are tested during the simulations and if possible, on real hardware:

- delayed delivery of messages
- out of order delivery of messages
- tests if timeouts due to a faulty component will trigger in a timely manner
- occasional sending of duplicate PDUs
- occasional sending of PDUs not appropriate for the receiver's current state
- occasional sending of PDUs with invalid CRCs
- occasional corruption of PDUs received
- occasional sending of unrecognizable PDUs

- sending of PDUs to the wrong intended recipient
- consistent sending of extra PDUs with wrong Session IDs during an operation
- sending of invalid security information
- all combinations of invalid, empty, truncated, and missing values for all PDUs used by the protocol

The system proposed and developed proved to be reliable and robust enough to survive all these problems presented to it.

While the system can cope up with hardware problems up to a limited extent, total hardware failure and severe network problems can still render it useless. For instance, if the SMS device used suddenly hangs up, the user can still intervene, reset the device, and retry the connection. Network outages, however, won't even let the system to send a single message in the first place.

Secure connections are also offered by the system as an additional feature. SMS, a relatively old technology, is not entirely secure when faced with an adversary equipped with specialized hardware such as a Stingray phone tracker and sniffer to perform a MiTM attacks. The protocol designed introduced security measures to protect sensitive data being transmitted from such adversaries. While attackers can read plaintext SMS data and possibly interrupt ongoing file transfers, integrity of the files and keys sent and received will not be compromised. These adversaries will never be able to successfully spoof the sender's or receiver's identity nor sniff the encrypted data as long as the security features are enabled.

Connection setup times are measured to be around 5 seconds up to a full minute. Transfer speeds can be variable as SMS transmission speeds can vary depending on the network condition or if the SMS device decides to slow down sending of messages. Assuming an uncompressed and unencrypted file is sent over transmission speeds of one SMS per 5 seconds, it can reach speeds up to 11.1 kbps sans the connection and teardown phases. That is around 20% of the maximum speed of a 56.6kbps dial-up modem.

Unfortunately, SMS's sluggishness as a data bearer still became apparent despite the fact that measures to reduce the total amount of data to be transmitted are done. However, such measures are still invaluable in making transmission speeds more palatable to the end user.

VII. Summary and Conclusions

A reliable and robust file transfer system over SMS has been developed to address the needs of users with limited or no access to technologies normally used to send and receive data. It was proven reliable and robust enough to survive problems related to SMS's unreliable nature on both simulated and real world tests.

Various methods to reduce the total size of data transmitted has been implemented. Additional security measures such as encryption and authentication has also been implemented to provide protection against well-funded adversaries.

Despite successes in file transmission over SMS, its speed remains as its greatest drawbacks. Setup and teardown times are less than ideal in some cases. On a very congested or faulty network, the system would probably be still running but would run on a greatly reduced speed.

It is not recommended to use this for everyday use if other faster technologies are readily available. This system can still prove useful as an effective alternative in remote places where such technologies won't be accessible or when physically transporting data to another location might be hazardous or impossible.

VIII. Recommendations

While the resulting protocol and system proved to be reliable and robust for file transfers via SMS messages, there are still other areas that can be further improved:

- Ability to pause and resume file transfer sessions especially for larger files.
- An Android implementation to eliminate the need for a separate computer when transferring files. The implementation must also consider quirks present when developing on Android. This includes reduced performance during sleep, applications and services being dynamically woken, paused, or killed on memory-intensive loads, and an entirely different UI framework that also affects decisions when implementing multithreaded tasks.
- The use of Content Providers, which are companies and organizations that are allowed by mobile network operators to send and receive SMS messages at a different priority and at a faster rate. The source code is made to be easily extensible in case a need to implement a new SMS Service for a new device or backend arises.
- Further improvements to speed up connection setup time and teardown.
- Fully secured communications which might be possibly provided using open-source projects such as Signal and Silence

- Applications other than file sharing that requires exchange of binary data.
- Possible reuse of the protocol for future iterations and successors of SMS over future generation networks.

Bibliography

[1] Augustyniak, P. & Tadeusiewicz, R. (2009). *Ubiquitous Cardiology: Emerging Wireless Telemedical Applications: Emerging Wireless Telemedical Applications.* IGI Global.

[2] Collin, L. (2005). A Quick Benchmark: GZIP vs BZIP2 vs LZMA. Retrieved from https://tukaani.org/lzma/benchmarks.html

[3] Collin, L., (2016). XZ Utils. Retrieved from https://tukaani.org/xz/

[4] Čihař, M. (2018). [GW]ammu. Retrieved from https://wammu.eu/gammu/

[5] Dimacali, T. (2010). Philippines still Text Messaging Champ -US Study. GMA News. Retrieved from http://www.gmanetwork.com/news/scitech/content/19883 2/philippines-still-text-messaging-champ-us-study/story/

[6] Dykimching, A.M., Lee, J.A.A., & Yu, W.E. (2011). A Study on the aspect of data transmission on SMS interface. *Journal of Information Security Research, vol. 2, no. 1, pp. 30-39.*

[7] Globe Telecom (2014). Globe Telecom completes HSPA+ rollout; network now 100% 4G. *Globe Telecom Press Release.* Retrieved from http://www.globe.com.ph/press-room/network-now-100-4g

[8] International Telecommunication Union (2010). ITU World Radiocommunication Seminar highlights future communication technologies. Retrieved from https://www.itu.int/net/pressoffice/press_releases/2010/48.aspx

[9] Internet World Stats (2017). Asia Marketing Research, Internet Usage, Population Statistics and Facebook Subscribers. Retrieved from https://www.internetworldstats.com/asia.htm#ph

[10] Klausmann, T. (2008). GZIP, BZIP2, and LZMA Compared. Retrieved from https://web.archive.org/web/20130106193958/blog.i-no.de//archives/2008/05/08/index.html

[11] McKinsey & Company (2015). Offline and falling behind: Barriers to Internet adoption. Retrieved from https://www.mckinsey.com/industries/high-tech/our-insights/offline-and-falling-behind-barriers-to-internet-adoption

[12] Meng, X., Zerfos, P., Samanta, V., Wong, S.H.Y., & Lu, S. (2007). Analysis of the Reliability of a Nationwide Short Message Service. *26th IEEE International Conference on Computer Communications, pp. 1811-1819.*

[13] OpenSignal (2016). Global State of Mobile Networks, August 2016. Retrieved from https://opensignal.com/reports/2016/08/global-state-of-the-mobile-network

[14] OpenSignal (2018). The State of LTE, February 2018. Retrieved from https://opensignal.com/reports/2018/02/state-of-lte

[15] Philippine Statistics Authority (2017). *Philippines in Figures 2017 p. 11.* Retrieved from https://psa.gov.ph/sites/default/files/PIF_2017.pdf

[16] Postel, J., & Reynolds, J. (1985). RFC 959. FILE TRANSFER PROTOCOL (FTP)(Oct. 1985), Retrieved from ftp://ftp.rfc-editor.org/innotes/rfc959.txt

[17] QaamGo Media GmbH. Average File Sizes. Retrieved from https://blog.online-convert.com/average-file-sizes

[18] Rao, S., Vora, R., Dhar, S., Salvi, S., & Kumar, V. (2008). Development of a Transport Layer using SMS. *7th International Conference on Cognitive Systems, India.*

[19] Sam, A. (2009). Free Wireless Internet through SMS. *ADSA Life Development, Sweden.*

[20] Scharnhorst, W., Hilty, & L., Jolliet, O. (2006). Life Cycle Assessment of Second Generation (2G) and Third Generation (3G) Mobile Phone Networks. *Environment International 32.5 (2006) pp. 656-675.*

[21] Shirali-Shahreza, M. & Shirali-Shahreza, S. (2009). Sending Pictures by SMS. *2009 11th International Conference on Advanced Communication Technology.*

[22] Tolstrup, M. (2015). *Indoor Radio Planning: A Practical Guide for 2G, 3G and 4G pp. 328-329, 349-350.* John Wiley & Sons.

[23] Yahoo! & The Nielsen Company (2009). Yahoo Nielsen Internet Habits Study for the Philippines. *Yahoo!-Nielsen Press Release.* Retrieved from https://www.scribd.com/doc/13661672/Yahoo-Nielsen-Internet-Habits-Study-for-the-Philippines

[24] *Collins English Dictionary - Complete & Unabridged 2012 Digital Edition.* William Collins Sons & Co. Ltd., HarperCollins Publishers.

[25] *GSM Doc 28/85 "Services and Facilities to be provided in the GSM System"* rev2, June 1985

[26] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990

[27] Reliable Request-Reply Patterns. *ØMQ - The Guide.* Retrieved from http://zguide.zeromq.org/page:all#reliable-request-reply

Appendix

A. Source Code

Dummy SMS Service Backend and Simulator:

```
from gevent import monkey; monkey.patch_all() # monkey all the things!
import zmq.green as zmq
import gevent



initiator_address = 'tcp://127.0.0.1:8768'
responder_address = 'tcp://127.0.0.1:8769'
mediator_address = 'inproc://dummymediator'



def filter_ping(address1, address2, identity):
    ctx = zmq.Context.instance()
    s1 = ctx.socket(zmq.PAIR)
    s1.bind(address1)
    s1.linger = 0

    s2 = ctx.socket(zmq.PAIR)
    if(identity == 1):
        s2.bind(address2)
    else:
        s2.connect(address2)
    s2.linger = 0

    while True:
        r = s1.poll(50)
        if r != 0:
            msg = s1.recv()
            if msg == "PING":
                s1.send("PONG")
            else:
                s2.send(msg)
        r = s2.poll(50)
        if r != 0:
            msg = s2.recv()
            s1.send(msg)
    print "end of " + identity


if __name__ == "__main__":
    g1 = gevent.spawn(filter_ping, initiator_address, mediator_address, 1)
    g2 = gevent.spawn(filter_ping, responder_address, mediator_address, 2)
    gs = [g1,g2]

    gevent.joinall(gs)
```

JavaFx UI:

```
--------------------------------------------------------------
Filename: build.gradle
--------------------------------------------------------------

apply plugin: 'java' // add java tasks
apply plugin: 'eclipse' // create eclipse project
apply plugin: 'application' // prepare for packaging as application
apply plugin: 'gradle-one-jar' // for single self-contained jar outputs

// java jar configuration (manifest.mf in jar)
//sourceCompatibility = '1.7'
sourceCompatibility = '1.8'
version = '0.9'
mainClassName = 'com.transmisms.ui.javafx.Main'
applicationDefaultJvmArgs = [] // set as empty for now

/*
compileJava {
    //options.compilerArgs << "-Xlint:unchecked" << "-Xlint:deprecation"
    options.compilerArgs << "-Xlint:all"
}
*/

jar {
    manifest {
        attributes 'Implementation-Title': 'Transmisms-core',
                   'Implementation-Version': version,
                   'Main-Class': mainClassName
    }
    exclude 'META-INF/*.RSA', 'META-INF/*.SF','META-INF/*.DSA' // exclude
digests to avoid problems
}

task fatJar(type:OneJar) {
    mainClass = mainClassName
}

artifacts {
    archives(jar) {
        group 'com.transmisms.smsftp'
        name 'smsftp-protocol'
    }
}

repositories {
    jcenter() // because we love https and supersets
    mavenCentral()
    mavenLocal()
}

configurations {
    integrationCompile.extendsFrom testCompile
    integrationRuntime.extendsFrom testRuntime
    stagingCompile.extendsFrom testCompile
    stagingRuntime.extendsFrom testRuntime
}

buildscript {
    repositories {
        jcenter() // because we love https and supersets
        mavenCentral()
        mavenLocal()
    }

    dependencies {
        classpath 'com.github.rholder:gradle-one-jar:1.0.4'
    }
}

dependencies {
    compile group: 'com.github.rholder', name: 'gradle-one-jar', version:
'1.0.4'

    compile group: 'org.zeromq', name: 'jeromq', version: '0.4.3'
    compile group: 'com.google.code.gson', name: 'gson', version: '2.8.1'
    compile group: 'org.apache.commons', name: 'commons-io', version: '1.3.2'
    compile group: 'commons-collections', name: 'commons-collections', version:
'3.2'
    compile group: 'commons-codec', name: 'commons-codec', version: '1.10'
    compile group: 'org.apache.commons', name: 'commons-compress', version:
'1.10'
    compile group: 'org.tukaani', name: 'xz', version: '1.5'
    compile group: 'org.apache.pdfbox', name: 'pdfbox', version: '1.8.11'
    compile group: 'net.sourceforge.javaflacencoder', name:
'java-flac-encoder', version: '0.3.7'
    compile group: 'org.apache.logging.log4j', name: 'log4j-core', version:
'2.1'
    compile group: 'org.apache.logging.log4j', name: 'log4j-api', version: '2.1'
    compile group: 'com.fasterxml.jackson.dataformat', name:
'jackson-dataformat-yaml', version: '2.5.5'
    compile group: 'org.yaml', name: 'snakeyaml', version: '1.12'
    compile group: 'technology.zeroalpha.security', name: 'j-pgp-wordlist',
version: '0.1.1'
    compile group: 'org.kordamp.ikonli', name: 'ikonli-javafx', version: '1.9.0'
    compile group: 'org.kordamp.ikonli', name: 'ikonli-material-pack', version:
'1.9.0'
    compile group: 'org.kordamp.ikonli', name: 'ikonli-materialdesign-pack',
version: '1.9.0'
    compile group: 'org.postgresql', name: 'postgresql', version: '42.1.4'


    // NOTE: version lock on 1.56 due to constant API changes
    //       See src/main/java/com/transmisms/core/util/crypto/RSA.java
    compile group: 'org.bouncycastle', name: 'bcprov-jdk15on', version: '1.56'
    compile group: 'org.bouncycastle', name: 'bcpkix-jdk15on', version: '1.56'
    testCompile group: 'junit', name: 'junit', version: '4.12'
    testCompile group: 'org.testng', name: 'testng', version: '6.9.6'
    testCompile group: 'org.mockito', name: 'mockito-core', version: '1.10.19'

    integrationCompile sourceSets.main.output
    integrationCompile sourceSets.test.output
    integrationCompile configurations.testCompile
    integrationCompile configurations.testRuntime

    stagingCompile sourceSets.main.output
    stagingCompile sourceSets.test.output
    stagingCompile configurations.testCompile
    stagingCompile configurations.testRuntime
}

test {
    include "**/**Suite.class"
}

sourceSets {
    test {
        java.srcDir 'src/test/unit/java'
    }

    integration {
        java.srcDir file('src/test/integration/java')
        resources.srcDir file('src/test/resources')
    }

    staging {
        java.srcDir file('src/test/staging/java')
        resources.srcDir file('src/test/resources')
    }
}

task integration(type: Test) {
    group 'Verification'
    description 'Runs the integration tests.'
    useTestNG { // enable TestNG
        //parallel = "classes"
        parallel = "methods"
        threadCount = 4
    }
    testClassesDir = sourceSets.integration.output.classesDir
    // NOTE: use the following instead if you're using gradle 5.0 and above
    //testClassesDirs = sourceSets.integration.output.classesDirs
    classpath = sourceSets.integration.runtimeClasspath
}

task staging(type: Test) {
    group 'Verification'
    description 'Runs the staging tests.'
    useTestNG { // enable TestNG
        //parallel = "classes"
        parallel = "classes"
        threadCount = 4
    }
    testClassesDir = sourceSets.staging.output.classesDir
    // NOTE: use the following instead if you're using gradle 5.0 and above
    //testClassesDirs = sourceSets.staging.output.classesDirs
    classpath = sourceSets.staging.runtimeClasspath
}


tasks.withType(Test) {
    // experiment with different values to see what works best
    maxParallelForks = 8
    // can also compute dynamically
    // maxParallelForks = Runtime.runtime.availableProcessors() / 2
}

javadoc {
    title = 'Transmisms API'
}


uploadArchives {
    repositories {
        flatDir {
            name 'local'
            dirs '../repos'
       }
    }
}

// this was added to demonstrate adding -X arguments to `javac`
/*compileTestJava {
    options.compilerArgs << "-Xlint:unchecked" // for unsafe thing javac is
complaining
}*/
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/BinaryPDUDecoder.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import com.transmisms.core.protocol.CoreProtocolFacade;
import com.transmisms.core.protocol.CoreProtocolFacade.DataSizes;
import com.transmisms.core.protocol.CoreProtocolFacade.DataStrSizes;
import com.transmisms.core.protocol.PDU;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.core.protocol.PDUMalformedException;
import com.transmisms.core.protocol.InvalidCRCException;
import com.transmisms.smsftp.protocol.SmsftpPDUType;

import com.transmisms.core.util.codec.Base85;
import net.sourceforge.javaflacencoder.CRC8;

import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.function.Function;

import java.io.IOException;
import java.io.UnsupportedEncodingException;


public class BinaryPDUDecoder {
    private Function<Character, PDUType> prefixToTypeFunc;
    private Function<PDUType, Character> typeToPrefixFunc;

    private static Function<Character, PDUType> basePrefixToTypeFunc;
    private static Function<PDUType, Character> baseTypeToPrefixFunc;

    private static final Map<Character, PDUType> prefixToTypeLookup =
            new Hashtable<>();
    private static final Map<PDUType, Character> typeToPrefixLookup =
            new Hashtable<>();
    // utility function for use for the maps above
    private static void biDiAddToLookup(Character c, PDUType p) {
        PDUType evalValue = BinaryPDUDecoder.prefixToTypeLookup.put(c, p);
        assert null == evalValue;
        Character evalValue2 = BinaryPDUDecoder.typeToPrefixLookup.put(p, c);
        assert null == evalValue2;
    }

    private static BinaryPDUDecoder thisInstance;
    static {
        // initialize instance
        BinaryPDUDecoder.thisInstance = new BinaryPDUDecoder(null, null);
        BinaryPDUDecoder.basePrefixToTypeFunc =
                (c) -> BinaryPDUDecoder.prefixToTypeLookup.get(c);
        BinaryPDUDecoder.baseTypeToPrefixFunc =
                (p) -> BinaryPDUDecoder.typeToPrefixLookup.get(p);

        // populate lookup maps
        BinaryPDUDecoder.biDiAddToLookup('r',
                CorePDUType.CORE_RETRANSMISSION);
        BinaryPDUDecoder.biDiAddToLookup('f',  CorePDUType.CORE_FIN);
        BinaryPDUDecoder.biDiAddToLookup('F',  CorePDUType.CORE_FINACK);
    }


    protected BinaryPDUDecoder(Function<Character, PDUType> prefixToTypeFunc,
            Function<PDUType, Character> typeToPrefixFunc) {
        this.prefixToTypeFunc = prefixToTypeFunc;
        this.typeToPrefixFunc = typeToPrefixFunc;
    }

    public static BinaryPDUDecoder getInstance() {
        return BinaryPDUDecoder.thisInstance;
    }

    public final PDUType prefixToPduType(Character prefix) {
        PDUType retType =
                BinaryPDUDecoder.basePrefixToTypeFunc.apply(prefix);
        if(retType == null && this.prefixToTypeFunc != null) {
            return this.prefixToTypeFunc.apply(prefix);
        }
        return retType;
    }
```

```
public final Character pduTypeToPrefix(PDUType pduType) {
    Character retPrefix =
            BinaryPDUDecoder.baseTypeToPrefixFunc.apply(pduType);
    if(retPrefix == null && this.typeToPrefixFunc != null) {
        return this.typeToPrefixFunc.apply(pduType);
    }
    return retPrefix;
}

public static int getHexSegmentId(String body, int maxSegmentCount,
        String encodedStr, String pduTypeName)
        throws PDUMalformedException {
    return getHexSegmentId(body, maxSegmentCount, encodedStr, pduTypeName,
            false);
}

public static int getHexSegmentId(String body, int maxSegmentCount,
        String encodedStr, String pduTypeName, boolean allowZeroId)
        throws PDUMalformedException {
    String segmentIdStr = body.substring(0,
            DataStrSizes.HEXSEGMENTID.getSize());

    Integer segmentId = -1; // preeempt for error, just in case
    boolean invalidNumberFormat = false;
    try {
        segmentId = Integer.valueOf(segmentIdStr, 16);
    }
    catch(NumberFormatException e) {
        invalidNumberFormat = true;
    }
    if(segmentId < 0 || (!allowZeroId && segmentId == 0)
            || segmentId > maxSegmentCount
            || invalidNumberFormat) {
        throw new PDUMalformedException(
                "Invalid " +
                ((pduTypeName == null) ? "" : pduTypeName+" ") +
                "Segment ID: " + segmentIdStr, encodedStr);
    }
    return segmentId;
}

public static UUID base85ToUUID(String b85String, String encodedStr,
        String pduTypeName)
        throws PDUMalformedException {
    // extract byte[] format from base85
    byte[] uuidBytes = null;
    try {
        uuidBytes = Base85.decode(b85String);
    }
    catch(IOException e) {
        throw new PDUMalformedException(
               pduTypeName + "'s Session ID is corrupt", encodedStr);
    }
    //  verify if sessionId's length is correct
    if(uuidBytes.length != DataSizes.SESSIONID.getSize()) {
        throw new PDUMalformedException(
               pduTypeName + "'s Session ID's byte length is incorrect",
               null);
    }

    // split the byte[] into msb and lsb
    byte[] msb = Arrays.copyOfRange(uuidBytes, 0, 8);
    byte[] lsb = Arrays.copyOfRange(uuidBytes, 8, 16);
    // convert the msb and lsb byte[] into long and return afterwards
    ByteBuffer msbBuffer = ByteBuffer.allocate(Long.BYTES).put(msb);
    ByteBuffer lsbBuffer = ByteBuffer.allocate(Long.BYTES).put(lsb);
    msbBuffer.flip(); // required for reading
    lsbBuffer.flip(); // required for reading
    return new UUID(msbBuffer.getLong(), lsbBuffer.getLong());
}

public static int base85ToInteger(String b85String, String encodedStr,
        String pduTypeName)
        throws PDUMalformedException {
    // extract bytes first from base85
    byte[] segmentNumBytes = null;
    try {
        segmentNumBytes = Base85.decode(b85String);
    }
    catch(IOException e) {
        throw new PDUMalformedException(
                pduTypeName + "'s Segment Id/Count is corrupt",
                encodedStr);
    }
    ByteBuffer bb = ByteBuffer.allocate(Integer.BYTES);
    // pad with zeroes if there's still space for Integer
    for(int i = 0; i < Integer.BYTES - segmentNumBytes.length; i++) {
        bb.put((byte)0);
    }
    // then add the bytes
    bb.put(segmentNumBytes).flip(); // flip() is required for reading
    return bb.getInt(); // then return the value
}

protected List<Object> decodeBody(
        PDUType pduType, String encoded)
        throws PDUMalformedException {
    return null;
}

private final List<Object> defaultDecodeBody(
        PDUType pduType, String encoded)
        throws PDUMalformedException {
    String body = encoded.substring(1, encoded.length()-2);
    List<Object> decodedData = new ArrayList<>();

    if(pduType instanceof CorePDUType) {
        CorePDUType cPduType = (CorePDUType)pduType;
        switch(cPduType) { // fill up data depending on the pduType
            case CORE_FIN: {
                // 1st form: f/<session_id>
                // check if size is less than SESSIONID
                if(body.length() < DataStrSizes.SESSIONID.getSize()) {
                    throw new PDUMalformedException(
                            "CORE_FIN too short to fit Session ID",
                            encoded);
                }
                String sessionIdStr = body.substring(0,
                        DataStrSizes.SESSIONID.getSize());
                // decode sessionId from here
                UUID sessionId = base85ToUUID(sessionIdStr, encoded,
                        "CORE_FIN");
                decodedData.add(sessionId); // and add to decoded data

                if(body.length() > DataStrSizes.SESSIONID.getSize()) {
                    // 2nd form a: f/<session_id>/<status_code>/<o_message>
                    // 2nd form b:
f/<session_id>/<status_code>/<user_message>

                    // check if size is less than SESSIONID+STATUSCODE
                    if(body.length() < DataStrSizes.SESSIONID.getSize() +
                            DataStrSizes.STATUSCODE.getSize()) {
                        throw new PDUMalformedException(
                                "CORE_FIN too short to fit Status Code",
                                encoded);
                    }
                    // get status code
                    String statusCode = body.substring(
                            DataStrSizes.SESSIONID.getSize(),
                            DataStrSizes.SESSIONID.getSize()+
                            DataStrSizes.STATUSCODE.getSize());
                    if(CoreProtocolFacade.getStatusMessage(
                            statusCode) == null) {
                        throw new PDUMalformedException(
                                "Invalid status code from " +
                                pduType.toString() + ": " + statusCode,
                                encoded);
                    }
                    decodedData.add(statusCode);
                    // check if there is an optional message
                    String message = null;
                    if(body.length() > DataStrSizes.SESSIONID.getSize() +
                            DataStrSizes.STATUSCODE.getSize()) {
                        message = body.substring(
                            DataStrSizes.SESSIONID.getSize()+
                            DataStrSizes.STATUSCODE.getSize(),
                            body.length());
                        decodedData.add(message);
                    }
                }
                break;
            }


            case CORE_FINACK: {
                // check if size is less than SESSIONID
                if(body.length() < DataStrSizes.SESSIONID.getSize()) {
                    throw new PDUMalformedException(
                            "CORE_FINACK too short to fit Session ID",
                            encoded);
                }
                // try to parseSessionId
                String sessionIdStr = body.substring(0,
                        DataStrSizes.SESSIONID.getSize());

                UUID sessionId = base85ToUUID(sessionIdStr, encoded,
                        "CORE_FINACK");
                decodedData.add(sessionId); // and add to decoded data
                break;
            }
```

```
                case CORE_RETRANSMISSION: {
                    AtomicInteger pointer = new AtomicInteger(0);

                    while(pointer.get() < body.length()) {
                        char retPrefix = body.charAt(
                                pointer.getAndIncrement());
                        PDUType retPduType = this.prefixToPduType(retPrefix);

                        if(retPduType != null) { // if valid
                            Object segmentId = null;
                            try {
                                if(retPduType == SmsftpPDUType.SMSFTP_DATA) {
                                    String b85SegmentIdStr = body.substring(
                                            pointer.getAndAdd(
                                            DataStrSizes.SEGMENTID.getSize()),
                                            pointer.get());
                                    Integer b85SegmentId = base85ToInteger(
                                            b85SegmentIdStr, encoded,
                                            pduType.toString());
                                    segmentId = b85SegmentId;
                                }
                                else {
                                    String hexSegmentIdStr = body.substring(
                                            pointer.getAndAdd(
DataStrSizes.HEXSEGMENTID.getSize()),
                                            pointer.get());
                                    Integer hexSegmentId = getHexSegmentId(
                                            hexSegmentIdStr, 256, encoded,
                                            pduType.toString(), true);
                                    segmentId = hexSegmentId;
                                }
                            }
                            // do nothing on exceptions
                            catch(PDUMalformedException e) {}
                            catch(StringIndexOutOfBoundsException e) {}
                            // add if no errors encountered
                            if(segmentId != null) {
                                decodedData.add(retPduType);
                                decodedData.add(segmentId);
                            }
                        }
                        else { // just increment and go
                            pointer.getAndAdd(
                                    DataStrSizes.HEXSEGMENTID.getSize());
                            // assume all other PDUs will be hex
                        }
                    }

                    break;
                }
                default: {
                    // code should NEVER reach here
                    throw new AssertionError("Uncaught PDU Type: " + pduType);
                }
            }
        }
        else {
            throw new PDUMalformedException("Uncaught PDU Type: " + pduType,
                    encoded);
        }
        return decodedData;
    }

    public final PDU decodeBinary(String encoded)
            throws PDUMalformedException, InvalidCRCException {
        List<Object> decodedData = new ArrayList<>();

        // some sanity checks before parsing
        if(encoded == null) {
            throw new NullPointerException(
                    "Binary encoded string cannot be null");
        }
        else if(encoded.equals("")) {
            throw new PDUMalformedException(
                    "Binary encoded string cannot be empty", encoded);
        }
        else if(encoded.length() < DataStrSizes.CRC.getSize()+1) {
            throw new PDUMalformedException("Binary encoded string too short",
                    encoded);
        }

        try { // determine if checksum matched PDU body
            // it doesn't matter that much if the last 2 chars of the string is
            //      a valid hex string; they are just compared as strings
            //      anyways
            String givenCRCStr = encoded.substring(encoded.length()-
                    DataStrSizes.CRC.getSize());
            CRC8 crc = new CRC8();
            String crcBody = encoded.substring(0, encoded.length()-2);
            crc.updateCRC8(crcBody.getBytes("UTF-8"), 0,
                    crcBody.getBytes("UTF-8").length);
            byte computedCRC = crc.checksum();
            String computedCrcStr = String.format("%02x", computedCRC);
            if(!givenCRCStr.equalsIgnoreCase(computedCrcStr)) {
                throw new InvalidCRCException("Invalid CRC! Got: " +
                        givenCRCStr + "; Expected: " + computedCrcStr,
                        encoded);
            }
        }
        catch(UnsupportedEncodingException e) {
            // this should never happen; crash early
            throw new IllegalStateException(e.getMessage(), e);
        }

        String body = encoded.substring(1, encoded.length()-2);
        // just to make sure that pduType is "initialized"
        PDUType pduType = null;
        { // get PDU type
            char firstChar = encoded.charAt(0);
            pduType = this.prefixToPduType(firstChar);
            if(pduType == null) {
                throw new PDUMalformedException(
                        "Invalid Binary PDU Prefix: " + firstChar, encoded);
            }
        }

        try {
            decodedData.addAll(
                    this.defaultDecodeBody(pduType, encoded));
        }
        catch(PDUMalformedException e) {
            List<Object> l = this.decodeBody(pduType, encoded);
            if(l != null) {
                decodedData.addAll(l);
            }
            else {
                throw e;
            }
        }

        return new PDU(pduType, decodedData.toArray());
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/Connection.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.atomic.AtomicReference;


public class Connection {
    public enum Role { INITIATOR, RESPONDER };

    public enum ConnectionState {
            CLOSED,
            ESTABLISHED,

            LISTENING,
            REQUEST_RECEIVED,

            REQUEST_SENT,
            REQUEST_SENT_2, // ignore for now
            REQUEST_SENT_3, // ignore for now

            CLOSE_WAIT,
            FIN_WAIT,
            CLOSING_1,
            CLOSING_2,
            TIME_WAIT
    };

    public final Role role;
    public final String localMin;
    public String peerMin;
    public int maxPduLifetime = 60000;

    private final AtomicReference<ConnectionState> connectionStateRef =
            new AtomicReference<>(ConnectionState.CLOSED);
    private final AtomicInteger errorCount = new AtomicInteger(0);

    // buffer variables for use by core protocol
    protected PDUType lastPduBeforeEst = null;
    protected String finStatusComboReceived;
    protected String encodedRepPdu;
    protected String encodedHeadPdu;
    protected String encodedFinPdu;
    protected String encodedFinackPdu;

    private Connection(String localMin, String peerMin, Role role) {
        this.localMin = localMin;
        this.peerMin = peerMin;
        this.role = role;
    }

    public static Connection createResponderConnection(String localMin) {
        return new Connection(localMin, null, Role.RESPONDER);
    }
    public static Connection createInitiatorConnection(String localMin,
            String peerMin) {
        return new Connection(localMin, peerMin, Role.INITIATOR);
    }



    public int getErrorCount() {
        return this.errorCount.get();
    }

    public int incrementErrorCount() {
        return this.errorCount.incrementAndGet();
    }

    public ConnectionState getConnectionState() {
        return this.connectionStateRef.get();
    }

    public void setConnectionState(ConnectionState connectionState) {
        if(connectionState == null) {
            throw new NullPointerException("Connection state cannot be null");
        }
        this.connectionStateRef.set(connectionState);
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/CoreLookupTable.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import com.transmisms.core.protocol.Connection.ConnectionState;
import com.transmisms.core.protocol.Presenter.MessageType;
import com.transmisms.core.protocol.Presenter.PromptType;
import com.transmisms.core.protocol.Presenter.StatusType;

import java.util.Hashtable;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.concurrent.Future;
import java.util.function.BiFunction;



public class CoreLookupTable {
    private static final Map<PdlEntry,
            BiFunction<CoreProtocolFacade, PDU, Runnable>> lookupTable =
            new Hashtable<>();

    static { // initialize lookupTable
        BiFunction<CoreProtocolFacade, PDU, Runnable> evalValue =
                null; // only used for asserts
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.LISTENING,
                CorePDUType.CORE_CONNECTION_REQUEST),
                (t, u) -> CoreLookupTable.createRequestReceivedTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.REQUEST_RECEIVED,
                CorePDUType.CORE_CONNECTION_REQUEST),
                (t, u) ->
                        CoreLookupTable.createResendCoreRepTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.REQUEST_RECEIVED,
                CorePDUType.CORE_FIN), // from extended Handshake FSM draft
                (t, u) ->
                        CoreLookupTable.createResendCoreRepTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.REQUEST_RECEIVED, CorePDUType.CORE_HEAD),
                (t, u) ->
                        CoreLookupTable.createEstConnAsResponderTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.REQUEST_SENT,
                CorePDUType.CORE_CONNECTION_RESPONSE),
                (t, u) ->
                        CoreLookupTable.createEstConnAsInitiatorTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.ESTABLISHED,
                CorePDUType.CORE_CONNECTION_RESPONSE),
                (t, u) ->CoreLookupTable.createResendCoreHeadIfValidTask(
                t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.ESTABLISHED, CorePDUType.CORE_HEAD),
                (t, u) -> CoreLookupTable.createDoNothingTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.ESTABLISHED, CorePDUType.CORE_FIN),
                (t, u) -> CoreLookupTable.createFinReceivedTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.FIN_WAIT, CorePDUType.CORE_FIN),
                (t, u) -> CoreLookupTable.createSimulFinReceivedTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.FIN_WAIT, CorePDUType.CORE_FINACK),
                (t, u) ->
                        CoreLookupTable.createFinwaitDisconnectTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.FIN_WAIT, CorePDUType.CORE_LAST_ACK),
                (t, u) -> CoreLookupTable.createSimulLastackReceivedTask(
                t, u));
        assert null == evalValue;
        evalValue = lookupTable.put( // from extended Termination FSM draft
                new PdlEntry(
                ConnectionState.FIN_WAIT,
                CorePDUType.CORE_CONNECTION_RESPONSE),
                (t, u) ->
                        CoreLookupTable.createResendCoreHeadIfValidTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.CLOSE_WAIT, CorePDUType.CORE_FIN),
                (t, u) -> CoreLookupTable.createResendFinackTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.CLOSE_WAIT, CorePDUType.CORE_LAST_ACK),
                (t, u) -> CoreLookupTable.createDisconnectTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.CLOSING_1, CorePDUType.CORE_FIN),
                (t, u) -> CoreLookupTable.createResendFinTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.CLOSING_1, CorePDUType.CORE_LAST_ACK),
                (t, u) ->
                        CoreLookupTable.createClosing1DisconnectTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.CLOSING_2, CorePDUType.CORE_FIN),
                (t, u) ->
                        CoreLookupTable.createClosing2DisconnectTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.CLOSING_2, CorePDUType.CORE_LAST_ACK),
                (t, u) -> CoreLookupTable.createResendFinTask(t, u));
        assert null == evalValue;
        evalValue = lookupTable.put(
                new PdlEntry(
                ConnectionState.TIME_WAIT, CorePDUType.CORE_FIN),
        (t, u) -> CoreLookupTable.createResendLastackTask(t, u));
assert null == evalValue;
evalValue = lookupTable.put(
        new PdlEntry(
        ConnectionState.TIME_WAIT, CorePDUType.CORE_FINACK),
        (t, u) -> CoreLookupTable.createResendLastackTask(t, u));
assert null == evalValue;

//// cases for retransmission
ConnectionState[] retransStates = { ConnectionState.ESTABLISHED,
                                  ConnectionState.CLOSE_WAIT,
                                  ConnectionState.FIN_WAIT,
                                  ConnectionState.CLOSING_1,
                                  ConnectionState.CLOSING_2,
                                  ConnectionState.TIME_WAIT };
for(ConnectionState cs : retransStates) {
    evalValue = lookupTable.put(new PdlEntry(cs,
            CorePDUType.CORE_RETRANSMISSION),
            (t, u) -> CoreLookupTable.createRetransmissionTask(t, u));
    assert null == evalValue;
}

//// error cases for lookupTable
PdlEntry[] errorEntries = {
        new PdlEntry(
        ConnectionState.LISTENING,
        CorePDUType.CORE_CONNECTION_RESPONSE),
        new PdlEntry(
        ConnectionState.LISTENING, CorePDUType.CORE_HEAD),
        new PdlEntry(
        ConnectionState.LISTENING, CorePDUType.CORE_FIN),
        new PdlEntry(
        ConnectionState.LISTENING, CorePDUType.CORE_FINACK),
        new PdlEntry(
        ConnectionState.LISTENING, CorePDUType.CORE_LAST_ACK),
        new PdlEntry(
        ConnectionState.REQUEST_RECEIVED,
        CorePDUType.CORE_CONNECTION_RESPONSE),
        /*
        //commented out due to changes from extended Handshake FSM
        new PdlEntry(
        ConnectionState.REQUEST_RECEIVED, CorePDUType.CORE_FIN),
        */
        new PdlEntry(
        ConnectionState.REQUEST_RECEIVED, CorePDUType.CORE_FINACK),
        new PdlEntry(
        ConnectionState.REQUEST_RECEIVED, CorePDUType.CORE_LAST_ACK),
        new PdlEntry(
        ConnectionState.REQUEST_SENT, CorePDUType.CORE_FIN),
        new PdlEntry(
        ConnectionState.REQUEST_SENT, CorePDUType.CORE_FINACK),
        new PdlEntry(
        ConnectionState.REQUEST_SENT, CorePDUType.CORE_LAST_ACK),
        new PdlEntry(
        ConnectionState.ESTABLISHED, CorePDUType.CORE_FINACK),
        new PdlEntry(
        ConnectionState.ESTABLISHED, CorePDUType.CORE_LAST_ACK),
        new PdlEntry(
        ConnectionState.FIN_WAIT, CorePDUType.CORE_CONNECTION_REQUEST),
        /*
        //commented out due to changes from extended Termination FSM
        new PdlEntry(
        ConnectionState.FIN_WAIT,
        CorePDUType.CORE_CONNECTION_RESPONSE),
        */
        new PdlEntry(
        ConnectionState.FIN_WAIT, CorePDUType.CORE_HEAD),
        new PdlEntry(
        ConnectionState.CLOSE_WAIT,
        CorePDUType.CORE_CONNECTION_REQUEST),
        new PdlEntry(
        ConnectionState.CLOSE_WAIT,
        CorePDUType.CORE_CONNECTION_RESPONSE),
        new PdlEntry(
        ConnectionState.CLOSE_WAIT, CorePDUType.CORE_HEAD),
        new PdlEntry(
        ConnectionState.CLOSE_WAIT, CorePDUType.CORE_FINACK),
        new PdlEntry(
        ConnectionState.CLOSING_1,
        CorePDUType.CORE_CONNECTION_REQUEST),
        new PdlEntry(
        ConnectionState.CLOSING_1,
        CorePDUType.CORE_CONNECTION_RESPONSE),
        new PdlEntry(
        ConnectionState.CLOSING_1, CorePDUType.CORE_HEAD),
        new PdlEntry(
        ConnectionState.CLOSING_1, CorePDUType.CORE_FINACK),
        new PdlEntry(
        ConnectionState.CLOSING_2,
        CorePDUType.CORE_CONNECTION_REQUEST),
        new PdlEntry(
        ConnectionState.CLOSING_2,
        CorePDUType.CORE_CONNECTION_RESPONSE),
        new PdlEntry(
        ConnectionState.CLOSING_2, CorePDUType.CORE_HEAD),
        new PdlEntry(
        ConnectionState.CLOSING_2, CorePDUType.CORE_FINACK),
    };
    for(PdlEntry e : errorEntries) {
        evalValue = lookupTable.put(e,
                (t, u) -> CoreLookupTable.createPDUMismatchTask(t, u));
        assert null == evalValue;
    }
}


//// "action" functions
public static Runnable createDoNothingTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {};
}

public static Runnable createHumanRejectTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // simply "close" the connection
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.CLOSED);
    };
}

public static Runnable createPeerTimeoutTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // pause all operations (sms consumption only)
        tObject.waitingForPrompt.set(true);
        // notify observers
        Object[] timeoutPrompt = {MessageType.PROMPT, PromptType.TIMEOUT};
        CoreProtocolFacade.setChangedAndNotifyObservers(tObject,
                timeoutPrompt);
    };
}

// NOTE: being used by resend*Tasks
public static Runnable createPDUMismatchTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        Connection connection =
                tObject.coreSession.connection;
        connection.incrementErrorCount();
        if(connection.getErrorCount() > 10) {
            CoreProtocolFacade.logAndNotify(tObject, MessageType.WARN,
                    "Connection got too many errors");
        }
    };
}

public static Runnable createRequestReceivedTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                "Received CONNECTION_REQUEST");
        String[] payloadArr = tObject.processReqAndGenRepPayload(
                uObject.getData());

        String repPayload;
        if(payloadArr.length == 1) {
            repPayload = generateStatusPayload(payloadArr[0]);
        }
        else if(payloadArr.length == 2) {
            repPayload = payloadArr[0] + "/" + payloadArr[1];
        }
        else { // invalid
            return; // do nothing
        }
        // use the received Session ID from sender
        if(uObject == null || uObject.getData() == null ||
                uObject.getData().length < 2 ||
                !(uObject.getData()[1] instanceof UUID)) { // invalid
            return; // do nothing
        }
        tObject.coreSession.sessionId = (UUID)(uObject.getData()[1]);
        // respond with REP
        PDUEncoder encoder = new PDUEncoder(false);
        encoder.appendVersion(CoreProtocolFacade.VERSION)
                .appendSessionId(tObject.coreSession
                .sessionId).appendPayload(
                "rep/" + tObject.getSubprotocolName() + "/" + repPayload)
                .appendAdditionalMessage(
                CoreProtocolFacade.DEFAULT_ADDITIONAL_MESSAGE)
                .finalizePDU();
        tObject.sendMessageRepeatedly(encoder.getEncoded());
        tObject.coreSession.connection.encodedRepPdu =
                encoder.getEncoded();
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.REQUEST_RECEIVED);
    };
}

public static Runnable createResendCoreRepTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                "Resending CONNECTION_RESPONSE");
        // just resend stored REP
        if(tObject.coreSession.connection.encodedRepPdu != null) {
            tObject.smsService.sendMessage(
                    tObject.coreSession.connection.encodedRepPdu);
        }
        // just maintain state
    };
}

public static Runnable createEstConnAsInitiatorTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        Object[] data = uObject.getData();
        // if peer accepted the connection
        if(data != null && data.length >= 4 &&
                data[3] instanceof String &&
                CoreProtocolFacade.USER_ACCEPT_STATUS_CODE.equals(
                    (String)data[3])) {
            CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Responder has accepted the connection request");
        }
        else if(data != null && data.length >= 4 &&
                data[3] instanceof String) { // non-accept status code
            String bufferPlus = (String)data[3] + " " +
                    ((data.length >= 5 && data[4] instanceof String) ?
                    (String)data[4] : "");
            CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Peer has rejected the connection request with " +
                    "status: " + bufferPlus);
        }
        else { // something else
            return; // exit asap
        }
        // respond with HEAD
        PDUEncoder encoder = new PDUEncoder(false);
        encoder.appendVersion(CoreProtocolFacade.VERSION)
                .appendSessionId(tObject.coreSession
                .sessionId).appendPayload(
                "head/" + tObject.getSubprotocolName() + "/" +
                tObject.generateHeadPayload())
                .appendAdditionalMessage(
                CoreProtocolFacade.DEFAULT_ADDITIONAL_MESSAGE)
                .finalizePDU();
        tObject.sendMessageRepeatedly(encoder.getEncoded());
        tObject.coreSession.connection.encodedHeadPdu =
                encoder.getEncoded();
        tObject.coreSession.connection.lastPduBeforeEst =
                CorePDUType.CORE_CONNECTION_RESPONSE;
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.ESTABLISHED);
        // finally, execute the onEstablished() hook
        tObject.onEstablished();
        // start the retransmission thread
        tObject.startRetransLoop();
    };
}

public static Runnable createEstConnAsResponderTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        Object[] data = uObject.getData();
        // if we accepted the connection
        if(data != null && data.length >= 4 && data[3] instanceof String) {
            tObject.processHeadPayload(data);
            CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Initiator has completed the handshake");
            // stop repeated sending of REP ASAP
            tObject.sendMessageRepeatedly(null);
        }
        else { // something else
            return; // exit asap
        }
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.ESTABLISHED);
        // finally, execute the onEstablished() hook
        tObject.onEstablished();
        // start the retransmission thread
        tObject.startRetransLoop();
    };
}

public static Runnable createResendCoreHeadIfValidTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // check if ESTABLISHED state is acquired via a
        //          REP PDU from a REQUEST_SENT state
        if(tObject.coreSession.connection.lastPduBeforeEst ==
                CorePDUType.CORE_CONNECTION_RESPONSE &&
                tObject.coreSession.connection.encodedHeadPdu != null) {
            CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Resending HEAD");
            tObject.smsService.sendMessage(
                    tObject.coreSession.connection.encodedHeadPdu);
        }
        else { // increment error count and notify listeners for
              // subsequent actions, if needed
            CoreLookupTable.createPDUMismatchTask(
                    tObject, uObject).run();
        }
        // just maintain state
    };
}

public static Runnable createRetransmissionTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        Object[] data = uObject.getData();
        if(data == null || data.length < 2) {
            return;
        }
        // loop through the PDUType, Integer pairs
        for(int i = 0; i < data.length/2; i++) {
            if(!(data[0] instanceof PDUType
                    && data[1] instanceof Integer)) {
                return; // exit asap on errors
            }
            // extract data
            PDUType t = (PDUType)data[0];
            int n = (Integer)data[1];

            // determine if n is for a multipart
            List<String> l = tObject.getRetransList(t);
            if(l == null) { // PDUType not yet registered
                CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                        "Requested PDU not yet available: " +
                        t + " part " + n);
                continue; // try next pair
            }
            if(n == 0 && l.size() == 1) { // single-part PDUs
                // log received 'n' first
                CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                        "Resending " + t + "'s only part ");
                // then leave it as it is
            }
            // check for errors first
            else if((n != 0 && n > l.size()+1) ||  // exceeded l.size()
                    (n == 0 && l.size() > 1)) {    // '0' on multipart PDUs
                CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                        "Invalid requested PDU: " + t + " part " + n);
                return; // exit asap on errors
            }
            else { // multi-part PDUs
                // log received 'n' first
                CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                        "Resending " + t + " part " + n);
                n--; // then decrement for real value
            }
            // notify observers about the resent PDU
            CoreProtocolFacade.setChangedAndNotifyObservers(tObject,
                    MessageType.STATUS, StatusType.NOTICE,
                    tObject.coreSession.connection.getConnectionState(),
                    t, n);
            // then retransmit the PDU
            tObject.smsService.sendMessage(l.get(n));
        }
    };
}

public static Runnable createFinReceivedTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        Object[] data = uObject.getData();
        String statusCode = (String)data[1];
        String statusMessage = (data.length == 3 && data[2]
                instanceof String) ? (String)data[2] : "";
        CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                "Peer has requested disconnection. Reason: " +
                statusCode + " " + statusMessage);
        // respond with FINACK
        PDUEncoder encoder = new PDUEncoder(true);
        encoder.appendPrefix(CorePDUType.CORE_FINACK,
                BinaryPDUDecoder.getInstance())
                .appendSessionId(tObject.coreSession.sessionId)
                .finalizePDU();
        tObject.sendMessageRepeatedly(encoder.getEncoded());
        // store the message somewhere
        tObject.coreSession.connection.encodedFinackPdu =
                encoder.getEncoded();
        // notify observers of the received statusCode and message
        CoreProtocolFacade.setChangedAndNotifyObservers(tObject,
                MessageType.STATUS, StatusType.NOTICE,
                ConnectionState.CLOSE_WAIT, statusCode, statusMessage);
        // then set to CLOSE_WAIT state afterwards
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.CLOSE_WAIT);
    };
}

public static Runnable createSimulFinReceivedTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // respond with LAST_ACK
        String lastackStr = generateEncodedLastack(tObject);
        tObject.smsService.sendMessage(lastackStr);
        // set to CLOSING_1 afterwards
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.CLOSING_1);
    };
}

public static Runnable createFinwaitDisconnectTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                "Peer acknowledged disconnection request");
        // respond with LAST_ACK
        String lastackStr = generateEncodedLastack(tObject);
        tObject.smsService.sendMessage(lastackStr);
        // set to TIME_WAIT afterwards
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.TIME_WAIT);

        // transition from a proper TIME_WAIT to CLOSED transition on a
        // background thread
        Future<?> delayedFuture = tObject.pool.submit(() -> {
            try { // wait until the maxPduLifetime*3/2
                // trip timeoutLoopFuture
                tObject.timeoutLoopFuture.cancel(true);
                // then sleep for a while before triggering disconnection
                Thread.sleep(
                        tObject.coreSession.connection.maxPduLifetime);
            }
            catch(InterruptedException e) {
                // do nothing; just proceed with shutdown procedure
            }
            CoreLookupTable.createDisconnectTask(
                    tObject, uObject).run();
        });
    };
}

public static Runnable createSimulLastackReceivedTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // set to CLOSING_2 afterwards
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.CLOSING_2);
    };
}

public static Runnable createResendFinackTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // just resend stored FINACK
        if(tObject.coreSession.connection.encodedFinackPdu != null) {
            CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Resending FINACK");
            tObject.smsService.sendMessage(
                    tObject.coreSession.connection.encodedFinackPdu);
        }
        // just maintain state
    };
}

public static Runnable createResendFinTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // just resend stored FIN
        if(tObject.coreSession.connection.encodedFinPdu != null) {
            CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Resending FIN");
            tObject.smsService.sendMessage(
                    tObject.coreSession.connection.encodedFinPdu);
        }
        // just maintain state
    };
}

public static Runnable createResendLastackTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        CoreProtocolFacade.logAndNotify(tObject, MessageType.DEBUG,
                "Resending LAST_ACK");
        // respond with LAST_ACK
        String lastackStr = generateEncodedLastack(tObject);
        tObject.smsService.sendMessage(lastackStr);
        // just maintain state
    };
}

public static Runnable createClosing1DisconnectTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // set to TIME_WAIT afterwards
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.TIME_WAIT);
    };
}

public static Runnable createClosing2DisconnectTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // respond with LAST_ACK
        String lastackStr = generateEncodedLastack(tObject);
        tObject.smsService.sendMessage(lastackStr);
        // set to TIME_WAIT afterwards
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.TIME_WAIT);
    };
}

public static Runnable createDisconnectTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // just disconnect (set state to CLOSED)
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.CLOSED);
        CoreProtocolFacade.logAndNotify(tObject, MessageType.INFO,
                "Disconnected");
        // kill all unneeded threads
        tObject.shutdownNow();
    };
}

public static Runnable createTimeoutDisconnectTask(
        CoreProtocolFacade tObject, PDU uObject) {
    return () -> {
        // send a LAST_ACK message with timeout as reason
        String statusCombo =
                CoreProtocolFacade.REQUEST_TIMEOUT_STATUS_CODE + "/" +
                CoreProtocolFacade.getStatusMessage(
                CoreProtocolFacade.REQUEST_TIMEOUT_STATUS_CODE);
        tObject.smsService.sendMessage(generateEncodedLastack(tObject,
                statusCombo));
        // finaly, disconnect (set state to CLOSED)
        CoreProtocolFacade.setStateAndNotify(tObject,
                ConnectionState.CLOSED);
        CoreProtocolFacade.logAndNotify(tObject, MessageType.INFO,
                "Disconnected");
        // and kill all unneeded threads
        tObject.shutdownNow();
    };
}



//// utility functions
private static String generateEncodedLastack(CoreProtocolFacade tObject) {
    return generateEncodedLastack(tObject,
            tObject.coreSession.connection.finStatusComboReceived);
}

private static String generateEncodedLastack(CoreProtocolFacade tObject,
        String statusCombo) {
    PDUEncoder encoder = new PDUEncoder(false);
    encoder.appendVersion(CoreProtocolFacade.VERSION)
            .appendSessionId(tObject.coreSession.sessionId)
            .appendPayload("last/" + tObject.getSubprotocolName() + "/" +
            statusCombo)
            .appendAdditionalMessage(
                CoreProtocolFacade.DEFAULT_ADDITIONAL_MESSAGE)
                .finalizePDU();
        return encoder.getEncoded();
    }

    private static String generateStatusPayload(String statusCode) {
        return statusCode + "/" +
                CoreProtocolFacade.getStatusMessage(statusCode);
    }


    // convenience get() function
    public static BiFunction<CoreProtocolFacade, PDU, Runnable>
            get(PdlEntry e) {
        return CoreLookupTable.lookupTable.get(e);
    }
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/CorePDUType.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import com.transmisms.core.protocol.PDUType.PDUSubType;
import com.transmisms.core.protocol.PDUType.PduSegmentIdFormat;



public enum CorePDUType implements PDUType<CorePDUType> {
    HUMAN_REJECT                (PDUSubType.HUMAN),
    CORE_CONNECTION_REQUEST     (PDUSubType.TEXT_BASED, 1),
    CORE_CONNECTION_RESPONSE    (PDUSubType.TEXT_BASED, 1),
    CORE_HEAD                   (PDUSubType.TEXT_BASED, 1),
    CORE_FIN                    (PDUSubType.BINARY, 0),
    CORE_FINACK                 (PDUSubType.BINARY, 0),
    CORE_LAST_ACK               (PDUSubType.TEXT_BASED, 1),
    CORE_RETRANSMISSION         (PDUSubType.BINARY);



    public final PDUSubType pduSubType;

    public final PduSegmentIdFormat segIdFormat;

    public final int sessionIdPosition;



    // default access control since enums are restricted
    CorePDUType(PDUSubType pduSubType) {
        this(pduSubType, -1);
    }

    CorePDUType(PDUSubType pduSubType, int sessionIdPosition) {
        this.pduSubType = pduSubType;
        this.segIdFormat = PduSegmentIdFormat.NONE;
        this.sessionIdPosition = sessionIdPosition;
    }



    @Override
    public CorePDUType valueOf() {
        return valueOf(this.name());
    }

    @Override
    public PDUSubType getPduSubType() {
        return this.pduSubType;
    }

    @Override
    public PduSegmentIdFormat getPduSegmentIdFormat() {
        return this.segIdFormat;
    }

    @Override
    public int getSessionIdPosition() {
        return this.sessionIdPosition;
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/CoreProtocolFacade.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import com.transmisms.core.protocol.Connection.ConnectionState;
import com.transmisms.core.protocol.Presenter.MessageType;
import com.transmisms.core.protocol.Presenter.PromptType;
import com.transmisms.core.util.commons.FIFOEntry;

import java.util.Arrays;
import java.util.Hashtable;
import java.util.List;
import java.util.Map;
import java.util.Observable;
import java.util.Set;
import java.util.UUID;
import java.util.concurrent.Callable;
import java.util.concurrent.ConcurrentSkipListMap;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Future;
import java.util.concurrent.Phaser;
import java.util.concurrent.PriorityBlockingQueue;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicReference;
import java.util.function.BiFunction;


import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;

import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeoutException;




public abstract class CoreProtocolFacade extends Observable {
    private final String SUBPROTOCOL_NAME = null;

    public enum DataSizes {
        CRC                         (1),
        SESSIONID                   (16),
        HEXSEGMENTID                (1),
        SEGMENTID                   (3),
        SEGMENTCOUNT                (3),
        SHA1_SUM                    (20),
        AES128_KEY                  (16),
        RSA2048_PUBKEY              (294),
        RSA2048ENC_AES128_KEY       (256),
        RSA2048_SHA512_SIG          (256),
        FILESIZE                    (3);

        private final int size;

        private DataSizes(int size) {
            this.size = size;
        }

        public int getSize() { return this.size; }
    }

    public enum DataStrSizes {
        PREFIX                      (1),
        CRC                         (2),
        SESSIONID                   (20),
        STATUSCODE                  (4),
        CEC                         (3),
        HEXSEGMENTID                (2),
        SEGMENTCOUNT                (4),
        SHA1_SUM                    (25),
        RSA2048_PUBKEY              (368),
        RSA2048_PUBKEY_PART1        (141),
        RSA2048ENC_AES128_KEY       (320),
        RSA2048ENC_AES128_KEY_PART3 (12),
        RSA2048_SHA512_SIG          (320),
        SEGMENTID                   (4);

        private final int size;

        private DataStrSizes(int size) {
            this.size = size;
        }

        public int getSize() { return this.size; }
    }

    protected static final Map<String, String> statusCodes =
            new Hashtable<String, String>();
    static {
        // initialize statusCodes
    statusCodes.put("0001", "OK; Completed without error");
    statusCodes.put("0002", "Accepted");

    statusCodes.put("1001", "User timeout");
    statusCodes.put("1002", "Cancelled by user");
    statusCodes.put("1003", "Rejected by user");

    statusCodes.put("1101", "Blacklisted");
    statusCodes.put("1102", "Peer unknown");

    statusCodes.put("2001", "PDU not acceptable");
    statusCodes.put("2002", "Request timeout");

    statusCodes.put("3001", "Protocol not supported");
    statusCodes.put("3002", "Version not supported");
    statusCodes.put("3003", "Filesystem error");
    statusCodes.put("3004", "Device error");

    statusCodes.put("4001", "Remote host identification mismatch");
    statusCodes.put("4101", "Key malformed");
    statusCodes.put("4201", "File decryption failed");
    statusCodes.put("4202", "File authentication failed");
    statusCodes.put("4203", "File signature mismatch");
    statusCodes.put("4901", "Too many PDU errors");
}

public static final String COMPLETED_STATUS_CODE = "0001";
public static final String USER_ACCEPT_STATUS_CODE = "0002";
public static final String REQUEST_TIMEOUT_STATUS_CODE = "2002";
public static final String DEFAULT_ADDITIONAL_MESSAGE =
        "To stop receiving messages, reply with NO";

public static final Version VERSION = new Version(0, 7, 0);
public static final Version MIN_VERSION = new Version(0, 7, 0);
public static final Version MAX_VERSION = new Version(0, 7, 0);



// non-constant members start here

public final Session coreSession;
private final BinaryPDUDecoder binaryDecoder;
private final TextBasedPDUDecoder textBasedDecoder;

public final SmsService smsService;

protected final Map<PDUType, SegmentManager> segmanMap =
        new ConcurrentSkipListMap<>();
protected final Map<PDUType, List<String>> retransMap =
        new ConcurrentSkipListMap<>();

protected final ExecutorService pool;
protected final ExecutorService pduThreadPool =
        Executors.newSingleThreadExecutor();

private Future<?> smsConsumerFuture;
private Future<?> pduConsumerFuture;
protected Future<?> timeoutLoopFuture;
private Future<?> retransLoopFuture;
private Future<?> classMainFuture;

private final Phaser timeoutPhaser = new Phaser(1);
private final AtomicReference<CountDownLatch> promptLatch =
        new AtomicReference<>(new CountDownLatch(1));
protected final AtomicBoolean waitingForPrompt = new AtomicBoolean(false);

private final Phaser dumbRetransPhaser = new Phaser(1);
private final AtomicReference<String> dumbRetransMsg =
        new AtomicReference<>(null);



protected CoreProtocolFacade(Session coreSession,
        BinaryPDUDecoder binaryDecoder,
        TextBasedPDUDecoder textBasedDecoder, SmsService smsService) {
    this(coreSession, binaryDecoder, textBasedDecoder, smsService,
            Executors.newCachedThreadPool(), -1);
}

protected CoreProtocolFacade(Session coreSession,
        BinaryPDUDecoder binaryDecoder,
        TextBasedPDUDecoder textBasedDecoder, SmsService smsService,
        int maxPduLifetime) {
    this(coreSession, binaryDecoder, textBasedDecoder, smsService,
            Executors.newCachedThreadPool(), maxPduLifetime);
}

/*
protected CoreProtocolFacade(Session coreSession,
        BinaryPDUDecoder binaryDecoder,
        TextBasedPDUDecoder textBasedDecoder, SmsService smsService,
        ExecutorService pool) {
    this(coreSession, binaryDecoder, textBasedDecoder, smsService,
            pool, -1);
}
*/

// NOTE: only subclasses should be able to use the base class's
//       constructor(s)
/**
 *
 * @param maxPduLifetime time in seconds to indicate the Maximum PDU
 *                       Lifetime
 */
protected CoreProtocolFacade(Session coreSession,
        BinaryPDUDecoder binaryDecoder,
        TextBasedPDUDecoder textBasedDecoder,
        SmsService smsService, ExecutorService pool,
        int maxPduLifetime) {
    this.coreSession = coreSession;
    this.binaryDecoder = binaryDecoder;
    this.textBasedDecoder = textBasedDecoder;
    this.smsService = smsService;
    this.pool = pool;
    if(maxPduLifetime >= 0) {
        this.coreSession.connection.maxPduLifetime = maxPduLifetime * 1000;
    }
}



//// member methods
public abstract String getSubprotocolName();

protected abstract String generateHeadPayload();

/**
 *
 * @return the default status message String for the statusCode
 */
public static String getStatusMessage(String statusCode) {
    return CoreProtocolFacade.statusCodes.get(statusCode);
}

protected abstract void processHeadPayload(Object[] headData);

protected abstract String[] processReqAndGenRepPayload(
        Object[] requestData);

protected abstract void onEstablished();

public PDU decode(String encoded)
        throws PDUMalformedException, InvalidCRCException {
    // some sanity checks before parsing
    if(encoded == null) {
        throw new NullPointerException("Encoded string cannot be null");
    }
    else if(encoded.equals("")) {
        throw new PDUMalformedException("Encoded string cannot be empty",
                encoded);
    }

    // determine if text-based, human, or binary
    if(encoded.startsWith("?transmismsv")) { // if text based protocol
        return this.textBasedDecoder.decodeTextBased(encoded);
    }
    else if(encoded.trim().toUpperCase().equals("NO")) { // human
        return new PDU(CorePDUType.HUMAN_REJECT, null);
    }
    else { // if binary protocol
        return this.binaryDecoder.decodeBinary(encoded);
    }
}

protected Runnable processPdu(PDU pduObject) {
    return null;
}

// this is where the real action is
private final Runnable processCorePdu(PDU pduObject) {
    { // process PDU from subprotocols first
        Runnable result = this.processPdu(pduObject);
        if(result != null) {
            return result; // return if subprotocol can parse
        }
    } // otherwise, let the core handle the PDU

    switch(this.coreSession.connection.getConnectionState()) {
        case CLOSED: // do nothing; we're supposedly closed
        case REQUEST_SENT_2:
        case REQUEST_SENT_3:
            return null;
        default: {
            BiFunction<CoreProtocolFacade, PDU, Runnable> pdlFunction =
                    CoreLookupTable.get(new PdlEntry(
                    this.coreSession.connection.getConnectionState(),
                    pduObject.getPduType()));
            if(pdlFunction == null) { // not in lookup table
                return null; // do nothing
            }
            else {
                return pdlFunction.apply(this, pduObject);
            }
        }
    }
}

/**
 *
 * @throws NullPointerException if there are no observers for this object
 */
private void initializeConnection() throws NullPointerException {
    if(this.countObservers() <= 0) {
        throw new NullPointerException("No observers set for this Facade");
    }
    logAndNotify(this, MessageType.TRACE, "Initializing connection");

    // initialize required objects
    final CoreProtocolFacade self = this;
    PriorityBlockingQueue<FIFOEntry<PDU>> pbq =
            new PriorityBlockingQueue<>();

    // sms consumer thread
    this.smsConsumerFuture = this.pool.submit(() -> {
        // loop until interrupt or SmsService error
        while(!Thread.currentThread().isInterrupted()) {
            try {
                // await while operation is paused
                if(this.waitingForPrompt.get()) {
                    this.promptLatch.get().await();
                }

                // get String messages from SmsService
                SmsEntry entry = self.smsService.pollMessage(
                        100, TimeUnit.MILLISECONDS);
                if(entry == null) { // if timed out
                    // continue
                    continue;
                }
                /*
                // NOTE: You can uncomment this if needed
                logAndNotify(this, MessageType.TRACE,
                        "got message: " + entry.getBody());
                */

                PDU messagePdu = null;
                { // parse then put; on errors, log erors
                    // entry is guaranteed to be non-null
                    String encoded = entry.getBody();
                    try {
                        messagePdu = this.decode(encoded);
                        pbq.put(new FIFOEntry<>(messagePdu));
                    }
                    catch(InvalidCRCException e) {
                        logAndNotify(this, MessageType.DEBUG,
                                "Invalid CRC: " + e.toString());
                    }
                    catch(PDUMalformedException e) {
                        logAndNotify(this, MessageType.DEBUG,
                                "Malformed PDU: " + e.toString());
                    }
                }
            }
            catch(SmsServiceException e) {
            }
            catch(InterruptedException e) {
                logAndNotify(this, MessageType.TRACE,
                        "SmsConsumer interrupted");
                break; // just break on interrupt
            }
        }
    });

    // pdu consumer thread
    this.pduConsumerFuture = this.pool.submit(() -> {
        // loop until interrupt or SmsService error
        while(!Thread.currentThread().isInterrupted()) {
            FIFOEntry<PDU> wrapper = null;
            try {
                wrapper = pbq.take();
            }
            catch(InterruptedException e) {
                logAndNotify(this, MessageType.TRACE,
                        "PduConsumer interrupted while waiting");
                break; // just break on interrupt
            }
            if(wrapper != null) { // should not be the other way since you
                                  // can't offer a null object
                // notify timeout loop first
                int phaseNumber = this.timeoutPhaser.arrive();

                // process the received pdu
                PDU entryPdu = wrapper.getEntry();


                // filter out PDUs that are not for this session
                Runnable runThis = null;
                // get session Id for the PDU
                int sessionIdPos =
                        entryPdu.getPduType().getSessionIdPosition();
                UUID sessionId = null;
                if(this.coreSession.sessionId != null &&
                        sessionIdPos != -1) {
                    Object o = entryPdu.getData()[sessionIdPos];
                    if(o != null && o instanceof UUID) {
                        sessionId = (UUID)o;
                    }
                }
                // set as null if invalid, else process
                if(sessionId != null &&
                        this.coreSession.sessionId != null &&
                        !this.coreSession.sessionId.equals((sessionId))) {
                    if(entryPdu.getPduType() !=
com.transmisms.smsftp.protocol.SmsftpPDUType.SMSFTP_DATA) {
                        logAndNotify(this, MessageType.TRACE,
                                "Received old PDU " + phaseNumber +
                                ": " + entryPdu.getPduType());
                    }
                    runThis = null;
                }
                else {
                    if(entryPdu.getPduType() !=
com.transmisms.smsftp.protocol.SmsftpPDUType.SMSFTP_DATA) {
                        logAndNotify(this, MessageType.TRACE,
                                "Received PDU " + phaseNumber +
                                ": " + entryPdu.getPduType());
                    }
                    runThis = this.processCorePdu(entryPdu);
                }

                // run and wait
                if(runThis != null) {
                    Future<?> runThisFuture = self.pduThreadPool.submit(
                            runThis);
                    try {
                        runThisFuture.get();
                    }
                    catch(ExecutionException e) {
                        logAndNotify(this, MessageType.TRACE,
                                "Caught: " + e);
                        e.printStackTrace();
                    }
                    catch(InterruptedException e) {
                        logAndNotify(this, MessageType.TRACE,
                                "PduConsumer interrupted");
                        break; // just break on interrupt
                    }
                }
            }
        }
    });

    // timeout loop thread
    this.timeoutLoopFuture = this.pool.submit(() -> {
        // loop until interrupt or SmsService error
        while(!Thread.currentThread().isInterrupted()) {
            ConnectionState cs = null;
            try {
                // wait for maxPduLifetime*2 seconds or until triggered
                int prevPhase = this.timeoutPhaser.getPhase();
                this.timeoutPhaser.awaitAdvanceInterruptibly(prevPhase,
                        this.coreSession.connection.maxPduLifetime*2,
                        TimeUnit.MILLISECONDS);
            }
            // if no PDUs received after waiting
            catch(TimeoutException e) {
                cs = this.coreSession.connection.getConnectionState();
                if(!(cs.equals(ConnectionState.CLOSED) ||
                        cs.equals(ConnectionState.TIME_WAIT) ||
                        cs.equals(ConnectionState.CLOSE_WAIT))) {
                    logAndNotify(this, MessageType.INFO,
                            "Timed out");
                    // schedule timeout procedure
                    this.pduThreadPool.submit(
                            CoreLookupTable.createPeerTimeoutTask(
                            this, null));
                }
            }
            catch(InterruptedException e) {
                // Thread.interrupt(); // preserve interrupt status
            }

            // check if we need to exit the loop
            cs = this.coreSession.connection.getConnectionState();
            if(cs.equals(ConnectionState.CLOSED) ||
                    cs.equals(ConnectionState.TIME_WAIT) ||
                    cs.equals(ConnectionState.CLOSE_WAIT)) {
                logAndNotify(this, MessageType.TRACE,
                        "Exiting TimeoutLoop...");
                break; // then break the loop
            }
            else {
                continue; // just loop and reset again
            }
        }
    });
}

/**
 *
 */
protected void startRetransLoop() {
    if(this.retransLoopFuture == null) {
        this.retransLoopFuture = this.pool.submit(() -> {
            // loop until interrupt or SmsService error
            while(!Thread.currentThread().isInterrupted() &&
                    !this.coreSession.connection.getConnectionState()
                    .equals(ConnectionState.TIME_WAIT)) {
                try {
                    // sleep before sending any retransmission PDU(s)
                    Thread.sleep(
                            this.coreSession.connection.maxPduLifetime);
                    // check if we need to exit ASAP
                    if(this.coreSession.connection.getConnectionState()
                            .equals(ConnectionState.TIME_WAIT)) {
                        break;
                    }
                    this.sendRetransPdu();
                }
                catch(InterruptedException e) {
                    // preserve interrupt status
                    Thread.currentThread().interrupt();
                    break; // then break the loop
                }
            }
        });
    }
}

/**
 *
 */
public void sendRetransPdu() {
    final PDUEncoder[] encoderC = { new PDUEncoder(true)
            .appendPrefix(
            CorePDUType.CORE_RETRANSMISSION,
            BinaryPDUDecoder.getInstance()) };
    // get each missed pdu type and segment number
    this.segmanMap.forEach((k, v) -> {
        for(Integer n : v.getSkippedSegNums()) {
            // cut into parts with 152-character limit
            if(encoderC[0].getEncoded().length() >= 152) {
                // send the message
                this.smsService.sendMessage(
                        encoderC[0].finalizePDU());
                // then set up another instance of encoder
                encoderC[0] = new PDUEncoder(true)
                        .appendPrefix(
                        CorePDUType.CORE_RETRANSMISSION,
                        BinaryPDUDecoder.getInstance());
            }

            encoderC[0].appendPrefix(k,
                    this.binaryDecoder);
            switch(k.getPduSegmentIdFormat()) {
                case BINARY:
                    encoderC[0].appendBinarySegmentId(n+1);
                    break;
                case HEX:
                    encoderC[0].appendHexSegmentId(n+1);
                    break;
                case NONE:
                default: // should never happen
                    encoderC[0].appendHexSegmentId(0);
            }
        }
    });
    // finally send the last message
    this.smsService.sendMessage(encoderC[0].finalizePDU());
}

/**
 *
 */
protected void sendMessageRepeatedly(String msg) {
    // set new value first
    this.dumbRetransMsg.set(msg);
    this.dumbRetransPhaser.arrive();

    Future msgRepeatFuture = this.pool.submit(() -> {
        while(msg != null && this.dumbRetransMsg.get() != null &&
                msg.equals(this.dumbRetransMsg.get()) &&
                !this.coreSession.connection.getConnectionState().equals(
                ConnectionState.TIME_WAIT)) {
            try {
                // send message before "sleeping"
                this.smsService.sendMessage(msg);
                // wait for maxPduLifetime*2 seconds or until interrupt
                int prevPhase = this.dumbRetransPhaser.getPhase();
                this.dumbRetransPhaser.awaitAdvanceInterruptibly(prevPhase,
                        this.coreSession.connection.maxPduLifetime*2,
                        TimeUnit.MILLISECONDS);
            }
            catch(TimeoutException e) {
                // do nothing
                // just loop and retry on timeout
            }
            catch(InterruptedException e) {
                // just break after getting the signal
                break;
            }
        }
    });
}

/**
 *
 * @throws NullPointerException if there are no observers for this object
 */
protected void listen() throws NullPointerException {
    // set state to listening
    CoreProtocolFacade.setStateAndNotify(this, ConnectionState.LISTENING);
    this.initializeConnection();
    logAndNotify(this, MessageType.INFO,
            "Listening for connection requests");
    // wait for request (via the smsConsumer and pduConsumer threads
    // and do nothing on the "main thread"

    // finally, just wait for other threads
    this.joinAll();
}

/**
 *
 * @throws NullPointerException if there are no observers for this object
 */
protected void connect(String additonalPayload)
        throws NullPointerException {
    this.initializeConnection();

    // class "main thread"
    this.classMainFuture = this.pool.submit(() -> {
        logAndNotify(this, MessageType.INFO,
                "Connecting to peer");
        if(this.coreSession.connection.role ==
                Connection.Role.INITIATOR) {
            // construct string to be appended later
            String addStr = "";
            if(additonalPayload != null) {
                addStr = "/" + additonalPayload;
            }
            // construct and try to send REQ
            PDUEncoder encoder = new PDUEncoder(false);
            encoder.appendVersion(CoreProtocolFacade.VERSION)
                    .appendSessionId(this.coreSession
                    .sessionId).appendPayload(
                    "req/" + this.getSubprotocolName() + addStr)
                    .appendAdditionalMessage(
                    CoreProtocolFacade.DEFAULT_ADDITIONAL_MESSAGE)
                    .finalizePDU();
            this.sendMessageRepeatedly(encoder.getEncoded());
            CoreProtocolFacade.setStateAndNotify(this,
                    ConnectionState.REQUEST_SENT);
        }
        else { // if receiver
        }
    });
    // finally, just wait for other threads
    this.joinAll();
}

public final void disconnect(String statusCode) {
    this.disconnect(statusCode, null);
}

// NOTE: this can be safely not subprotocol-specific
public final void disconnect(String statusCode, String statusStr) {
    // make sure that we are ESTABLISHED in the first place
    if(this.coreSession.connection.getConnectionState() ==
            ConnectionState.ESTABLISHED) {
        logAndNotify(this, MessageType.INFO,
                "Requesting disconnection");
        // if statusStr is null, use the default one
        String bufferStr = statusStr;
        if(bufferStr == null) {
            bufferStr =
                    CoreProtocolFacade.getStatusMessage(statusCode);
        }

        // construct and try to send FIN
        String addStr = statusCode + "/" + bufferStr;
        PDUEncoder encoder = new PDUEncoder(true);
        encoder.appendPrefix(CorePDUType.CORE_FIN,
                this.binaryDecoder)
                .appendSessionId(this.coreSession.sessionId)
                .appendStatusCode(statusCode);
        if(bufferStr != null) {
            encoder.appendString(bufferStr);
        }
        encoder.finalizePDU();
        this.sendMessageRepeatedly(encoder.getEncoded());
        this.coreSession.connection.encodedFinPdu = encoder.getEncoded();
        this.coreSession.connection.finStatusComboReceived = statusCode +
                (bufferStr != null ? "/" + bufferStr : "");
        // set to FIN_WAIT afterwards
        CoreProtocolFacade.setStateAndNotify(this,
                ConnectionState.FIN_WAIT);
    }
    // stop any attempts/operations to establish the connection
    else if(this.coreSession.connection.getConnectionState() ==
            ConnectionState.LISTENING) {
        // start hard disconnect sequence instead which includes:
        // - setting ConnectionState to CLOSED
        // - stopping any threads started, if any
        this.pool.submit(CoreLookupTable.createDisconnectTask(this, null));
        // NOTE: null PDU is still okay since it's an unused parameter
    }
    else {
    }
}

public void continueOperationAfterPrompt() {
    logAndNotify(this, MessageType.INFO,
            "User decided to continue waiting after timeout");
    // continue all operations if needed
    this.waitingForPrompt.set(false);

    CountDownLatch tmp = this.promptLatch.get();
    // prepare a new CountDownLatch
    this.promptLatch.set(new CountDownLatch(1));
    // wake the awaiting thread
    tmp.countDown();
}

public void cancelOperationAfterPrompt() {
    logAndNotify(this, MessageType.INFO,
            "User requested to cancel after timeout");
    // start shutdown sequence
    Future<?> timeoutDisconnectFuture = this.pool.submit(
            CoreLookupTable.createTimeoutDisconnectTask(this, null));
}


//// exposed functions for use by package and subclasses
protected List registerRetrans(PDUType k, List<String> v) {
    return this.retransMap.put(k, v);
}

protected List registerRetrans(PDUType k, String[] a) {
    return this.retransMap.put(k, Arrays.asList(a));
}

protected List<String> getRetransList(PDUType k) {
    return this.retransMap.get(k);
}

protected SegmentManager registerSegMan(PDUType k, SegmentManager v) {
    return this.segmanMap.put(k, v);
}

protected <T> Future<T> submitToInternalPool(Callable<T> task) {
    return this.pool.submit(task);
}

protected Future<?> submitToInternalPool(Runnable task) {
    return this.pool.submit(task);
}



//// utility functions
protected static void setChangedAndNotifyObservers(CoreProtocolFacade self,
        Object ... o) {
    self.setChanged();
    self.notifyObservers(o);
}

protected static void logAndNotify(CoreProtocolFacade self, MessageType t,
        String message) {
    Object[] logArr = {t, message};
    CoreProtocolFacade.setChangedAndNotifyObservers(self, logArr);
}

protected static void setStateAndNotify(CoreProtocolFacade self,
        ConnectionState newState) {
    ConnectionState oldState =
            self.coreSession.connection.getConnectionState();
    self.coreSession.connection.setConnectionState(newState);
    CoreProtocolFacade.setChangedAndNotifyObservers(self,
            MessageType.STATUS, Presenter.StatusType.STATE_CHANGED,
            oldState, newState);
}

protected void shutdownNow() {
    /*
    // stop smsService from sending and receiving messages
    this.smsService.stop();
    */

    // stop sendMessageRepeatedly() ASAP
    this.sendMessageRepeatedly(null);

    // shutdown all threadPools registered on this facade
    ExecutorService[] deathList = { this.pool, this.pduThreadPool };
    for(int i = 0; i < deathList.length; i++) {
        deathList[i].shutdown(); // disable new tasks from being submitted
        try {
            deathList[i].shutdownNow(); // try to cancel currently running
                                        // tasks
            // wait until tasks respond to being cancelled
            if(!deathList[i].awaitTermination(
                    500, TimeUnit.MILLISECONDS)) {
                logAndNotify(this, MessageType.DEBUG,
                        "Pool " + i + " did not terminate!");
            }
        }
        catch(InterruptedException e) {
            logAndNotify(this, MessageType.DEBUG,
                    "Pool " + i + " interrupted; Retrying shutdown");
            // retry cancelling current tasks again
            deathList[i].shutdownNow();
        }
    }
}

private void joinAll() {
    try {
        if(this.classMainFuture != null) {
            this.classMainFuture.get();
        }
        this.smsConsumerFuture.get();
        this.pduConsumerFuture.get();
    }
    catch(ExecutionException e) {
        System.out.println("thrown " + e);
        e.printStackTrace();
    }
    catch(InterruptedException e) {
        logAndNotify(this, MessageType.DEBUG,
                "CoreProtocolFacade join()s interrupted!");
        // preserve interrupt status
        Thread.currentThread().interrupt();
    }
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/InvalidCRCException.java
--------------------------------------------------------------

package com.transmisms.core.protocol;


public class InvalidCRCException extends Exception {

    private String malformedPDU = null;

    public InvalidCRCException(String message, String malformedPDU) {
        super(message);
        this.malformedPDU = malformedPDU;
    }

    public String getMalformedPDU() {
        return this.malformedPDU;
    }
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/PdlEntry.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import com.transmisms.core.protocol.Connection.ConnectionState;
import com.transmisms.core.protocol.PDUType;



// access is between public and protected, see Java documentation
final class PdlEntry {
    public final ConnectionState s;
    public final PDUType t;



    public PdlEntry(ConnectionState s, PDUType t) {
        this.s = s;
        this.t = t;
    }



    @Override
    public boolean equals(Object o) {
        if(o instanceof PdlEntry) {
            PdlEntry po = (PdlEntry)o;
            // check for nulls before proceeding with equals()
            if(po.s == null || po.t == null ||
                    this.s == null || this.t == null) {
                return false;
            }
            return this.s.equals(po.s) && this.t.equals(po.t);
        }
        else {
            return false;
        }
    }

    @Override
    public int hashCode() {
        return this.s.hashCode() + this.t.hashCode();
    }
    // CAUTION: this has no protection against nulls and problematic types
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/PDU.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import com.transmisms.core.protocol.PDUType;
import com.transmisms.smsftp.protocol.SmsftpPDUType;

import java.util.Hashtable;
import java.util.Map;
public class PDU implements Comparable<PDU> {
    public static final String transmismsCoreProtocolStr = "core";
    public static final String transmismsSmsftpProtocolStr = "smsftp";

    private final PDUType pduType;
    private final Object[] data;

    private static final Map<PDUType, Integer> pduTypePriorities =
            new Hashtable<PDUType, Integer>();
    static {
        // initialize pduType priorities
        //// human reject PDU
        pduTypePriorities.put(CorePDUType.HUMAN_REJECT, 50);
        //// core connection/termination PDUs
        pduTypePriorities.put(CorePDUType.CORE_CONNECTION_REQUEST, 40);
        pduTypePriorities.put(CorePDUType.CORE_CONNECTION_RESPONSE, 40);
        pduTypePriorities.put(CorePDUType.CORE_HEAD, 40);
        pduTypePriorities.put(CorePDUType.CORE_FIN, 40);
        pduTypePriorities.put(CorePDUType.CORE_FINACK, 40);
        pduTypePriorities.put(CorePDUType.CORE_LAST_ACK, 40);
        //// retransmission PDUs
        pduTypePriorities.put(CorePDUType.CORE_RETRANSMISSION, 30);
        //// smsftp control PDUs
        pduTypePriorities.put(SmsftpPDUType.SMSFTP_INIT, 20);
        pduTypePriorities.put(SmsftpPDUType.SMSFTP_META, 20);
        pduTypePriorities.put(SmsftpPDUType.SMSFTP_READY, 20);
        pduTypePriorities.put(SmsftpPDUType.SMSFTP_END, 20);
        pduTypePriorities.put(SmsftpPDUType.SMSFTP_END_WAIT, 20);
        //// blobs
        pduTypePriorities.put(SmsftpPDUType.SMSFTP_DATA, 10);
        pduTypePriorities.put(SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY, 10);
        pduTypePriorities.put(SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY, 10);
    }

    public PDU(PDUType pduType, Object[] data) {
        this.pduType = pduType;
        this.data = data;
    }

    public PDUType getPduType() {
        return this.pduType;
    }

    public PDUType.PDUSubType getPduSubType() {
        return this.pduType.getPduSubType();
    }

    public Object[] getData() {
        return this.data;
    }

    // for priority comparisons only
    public int compareTo(PDU o) {
        PDUType ownPDUType = this.getPduType();
        PDUType otherPDUType = o.getPduType();
        Integer ownPriority = pduTypePriorities.get(ownPDUType);
        Integer otherPriority = pduTypePriorities.get(otherPDUType);
        if(ownPriority == null) {
            ownPriority = 0;
        }
        if(otherPriority == null) {
            otherPriority = 0;
        }

        return otherPriority-ownPriority;
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/PDUEncoder.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import com.transmisms.core.protocol.CoreProtocolFacade.DataSizes;
import com.transmisms.core.util.codec.Base85;

import net.sourceforge.javaflacencoder.CRC8;

import java.nio.ByteBuffer;
import java.util.Arrays;
import java.util.UUID;

import java.io.UnsupportedEncodingException;
```

```
// convenience class to construct PDUs
public class PDUEncoder {

    private boolean isBinary;
    private boolean isFinalized;
    private String encoded;

    public PDUEncoder(boolean isBinary) {
        this.isBinary = isBinary;
        this.isFinalized = false;
        this.encoded = "";

        if(!isBinary) {
            this.encoded+="?transmismsv";
        }
    }

    public String getEncoded() {
        return this.encoded;
    }

    public String finalizePDU() {
        if(!this.isFinalized) {
            if(this.isBinary) {
                // append the CRC for this PDU
                CRC8 crc = new CRC8();
                try {
                    crc.updateCRC8(this.getEncoded().getBytes("UTF-8"), 0,
                            this.getEncoded().getBytes("UTF-8").length);
                    this.appendString(String.format("%02x", crc.checksum()));
                }
                catch(UnsupportedEncodingException e) {
                    // this should never happen; crash early
                    throw new IllegalStateException(e.getMessage(), e);
                }
            }
            this.isFinalized = true;
        }
        return this.getEncoded();
    }

    public PDUEncoder appendString(String s) {
        if(!this.isFinalized) {
            this.encoded+=s;
        }
        return this;
    }

    /* TEXT-BASED PDUS */

    public PDUEncoder appendVersion(Version v) {
        this.appendString(v + "\n");
        return this;
    }

    public PDUEncoder appendSessionId(UUID sessionId) {
        this.appendString(PDUEncoder.UUIDToBase85(sessionId));
        if(!this.isBinary) {
            this.appendString("\n");
        }
        return this;
    }

    public PDUEncoder appendPayload(String payload) {
        this.appendString(payload + "\n");
        return this;
    }

    public PDUEncoder appendAdditionalMessage(String message) {
        this.appendString("\n" + message);
        return this;
    }

    /* BINARY PDUS */

    public PDUEncoder appendPrefix(PDUType x, BinaryPDUDecoder b) {
        this.appendString(b.pduTypeToPrefix(x) + "");
        return this;
    }

    public PDUEncoder appendHexSegmentId(int segmentId) {
        this.appendString(String.format("%02x", segmentId));
        return this;
    }

    public PDUEncoder appendBinarySegmentId(int segmentId) {
        ByteBuffer buffer = ByteBuffer.allocate(Integer.BYTES);
        buffer.putInt(segmentId);
        byte[] raw =  buffer.array();
        byte[] truncated = Arrays.copyOfRange(raw,
                raw.length-DataSizes.SEGMENTID.getSize(), raw.length);
        this.appendString(Base85.encode(truncated));
        return this;
    }

    public PDUEncoder appendStatusCode(String statusCode) {
        this.appendString(statusCode);
        return this;
    }
    // the rest can just use the appendString() method

    /* UTILITY FUNCTIONS */

    public static String UUIDToBase85(UUID uuidObject) {
        Long msb = uuidObject.getMostSignificantBits();
        Long lsb = uuidObject.getLeastSignificantBits();
        byte[] bytes = ByteBuffer.allocate(2*Long.BYTES).putLong(msb)
                .putLong(lsb).array();
        return Base85.encode(bytes);
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/PDUMalformedException.java
--------------------------------------------------------------
package com.transmisms.core.protocol;


public class PDUMalformedException extends Exception {

    private String malformedPDU = null;

    public PDUMalformedException(String message, String malformedPDU) {
        super(message);
        this.malformedPDU = malformedPDU;
    }

    public String getMalformedPDU() {
        return this.malformedPDU;
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/PDUType.java
--------------------------------------------------------------

package com.transmisms.core.protocol;



public interface PDUType<T extends Enum> {

    public enum PDUSubType {    BINARY,
                                TEXT_BASED,
                                HUMAN };

    public enum PduSegmentIdFormat {    NONE,
                                        HEX,
                                        BINARY };


    public T valueOf();

    public PDUSubType getPduSubType();
    public PduSegmentIdFormat getPduSegmentIdFormat();
    public int getSessionIdPosition();
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/Presenter.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import java.util.Observable;
import java.util.Observer;



public abstract class Presenter implements Observer {
    public enum MessageType {
        // UI-related
        PROMPT,
        STATUS,
        // logging
        FATAL,
        ERROR,
        WARN,
        INFO,
        DEBUG,
        TRACE
    }

    public enum PromptType {
        CONNECTION_REQUEST,
        TIMEOUT,
        SMS_SERVICE_ERROR,
        PEER_ERROR_EXCEEDED_LIMIT
    }

    public enum StatusType {
        STATE_CHANGED,
        NOTICE
    }

    @Override
    public abstract void update(Observable o, Object arg); // do nothing
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/SegmentManager.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import java.util.concurrent.ConcurrentSkipListSet;
import java.util.concurrent.atomic.AtomicReference;


public class SegmentManager {
    private final String[] segments;
    public final int segCount;

    private final AtomicReference<String> completedString =
            new AtomicReference<>();

    // properties used for tracking 'lost' segments
    private final ConcurrentSkipListSet<Integer> missingnos =
            new ConcurrentSkipListSet<>();
    private int lastSegmentReceived = -1;



    public SegmentManager(int segCount) {
        if(segCount <= 0) {
            throw new IllegalArgumentException(
                    "Segment count should be greater than 0");
        }
        this.missingnos.add(0); // seed first segment
        this.segCount = segCount;
        this.segments = new String[this.segCount];
    }



    private synchronized String getSegment(int segNum) {
        if(segNum < 0) {
            throw new IllegalArgumentException(
                    "Segment number should not be negative");
        }
        if(segNum >= this.segments.length) {
            throw new IllegalArgumentException(
                    "Segment number should not be greater than or equal " +
                    "to the number of segments");
        }
        return this.segments[segNum];
    }

    /**
     *
     * @return true if segment is updated; false if segment already exists
     */
    public synchronized boolean updateSegment(String segment, int segNum) {
        if(segment == null) {
            throw new IllegalArgumentException(
                    "Segment cannot be null");
        }
        if(segNum < 0) {
            throw new IllegalArgumentException(
                    "Segment number should not be negative");
        }
        if(segNum >= this.segments.length) {
            throw new IllegalArgumentException(
                    "Segment number should not be greater than or equal " +
                    "to the number of segments");
        }

        // check if can be updated
        if(this.segments[segNum] != null) {
            return true;
        }
        // add if null/update if existing
        this.segments[segNum] = segment;

        // update missingnos
        this.missingnos.remove(segNum);
        if(this.lastSegmentReceived < segNum) {
            // add skipped segments to missingnos, if any
            for(int i = lastSegmentReceived+1; i < segNum; i++) {
                if(this.segments[i] == null) {
                    this.missingnos.add(i);
                }
            }
            // add the next segNum to missingnos
            if(segNum+1 < this.segCount) {
                this.missingnos.add(segNum+1);
            }
            this.lastSegmentReceived = segNum; // update last segment number
        }

        // preemptive consolidation
        this.getCompletedString();

        return true;
    }

    public Integer[] getSkippedSegNums() {
        return this.missingnos.toArray(new Integer[0]);
    }

    public int getLastSegmentReceived() {
        return this.lastSegmentReceived;
    }

    /**
     *  @return the combined segments if completed; null otherwise
     */
    // Consolidates segments if not yet completed
    public synchronized String getCompletedString() {
        // check for segments complete conditions and do appropriate
        //     actions
        if(this.completedString.get() == null &&
                this.missingnos.isEmpty() &&
                this.lastSegmentReceived+1 == this.segments.length) {
            // consolidate and convert String data
            StringBuilder sb = new StringBuilder();
            for(String s : this.segments) {
                sb.append(s);
            }
            this.completedString.set(sb.toString());
        }

        return this.completedString.get();
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/Session.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import java.util.UUID;
import java.nio.ByteBuffer;


public abstract class Session {

    public final Connection connection;

    public UUID sessionId;



    /**
```

```
     * Creates a new Session with the specified Session Id
     */
    public Session(Connection connection, UUID sessionId) {
        this.sessionId = sessionId;
        this.connection = connection;
    }
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/SmsEntry.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import java.util.Date;



public class SmsEntry {
    private String sender;
    private String body;
    private Date timestamp;

    public SmsEntry(String sender, String body) {
        this(sender, body, new Date());
    }

    public SmsEntry(String sender, String body, Date timestamp) {
        this.sender = sender;
        this.body = body;
        this.timestamp = timestamp;
    }

    public String getSender() {
        return this.sender;
    }

    public String getBody() {
        return this.body;
    }

    public Date getTimestamp() {
        return this.timestamp;
    }
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/SmsService.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import java.util.concurrent.TimeUnit;



// NOTE: handling queuing, setting destination number, etc. should be done
//       by the implementer
public interface SmsService {

    public void sendMessage(String message);

    public SmsEntry pollMessage(long timeout, TimeUnit unit)
            throws SmsServiceException, InterruptedException;

    // for running methods that the service might require (to be run from
    // background threads)
    public boolean start() throws SmsServiceException;
    public boolean stop();

    public boolean isRunning();

    // NOTE: we don't need to check for a ready state here as the service might
    //       become unavailable immediately after we have verified that the
    //       service is available
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/SmsServiceException.java
--------------------------------------------------------------

package com.transmisms.core.protocol;



public class SmsServiceException extends Exception {
    public interface Reason<T extends Enum> {
        public T valueOf();
    }

    public final Reason reason;

    public SmsServiceException(Reason reason) {
        super(reason.toString());
        this.reason = reason;
    }

    public Reason getReason() {
        return this.reason;
    }
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/TextBasedPDUDecoder.java
--------------------------------------------------------------

package com.transmisms.core.protocol;

import com.transmisms.core.protocol.PDU;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.core.protocol.PDUMalformedException;

import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;
import java.util.regex.Pattern;

import java.util.UUID;
import java.nio.ByteBuffer;
import java.io.IOException;
import com.transmisms.core.util.codec.Base85;
import com.transmisms.core.protocol.CoreProtocolFacade.DataSizes;



public class TextBasedPDUDecoder {
    private static final Pattern LF_PATTERN = Pattern.compile("\n");
    private static final Pattern LFLF_PATTERN = Pattern.compile("\n\n");
    private static final Pattern DOT_PATTERN = Pattern.compile("\\.");
    private static final Pattern SLASH_PATTERN = Pattern.compile("/");

    private String[] knownProtocols;

    private static TextBasedPDUDecoder thisInstance;
    static {
        String[] emptyArray = {};
        TextBasedPDUDecoder.thisInstance = new TextBasedPDUDecoder(emptyArray);
    }

    protected TextBasedPDUDecoder(String[] knownProtocols) {
        this.knownProtocols = knownProtocols;
    }

    public static TextBasedPDUDecoder getInstance() {
        return TextBasedPDUDecoder.thisInstance;
    }



    private static UUID base85ToUUID(String b85String, String encodedStr,
            String pduTypeName)
            throws PDUMalformedException {
        // extract byte[] format from base85
        byte[] uuidBytes = null;
        try {
            uuidBytes = Base85.decode(b85String);
        }
        catch(IOException e) {
            throw new PDUMalformedException(
                    pduTypeName + "'s Session ID is corrupt", encodedStr);
        }
        //  verify if sessionId's length is correct
        if(uuidBytes.length != DataSizes.SESSIONID.getSize()) {
            throw new PDUMalformedException(
                    pduTypeName + "'s Session ID's byte length is incorrect",
                    null);
        }

        // split the byte[] into msb and lsb
        byte[] msb = Arrays.copyOfRange(uuidBytes, 0, 8);
        byte[] lsb = Arrays.copyOfRange(uuidBytes, 8, 16);
        // convert the msb and lsb byte[] into long and return afterwards
        ByteBuffer msbBuffer = ByteBuffer.allocate(Long.BYTES).put(msb);
        ByteBuffer lsbBuffer = ByteBuffer.allocate(Long.BYTES).put(lsb);
        msbBuffer.flip(); // required for reading
        lsbBuffer.flip(); // required for reading
        return new UUID(msbBuffer.getLong(), lsbBuffer.getLong());
    }

    protected Object[] decodePayloadSubparts(PDUType pduType,
            String[] payloadSubparts, String encoded)
            throws PDUMalformedException {
        // does nothing alone; just return back the payloadSubparts
        return payloadSubparts;
    }

    public final PDU decodeTextBased(String encoded)
            throws PDUMalformedException {
        CorePDUType pduType = null;
        List<Object> decodedData = new ArrayList<Object>();

        String machineMsg = "";
        String humanMsg = null;
        { // split machine and human-readable messages
            String[] lflfParts = LFLF_PATTERN.split(encoded);
            if(lflfParts.length > 1) {
                // last LFLF separated string is the human-readable message
                humanMsg = lflfParts[lflfParts.length-1];
                // join back remaining parts as machine-readable message
                machineMsg = "";
                for(int i = 0; i < lflfParts.length-1; i++) {
                    machineMsg += lflfParts[i];
                }
            }
            else { // no additional message
                machineMsg = lflfParts[0];
            }
        }

        // split machine message into segments
        String[] segments = LF_PATTERN.split(machineMsg);
        // check for "?transmismsvx.x.x" header
        if(!segments[0].startsWith("?transmismsv")) {
            throw new PDUMalformedException("Invalid Header: " +  segments[0],
                    encoded);
        }
        // get Version
        try {
            String[] versionParts = TextBasedPDUDecoder.DOT_PATTERN.split(
                segments[0].substring(12));
            if(versionParts.length != 3) {
                throw new PDUMalformedException("Invalid Version format: " +
                        segments[0].substring(12), encoded);
            }
            int a = new Integer(versionParts[0]);
            int b = new Integer(versionParts[1]);
            int c = new Integer(versionParts[2]);
            if(a < 0 || b < 0 || c < 0) {
                throw new NumberFormatException(); // pass responsibility
                                                   // to the catcher below
            }
            Version ver = new Version(a, b, c);
            decodedData.add(ver);
        }
        catch(NumberFormatException e) {
            throw new PDUMalformedException("Invalid Version: " +
                    segments[0].substring(12), encoded);
        }

        { // get Session Id
            String sessionIdStr = segments[1];
            UUID sessionId = base85ToUUID(sessionIdStr, encoded,
                    "Text-based PDU");
            decodedData.add(sessionId);
        }

        { // parse payload
            String[] payloadParts = TextBasedPDUDecoder.SLASH_PATTERN.split(
                    segments[2]);
            // get PDU Type
            if(payloadParts[0].equals("req")) {
                pduType = CorePDUType.CORE_CONNECTION_REQUEST;
            }
            else if(payloadParts[0].equals("rep")) {
                pduType = CorePDUType.CORE_CONNECTION_RESPONSE;
            }
            else if(payloadParts[0].equals("head")) {
                pduType = CorePDUType.CORE_HEAD;
            }
            else if(payloadParts[0].equals("last")) {
                pduType = CorePDUType.CORE_LAST_ACK;
            }
            else {
                throw new PDUMalformedException("Invalid Payload Type: " +
                        payloadParts[0], encoded);
            }

            { // check if protocol name is valid
                String protocolName = payloadParts[1];
                boolean validProtocolName = false;
                for(int i = 0; i < this.knownProtocols.length; i++) {
                    if(this.knownProtocols[i].equals(protocolName)) {
                        validProtocolName = true;
                    }
                }
                if(validProtocolName) {
                    decodedData.add(protocolName);
                }
                else {
                    throw new PDUMalformedException("Invalid Protocol name: " +
                            protocolName, encoded);
                }
            }

            {
                String[] subParts = {};
                if(payloadParts.length > 2) {
                    subParts = Arrays.copyOfRange(payloadParts, 2,
                        payloadParts.length);
                }

                Object[] addLater =
                        this.decodePayloadSubparts(pduType, subParts, encoded);
                for(Object o : addLater) {
                    decodedData.add(o);
                }
            }
        }
        // finally, add human message
        decodedData.add(humanMsg);

        return new PDU(pduType, decodedData.toArray());
    }
}
```

```
-------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/protocol/Version.java
-------------------------------------------------------------

package com.transmisms.core.protocol;

public class Version implements Comparable<Version> {
    protected final int[] data = new int[3];

    public Version(int a, int b, int c) throws IllegalArgumentException {
        if(a < 0 || b < 0 || c < 0) {
            throw new IllegalArgumentException(
                    "Version numbers cannot be negative");
        }
        this.data[0] = a;
        this.data[1] = b;
        this.data[2] = c;
    }

    @Override
    public int compareTo(Version o) {
        return compare(o, 0);
    }

    @Override
    public boolean equals(Object o) {
        if(!(o instanceof Version)) {
            return false;
        }
        else {
            return (0 == compareTo((Version)o));
        }
    }

    @Override
    public int hashCode() {
        // prime numbers arbitrarily chosen at random in descending order
        return (7919*this.data[0]) + (809*this.data[1]) + (37*this.data[2]);
    }

    private int compare(Version o, int start) {
        if(this.data[start] < o.data[start]) {
            return -1;
        }
        else if(this.data[start] > o.data[start]) {
            return 1;
        }
        else {
            if(start == 2) {
                return 0;
            }
            else {
                return this.compare(o, start+1);
            }
        }
    }

    @Override
    public String toString() {
        return this.data[0] + "." + this.data[1] + "." + this.data[2];
    }
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/util/codec/Base85.java
--------------------------------------------------------------

package com.transmisms.core.util.codec;

import org.apache.pdfbox.io.ASCII85InputStream;
import org.apache.pdfbox.io.ASCII85OutputStream;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.util.ArrayList;

import java.io.IOException;

/**
 *
 */
public class Base85 {

    /**
     * Encodes data to ASCII85
     *
     * @param data data to be encoded into ASCII85
     * @return ASCII85 encoded string
     */
    public static String encode(byte[] data) {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ASCII85OutputStream a85os = new ASCII85OutputStream(baos);

        String encoded = null;
        try {
            a85os.write(data);
            a85os.flush();

            encoded = baos.toString("UTF-8").trim();
                                            // because pdfbox's ascii85 appends
                                            // an annoying \n (int 10) every
                                            // ducking single time

            baos.close();
            a85os.close();
        }
        catch(IOException e) { // should never happen since we are just using
                               // mostly safe ByteArrayOutputStream
            assert false;
            return null;
        }

        // remove ~ and ~> suffixes, if present
        if(encoded.endsWith("~")) {
            encoded = encoded.substring(0, encoded.length()-1);
        }
        else if(encoded.endsWith("~>")) {
            encoded = encoded.substring(0, encoded.length()-2);
        }

        // remove all unused whitespaces
        return encoded.replace("\n", "")
                      .replace("\r", "")
                      .replace(" ",  "")
        // and replace all non-GSM 7-bit alphabet characters with compliant
        // ones
                      .replace("[",  "v")
                      .replace("\\", "w")
                      .replace("]",  "x")
                      .replace("^",  "y")
                      .replace("`",  "£");
    }

    /**
     * Decodes ASCII85 encoded string
     *
     * @param data String to be decoded from ASCII85
     * @return decoded data
     */
    public static byte[] decode(String data) throws IOException {
        // remove all unused whitespaces
        String encoded = data.trim()
        // and replace back all replaced characters on encoding to ASCII
                             .replace("v", "[")
                             .replace("w", "\\")
                             .replace("x", "]")
                             .replace("y", "^")
                             .replace("£", "`");

        // re-add the ~ suffix if missing
        if(!encoded.endsWith("~") || !encoded.endsWith("~>")) {
            encoded = encoded + "~";
        }

        ByteArrayInputStream bais = new
ByteArrayInputStream(encoded.getBytes("UTF-8"));
        ASCII85InputStream a85is = new ASCII85InputStream(bais);

        ArrayList<Byte> buff = new ArrayList<Byte>();

        int d;
        while((d = a85is.read()) != -1) {
            buff.add((byte)d);
        }
        byte[] decoded = new byte[buff.size()];
        for(int i = 0; i < buff.size(); i++) {
            decoded[i] = buff.get(i);
        }

        a85is.close();

        return decoded;
    }

}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/util/commons/FIFOEntry.java
--------------------------------------------------------------

package com.transmisms.core.util.commons;


import java.util.concurrent.atomic.AtomicLong;


public class FIFOEntry<E extends Comparable<? super E>>
        implements Comparable<FIFOEntry<E>> {
    static final AtomicLong seq = new AtomicLong(0);
    final long seqNum;
    final private E entry;

    public FIFOEntry(E entry) {
        this.seqNum = FIFOEntry.seq.getAndIncrement();
        this.entry = entry;
    }

    public int compareTo(FIFOEntry<E> other) {
        int res = this.entry.compareTo(other.entry);
        if(res == 0 && other.entry != this.entry) {
            res = (this.seqNum < other.seqNum ? -1 : 1);
        }
        return res;
    }

    public E getEntry() {
        return this.entry;
    }
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/util/compression/XZ.java
--------------------------------------------------------------

package com.transmisms.core.util.compression;
```

```
import org.apache.commons.io.IOUtils;
import org.apache.commons.compress.compressors.xz.XZCompressorInputStream;
import org.apache.commons.compress.compressors.xz.XZCompressorOutputStream;

import java.io.InputStream;
import java.io.OutputStream;

import java.io.IOException;

/**
 *
 */
public class XZ {

    /**
     *
     */
    public static void compress(InputStream is, OutputStream os)
            throws IOException {
        // we're going to use default compression level: 6
        XZCompressorOutputStream xzos = new XZCompressorOutputStream(os);

        xzos.write(IOUtils.toByteArray(is));
        xzos.flush();

        xzos.close();
    }

    /**
     *
     */
    public static void decompress(InputStream is, OutputStream os)
            throws IOException {
        XZCompressorInputStream xzis = new XZCompressorInputStream(is);

        os.write(IOUtils.toByteArray(xzis));

        xzis.close();
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/util/crypto/AES.java
--------------------------------------------------------------

package com.transmisms.core.util.crypto;

import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

import org.apache.commons.io.IOUtils;

import java.io.InputStream;
import java.io.OutputStream;

import java.security.GeneralSecurityException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.io.IOException;



/**
 *
 */
public final class AES {
    //       see AES() cipher.getInstance()
    private final static String _AES_KEYGEN_ALGO = "AES";
    private final static String _AES_CIPHER_ALGO = "AES";
    private final static String _AES_CIPHER_ALGO_MODE_CBC = "CBC";
    // NOTE: this is actually a PKCS7Padding internally, see Sun's padding
    //         security documentation (more of a historical artifact to prevent
    //         breaking things; > 8 bytes would yield as PKCS7 instead of
    //         PKCS5)
    private final static String _AES_CIPHER_ALGO_PADDING_PKCS7 =
            "PKCS5Padding";
    private final static int _AES_DEFAULT_KEY_SIZE = 128;



    /**
     * Encrypts InputStream is with the object's key and outputs it to
     * OutputStream os.
     *
     * @param is data to be encrypted will be read from
     * @param os encrypted data will be written to
     */
    public static void encrypt(byte[] key, byte[] iv, InputStream is,
            OutputStream os) throws IOException {
        // read key from input stream
        SecretKeySpec sks = byteArrayToKeySpec(key);
        Cipher c = AES.getCipher();
        // encrypt the stream
        try {
            c.init(Cipher.ENCRYPT_MODE, sks, new IvParameterSpec(iv));
        }
        catch(InvalidKeyException e) {
            System.err.println("InvalidKeyException at AES.encrypt() !!!");
        }
        catch(InvalidAlgorithmParameterException e) {
            System.err.println(
                    "InvalidAlgorithmParameterException at AES.encrypt() !!!");
        }
        CipherOutputStream cos = new CipherOutputStream(os, c);
        IOUtils.copy(is, cos);
        cos.close();
    }

    /**
     * Decrypts InputStream is with the object's key and outputs it to
     * OutputStream os.
     *
     * @param is data to be decrypted will be read from
     * @param os decrypted data will be written to
     */
    public static void decrypt(byte[] key, byte[] iv, InputStream is,
            OutputStream os) throws IOException {
        // read key from input stream
        SecretKeySpec sks = byteArrayToKeySpec(key);
        Cipher c = AES.getCipher();

        // decrypt the stream
        try {
            c.init(Cipher.DECRYPT_MODE, sks, new IvParameterSpec(iv));
        }
        catch(InvalidKeyException e) {
            System.err.println("InvalidKeyException at AES.decrypt() !!!");
        }
        catch(InvalidAlgorithmParameterException e) {
            System.err.println(
                    "InvalidAlgorithmParameterException at AES.decrypt() !!!");
        }
        CipherInputStream cis = new CipherInputStream(is, c);
        IOUtils.copy(cis, os);
        cis.close();
    }

    /**
     * Generates a 128-bit AES key
     *
     * @return a 128-bit AES key
     */
    public static byte[] generateKey() {
        SecretKey key = null;
        try {
            KeyGenerator kg = KeyGenerator.getInstance(_AES_KEYGEN_ALGO);
            kg.init(_AES_DEFAULT_KEY_SIZE, new SecureRandom());
            key = kg.generateKey();
        }

        // this arises from KeyGenerator.getInstance()
        // this should be impossible to reach!
        catch(NullPointerException e) {
            System.err.println(
                    "NullPointerException at AES.generateKey() !!!");
        }
        /*
         * Covers the following:
         *  - NoSuchAlgorithmException from KeyGenerator.getInstance()
         *  - InvalidAlgorithmParameterException from kg.init()
         */
        // this should be impossible to reach!
        catch(GeneralSecurityException e) {
            System.err.println(
                    "GeneralSecurityException at AES.generateKey() !!!");
        }

        return key.getEncoded();
    }

    /**
     * Generates a SecretKeySpec using this class's parameters from the
     * provided key
     *
     * @return a SecretKeySpec using this class's parameters from the
     *         provided key
     */
    public static SecretKeySpec byteArrayToKeySpec(byte[] key) {
        return new SecretKeySpec(key, _AES_KEYGEN_ALGO);
    }


    private static Cipher getCipher() {
        try {
            Cipher c = Cipher.getInstance(_AES_CIPHER_ALGO + "/" +
                                          _AES_CIPHER_ALGO_MODE_CBC + "/" +
                                          _AES_CIPHER_ALGO_PADDING_PKCS7);
            return c;
        }
        /*
         * Covers the following:
         *  - NoSuchAlgorithmException from Cipher.getInstance()
         *  - NoSuchPaddingException from Cipher.getInstance()
         */
        // this should be impossible to reach!
        catch(GeneralSecurityException e) {
            assert false; // this should never happen
            return null;
        }
    }
}



---------------------------------------------------------------
Filename:
src/main/java/com/transmisms/core/util/crypto/BiometricFingerprint.java
---------------------------------------------------------------

package com.transmisms.core.util.crypto;

import technology.zeroalpha.security.pgpwordlist.PGPWordListConverter;
import technology.zeroalpha.security.pgpwordlist.InvalidHexValueException;
import org.apache.commons.codec.digest.DigestUtils;


/**
 *
 */
public final class BiometricFingerprint {

    private static final PGPWordListConverter converter =
            new PGPWordListConverter();

    /**
     *
     */
    public static String getBiometricFingerprint(byte[] data)
            throws NullPointerException {
        // get SHA1 checksum of data
        String sha1Hex = DigestUtils.sha1Hex(data);

        // translate hex string fingerprint to PGP words
        boolean oddWord = true;
        String bioFingerprint = "";
        try {
            for(int i = 0; i < sha1Hex.length()/2 ; i++) {
                String currentWord = sha1Hex.substring(i*2,(i*2)+2);
                bioFingerprint += i==0 ? "" : " ";
                bioFingerprint += oddWord ?
                        BiometricFingerprint.converter.getOddWordForHexValue(
                        currentWord) :
                        BiometricFingerprint.converter.getEvenWordForHexValue(
                        currentWord);
                oddWord = !oddWord; // next odd/even word
            }
        }
        // this should never happen as the hex returned by DigestUtils should
        // always be a valid hex
        catch(InvalidHexValueException e) {
            throw new NullPointerException("Got InvalidHexValueException!");
        }

        return bioFingerprint;
    }
}



---------------------------------------------------------------
```

```
Filename: src/main/java/com/transmisms/core/util/crypto/RSA.java
---------------------------------------------------------------

package com.transmisms.core.util.crypto;

import org.bouncycastle.util.io.pem.PemReader;
import org.bouncycastle.util.io.pem.PemWriter;
import org.bouncycastle.util.io.pem.PemObject;
import org.bouncycastle.util.encoders.DecoderException;

import java.security.KeyFactory;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.spec.PKCS8EncodedKeySpec;
import java.security.spec.X509EncodedKeySpec;
import javax.crypto.Cipher;

import org.apache.commons.io.IOUtils;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;

import java.security.GeneralSecurityException;
import java.security.NoSuchAlgorithmException;
import java.security.spec.InvalidKeySpecException;
import java.io.IOException;


/**
 *
 */
public final class RSA {
    private final static int _RSA_DEFAULT_KEY_SIZE = 2048;
    private final static String _RSA_ALGO = "RSA";
    private final static String _RSA_CIPHER_ALGO = "RSA";
    // NOTE: ECB on RSA is interpreted by java security as a null cipher algo
    private final static String _RSA_CIPHER_ALGO_MODE_ECB = "ECB";
    private final static String _RSA_ALGO_PADDING_OAEP =
            "OAEPWithSHA-256AndMGF1Padding";
    private final static String _SHA512_WITH_RSA = "SHA512withRSA";

    private static KeyFactory keyFactory;

    static {
        try {
            RSA.keyFactory = KeyFactory.getInstance(_RSA_ALGO);
        }
        catch(NoSuchAlgorithmException e) { // this should never happen
            assert false;
            RSA.keyFactory = null;
        }
    }

    private static Cipher generateCipher() {
        try {
            return Cipher.getInstance(_RSA_CIPHER_ALGO + "/" +
                    _RSA_CIPHER_ALGO_MODE_ECB + "/" + _RSA_ALGO_PADDING_OAEP);
        }
        /*
         * Covers the following:
         *  - NoSuchAlgorithmException
         *  - NoSuchPaddingException
         */
        catch(GeneralSecurityException e) { // this should never happen
            assert false;
            return null;
        }
    }

    private static Signature generateSignature() {
        try {
            return Signature.getInstance(_SHA512_WITH_RSA);
        }
        catch(NoSuchAlgorithmException e) { // this should never happen
            assert false;
            return null;
        }
    }

    public static PrivateKey generatePrivate(byte[] key)
            throws InvalidKeySpecException {
        PKCS8EncodedKeySpec pkcs8Spec = new PKCS8EncodedKeySpec(key);
        return RSA.keyFactory.generatePrivate(pkcs8Spec);
    }

    public static PublicKey generatePublic(byte[] key)
```

```
        throws InvalidKeySpecException {
    X509EncodedKeySpec x509Spec = new X509EncodedKeySpec(key);
    return RSA.keyFactory.generatePublic(x509Spec);
}

/**
 *
 */
public static byte[] sign(byte[] key, InputStream is) throws IOException {
    Signature sig = RSA.generateSignature();
    try {
        PrivateKey privKey = RSA.generatePrivate(key);
        sig.initSign(privKey);
        sig.update(IOUtils.toByteArray(is));
        return sig.sign();
    }
    /*
     * Covers the following:
     *  - InvalidKeySpecException from (RSA.)KeyFactory.generatePrivate()
     *  - InvalidKeyException from Signature.initSign()
     *  - SignatureException from Signature.update() and Signature.sign()
     */
    catch(GeneralSecurityException e) {
        System.err.println("GeneralSecurityException at sign() !!!");
    }
    return null; // catch all scenario if something has gone wrong
}


/**
 *
 */
public static boolean verify(byte[] key, InputStream is, byte[] sigToVer)
        throws IOException {
    Signature sig = RSA.generateSignature();
    try {
        PublicKey pubKey = RSA.generatePublic(key);
        sig.initVerify(pubKey);
        sig.update(IOUtils.toByteArray(is));
        return sig.verify(sigToVer);
    }
    /*
     * Covers the following:
     *  - InvalidKeySpecException from (RSA.)KeyFactory.generatePublic()
     *  - InvalidKeyException from Signature.initVerify()
     *  - SignatureException from Signature.update() and Signature.verify()
     */
    catch(GeneralSecurityException e) {
        System.err.println("GeneralSecurityException at verify() !!!");
    }
    return false; // catch all scenario if something has gone wrong
}

/**
 *
 */
public static void encrypt(byte[] key, InputStream is, OutputStream os)
        throws IOException {
    byte[] src = IOUtils.toByteArray(is);

    Cipher cipher = RSA.generateCipher();
    try {
        PublicKey pubKey = RSA.generatePublic(key);
        cipher.init(Cipher.ENCRYPT_MODE, pubKey);
        os.write(cipher.doFinal(src));
        os.flush();
    }
    /*
     * Covers the following:
     *  - InvalidKeySpecException from (RSA.)KeyFactory.generatePublic()
     *  - InvalidKeyException from Cipher.init()
     *  - IllegalBlockSizeException from Cipher.doFinal()
     */
    catch(GeneralSecurityException e) {
        System.err.println("GeneralSecurityException at encrypt() !!!");
    }
}

/**
 *
 */
public static void decrypt(byte[] key, InputStream is, OutputStream os)
        throws IOException {
    byte[] src = IOUtils.toByteArray(is);

    Cipher cipher = RSA.generateCipher();
    try {
        PrivateKey privKey = RSA.generatePrivate(key);
        cipher.init(Cipher.DECRYPT_MODE, privKey);
        os.write(cipher.doFinal(src));
        os.flush();
    }
    /*
     * Covers the following:
     *  - InvalidKeySpecException from (RSA.)KeyFactory.generatePrivate()
     *  - InvalidKeyException from Cipher.init()
     *  - IllegalBlockSizeException from Cipher.doFinal()
     */
    catch(GeneralSecurityException e) {
        System.err.println("GeneralSecurityException at decrypt() !!!");
    }
}

/**
 *
 */
// this does not guarantee however that the read data will be correct
public static byte[] readPublicKeyFromPEM(InputStream source)
        throws RSADecoderException {
    PemReader pemReader = new PemReader(new InputStreamReader(source));
    PemObject obj = null;
    try {
        obj = pemReader.readPemObject();
    }
    catch(DecoderException e) {
        throw new RSADecoderException(e.getMessage(), e.getCause());
    }
    catch(IOException e) {
        // NOTE: this method of "filtering" exceptions is only
        for(StackTraceElement el : e.getStackTrace()) {
            if(el.getClassName() ==
                    "org.bouncycastle.util.io.pem.PemReader"
                    && el.getMethodName() == "loadObject") {
                throw new RSADecoderException(e.getMessage(),
                        e.getCause());
            }
        }
    }
    if(obj == null) {
        throw new RSADecoderException("Empty key read from PEM file");
    }
    return obj.getContent();
}

/**
 *
 */
// this does not guarantee that the bytes written are valid public keys
public static void writePublicKeyToPEM(byte[] pubKey,
        OutputStream destination) throws IOException {
    PemWriter pemWriter = new PemWriter(
            new OutputStreamWriter(destination));
    pemWriter.writeObject(new PemObject("PUBLIC KEY", pubKey));
    pemWriter.close();
}

/**
 *
 */
public static void generateKeys(OutputStream privKeyOs,
        OutputStream pubKeyOs) throws IOException {
    KeyPairGenerator kpg = null;
    try {
        kpg = KeyPairGenerator.getInstance("RSA");
    }
    catch(NoSuchAlgorithmException e) {
        System.err.println(
                "NoSuchAlgorithmException at RSA.generateKeys() !!!");
    }

    kpg.initialize(_RSA_DEFAULT_KEY_SIZE);
    KeyPair kp = kpg.genKeyPair();

    byte[] pubKey = kp.getPublic().getEncoded();
    byte[] privKey = kp.getPrivate().getEncoded();

    pubKeyOs.write(pubKey);
    pubKeyOs.flush();
    privKeyOs.write(privKey);
    privKeyOs.flush();
}
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/core/util/crypto/RSADecoderException.java
--------------------------------------------------------------

package com.transmisms.core.util.crypto;
```

```
public final class RSADecoderException extends Exception {

    /**
     * @see Exception#Exception(String) Exception
     */
    public RSADecoderException(String message) {
        super(message);
    }

    /**
     * @see Exception#Exception(String, Throwable) Exception
     */
    public RSADecoderException(String message, Throwable cause) {
        super(message, cause);
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/smsftp/protocol/PdlEntry.java
--------------------------------------------------------------

package com.transmisms.smsftp.protocol;

import com.transmisms.core.protocol.PDUType;
import com.transmisms.smsftp.protocol.SmsftpSession.SessionState;



// access is between public and protected, see Java documentation
final class PdlEntry {
    public final SessionState s;
    public final PDUType t;



    public PdlEntry(SessionState s, PDUType t) {
        this.s = s;
        this.t = t;
    }



    @Override
    public boolean equals(Object o) {
        if(o instanceof PdlEntry) {
            return this.s == ((PdlEntry)o).s && this.t == ((PdlEntry)o).t;
        }
        else {
            return false;
        }
    }

    @Override
    public int hashCode() {
        return this.s.hashCode() + this.t.hashCode();
    }
    // CAUTION: this has no protection against nulls and problematic types
}



--------------------------------------------------------------
Filename:
src/main/java/com/transmisms/smsftp/protocol/SmsftpBinaryPduDecoder.java
--------------------------------------------------------------

package com.transmisms.smsftp.protocol;

import com.transmisms.core.protocol.BinaryPDUDecoder;
import com.transmisms.core.protocol.PDU;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.core.protocol.PDUMalformedException;
import com.transmisms.core.protocol.InvalidCRCException;
import static com.transmisms.smsftp.protocol.SmsftpFacade.DataSizes;
import static com.transmisms.smsftp.protocol.SmsftpFacade.DataStrSizes;

import com.transmisms.core.util.codec.Base85;

import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;
import java.util.Map;
import java.util.UUID;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.function.Function;
import java.io.IOException;


public class SmsftpBinaryPduDecoder extends BinaryPDUDecoder {
    private final static List<String> validNoencCec = new ArrayList<String>();
    private final static List<String> validEncCec = new ArrayList<String>();

    private static final Map<Character, PDUType> prefixToTypeLookup =
            new Hashtable<>();
    private static final Map<PDUType, Character> typeToPrefixLookup =
            new Hashtable<>();
    // utility function for use for the maps above
    private static void biDiAddToLookup(Character c, PDUType p) {
        PDUType evalValue =
                SmsftpBinaryPduDecoder.prefixToTypeLookup.put(c, p);
        assert null == evalValue;
        Character evalValue2 =
                SmsftpBinaryPduDecoder.typeToPrefixLookup.put(p, c);
        assert null == evalValue2;
    }

    private static SmsftpBinaryPduDecoder thisInstance;
    static {
        // initialize instance
        SmsftpBinaryPduDecoder.thisInstance = new SmsftpBinaryPduDecoder(
                (c) -> SmsftpBinaryPduDecoder.prefixToTypeLookup.get(c),
                (p) -> SmsftpBinaryPduDecoder.typeToPrefixLookup.get(p));

        // populate custom lookup maps
        SmsftpBinaryPduDecoder.biDiAddToLookup('a',
                SmsftpPDUType.SMSFTP_INIT);
        SmsftpBinaryPduDecoder.biDiAddToLookup('b',
                SmsftpPDUType.SMSFTP_META);
        SmsftpBinaryPduDecoder.biDiAddToLookup('c',
                SmsftpPDUType.SMSFTP_READY);
        SmsftpBinaryPduDecoder.biDiAddToLookup('d',
                SmsftpPDUType.SMSFTP_DATA);
        SmsftpBinaryPduDecoder.biDiAddToLookup('e',
                SmsftpPDUType.SMSFTP_END);
        SmsftpBinaryPduDecoder.biDiAddToLookup('E',
                SmsftpPDUType.SMSFTP_END_WAIT);
        SmsftpBinaryPduDecoder.biDiAddToLookup('i',
                SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY);
        SmsftpBinaryPduDecoder.biDiAddToLookup('j',
                SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY);

        // populate valid CEC values
        validNoencCec.add("XXC");
        validNoencCec.add("XXX");
        validEncCec.add("XEX");
        validEncCec.add("CEX");
        validEncCec.add("XEC");
        validEncCec.add("CEC");
    }

    protected SmsftpBinaryPduDecoder(
            Function<Character, PDUType> prefixToTypeFunc,
            Function<PDUType, Character> typeToPrefixFunc) {
        super(prefixToTypeFunc, typeToPrefixFunc);
    }

    public static SmsftpBinaryPduDecoder getInstance() {
        return SmsftpBinaryPduDecoder.thisInstance;
    }

    private static String getSubstrAndIncrement(String body,
            AtomicInteger pointer, int substrSize) {
        return body.substring(pointer.get(), pointer.addAndGet(substrSize));
    }

    @Override
    protected List<Object> decodeBody(
            PDUType pduType, String encoded)
            throws PDUMalformedException {
        String body = encoded.substring(1, encoded.length()-2);
        List<Object> decodedData = new ArrayList<>();

        if(pduType instanceof SmsftpPDUType) {
            SmsftpPDUType sPduType = (SmsftpPDUType)pduType;
            switch(sPduType) {
                case SMSFTP_INIT: {
                    AtomicInteger pointer = new AtomicInteger(
                            DataStrSizes.HEXSEGMENTID.getSize());

                    if(body.length() < 1) {
                        throw new PDUMalformedException(
                                "SMSFTP_INIT too short", encoded);
                    }

                    // get Segment Id
                    int segmentId = getHexSegmentId(body, 3, encoded,
                            "SMSFTP_INIT", true);
                    decodedData.add(segmentId);

                    // get Session Id
                    String sessionIdStr = getSubstrAndIncrement(body,
                            pointer, DataStrSizes.SESSIONID.getSize());
                    UUID sessionId = base85ToUUID(sessionIdStr, encoded,
                            "SMSFTP_INIT");
                    // then add the extracted Session Id
                    decodedData.add(sessionId);

                    if(segmentId != 0) { // encrypted connections
                        String excessData = body.substring(pointer.get());
                        decodedData.add(excessData);
                    } // do nothing for unencrypted connections
                    break;
                }


                case SMSFTP_META: {
                    AtomicInteger pointer = new AtomicInteger(
                            DataStrSizes.HEXSEGMENTID.getSize());

                    if(body.length() < 1) {
                        throw new PDUMalformedException(
                                "SMSFTP_META too short", encoded);
                    }

                    // get Segment Id
                    int segmentId = getHexSegmentId(body, 3, encoded,
                            "SMSFTP_META", true);
                    decodedData.add(segmentId);

                    // get Session Id
                    String sessionIdStr = getSubstrAndIncrement(body,
                            pointer, DataStrSizes.SESSIONID.getSize());
                    UUID sessionId = base85ToUUID(sessionIdStr, encoded,
                            "SMSFTP_META");
                    decodedData.add(sessionId);
                    if(segmentId == 1 || segmentId == 0) {
                        // get CEC
                        String cecStr = getSubstrAndIncrement(body, pointer,
                                DataStrSizes.CEC.getSize());
                        // verify CEC
                        if(!(segmentId == 1 ? validEncCec :
                                validNoencCec).contains(cecStr)) {
                            throw new PDUMalformedException(
                                    pduType.toString() +
                                    " invalid CEC flag: " + cecStr, encoded);
                        }
                        // get Segment Count
                        String segmentCountStr = getSubstrAndIncrement(body,
                                pointer, DataStrSizes.SEGMENTCOUNT.getSize());
                        Integer segmentCount = base85ToInteger(segmentCountStr,
                                encoded, "SMSFTP_META");

                        // add values afterwards
                        decodedData.add(cecStr);
                        decodedData.add(segmentCount);
                    }

                    // extract checksum/signature
                    String excessDataStr = body.substring(pointer.get());
                    Object excessData = null;
                    if(segmentId == 0) { // unencrypted connection
                        try {
                            byte[] excessDataBytes
                                    = Base85.decode(excessDataStr);
                            if(excessDataBytes.length !=
                                    DataSizes.SHA1_SUM.getSize()) {
                                throw new PDUMalformedException(
                                        "SMSFTP_META's SHA1 checksum's size
invalid",
                                        encoded);
                            }
                            excessData = excessDataBytes;
                        }
                        catch(IOException e) {
                            throw new PDUMalformedException(
                                    "SMSFTP_META's SHA1 checksum is corrupt",
                                    encoded);
                        }
                    }
                    else { // encrypted connection
                        excessData = excessDataStr;
                    }
                    // finally, add excess data
                    decodedData.add(excessData);
                    break;
                }

                case SMSFTP_END:
                case SMSFTP_END_WAIT:
                case SMSFTP_READY: {
                    if(body.length() < DataStrSizes.SESSIONID.getSize()) {
                        throw new PDUMalformedException(
                                pduType.toString() +
                                " too short to fit Session ID", encoded);
                    }
                    String sessionIdStr = body.substring(0,
                            DataStrSizes.SESSIONID.getSize());
                    // decode sessionId from here
                    UUID sessionId = base85ToUUID(sessionIdStr, encoded,
                            pduType.toString());
                    decodedData.add(sessionId); // and add to decoded data
                    break;
                }


                case SMSFTP_DATA: {
                    // check if size is less than SEGMENTID
                    if(body.length() < DataStrSizes.SEGMENTID.getSize()) {
                        throw new PDUMalformedException(
                                "SMSFTP_DATA too short to fit Segment ID",
                                encoded);
                    }
                    String segmentIdStr = body.substring(0,
                            DataStrSizes.SEGMENTID.getSize());
                    // decode segmentId from here
                    Integer segmentId = null;
                    try {
                        byte[] segmentIdBytes = Base85.decode(segmentIdStr);
                        ByteBuffer bb = ByteBuffer.allocate(Integer.BYTES);
                        for(int i = 0;
                                i < Integer.BYTES - segmentIdBytes.length;
                                i++) {
                            bb.put((byte)0);
                        }
                        // flip() is required for reading
                        bb.put(segmentIdBytes).flip();

                        segmentId = bb.getInt();
                    }
                    catch(IOException e) {
                        throw new PDUMalformedException(
                                "SMSFTP_DATA's Segment ID is corrupt",
                                encoded);
                    }
                    decodedData.add(segmentId); // and add to decoded data
                    // get data body from here
                    String dataBody = body.substring(
                            DataStrSizes.SEGMENTID.getSize(), body.length());
                    decodedData.add(dataBody);
                    break;
                }


                case SMSFTP_INITIATOR_IDENTITY:
                case SMSFTP_RESPONDER_IDENTITY: {
                    AtomicInteger pointer = new AtomicInteger(
                            DataStrSizes.HEXSEGMENTID.getSize());

                    int segmentId = getHexSegmentId(body, 5, encoded,
                            pduType.toString(), false);
                    decodedData.add(segmentId);

                    String excessData = body.substring(pointer.get());
                    decodedData.add(excessData);
                    break;
                }
                default: {
                    // code should NEVER reach here
                    throw new AssertionError("Uncaught PDU Type: " + pduType);
                }
            }
        }
        else {
            return null;
        }
        return decodedData;
    }
}

--------------------------------------------------------------
Filename: src/main/java/com/transmisms/smsftp/protocol/SmsftpFacade.java
--------------------------------------------------------------
```

```java
package com.transmisms.smsftp.protocol;

import com.transmisms.core.protocol.CoreProtocolFacade;
import com.transmisms.core.protocol.PDU;
import com.transmisms.core.protocol.PDUEncoder;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.core.protocol.Presenter.MessageType;
import com.transmisms.core.protocol.Presenter.PromptType;
import com.transmisms.core.protocol.Presenter.StatusType;
import com.transmisms.core.protocol.SegmentManager;
import com.transmisms.core.protocol.SmsService;
import com.transmisms.core.util.codec.Base85;
import com.transmisms.core.util.crypto.AES;
import com.transmisms.core.util.crypto.RSA;
import com.transmisms.smsftp.protocol.SmsftpSession.Role;
import com.transmisms.smsftp.protocol.SmsftpSession.SessionIntent;
import com.transmisms.smsftp.protocol.SmsftpSession.SessionState;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.nio.ByteBuffer;
import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.util.List;
import java.util.Map;
import java.util.concurrent.Callable;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Future;
import java.util.function.BiFunction;

import java.io.IOException;
import java.security.spec.InvalidKeySpecException;



public class SmsftpFacade extends CoreProtocolFacade {
    private static final String USER_ACCEPT_STATUS_CODE = "0002";
    public final String SUBPROTOCOL_NAME = "smsftp";

    public final SmsftpSession smsftpSession;

    protected final CountDownLatch processingLatch = new CountDownLatch(1);
    private final CountDownLatch acceptPromptLatch = new CountDownLatch(1);



    public SmsftpFacade(SmsftpSession smsftpSession, SmsService smsService,
            int maxPduLifetime) {
        super(smsftpSession,
                SmsftpBinaryPduDecoder.getInstance(),
                SmsftpTextBasedPduDecoder.getInstance(),
                smsService,
                maxPduLifetime);
        this.smsftpSession = smsftpSession;
    }

    public SmsftpFacade(SmsftpSession smsftpSession, SmsService smsService) {
        this(smsftpSession, smsService, -1);
    }



    public SmsftpSession getSmsftpSession() {
        return this.smsftpSession;
    }

    @Override
    public String getSubprotocolName() {
        return this.SUBPROTOCOL_NAME;
    }

    @Override
    protected String generateHeadPayload() {
        if(this.getSmsftpSession().getSessionIntent() ==
                SessionIntent.DATA_TRANSFER) {
            if(this.getSmsftpSession().getEncryptionFlag()) { // encrypted
                return "enc";
            }
            return "noenc"; // unencrypted
        }
        else if(this.getSmsftpSession().getSessionIntent()
                == SessionIntent.PAIR) {
            return "noenc"; // technically noenc, but not practically necessary
        }
        else { // should never happen
            return null;
        }
    }

    @Override
    protected void processHeadPayload(Object[] headData) {
        if(this.getSmsftpSession().getSessionIntent() ==
                SessionIntent.DATA_TRANSFER) {
            if(headData != null && headData.length >= 5 &&
                    headData[3] instanceof String) {
                String encryptionMode = (String)headData[3];
                if(encryptionMode.equals("enc")) {
                    this.getSmsftpSession().setCecEncryption(true);
                }
                else if(encryptionMode.equals("noenc")) {
                    this.getSmsftpSession().setCecEncryption(false);
                }
            }
        }
        else if(this.getSmsftpSession().getSessionIntent()
                == SessionIntent.PAIR) {
            // do nothing
        }
        else { // should never happen
        }
    }

    @Override
    protected String[] processReqAndGenRepPayload(Object[] requestData) {
        if(requestData != null && requestData.length >= 6) {
            if(requestData.length >= 6 && requestData[4] instanceof String &&
                    requestData[4].equals("send") &&
                    requestData[5] instanceof String) { // send intent
                this.getSmsftpSession().setSessionIntent(
                        SessionIntent.DATA_TRANSFER);
                // store filename in the SmsftpSession object
                this.getSmsftpSession().setFilename((String)requestData[5]);
            }
            else if(requestData.length >= 5 &&
                    requestData[4] instanceof String &&
                    requestData[4].equals("pair")) { // pair intent
                this.getSmsftpSession().setSessionIntent(SessionIntent.PAIR);
            }
            else { // error
            }

            // just accept all valid connections
            String[] retStr = { USER_ACCEPT_STATUS_CODE };
            return retStr;
        }
        return null;
    }

    @Override
    protected final Runnable processPdu(PDU pduObject) {
        switch(this.smsftpSession.getSessionState()) {
            case TERMINATING:
                return null;
            default: {
                BiFunction<SmsftpFacade, PDU, Runnable> pdlFunction =
                        SmsftpLookupTable.get(new PdlEntry(
                        this.smsftpSession.getSessionState(),
                        pduObject.getPduType()));
                if(pdlFunction == null) { // not in lookup table
                    return null; // do nothing
                }
                else {
                    return pdlFunction.apply(this, pduObject);
                }
            }
        }
    }

    @Override
    final protected void onEstablished() {
        SmsftpSession sObject = this.getSmsftpSession();
        // extract IV from SessionIntent.DATA_TRANSFER sessions
        if(sObject.getSessionIntent() != null &&
                sObject.getSessionIntent().equals(
                SessionIntent.DATA_TRANSFER)) {
            long msb = sObject.sessionId.getMostSignificantBits();
            long lsb = sObject.sessionId.getLeastSignificantBits();
            ByteBuffer buffer = ByteBuffer.allocate(Long.BYTES*2);
            buffer.putLong(msb);
            buffer.putLong(lsb);
            sObject.iv = buffer.array();
        }
        if(sObject.getRole() == Role.RESPONDER) {
            Object[] promptArr = {MessageType.PROMPT,
                    PromptType.CONNECTION_REQUEST};
            this.setChanged();
            this.notifyObservers(promptArr);

            Future<?> userConnReqWaitFuture =  this.pool.submit(() -> {
                this.acceptPromptLatch.countDown();
            });
        }
        else { // INITIATOR
        }
    }



    public void receive() {
        this.listen();
    }

    public void insecureSendFile() {
        this.connect(this.getSmsftpSession().connection.peerMin +
                "/send/" + this.getSmsftpSession().getFilename());
    }

    public void secureSendFile() {
        this.connect(this.getSmsftpSession().connection.peerMin +
                "/send/" + this.getSmsftpSession().getFilename());
    }

    public void pair() {
        // register our own segment manager
        this.registerSegMan(SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY,
                this.smsftpSession.identitySegments);
        // set session waiting state first (similar to CONNECTED)
        SmsftpFacade.setStateAndNotify(this, SessionState.INITIATOR_WAIT);
        // then connect as usual
        this.connect(this.getSmsftpSession().connection.peerMin +
                "/pair");
    }

    public void acceptRequestAfterPrompt(byte[] enckey, byte[] authkey) {
        SmsftpFacade.logAndNotify(this, MessageType.INFO,
                "User accepted the smsftp request");

        // continue all operations if needed
        // wake the awaiting thread
        this.acceptPromptLatch.countDown();

        if(this.getSmsftpSession().getSessionIntent() ==
                SessionIntent.DATA_TRANSFER) {
            // send INIT PDU(s) depending on encryption
            if(this.smsftpSession.getEncryptionFlag()) { // encrypted
                // check first if the keys given are legitimate
                if(enckey == null || authkey == null) {
                    assert false;
                }
                try {
                    RSA.generatePublic(enckey);
                    RSA.generatePublic(authkey);
                }
                catch(InvalidKeySpecException e) {
                    assert false;
                }

                // set session's authkey
                this.getSmsftpSession().authkey = authkey;
                // construct AES key to be used
                this.getSmsftpSession().fileEncKey = AES.generateKey();
                // encrypt aesKey
                ByteArrayInputStream is = new ByteArrayInputStream(
                        this.getSmsftpSession().fileEncKey);
                ByteArrayOutputStream os = new ByteArrayOutputStream();
                try {
                    RSA.encrypt(enckey, is, os);
                    os.flush();
                }
                catch(IOException e) {
                    assert false; // NOTE: should never happen since we are
                                  //       just using ByteArray streams
                }
                String encEncAesKey = Base85.encode(os.toByteArray());
                String[] encEncAesKeyParts = new String[3];
                encEncAesKeyParts[1] = "";
                encEncAesKeyParts[2] = "";
                if(encEncAesKey.length() < 135) {
                    encEncAesKeyParts[0] = encEncAesKey;
                }
                else if(encEncAesKey.length() >= 135 &&
                        encEncAesKey.length() < 270) {
                    encEncAesKeyParts[0] = encEncAesKey.substring(0, 135);
                    encEncAesKeyParts[1] = encEncAesKey.substring(135);
                }
                else {
                    encEncAesKeyParts[0] = encEncAesKey.substring(0, 135);
                    encEncAesKeyParts[1] = encEncAesKey.substring(135, 270); //
135
                    encEncAesKeyParts[2] = encEncAesKey.substring(270,
                            encEncAesKey.length());
                }
                this.smsftpSession.encodedInitPdu = new String[3];
                // construct the INIT PDUs to be sent
                this.smsftpSession.encodedInitPdu[0] = (new PDUEncoder(true))
                        .appendPrefix(SmsftpPDUType.SMSFTP_INIT,
                        SmsftpBinaryPduDecoder.getInstance())
                        .appendHexSegmentId(1) // multipart ids start at 1
                        .appendSessionId(this.smsftpSession.sessionId)
                        .appendString(encEncAesKeyParts[0])
                        .finalizePDU();
                this.smsftpSession.encodedInitPdu[1] = (new PDUEncoder(true))
                        .appendPrefix(SmsftpPDUType.SMSFTP_INIT,
                        SmsftpBinaryPduDecoder.getInstance())
                        .appendHexSegmentId(2) // multipart ids start at 1
                        .appendSessionId(this.smsftpSession.sessionId)
                        .appendString(encEncAesKeyParts[1])
                        .finalizePDU();
                this.smsftpSession.encodedInitPdu[2] = (new PDUEncoder(true))
                        .appendPrefix(SmsftpPDUType.SMSFTP_INIT,
                        SmsftpBinaryPduDecoder.getInstance())
                        .appendHexSegmentId(3) // multipart ids start at 1
                        .appendSessionId(this.smsftpSession.sessionId)
                        .appendString(encEncAesKeyParts[2])
                        .finalizePDU();
                // finally, send the messages
                this.smsService.sendMessage(
                        this.smsftpSession.encodedInitPdu[0]);
                this.smsService.sendMessage(
                        this.smsftpSession.encodedInitPdu[1]);
                this.smsService.sendMessage(
                        this.smsftpSession.encodedInitPdu[2]);

            }
            else { // unencrypted
                this.smsftpSession.encodedInitPdu = new String[1];
                // construct and send the INIT PDU
                this.smsftpSession.encodedInitPdu[0] = (new PDUEncoder(true))
                        .appendPrefix(SmsftpPDUType.SMSFTP_INIT,
                        SmsftpBinaryPduDecoder.getInstance())
                        .appendHexSegmentId(0) // non-multipart ids start at 0
                        .appendSessionId(this.smsftpSession.sessionId)
                        .finalizePDU();
                this.smsService.sendMessage(
                        this.smsftpSession.encodedInitPdu[0]);
            }
            // register encodedInitPdu Array for retransmission
            this.registerRetrans(SmsftpPDUType.SMSFTP_INIT,
                    this.smsftpSession.encodedInitPdu);
            // finally, set state
            SmsftpFacade.setStateAndNotify(this, SessionState.INIT_SENT);
        }
        else if(this.getSmsftpSession().getSessionIntent()
                == SessionIntent.PAIR) {
            // generate keys and encode them properly
            this.smsftpSession.generatePairingIdentity();
            byte[] pubEncKey =
                    this.smsftpSession.localEncKeypair.getPublic()
                    .getEncoded();
            byte[] pubAuthKey =
                    this.smsftpSession.localAuthKeypair.getPublic()
                    .getEncoded();
            String concatenatedKeys =
                    Base85.encode(pubEncKey) + "¥" + Base85.encode(pubAuthKey);

            // construct and send the SMSFTP_RESPONDER_IDENTITY PDUs
            int p = 0;
            for(int i = 0; i < 5; i++) {
                // construct each PDU
                String payload = "";
                if(p < concatenatedKeys.length()) {
                    payload = concatenatedKeys.substring(p,
                            (p+155 < concatenatedKeys.length()) ? p+155 :
                            concatenatedKeys.length());
                }
                this.smsftpSession.encodedKeyPdu[i] = (new PDUEncoder(true))
                        .appendPrefix(SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY,
                        SmsftpBinaryPduDecoder.getInstance())
                        .appendHexSegmentId(i+1) // multipart ids start at 1
                        .appendString(payload)
                        .finalizePDU();
                // then send each messasge
                this.smsService.sendMessage(
                        this.smsftpSession.encodedKeyPdu[i]);
                p=p+155;
            }
            SmsftpFacade.logAndNotify(this, MessageType.INFO,
                    "Sent last responder identity PDU to receiver");
            // register encodedKeyPdu Array for retransmission
            this.registerRetrans(SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY,
                    this.smsftpSession.encodedKeyPdu);
            // register our own segment manager
        this.registerSegMan(SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY,
                this.smsftpSession.identitySegments);
        // start a new thread for repeated sending of END PDUs
        this.sendEndPduRepeatedly();
        // finally, set state
        SmsftpFacade.setStateAndNotify(this,
                SessionState.RESPONDER_KEY_EXCHANGE);
    }
    else { // should never happen
    }
}

public void rejectRequestAfterPrompt(String reason) {
}

public void rejectRequestAfterPrompt() {
    SmsftpFacade.logAndNotify(this, MessageType.INFO,
            "User rejected the smsftp request");
    this.rejectRequestAfterPrompt(null);
}

/**
 *
 * @return a reference to the internal byte[] object used for storing
 *         data to be sent/received
 */
public byte[] getDataBytes() {
    return this.smsftpSession.dataBytes;
}

protected void sendEndPduRepeatedly() {
    this.submitToInternalPool(() -> {
        SmsftpFacade.logAndNotify(this, MessageType.INFO,
                "Waiting until peer has completed");
        // loop until interrupt or SmsService error
        while(!Thread.currentThread().isInterrupted() &&
                !this.smsftpSession.getSessionState().equals(
                SessionState.TERMINATING)) {
            try {
                this.smsService.sendMessage(new PDUEncoder(true)
                        .appendPrefix(SmsftpPDUType.SMSFTP_END,
                        SmsftpBinaryPduDecoder.getInstance())
                        .appendSessionId(this.smsftpSession.sessionId)
                        .finalizePDU());
                // try again after maxPduLifetime
                Thread.sleep(
                        this.coreSession.connection.maxPduLifetime);
            }
            catch(InterruptedException e) {
                // preserve interrupt status
                Thread.currentThread().interrupt();
                break; // then break the loop
            }
        }
    });
}

/*
// operations for prompt feedbacks (for UI, CLI, tester, etc.)
public void continueOperation();
public void cancelOperation(); //
                               // errcode depends on the situation
                               // - no prompt: user cancel
                               // - prompt: the real cause
                               //           i.e. timeout
                               //
*/



//// exposed functions for use by package and subclasses
@Override
protected List registerRetrans(PDUType k, List<String> v) {
    return super.registerRetrans(k, v);
}

@Override
protected List registerRetrans(PDUType k, String[] a) {
    return super.registerRetrans(k, a);
}

@Override
protected SegmentManager registerSegMan(PDUType k, SegmentManager v) {
    return super.registerSegMan(k, v);
}

@Override
protected <T> Future<T> submitToInternalPool(Callable<T> task) {
    return super.submitToInternalPool(task);
}
```

```
    @Override
    protected Future<?> submitToInternalPool(Runnable task) {
        return super.submitToInternalPool(task);
    }



    //// utility functions
    protected static void logAndNotify(SmsftpFacade self, MessageType t,
            String message) {
        Object[] logArr = {t, message};
        SmsftpFacade.setChangedAndNotifyObservers(self, logArr);
    }

    protected static void setStateAndNotify(SmsftpFacade self,
            SessionState newState) {
        SmsftpFacade.getAndSetStateAndNotify(self, newState);
    }

    protected static SessionState getAndSetStateAndNotify(SmsftpFacade self,
            SessionState newState) {
        SessionState oldState =
                self.smsftpSession.getAndSetSessionState(newState);
        SmsftpFacade.setChangedAndNotifyObservers(self, MessageType.STATUS,
                StatusType.STATE_CHANGED, oldState, newState);
        return oldState;
    }

    protected static void setChangedAndNotifyObservers(CoreProtocolFacade self,
            Object ... o) {
        CoreProtocolFacade.setChangedAndNotifyObservers(self, o);
    }
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/smsftp/protocol/SmsftpLookupTable.java
--------------------------------------------------------------

package com.transmisms.smsftp.protocol;

import com.transmisms.core.protocol.CoreProtocolFacade;
import com.transmisms.core.protocol.PDU;
import com.transmisms.core.protocol.PDUEncoder;
import com.transmisms.core.protocol.Presenter.MessageType;
import com.transmisms.core.protocol.Presenter.StatusType;
import com.transmisms.core.protocol.SegmentManager;
import com.transmisms.smsftp.protocol.SmsftpSession.SessionState;

import com.transmisms.core.util.codec.Base85;
import com.transmisms.core.util.compression.XZ;
import com.transmisms.core.util.crypto.AES;
import com.transmisms.core.util.crypto.RSA;
import com.transmisms.core.util.crypto.RSA;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.security.MessageDigest;
import java.util.Arrays;
import java.util.Hashtable;
import java.util.Map;
import java.util.concurrent.Future;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.function.BiFunction;
import org.apache.commons.io.output.CountingOutputStream;

import java.io.IOException;
import java.security.NoSuchAlgorithmException;
import java.util.concurrent.ExecutionException;


public class SmsftpLookupTable {
    /**
     * Container class for use by encryptAndCompact()
     */
    private static class CompEncData {
        public boolean compressAfter = false;
        public byte[] data = null;
    }


    private static final Map<PdlEntry,
            BiFunction<SmsftpFacade, PDU, Runnable>> lookupTable =
            new Hashtable<>();
```

```
static { // initialize lookupTable
    BiFunction<SmsftpFacade, PDU, Runnable> evalValue =
            null; // only used for asserts
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.CONNECTED, SmsftpPDUType.SMSFTP_INIT),
            (t, u) -> SmsftpLookupTable.createInitReceivedTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.INIT_RECEIVED, SmsftpPDUType.SMSFTP_INIT),
            (t, u) -> SmsftpLookupTable.createResendMetaTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.INIT_RECEIVED, SmsftpPDUType.SMSFTP_READY),
            (t, u) -> SmsftpLookupTable.createReadyReceivedTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.INIT_SENT, SmsftpPDUType.SMSFTP_META),
            (t, u) -> SmsftpLookupTable.createMetaReceivedTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.RECEIVING_DATA, SmsftpPDUType.SMSFTP_META),
            (t, u) -> SmsftpLookupTable.createResendReadyTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.RECEIVING_DATA,
            SmsftpPDUType.SMSFTP_DATA),
            (t, u) -> SmsftpLookupTable.createProcessDataTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.RECEIVING_DATA, SmsftpPDUType.SMSFTP_END),
            (t, u) -> SmsftpLookupTable.createSendWaitTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.INITIATOR_WAIT,
            SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
            (t, u) -> SmsftpLookupTable.createProcFirstRespDataTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.INITIATOR_WAIT, SmsftpPDUType.SMSFTP_END),
            (t, u) -> SmsftpLookupTable.createSendWaitTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.INITIATOR_KEY_EXCHANGE,
            SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
            (t, u) -> SmsftpLookupTable.createProcessRespDataTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.INITIATOR_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_END),
            (t, u) -> SmsftpLookupTable.createSendWaitTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.RESPONDER_KEY_EXCHANGE,
            SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
            (t, u) -> SmsftpLookupTable.createProcessInitDataTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.RESPONDER_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_END),
            (t, u) -> SmsftpLookupTable.createSendWaitTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.COMPLETED, SmsftpPDUType.SMSFTP_END),
            (t, u) ->
            SmsftpLookupTable.createCompletedEndReceivedTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.COMPLETE_WAIT, SmsftpPDUType.SMSFTP_END),
            (t, u) ->
            SmsftpLookupTable.createCompleteWaitEndReceivedTask(t, u));
    assert null == evalValue;
    evalValue = lookupTable.put(
            new PdlEntry(
            SessionState.TERMINATING, SmsftpPDUType.SMSFTP_END),
            (t, u) ->
            SmsftpLookupTable.createCompletedEndReceivedTask(t, u));
    assert null == evalValue;

    PdlEntry[] doNothingEntries = {
        new PdlEntry(
        SessionState.SENDING_DATA, SmsftpPDUType.SMSFTP_INIT),
        new PdlEntry(
        SessionState.SENDING_DATA, SmsftpPDUType.SMSFTP_READY),
        new PdlEntry(
        SessionState.COMPLETED, SmsftpPDUType.SMSFTP_INIT),
        new PdlEntry(
        SessionState.COMPLETED, SmsftpPDUType.SMSFTP_META),
        new PdlEntry(
        SessionState.COMPLETED, SmsftpPDUType.SMSFTP_READY),
        new PdlEntry(
        SessionState.COMPLETED, SmsftpPDUType.SMSFTP_DATA),
        new PdlEntry(
        SessionState.COMPLETED, SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
        new PdlEntry(
        SessionState.COMPLETED, SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
        new PdlEntry(
        SessionState.COMPLETE_WAIT, SmsftpPDUType.SMSFTP_INIT),
        new PdlEntry(
        SessionState.COMPLETE_WAIT, SmsftpPDUType.SMSFTP_META),
        new PdlEntry(
        SessionState.COMPLETE_WAIT, SmsftpPDUType.SMSFTP_READY),
        new PdlEntry(
        SessionState.COMPLETE_WAIT, SmsftpPDUType.SMSFTP_DATA),
        new PdlEntry(
        SessionState.COMPLETE_WAIT,
        SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
        new PdlEntry(
        SessionState.COMPLETE_WAIT,
        SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
        new PdlEntry(
        SessionState.COMPLETE_WAIT, SmsftpPDUType.SMSFTP_END_WAIT),
        new PdlEntry(
        SessionState.TERMINATING, SmsftpPDUType.SMSFTP_INIT),
        new PdlEntry(
        SessionState.TERMINATING, SmsftpPDUType.SMSFTP_META),
        new PdlEntry(
        SessionState.TERMINATING, SmsftpPDUType.SMSFTP_READY),
        new PdlEntry(
        SessionState.TERMINATING, SmsftpPDUType.SMSFTP_DATA),
        new PdlEntry(
        SessionState.TERMINATING, SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
        new PdlEntry(
        SessionState.TERMINATING, SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
        new PdlEntry(
        SessionState.TERMINATING, SmsftpPDUType.SMSFTP_END_WAIT),
    };
    for(PdlEntry e : doNothingEntries) {
        evalValue = lookupTable.put(e,
                (t, u) -> SmsftpLookupTable.createDoNothingTask(t, u));
        assert null == evalValue;
    }

    PdlEntry[] errorEntries = {
        new PdlEntry(
        SessionState.CONNECTED, SmsftpPDUType.SMSFTP_META),
        new PdlEntry(
        SessionState.CONNECTED, SmsftpPDUType.SMSFTP_READY),
        new PdlEntry(
        SessionState.CONNECTED, SmsftpPDUType.SMSFTP_DATA),
        new PdlEntry(
        SessionState.CONNECTED, SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
        new PdlEntry(
        SessionState.CONNECTED, SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
        new PdlEntry(
        SessionState.CONNECTED, SmsftpPDUType.SMSFTP_END),
        new PdlEntry(
        SessionState.CONNECTED, SmsftpPDUType.SMSFTP_END_WAIT),
        new PdlEntry(
        SessionState.INIT_RECEIVED, SmsftpPDUType.SMSFTP_META),
        new PdlEntry(
        SessionState.INIT_RECEIVED, SmsftpPDUType.SMSFTP_DATA),
        new PdlEntry(
        SessionState.INIT_RECEIVED,
SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
        new PdlEntry(
        SessionState.INIT_RECEIVED,
SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
        new PdlEntry(
        SessionState.INIT_RECEIVED, SmsftpPDUType.SMSFTP_END),
        new PdlEntry(
        SessionState.INIT_RECEIVED, SmsftpPDUType.SMSFTP_END_WAIT),
        new PdlEntry(
        SessionState.INIT_SENT, SmsftpPDUType.SMSFTP_INIT),
        new PdlEntry(
        SessionState.INIT_SENT, SmsftpPDUType.SMSFTP_READY),
        new PdlEntry(
            SessionState.INIT_SENT, SmsftpPDUType.SMSFTP_DATA),
            new PdlEntry(
            SessionState.INIT_SENT, SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
            new PdlEntry(
            SessionState.INIT_SENT, SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
            new PdlEntry(
            SessionState.INIT_SENT, SmsftpPDUType.SMSFTP_END),
            new PdlEntry(
            SessionState.INIT_SENT, SmsftpPDUType.SMSFTP_END_WAIT),
            new PdlEntry(
            SessionState.SENDING_DATA, SmsftpPDUType.SMSFTP_META),
            new PdlEntry(
            SessionState.SENDING_DATA, SmsftpPDUType.SMSFTP_DATA),
            new PdlEntry(
            SessionState.SENDING_DATA,
            SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
            new PdlEntry(
            SessionState.SENDING_DATA,
            SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
            new PdlEntry(
            SessionState.SENDING_DATA, SmsftpPDUType.SMSFTP_END),
            new PdlEntry(
            SessionState.SENDING_DATA, SmsftpPDUType.SMSFTP_END_WAIT),
            new PdlEntry(
            SessionState.RECEIVING_DATA, SmsftpPDUType.SMSFTP_INIT),
            new PdlEntry(
            SessionState.RECEIVING_DATA, SmsftpPDUType.SMSFTP_READY),
            new PdlEntry(
            SessionState.RECEIVING_DATA,
            SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
            new PdlEntry(
            SessionState.RECEIVING_DATA,
            SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
            new PdlEntry(
            SessionState.RECEIVING_DATA, SmsftpPDUType.SMSFTP_END_WAIT),
            new PdlEntry(
            SessionState.INITIATOR_WAIT, SmsftpPDUType.SMSFTP_INIT),
            new PdlEntry(
            SessionState.INITIATOR_WAIT, SmsftpPDUType.SMSFTP_META),
            new PdlEntry(
            SessionState.INITIATOR_WAIT, SmsftpPDUType.SMSFTP_READY),
            new PdlEntry(
            SessionState.INITIATOR_WAIT, SmsftpPDUType.SMSFTP_DATA),
            new PdlEntry(
            SessionState.INITIATOR_WAIT,
SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
            new PdlEntry(
            SessionState.INITIATOR_WAIT, SmsftpPDUType.SMSFTP_END_WAIT),
            new PdlEntry(
            SessionState.INITIATOR_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_INIT),
            new PdlEntry(
            SessionState.INITIATOR_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_META),
            new PdlEntry(
            SessionState.INITIATOR_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_READY),
            new PdlEntry(
            SessionState.INITIATOR_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_DATA),
            new PdlEntry(
            SessionState.INITIATOR_KEY_EXCHANGE,
            SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY),
            new PdlEntry(
            SessionState.INITIATOR_KEY_EXCHANGE,
            SmsftpPDUType.SMSFTP_END_WAIT),
            new PdlEntry(
            SessionState.RESPONDER_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_INIT),
            new PdlEntry(
            SessionState.RESPONDER_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_META),
            new PdlEntry(
            SessionState.RESPONDER_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_READY),
            new PdlEntry(
            SessionState.RESPONDER_KEY_EXCHANGE, SmsftpPDUType.SMSFTP_DATA),
            new PdlEntry(
            SessionState.RESPONDER_KEY_EXCHANGE,
            SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY),
            new PdlEntry(
            SessionState.RESPONDER_KEY_EXCHANGE,
            SmsftpPDUType.SMSFTP_END_WAIT),
            new PdlEntry(
            SessionState.COMPLETED, SmsftpPDUType.SMSFTP_END_WAIT),
        };
        for(PdlEntry e : errorEntries) {
            evalValue = lookupTable.put(e,
                    (t, u) -> SmsftpLookupTable.createPDUMismatchTask(t, u));
            assert null == evalValue;
        }
    }



    //// "action" functions
    public static Runnable createDoNothingTask(
            SmsftpFacade tObject, PDU uObject) {
        return () -> {};
    }

    public static Runnable createPDUMismatchTask(
            SmsftpFacade tObject, PDU uObject) {
        return () -> {};
    }

    public static Runnable createInitReceivedTask(
            SmsftpFacade tObject, PDU uObject) {
        return () -> { tObject.submitToInternalPool(() -> {
            // initialize segment manager if not yet initialized
            SmsftpSession sObject = tObject.smsftpSession;

            boolean isChanged = false;
            if(sObject.getEncryptionFlag()) {
                isChanged = sObject.initSegments.compareAndSet(
                        null, new SegmentManager(3));
            }
            else {
                isChanged = sObject.initSegments.compareAndSet(
                        null, new SegmentManager(1));
            }
            if(isChanged) { // register new segment manager
                tObject.registerSegMan(
                        SmsftpPDUType.SMSFTP_INIT, sObject.initSegments.get());
            }

            synchronized(sObject.initSegments) {
                // validate uObject and extract available data
                Object[] uData = uObject.getData();
                if(uData == null ||
                        uData.length < 2 || !(uData[0] instanceof Integer)) {
                    SmsftpFacade.logAndNotify(tObject,
                            MessageType.TRACE,
                            "Got problematic INIT PDU data");
                    return; // just return on invalid cases
                }
                int segnum = (Integer)(uData[0]);
                SmsftpFacade.logAndNotify(tObject,
                        MessageType.TRACE, "Received INIT part " + segnum);

                // update segment and determine if all PDUs are completed
                final String[][] veriArrC = { null };
                if(sObject.getEncryptionFlag()) { // if encrypted/multipart
                    // get encrypted AES segment
                    String encAesSeg = null;
                    if(uData.length == 3 && uData[2] instanceof String) {
                        encAesSeg = (String)(uData[2]);
                    }
                    else { // invalid case, just return asap
                        SmsftpFacade.logAndNotify(tObject,
                                MessageType.TRACE, "Erratic INIT part " +
                                segnum);
                        return;
                    }

                    // update segment and check if completed
                    if(sObject.initSegments.get().updateSegment(
                            encAesSeg, segnum-1) &&
                            sObject.initSegments.get().getCompletedString()
                            != null) {
                        // set lock and exit if already locked
                        if(sObject.processingStarted.getAndSet(true)) {
                            SmsftpFacade.logAndNotify(tObject,
                                    MessageType.TRACE,
                                    "Got another INIT while processing");
                            return;
                        }

                        SmsftpFacade.logAndNotify(tObject, MessageType.DEBUG,
                                "INIT PDUs complete; reassembling segments");


                        // process INIT PDU(s) if already completed
                        final CompEncData[] xeRetValC = { null };
                        final CompEncData[] ceRetValC = { null };
                        final byte[][] aesKeyC = { null };

                        //// #p1
                        Future cecFuture = tObject.submitToInternalPool(
                                () -> {
                            // extract AES key #p1
                            try {
                                ByteArrayOutputStream baos =
                                        new ByteArrayOutputStream();
                                byte[] encAesKey = Base85.decode(
                                        sObject.initSegments.get()
                                        .getCompletedString());
                                // decrypt and validate consolidated key
                                RSA.decrypt(sObject.enckey,
            new ByteArrayInputStream(encAesKey),
            baos);
    aesKeyC[0] = baos.toByteArray();
}
catch(IOException e) {
    assert false; // should never happen
}


// check cases for CEC, CEX, and XEC #p1x3
Future xeFuture = tObject.submitToInternalPool(
        () -> {
    SmsftpFacade.logAndNotify(tObject,
            MessageType.INFO,
            "Encrypting file...");
    // encrypt file #p1.1 XEX + XEC
    xeRetValC[0] = encryptAndCompact(
            new ByteArrayInputStream(
            sObject.dataBytes), aesKeyC[0],
            sObject.iv);
    SmsftpFacade.logAndNotify(tObject,
            MessageType.INFO, "Encryption done");
});

Future ceFuture = tObject.submitToInternalPool(
        () -> {
    SmsftpFacade.logAndNotify(tObject,
            MessageType.INFO,
            "Compressing file (first pass)...");
    // compress file #p1.2 CE?
    // prepare Piped streams for use later
    final PipedInputStream pis =
            new PipedInputStream();
    PipedOutputStream tpos = null;
    try {
        tpos = new PipedOutputStream(pis);
    }
    catch(IOException e) {
            assert false; // should never happen
    }
    final PipedOutputStream pos = tpos;
    // spawn to a new thread to avoid
    // Piped* deadlocks
    Future pipedFuture =
            tObject.submitToInternalPool(() -> {
        try {
            XZ.compress(new ByteArrayInputStream(
                    sObject.dataBytes), pos);
        }
        catch(IOException e) {
            assert false; // should never happen
                          // since we're just using
                          // byte arrays or similar
        }
        SmsftpFacade.logAndNotify(tObject,
                MessageType.INFO,
                "Compression (first pass) done");
        SmsftpFacade.logAndNotify(tObject,
                MessageType.INFO,
                "Compressing file (second
pass)...");
    });
    try {
        // encrypt and try to further compress
        // compressed file #p1.2 CEC + CEX
        ceRetValC[0] = encryptAndCompact(pis,
                aesKeyC[0], sObject.iv);
        // while processing pipedFuture
        pipedFuture.get();
    }
    catch(InterruptedException e) {
        // log event
        SmsftpFacade.logAndNotify(tObject,
                MessageType.TRACE,
                "CEC processing 2 interrupted");
        // preserve interrupt status
        Thread.currentThread().interrupt();
        return; // return ASAP
    }
    // should never happen
    catch(ExecutionException e) {
        // log event
        SmsftpFacade.logAndNotify(tObject,
                MessageType.TRACE,
                "CEC processing 2 exception: " +
                e);
        return; // return ASAP
    }
    SmsftpFacade.logAndNotify(tObject,
            MessageType.INFO,
            "Compression (second pass) done");
```

```
    });

    // join here and compare sizes #p1
    try {
        xeFuture.get();
        ceFuture.get();
    }
    catch(InterruptedException e) {
        // log event
        SmsftpFacade.logAndNotify(tObject,
                MessageType.TRACE,
                "CEC processing interrupted");
        // preserve interrupt status
        Thread.currentThread().interrupt();
        return; // return ASAP
    }
    catch(ExecutionException e) { // should never happen
        // log event
        SmsftpFacade.logAndNotify(tObject,
                MessageType.TRACE,
                "CEC processing exception: " + e);
        return; // return ASAP
    }
    { // compare sizes and select the smallest one
        boolean compressPrior =
                ceRetValC[0].data.length >
                xeRetValC[0].data.length;

        sObject.cecPreCompression = compressPrior;
        sObject.cecPostCompression = compressPrior ?
                ceRetValC[0].compressAfter :
                xeRetValC[0].compressAfter;

        SmsftpFacade.logAndNotify(tObject,
                MessageType.DEBUG,
                "Converted data length: " +
                (compressPrior ?
                ceRetValC[0].data.length :
                xeRetValC[0].data.length));

        // convert to Base85 #p1
        SmsftpFacade.logAndNotify(tObject,
                MessageType.INFO,
                "Encoding data...");
        String encoded = Base85.encode(compressPrior ?
                ceRetValC[0].data : xeRetValC[0].data);
        SmsftpFacade.logAndNotify(tObject,
                MessageType.INFO, "Encoding done");
        SmsftpFacade.logAndNotify(tObject,
                MessageType.DEBUG,
                "Encoded length: " + encoded.length());

        // split encoded data
        int index = 0;
        final AtomicInteger i = new AtomicInteger(1);
        while(index < encoded.length()) {
            String str = encoded.substring(index,
                    Math.min(index+153,
                    encoded.length()));
            sObject.dataPdu.add(new PDUEncoder(true)
                    .appendPrefix(
                    SmsftpPDUType.SMSFTP_DATA,
                    SmsftpBinaryPduDecoder
                    .getInstance())
                    .appendBinarySegmentId(
                    i.getAndIncrement())
                    .appendString(str)
                    .finalizePDU());
            index+=153;
        }
        SmsftpFacade.logAndNotify(tObject,
                MessageType.DEBUG,
                "Segment count: " +
                sObject.dataPdu.size());
        // register dataPdu List for retransmission
        tObject.registerRetrans(
                SmsftpPDUType.SMSFTP_DATA,
                sObject.dataPdu);

        encoded = null;
    }
});


//// #p2
Future sigFuture = tObject.submitToInternalPool(
        () -> { // generate RSA signature #p2
    SmsftpFacade.logAndNotify(tObject,
            MessageType.INFO, "Signing...");
    try {
        byte[] signature = RSA.sign(sObject.authkey,
                        new ByteArrayInputStream(
                        sObject.dataBytes));
                String veriString = Base85.encode(signature);
                veriArrC[0] = new String[3];
                veriArrC[0][0] = veriString.substring(0, 128);
                veriArrC[0][1] =
                        veriString.substring(128, 263);
                veriArrC[0][2] = veriString.substring(263,
                        veriString.length());
            }
            catch(IOException e) {
                assert false; // should never happen
            }
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.INFO, "Signing done");
        });

        //// join p1 and p2
        try {
            cecFuture.get();
            sigFuture.get();
        }
        catch(InterruptedException e) {
            // log event
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.TRACE,
                    "CEC processing and signing interrupted");
            // preserve interrupt status
            Thread.currentThread().interrupt();
            return; // return ASAP
        }
        catch(ExecutionException e) { // should never happen
            // log event
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.TRACE,
                    "CEC processing/signing exception: " + e);
            return; // return ASAP
        }

    }
}
else { // unencrypted, single part
    // update with virtually empty contents (and can be only
    // executed once)
    if(sObject.initSegments.get().updateSegment("", segnum)) {
        // check cases for XXC and XXX #q1
        Future xxcFuture = tObject.submitToInternalPool(
                () -> {
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.INFO, "Compressing file...");
            ByteArrayOutputStream baos =
                    new ByteArrayOutputStream();
            try { // compress file
                XZ.compress(new ByteArrayInputStream(
                        sObject.dataBytes), baos);
            }
            catch(IOException e) {
                assert false; // should never happen
            }
            // compare, set flags, and encode
            String encoded = null;
            if(baos.size() < sObject.dataBytes.length) {
                sObject.cecPostCompression = true;
                encoded = Base85.encode(baos.toByteArray());
            }
            else {
                sObject.cecPostCompression = false;
                encoded = Base85.encode(sObject.dataBytes);
            }
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.INFO, "Compression done");

            // split encoded data
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.INFO, "Encoding data...");
            // split encoded data
            int index = 0;
            final AtomicInteger i = new AtomicInteger(1);
            while(index < encoded.length()) {
                String str = encoded.substring(index,
                        Math.min(index+153,
                        encoded.length()));
                sObject.dataPdu.add(new PDUEncoder(true)
                            .appendPrefix(
                            SmsftpPDUType.SMSFTP_DATA,
                            SmsftpBinaryPduDecoder
                            .getInstance())
                            .appendBinarySegmentId(
                            i.getAndIncrement())
                            .appendString(str)
                            .finalizePDU());
                index+=153;
            }
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.INFO, "Encoding done");
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.DEBUG, "Segment count: " +
                    sObject.dataPdu.size());
            // register dataPdu List for retransmission
            tObject.registerRetrans(SmsftpPDUType.SMSFTP_DATA,
                    sObject.dataPdu);
        });

        // compute SHA512 checksum #q2
        Future hashFuture = tObject.submitToInternalPool(
                () -> {
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.INFO,
                    "Computing checksum...");
            veriArrC[0] = new String[1];
            MessageDigest md = null;
            try {
                md = MessageDigest.getInstance("SHA-1");
            }
            catch(NoSuchAlgorithmException e) {
                assert false; // should never happen (min JDK7)
            }
            md.update(sObject.dataBytes);
            veriArrC[0][0] = Base85.encode(md.digest());
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.INFO,
                    "Checksum computation done");
        });

        //// join q1 and q2
        try {
            xxcFuture.get();
            hashFuture.get();
        }
        catch(InterruptedException e) {
            // log event
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.TRACE,
                    "XXC processing and hashing interrupted");
            // preserve interrupt status
            Thread.currentThread().interrupt();
            return; // return ASAP
        }
        catch(ExecutionException e) { // should never happen
            // log event
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.TRACE,
                    "XXC processing/hashing exception: " + e);
            return; // return ASAP
        }

    }
}

// if not yet complete, just do nothing and return asap
if(sObject.initSegments.get().getCompletedString() == null) {
    return;
}

// further processing for both encrypted and unencrypted

// generate CEC string from session properties
String cecStr = sObject.cecPreCompression ? "C" : "X";
cecStr += sObject.getEncryptionFlag() ? "E" : "X";
cecStr += sObject.cecPostCompression ? "C" : "X";

// construct and send META PDU(s)
PDUEncoder encoder = new PDUEncoder(true);
encoder.appendPrefix(SmsftpPDUType.SMSFTP_META,
        SmsftpBinaryPduDecoder.getInstance());
if(sObject.getEncryptionFlag()) { // encrypted
    encoder.appendHexSegmentId(1)
        .appendSessionId(sObject.sessionId)
        .appendString(cecStr)
        .appendBinarySegmentId(sObject.dataPdu.size())
        .appendString(veriArrC[0][0])
        .finalizePDU();
    sObject.encodedMetaPdu[0] = encoder.getEncoded();
    tObject.smsService.sendMessage(sObject.encodedMetaPdu[0]);
    // do the same for parts 2 and 3
    encoder = new PDUEncoder(true); // part 2
    encoder.appendPrefix(SmsftpPDUType.SMSFTP_META,
            SmsftpBinaryPduDecoder.getInstance())
            .appendHexSegmentId(2)
            .appendSessionId(sObject.sessionId)
            .appendString(veriArrC[0][1])
            .finalizePDU();
```

```
                sObject.encodedMetaPdu[1] = encoder.getEncoded();
                tObject.smsService.sendMessage(sObject.encodedMetaPdu[1]);
                encoder = new PDUEncoder(true); // part 3
                encoder.appendPrefix(SmsftpPDUType.SMSFTP_META,
                        SmsftpBinaryPduDecoder.getInstance())
                        .appendHexSegmentId(3)
                        .appendSessionId(sObject.sessionId)
                        .appendString(veriArrC[0][2])
                        .finalizePDU();
                sObject.encodedMetaPdu[2] = encoder.getEncoded();
                tObject.smsService.sendMessage(sObject.encodedMetaPdu[2]);
            }
            else { // unencrypted
                encoder.appendHexSegmentId(0) // use '0' for non-multipart
                    .appendSessionId(sObject.sessionId)
                    .appendString(cecStr)
                    .appendBinarySegmentId(sObject.dataPdu.size())
                    .appendString(veriArrC[0][0])
                    .finalizePDU();
                // finally, send the META PDU
                sObject.encodedMetaPdu[0] = encoder.getEncoded();
                tObject.smsService.sendMessage(sObject.encodedMetaPdu[0]);
            }

            // register encodedMetaPdu Array for retransmission
            tObject.registerRetrans(SmsftpPDUType.SMSFTP_META,
                    sObject.encodedMetaPdu);

            // release the processing lock here
            SmsftpFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Releasing data processing latch");
            tObject.processingLatch.countDown();

            // set state as INIT_RECEIVED afterwards
            SmsftpFacade.setStateAndNotify(
                    tObject, SessionState.INIT_RECEIVED);
        }
    });};
}

public static Runnable createResendMetaTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
    };
}

public static Runnable createReadyReceivedTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        SmsftpSession sObject = tObject.smsftpSession;

        // set state
        if(SmsftpFacade.getAndSetStateAndNotify(
                tObject, SessionState.SENDING_DATA)
                .equals(SessionState.SENDING_DATA)) {
            return; // and return if already set
        }

        // wait for the processing lock here
        try {
            SmsftpFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Waiting for data to be sent to be ready...");
            tObject.processingLatch.await();
            SmsftpFacade.logAndNotify(tObject, MessageType.DEBUG,
                    "Data ready");
        }
        catch(InterruptedException e) {
            // preserve interrupt status
            Thread.currentThread().interrupt();
            return; // exit ASAP
        }

        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Sending data to receiver...");
        sObject.dataPdu.forEach((item) -> {
            tObject.smsService.sendMessage(item);
        });
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Sent last data PDU to receiver");

        // set state as COMPLETE_WAIT afterwards
        SmsftpFacade.setStateAndNotify(
                tObject, SessionState.COMPLETE_WAIT);

        // start a new thread for repeated sending of END PDUs
        tObject.sendEndPduRepeatedly();
    };
}

public static Runnable createMetaReceivedTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        SmsftpSession sObject = tObject.smsftpSession;

        // initialize segment manager if not yet initialized
        boolean isChanged = false;
        if(sObject.getEncryptionFlag()) {
            isChanged = sObject.metaSegments.compareAndSet(
                    null, new SegmentManager(3));
        }
        else {
            isChanged = sObject.metaSegments.compareAndSet(
                    null, new SegmentManager(1));
        }
        if(isChanged) { // register new segment manager
            tObject.registerSegMan(
                    SmsftpPDUType.SMSFTP_META, sObject.metaSegments.get());
        }

        synchronized(sObject.metaSegments) {
            // validate uObject and extract available data
            Object[] uData = uObject.getData();
            if(uData == null ||
                    uData.length < 2 || !(uData[0] instanceof Integer)) {
                return; // just return on invalid cases
            }
            int segnum = (Integer)(uData[0]);
            SmsftpFacade.logAndNotify(tObject,
                    MessageType.TRACE, "Received META part " + segnum);

            // update segment and determine if all PDUs are completed
            if(sObject.getEncryptionFlag()) { // if encrypted/multipart
                // get encrypted AES segment and further validation
                String sigSeg = null;
                if(segnum == 1 && uData.length == 5 &&
                        uData[4] instanceof String) { // first part
                    if(!(uData[2] instanceof String &&
                            ((String)uData[2]).length() == 3) ||
                            !(uData[3] instanceof Integer)) {
                        return; // invalid, return asap
                    }
                    sigSeg = (String)(uData[4]);
                }
                else if((segnum == 2 || segnum == 3) &&
                        uData.length == 3 &&
                        uData[2] instanceof String) { // second+ part
                    sigSeg = (String)(uData[2]);
                }
                else { // invalid case, just return asap
                    return;
                }

                // update segment and check if completed
                if(sObject.metaSegments.get().updateSegment(
                        sigSeg, segnum-1) &&
                        sObject.metaSegments.get().getCompletedString()
                        != null) {
                    // process META PDU(s) if already completed

                    try { // extract signature and store on session object
                        sObject.signature = Base85.decode(
                                sObject.metaSegments.get()
                                .getCompletedString());
                    }
                    catch(IOException e) {
                        assert false; // should never happen
                    }
                }

                if(segnum == 1) {
                    // extract the session parameters
                    String cecStr = (String)(uData[2]);
                    if(cecStr.charAt(0) == 'C') {
                        sObject.cecPreCompression = true;
                    }
                    if(cecStr.charAt(2) == 'C') {
                        sObject.cecPostCompression = true;
                    }
                    // prepare the new segment manager, if not
                    // yet initialized
                    int segcount = (Integer)(uData[3]);
                    boolean dIsChanged =
                            sObject.dataSegments.compareAndSet(null,
                            new SegmentManager(segcount));
                    if(dIsChanged) {
                        tObject.registerSegMan(
                                SmsftpPDUType.SMSFTP_DATA,
                                sObject.dataSegments.get());
                    }
                }

            }
            else { // unencrypted, single part
                // get segment and further validation
                if(!(uData.length >= 5 && uData[4] instanceof byte[] &&
                        uData[2] instanceof String &&
                        uData[3] instanceof Integer)) {
                    return; // invalid, return asap
                }
                byte[] sigSeg = (byte[])(uData[4]);

                // "update" segment and force "completion"
                sObject.metaSegments.get().updateSegment("", 0);

                // extract signature and store on session object
                sObject.signature = sigSeg;

                // extract session parameters
                String cecStr = (String)(uData[2]);
                if(cecStr.charAt(0) == 'C') {
                    return; // invalid, return asap
                }
                if(cecStr.charAt(1) == 'E') {
                    return; // invalid, return asap
                }
                if(cecStr.charAt(2) == 'C') {
                    sObject.cecPostCompression = true;
                }

                // prepare the new segment manager, if not yet initialized
                int segCount = (Integer)(uData[3]);
                boolean dIsChanged =
                        sObject.dataSegments.compareAndSet(null,
                        new SegmentManager(segCount));
                if(dIsChanged) {
                    tObject.registerSegMan(
                            SmsftpPDUType.SMSFTP_DATA,
                            sObject.dataSegments.get());
                }
            }

            if(sObject.metaSegments.get().getCompletedString() != null) {
                // send READY PDU
                tObject.smsService.sendMessage(new PDUEncoder(true)
                        .appendPrefix(SmsftpPDUType.SMSFTP_READY,
                        SmsftpBinaryPduDecoder.getInstance())
                        .appendSessionId(sObject.sessionId)
                        .finalizePDU());
                // set state as RECEIVING_DATA afterwards
                SmsftpFacade.setStateAndNotify(
                        tObject, SessionState.RECEIVING_DATA);
            }
        }
    };
}

public static Runnable createResendReadyTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
    };
}

public static Runnable createProcessDataTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        SmsftpSession sObject = tObject.smsftpSession;
        // NOTE: no need to initialize since we have done that on
        // receipt of SMSFTP_META PDU

        // validate contents of PDU
        Object[] uData = uObject.getData();
        if(uData == null || uData.length < 2 ||
                !(uData[0] instanceof Integer &&
                    uData[1] instanceof String)) {
            return; // just return on invalid cases
        }
        int segnum = (Integer)(uData[0]);
        String dataSeg = (String)(uData[1]);

        // update extracted data and process the whole file on completion
        if(sObject.dataSegments.get().updateSegment(dataSeg, segnum-1) &&
                sObject.dataSegments.get().getCompletedString() != null) {
            byte[] rawBytes = null;
            try {
                rawBytes = Base85.decode(
                        sObject.dataSegments.get().getCompletedString());
            }
            catch(IOException e) {
                assert false; // should never happen
            }
            ByteArrayInputStream bais = new ByteArrayInputStream(rawBytes);
            ByteArrayOutputStream baos = null;
```

```
try {
    if(sObject.cecPostCompression) { // --C
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Decompressing file (second pass)...");
        baos = new ByteArrayOutputStream();
        XZ.decompress(new ByteArrayInputStream(rawBytes),
                baos);
        rawBytes = baos.toByteArray();
        // preemptive cleanup
        bais = null;
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Decompression (second pass) done");
    }
    if(sObject.getEncryptionFlag()) { // -E-
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Decrypting file...");
        baos = new ByteArrayOutputStream();
        AES.decrypt(sObject.fileEncKey, sObject.iv,
                new ByteArrayInputStream(rawBytes), baos);
        rawBytes = baos.toByteArray();
        // preemptive cleanup
        bais = null;
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Decryption done");
    }
    if(sObject.cecPreCompression) { // C--
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Decompressing file (first pass)...");
        baos = new ByteArrayOutputStream();
        XZ.decompress(new ByteArrayInputStream(rawBytes),
                baos);
        rawBytes = baos.toByteArray();
        // preemptive cleanup
        bais = null;
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Decompression (first pass) done");
    }
}
catch(IOException e) {
    assert false; // should never happen
}

// verify rawBytes
SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
        "Verifying...");
boolean authentic = false;
try {
    if(sObject.getEncryptionFlag()) { // encrypted
        authentic = RSA.verify(sObject.authkey,
                new ByteArrayInputStream(rawBytes),
                sObject.signature);
    }
    else { // unencrypted
        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("SHA-1");
        }
        catch(NoSuchAlgorithmException e) {
            assert false; // should never happen (min JDK7)
        }
        md.update(rawBytes);
        byte[] checksum = md.digest();
        authentic = Arrays.equals(checksum, sObject.signature);
    }
}
catch(IOException e) {
    assert false; // should never happen
}
SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
        "Verification done");
if(authentic) {
    SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
            "File complete and verified");
    // move rawBytes to SmsftpSession.dataBytes
    sObject.dataBytes = rawBytes;
    // notify presenters about the success
    Object[] noticeArr = { MessageType.STATUS,
            StatusType.NOTICE, SessionState.COMPLETED,
            sObject.dataBytes };
    SmsftpFacade.setChangedAndNotifyObservers(
            tObject, noticeArr);
    // reply with END PDU ASAP
    if(sObject.endReceived) {
        // for formalities sake, we'll transition to COMPLETED
        SmsftpFacade.setStateAndNotify(
                tObject, SessionState.COMPLETED);
        // send END PDU
        tObject.smsService.sendMessage(new PDUEncoder(true)
                .appendPrefix(SmsftpPDUType.SMSFTP_END,
                SmsftpBinaryPduDecoder.getInstance())
                .appendSessionId(
                            tObject.smsftpSession.sessionId)
                            .finalizePDU());
                    // set state as TERMINATING afterwards
                    SmsftpFacade.setStateAndNotify(
                            tObject, SessionState.TERMINATING);
                    return; // no need for other later calls
                }
            }
            else { // sender's file authentication failed
                SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                        "File failed verification");
                // notify presenters about the failure
                Object[] noticeArr = { MessageType.STATUS,
                        StatusType.NOTICE, SessionState.COMPLETED,
                        null };
                SmsftpFacade.setChangedAndNotifyObservers(
                        tObject, noticeArr);
            }

            // finally, set session state to COMPLETED
            SmsftpFacade.setStateAndNotify(
                    tObject, SessionState.COMPLETED);
        }
        else { // not yet complete; misc actions
            // force sending of CORE_RETRANSMISSION PDUs on
            // certain percentages (optional for spec; added for ui)
            int progress = (sObject.getLastDataSegNumReceived() -
                    sObject.getSkippedSegNumsLength())*100 /
                    sObject.getDataSegCount();
            if(progress - sObject.lastDataMilestone >= 20) {
                sObject.lastDataMilestone = progress;
                tObject.sendRetransPdu();
            }
        }

        // notify presenters about arrival of new DATA PDU
        SmsftpFacade.setChangedAndNotifyObservers(tObject,
                MessageType.STATUS, StatusType.NOTICE,
                SessionState.RECEIVING_DATA);
    };
}

public static Runnable createSendWaitTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Not yet finished; requesting peer to wait longer...");
        SmsftpSession sObject = tObject.smsftpSession;
        // enable endReceived flag
        sObject.endReceived = true;
        { // send an END_WAIT PDU
            tObject.smsService.sendMessage(new PDUEncoder(true)
                    .appendPrefix(SmsftpPDUType.SMSFTP_END_WAIT,
                    SmsftpBinaryPduDecoder.getInstance())
                    .appendSessionId(sObject.sessionId)
                    .finalizePDU());
        }
        // and request retransmission of missing PDUs
        tObject.sendRetransPdu();
    };
}

public static Runnable createProcFirstRespDataTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        // set to not waiting
        SmsftpFacade.setStateAndNotify(
                tObject, SessionState.INITIATOR_KEY_EXCHANGE);
        // process data received
        SmsftpLookupTable.createProcessRespDataTask(
                tObject, uObject).run();
        // generate keys and encode them properly
        tObject.smsftpSession.generatePairingIdentity();
        byte[] pubEncKey =
                tObject.smsftpSession.localEncKeypair.getPublic()
                .getEncoded();
        byte[] pubAuthKey =
                tObject.smsftpSession.localAuthKeypair.getPublic()
                .getEncoded();
        String concatenatedKeys =
                Base85.encode(pubEncKey) + "¥" + Base85.encode(pubAuthKey);

        // then start sending SMSFTP_INITIATOR_IDENTITY PDUs
        int p = 0;
        for(int i = 0; i < 5; i++) {
            // construct each PDU
            String payload = "";
            if(p < concatenatedKeys.length()) {
                payload = concatenatedKeys.substring(p,
                        (p+155 < concatenatedKeys.length()) ? p+155 :
                        concatenatedKeys.length());
            }
            tObject.smsftpSession.encodedKeyPdu[i] = (new PDUEncoder(true))
                    .appendPrefix(SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY,
                    SmsftpBinaryPduDecoder.getInstance())
                    .appendHexSegmentId(i+1) // multipart ids start at 1
                    .appendString(payload)
                    .finalizePDU();
            // then send each messasge
            tObject.smsService.sendMessage(
                    tObject.smsftpSession.encodedKeyPdu[i]);
            p=p+155;
        }
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Sent last initiator identity PDU to receiver");
        // register encodedKeyPdu Array for retransmission
        tObject.registerRetrans(SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY,
                tObject.smsftpSession.encodedKeyPdu);
    };
}

public static Runnable createProcessRespDataTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        SmsftpLookupTable.processIdentityData(tObject, uObject,
                SessionState.COMPLETE_WAIT);
    };
}

public static Runnable createProcessInitDataTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        SmsftpLookupTable.processIdentityData(tObject, uObject,
                SessionState.COMPLETED);
    };
}

public static Runnable createCompletedEndReceivedTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "Received END request. Finalizing session...");
        // send END PDU
        tObject.smsService.sendMessage(new PDUEncoder(true)
                .appendPrefix(SmsftpPDUType.SMSFTP_END,
                SmsftpBinaryPduDecoder.getInstance())
                .appendSessionId(tObject.smsftpSession.sessionId)
                .finalizePDU());
        // set state as TERMINATING afterwards
        SmsftpFacade.setStateAndNotify(
                tObject, SessionState.TERMINATING);
    };
}

public static Runnable createCompleteWaitEndReceivedTask(
        SmsftpFacade tObject, PDU uObject) {
    return () -> {
        SmsftpFacade.logAndNotify(tObject, MessageType.INFO,
                "All parties finished. Terminating session...");
        SmsftpFacade.setStateAndNotify(
                tObject, SessionState.TERMINATING);
        // then disconnect asap
        tObject.disconnect(CoreProtocolFacade.COMPLETED_STATUS_CODE);
    };
}



//// utility functions
private static void processIdentityData(SmsftpFacade tObject, PDU uObject,
        SessionState completeState) {
    SmsftpSession sObject = tObject.smsftpSession;
    // validate contents of PDU
    Object[] uData = uObject.getData();
    if(uData == null || uData.length < 2 ||
            !(uData[0] instanceof Integer &&
            uData[1] instanceof String)) {
        return; // just return on invalid cases
    }
    int segnum = (Integer)(uData[0]);
    if(segnum < 1 || segnum > 5) {
        return; // just return on invalid cases
    }
    String dataSeg = (String)(uData[1]);

    // update extracted data and process whole string on completion
    if(sObject.identitySegments.updateSegment(dataSeg, segnum-1) &&
            sObject.identitySegments.getCompletedString() != null) {
        String[] splitStr = sObject.identitySegments
                .getCompletedString().split("¥");
        String b85PubEncKey = splitStr[0];
        String b85PubAuthKey = splitStr[1];
```

```
            try {
                sObject.peerEnckey = Base85.decode(b85PubEncKey);
                sObject.peerAuthkey = Base85.decode(b85PubAuthKey);
            }
            catch(IOException e) {
                assert false; // should never happen
            }
            // finally, set necessary session state
            SmsftpFacade.setStateAndNotify(
                    tObject, completeState);
            // notify presenters about completion
            SmsftpFacade.setChangedAndNotifyObservers(
                    tObject, MessageType.STATUS, StatusType.NOTICE,
                    completeState);
            // start a new thread for repeated sending of END PDUs
            tObject.sendEndPduRepeatedly();
        }
    }

    /**
     * Encrypt stream using given AES key then tries compress data further, if
     * possible
     *
     * @return CompEncData containing details if resulting encrypted data is
     *         compressed and the actual data
     */
    private static CompEncData encryptAndCompact(InputStream is,
            byte[] aesKey, byte[] iv) {
        CompEncData retVal = new CompEncData();
        ByteArrayOutputStream encBaos = new ByteArrayOutputStream();

        // encrypt bytes -E?
        try {
            AES.encrypt(aesKey, iv, is, encBaos);
        }
        catch(IOException e) {
            assert false; // should never happen since we're just using
                          // byte arrays
        }
        byte[] encBytes = encBaos.toByteArray();
        encBaos = null; // cleanup asap

        // compress encrypted bytes -EC
        ByteArrayOutputStream compEncBaos = new ByteArrayOutputStream();
        CountingOutputStream compEncCos =
                new CountingOutputStream(compEncBaos);
        try {
            XZ.compress(new ByteArrayInputStream(encBytes), compEncCos);
        }
        catch(IOException e) {
            assert false; // should never happen since we're just using
                          // byte arrays
        }

        // set proper flags and return values
        if(compEncCos.getByteCount() < encBytes.length) { // use encryption
            retVal.compressAfter = true;
            retVal.data = compEncBaos.toByteArray();
        }
        else { // don't use encryption
            retVal.compressAfter = false;
            retVal.data = encBytes;
        }

        return retVal;
    }



    // convenience get() function
    public static BiFunction<SmsftpFacade, PDU, Runnable>
            get(PdlEntry e) {
        return SmsftpLookupTable.lookupTable.get(e);
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/smsftp/protocol/SmsftpPDUType.java
--------------------------------------------------------------

package com.transmisms.smsftp.protocol;

import com.transmisms.core.protocol.PDUType.PDUSubType;
import com.transmisms.core.protocol.PDUType;



public enum SmsftpPDUType implements PDUType<SmsftpPDUType> {
    SMSFTP_INIT                 (PDUSubType.BINARY, PduSegmentIdFormat.HEX, 1),
    SMSFTP_META                 (PDUSubType.BINARY, PduSegmentIdFormat.HEX, 1),
    SMSFTP_READY                (PDUSubType.BINARY, 0),
    SMSFTP_DATA                 (PDUSubType.BINARY, PduSegmentIdFormat.BINARY),
    SMSFTP_INITIATOR_IDENTITY   (PDUSubType.BINARY, PduSegmentIdFormat.HEX),
    SMSFTP_RESPONDER_IDENTITY   (PDUSubType.BINARY, PduSegmentIdFormat.HEX),
    SMSFTP_END                  (PDUSubType.BINARY, 0),
    SMSFTP_END_WAIT             (PDUSubType.BINARY, 0);



    public final PDUSubType pduSubType;

    public final PduSegmentIdFormat segIdFormat;

    public final int sessionIdPosition;



    // default access control since enums are restricted
    SmsftpPDUType(PDUSubType pduSubType) {
        this(pduSubType, PduSegmentIdFormat.NONE);
    }

    SmsftpPDUType(PDUSubType pduSubType, PduSegmentIdFormat segIdFormat) {
        this(pduSubType, segIdFormat, -1);
    }

    SmsftpPDUType(PDUSubType pduSubType, int sessionIdPosition) {
        this(pduSubType, PduSegmentIdFormat.NONE, sessionIdPosition);
    }

    SmsftpPDUType(PDUSubType pduSubType, PduSegmentIdFormat segIdFormat,
            int sessionIdPosition) {
        this.pduSubType = pduSubType;
        this.segIdFormat = segIdFormat;
        this.sessionIdPosition = sessionIdPosition;
    }



    @Override
    public SmsftpPDUType valueOf() {
        return valueOf(this.name());
    }

    @Override
    public PDUSubType getPduSubType() {
        return this.pduSubType;
    }

    @Override
    public PduSegmentIdFormat getPduSegmentIdFormat() {
        return this.segIdFormat;
    }

    @Override
    public int getSessionIdPosition() {
        return this.sessionIdPosition;
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/smsftp/protocol/SmsftpPresenter.java
--------------------------------------------------------------

package com.transmisms.smsftp.protocol;

import com.transmisms.core.protocol.Connection.ConnectionState;
import com.transmisms.core.protocol.Presenter;
import com.transmisms.core.protocol.Presenter.MessageType;
import com.transmisms.core.protocol.Presenter.PromptType;
import com.transmisms.core.protocol.Presenter.StatusType;
import com.transmisms.smsftp.protocol.SmsftpSession.SessionState;

import java.util.Arrays;
import java.util.Observable;



public /*abstract*/ class SmsftpPresenter extends Presenter {
    private final SmsftpFacade smsftpFacade;



    public SmsftpPresenter(SmsftpFacade smsftpFacade) {
        this.smsftpFacade = smsftpFacade;
    }
    public void receive() {
        this.smsftpFacade.receive();
    }

    public void insecureSendFile() {
        this.smsftpFacade.insecureSendFile();
    }

    public void secureSendFile() {
        this.smsftpFacade.secureSendFile();
    }

    public void pair() {
        this.smsftpFacade.pair();
    }

    @Override
    public final void update(Observable o, Object arg) {
        // check for unwanted cases
        if(arg == null || !(arg instanceof Object[])) {
            return; // do nothing
        }
        Object[] args = (Object[])arg;
        if(args.length <= 1 || args.length == 0 ||
                !(args[0] instanceof MessageType)) {
            return; // do nothing
        }
        this.processMessage(o, args);
    }

    final private void processMessage(Observable o, Object[] args) {
        // args[0] is guaranteed instanceof MessageType
        MessageType messageType = (MessageType)args[0];

        switch(messageType) {
            case PROMPT: {
                if(!(args[1] instanceof PromptType)) { // if invalid input
                    return; // just do nothing and return ASAP
                }
                PromptType promptType = (PromptType)args[1];
                String promptMessage = "";
                if(args.length >= 3 && args[2] != null
                        && args[2] instanceof String) {
                    promptMessage = (String)(args[2]);
                }
                this.onUserPrompt(promptType, promptMessage);
                break;
            }
            case STATUS: {
                if(args.length < 3 ||
                        !(args[1] instanceof StatusType)) { // if invalid input
                    return; // just do nothing and return ASAP
                }
                StatusType statusType = (StatusType)args[1];
                switch(statusType) {
                    case STATE_CHANGED: {
                        if(args.length < 4) {
                            return; // just do nothing and return ASAP
                        }
                        if((args[2] instanceof SessionState) &&
                                (args[3] instanceof SessionState)) {
                            SessionState oldState = (SessionState)(args[2]);
                            SessionState newState = (SessionState)(args[3]);
                            this.onStatusUpdate(statusType,
                                    oldState, newState);
                        }
                        else if((args[2] instanceof ConnectionState) &&
                                (args[3] instanceof ConnectionState)) {
                            ConnectionState oldState =
                                    (ConnectionState)(args[2]);
                            ConnectionState newState =
                                    (ConnectionState)(args[3]);
                            this.onStatusUpdate(statusType,
                                    oldState, newState);
                        }
                        else {
                            return; // just do nothing and return ASAP
                        }
                        break;
                    }
                    case NOTICE: {
                        if(args[2] instanceof SessionState) {
                            SessionState sState = (SessionState)(args[2]);
                            Object[] carrier = { };
                            if(args.length >= 4) {
                                carrier = Arrays.copyOfRange(args, 3,
                                        args.length);
                            }
                            this.onStatusUpdate(statusType, sState, carrier);
                        }
                        else if(args[2] instanceof ConnectionState) {
                            ConnectionState cState =
                                    (ConnectionState)(args[2]);
                            Object[] carrier = { };
                            if(args.length >= 4) {
                                carrier = Arrays.copyOfRange(args, 3,
                                        args.length);
                            }
                            this.onStatusUpdate(statusType, cState, carrier);
                        }
                        else {
                            return; // just do nothing and return ASAP
                        }
                        break;
                    }
                    default: {
                        assert false; // should never happen
                    }
                }
                break;
            }
            case FATAL:
            case ERROR:
            case WARN:
            case INFO:
            case DEBUG:
            case TRACE: {
                // check if the rest of the payload is valid
                if(!(args.length == 2 && args[1] instanceof String)) {
                    return; // do not log anything that is not loggable
                }
                // and get logMessage from args
                String logMessage = (String)(args[1]);
                // finally log message
                this.onLogMessage(messageType, logMessage);
                break;
            }
            default: {
                return; // just do nothing and return ASAP
            }
        }
    }

    protected void onLogMessage(MessageType mType, String message) {}

    protected void onStatusUpdate(StatusType sType, SessionState sState,
            Object ... args) {}

    protected void onStatusUpdate(StatusType sType, ConnectionState oldState,
            Object ... args) {}

    protected void onUserPrompt(PromptType pType, String message) {}
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/smsftp/protocol/SmsftpSession.java
--------------------------------------------------------------

package com.transmisms.smsftp.protocol;

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.SegmentManager;
import com.transmisms.core.protocol.Session;
import com.transmisms.core.util.crypto.RSA;

import java.security.KeyPair;
import java.security.PrivateKey;
import java.security.PublicKey;

import java.io.ByteArrayOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;
import java.util.concurrent.atomic.AtomicBoolean;
import java.util.concurrent.atomic.AtomicReference;

import java.io.IOException;
import java.security.spec.InvalidKeySpecException;


public class SmsftpSession extends Session {
    public enum Role {
            INITIATOR,     // SENDER
            RESPONDER      // RECEIVER
    }
```

```
public enum SessionState {
        CONNECTED,              // all
        INIT_RECEIVED,          // snd
        INIT_SENT,              // rcv
        SENDING_DATA,           // snd
        RECEIVING_DATA,         // rcv
        INITIATOR_KEY_EXCHANGE, // kx-ini
        INITIATOR_WAIT,         // kx-ini
        RESPONDER_KEY_EXCHANGE, // kx-rsp
        COMPLETED,              // rcv, kx-*
        COMPLETE_WAIT,          // snd, kx-*
        TERMINATING,            // all
}

public enum SessionIntent { DATA_TRANSFER, PAIR }



private final AtomicReference<SessionState> sessionStateRef =
        new AtomicReference<>(SessionState.CONNECTED);
private SessionIntent sessionIntent = null;
private final Role role;

private String filename = null;
private int fileSize = -1;

protected boolean cecPreCompression = false;
private boolean cecEncryption = false;
protected boolean cecPostCompression = false;

protected boolean endReceived = false; // for RESPONDER only

protected byte[] iv = null;
protected byte[] enckey = null;
protected byte[] authkey = null;
protected byte[] fileEncKey = null;
protected byte[] signature = null;

protected byte[] dataBytes = null;

// only applicable to DATA_TRANSFER intent
protected String[] encodedInitPdu; // only applicable to RECEIVER
protected String[] encodedMetaPdu; // only applicable to INITIATOR
protected String encodedReadyPdu; // only applicable to RECEIVER
protected final AtomicReference<SegmentManager> initSegments =
        new AtomicReference<>(); // only applicable to INITIATOR
protected final AtomicReference<SegmentManager> metaSegments =
        new AtomicReference<>(); // only applicable to RECEIVER
protected final AtomicReference<SegmentManager> dataSegments =
        new AtomicReference<>(); // only applicable to RECEIVER
protected final List<String> dataPdu = // only applicable to INITIATOR
        new ArrayList<>();
protected volatile int lastDataMilestone = -1;

protected final AtomicBoolean processingStarted = new AtomicBoolean(false);

// only applicable to PAIRING intent
public KeyPair localEncKeypair = null;
public KeyPair localAuthKeypair = null;
public byte[] peerEnckey = null;
public byte[] peerAuthkey = null;
protected final String[] encodedKeyPdu = new String[5];
protected final SegmentManager identitySegments = new SegmentManager(5);


/**
 * Constructs an SmsftpSession for sending data
 *
 * @param enckey private RSA key to be used in AES key decryption
 * @param authkey private RSA key to be used in creating the auth signature
 * @throws NullPointerException
 * @throws IllegalArgumentException
 */
private SmsftpSession(Connection connection, Role role, String filename,
        byte[] dataBytes, byte[] enckey, byte[] authkey)
        throws NullPointerException, IllegalArgumentException {
    super(connection, UUID.randomUUID());

    boolean cecEncryption;
    if(enckey == null && authkey == null) { // unencrypted
        cecEncryption = false;
    }
    else if(enckey != null && authkey != null) { // encrypted
        cecEncryption = true;
    }
    else { // one of the keys is missing!
        throw new IllegalArgumentException("enckey and authkey must " +
                "be either both non-null or both null");
    }
    this.cecEncryption = cecEncryption;

    if(role == Role.INITIATOR) {
        // check if dataBytes is non-null
        if(dataBytes == null) {
            throw new IllegalArgumentException("dataBytes cannot be null");
        }

        this.dataBytes = dataBytes;
        this.fileSize = dataBytes.length;
    }
    else {
        //this.dataBytes = null;
    }
    this.role = role;
    this.sessionIntent = SessionIntent.DATA_TRANSFER;
    this.filename = filename;

    // if sender on encrypted session, construct necessary keys
    if(role == Role.INITIATOR && this.cecEncryption) {
        // construct proper keypair from enckey and authkey
        try {
            PrivateKey aPrivKey = RSA.generatePrivate(authkey);
            PrivateKey ePrivKey = RSA.generatePrivate(enckey);
            // finally, set the keys if valid
            this.enckey = enckey;
            this.authkey = authkey;
        }
        catch(InvalidKeySpecException e) {
            throw new IllegalArgumentException(
                    "Invalid public/private key");
        }
    }

    // construct buffers
    if(this.role == Role.INITIATOR && this.cecEncryption) {
        this.encodedMetaPdu = new String[3];
    }
    else if(this.role == Role.INITIATOR && !this.cecEncryption) {
        this.encodedMetaPdu = new String[1];
    }
}

/**
 * Constructs an SmsftpSession for pairing
 */
private SmsftpSession(Connection connection, Role role) {
    super(connection, UUID.randomUUID());
    this.role = role;
    this.sessionIntent = null;

    if(role == Role.INITIATOR) {
        this.sessionIntent = SessionIntent.PAIR;
    }
}
/**
 * Constructs an SmsftpSession for receiving
 */
private SmsftpSession(Connection connection) {
    super(connection, null);

    this.role = Role.RESPONDER;
    this.sessionIntent = null;
}


/**
 *
 * @param enckey public RSA key to be used in AES key decryption
 * @param authkey public RSA key to be used in creating the auth signature
 * @throws NullPointerException
 * @throws IllegalArgumentException
 */
public static SmsftpSession generateSenderDataSession(
        Connection connection, String filename, byte[] dataBytes,
        byte[] enckey, byte[] authkey) {
    return new SmsftpSession(connection, Role.INITIATOR, filename,
            dataBytes, enckey, authkey);
}

/**
 *
 */
public static SmsftpSession generateInitiatorPairSession(
        Connection connection) {
    return new SmsftpSession(connection, Role.INITIATOR);
}

/**
```

```
     *
     */
    public static SmsftpSession generateReceiverSession(Connection connection) {
        return new SmsftpSession(connection);
    }

    public Role getRole() {
        return this.role;
    }

    public boolean getEncryptionFlag() {
        return this.cecEncryption;
    }

    public SessionIntent getSessionIntent() {
        return this.sessionIntent;
    }

    public int getDataSegCount() {
        return this.dataSegments.get().segCount;
    }

    public int getLastDataSegNumReceived() {
        return this.dataSegments.get().getLastSegmentReceived();
    }

    public int getSkippedSegNumsLength() {
        return this.dataSegments.get().getSkippedSegNums().length;
    }

    public int getDataSize() {
        return this.dataPdu.size();
    }

    /* old methods below */

    public String getFilename() {
        return this.filename;
    }

    public void generatePairingIdentity() {
        ByteArrayOutputStream privBaos = new ByteArrayOutputStream();
        ByteArrayOutputStream pubBaos = new ByteArrayOutputStream();
        try {
            // generate encryption key pair
            RSA.generateKeys(privBaos, pubBaos);
            privBaos.flush();
            pubBaos.flush();
            PrivateKey privKey = RSA.generatePrivate(privBaos.toByteArray());
            PublicKey pubKey = RSA.generatePublic(pubBaos.toByteArray());
            this.localEncKeypair = new KeyPair(pubKey, privKey);

            // generate authentication key pair
            privBaos = new ByteArrayOutputStream();
            pubBaos = new ByteArrayOutputStream();
            RSA.generateKeys(privBaos, pubBaos);
            privBaos.flush();
            pubBaos.flush();
            privKey = RSA.generatePrivate(privBaos.toByteArray());
            pubKey = RSA.generatePublic(pubBaos.toByteArray());
            this.localAuthKeypair = new KeyPair(pubKey, privKey);
        }
        catch(InvalidKeySpecException e) {
            assert false; // code should never reach here
            return;
        }
        catch(IOException e) {
            assert false; // code should never reach here
            return;
        }
    }

    public void setFilename(String filename) {
        this.filename = filename;
    }

    /**
     * Does nothing if this is an INITIATOR session.
     */
    public void setCecEncryption(boolean cecEncryption) {
        if(this.getRole() == Role.RESPONDER) {
            this.cecEncryption = cecEncryption;
        }
    }

    /**
     * Does nothing if sessionIntent is already set
     */
    public void setSessionIntent(SessionIntent sessionIntent) {
        if(this.sessionIntent == null) {
            this.sessionIntent = sessionIntent;
        }
    }

    public SessionState getSessionState() {
        return this.sessionStateRef.get();
    }

    public void setSessionState(SessionState sessionState) {
        this.getAndSetSessionState(sessionState);
    }

    public SessionState getAndSetSessionState(SessionState sessionState) {
        if(sessionState == null) {
            throw new NullPointerException("Session state cannot be null");
        }
        return this.sessionStateRef.getAndSet(sessionState);
    }
}
```

```
-------------------------------------------------------------
Filename:
src/main/java/com/transmisms/smsftp/protocol/SmsftpTextBasedPduDecoder.java
-------------------------------------------------------------

package com.transmisms.smsftp.protocol;

import com.transmisms.core.protocol.CorePDUType;
import com.transmisms.core.protocol.PDU;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.core.protocol.TextBasedPDUDecoder;
import com.transmisms.core.protocol.PDUMalformedException;
import com.transmisms.smsftp.protocol.SmsftpFacade;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Pattern;



public class SmsftpTextBasedPduDecoder extends TextBasedPDUDecoder {
    private static final Pattern FILENAME_PATTERN =
            Pattern.compile("[a-zA-Z0-9. \\-#$%&'()+,i=@_]+");

    private static SmsftpTextBasedPduDecoder thisInstance;
    static {
        String[] strArray = { "smsftp" };
        SmsftpTextBasedPduDecoder.thisInstance =
            new SmsftpTextBasedPduDecoder(strArray);
    }

    protected SmsftpTextBasedPduDecoder(String[] knownProtocols) {
        super(knownProtocols);
    }

    public static SmsftpTextBasedPduDecoder getInstance() {
        return SmsftpTextBasedPduDecoder.thisInstance;
    }

    @Override
    protected Object[] decodePayloadSubparts(PDUType pduType,
            String[] payloadSubparts, String encoded)
            throws PDUMalformedException {
        List<Object> returnData = new ArrayList<Object>();
        CorePDUType corePduType = (CorePDUType)pduType;
        switch(corePduType) {
            case CORE_CONNECTION_REQUEST: {
                if(payloadSubparts.length < 2 || payloadSubparts.length > 3) {
                    throw new PDUMalformedException("Payload parts for " +
                            corePduType.toString() + " too long or too short",
                            encoded);
                }

                // get Receiver MSISDN/MIN
                String receiverMsisdnMin = payloadSubparts[0];
                // validate msisdn/min
                boolean validMsisdnMin = true;
                if((receiverMsisdnMin.charAt(0) != '+' &&
                        !Character.isDigit(receiverMsisdnMin.charAt(0))) ||
                        !receiverMsisdnMin.substring(1).chars()
                        .allMatch(x -> Character.isDigit(x))) {
                    throw new PDUMalformedException(
                            "Invalid MSISDN/MIN for " +
                            corePduType.toString() + ": " + receiverMsisdnMin,
                            encoded);
                }
                returnData.add(receiverMsisdnMin);
                // get Intent
    String intent = payloadSubparts[1];
    if(!intent.equals("send") && !intent.equals("pair")) {
        throw new PDUMalformedException("Invalid Intent for " +
                corePduType.toString() + ": " + intent, encoded);
    }
    returnData.add(intent);
    // get Filename if send
    if(intent.equals("send")) {
        if(payloadSubparts.length != 3) {
            throw new PDUMalformedException(
                    "Payload parts for " +
                    corePduType.toString() +
                    " too long or too short", encoded);

        }
        String filename = payloadSubparts[2];
        // validate filename
        if(!FILENAME_PATTERN.matcher(filename).matches()) {
            throw new PDUMalformedException(
                    "Invalid Filename for " +
                    corePduType.toString() + ": " + filename,
                    encoded);

        }
        returnData.add(filename);
    }
    else {
        if(payloadSubparts.length != 2) {
            throw new PDUMalformedException(
                    "Payload parts for " +
                    corePduType.toString() +
                    " too long or too short", encoded);

        }
    }
    break;
}
case CORE_HEAD: {
    if(payloadSubparts.length != 1) {
        throw new PDUMalformedException("Payload parts for " +
                corePduType.toString() + " too long or too short",
                encoded);
    }
    // get Encryption Mode
    String encMode = payloadSubparts[0];
    if(encMode.equals("enc") || encMode.equals("noenc")) {
        returnData.add(encMode);
    }
    else {
        throw new PDUMalformedException(
                "Invalid Encryption Mode: " + encMode,
                encoded);
    }
    break;
}
case CORE_CONNECTION_RESPONSE:
case CORE_LAST_ACK: {
    // get Status Code
    String statusCode = "";
    if(payloadSubparts.length == 1 || payloadSubparts.length == 2) {
        statusCode = payloadSubparts[0];
        returnData.add(statusCode);
    }
    else {
        throw new PDUMalformedException("Payload parts for " +
                corePduType.toString() + " too long or too short",
                encoded);
    }
    // validate status code
    if(SmsftpFacade.getStatusMessage(statusCode)
            == null) {
        throw new PDUMalformedException(
                "Invalid status code from " +
                corePduType.toString() + ": " + statusCode,
                encoded);
    }

    if(payloadSubparts.length == 1) { // without status message
        returnData.add(null);
    }
    else if(payloadSubparts.length == 2) { // with status message
        // add Status Message
        returnData.add(payloadSubparts[1]);
    }
    break;
}
default: {
    // code should NEVER reach here
    throw new AssertionError("Uncaught PDU Type: " + corePduType);
}
```

```
        }
        // finally, return the data
        return returnData.toArray();
    }
}
```

```
--------------------------------------------------------------
Filename:
src/main/java/com/transmisms/smsftp/temputil/SmsftpLoggerPresenter.java
--------------------------------------------------------------

package com.transmisms.smsftp.temputil;

import com.transmisms.core.protocol.Presenter.MessageType;
import com.transmisms.smsftp.protocol.SmsftpPresenter;

import java.util.Observable;

import org.apache.logging.log4j.Logger;


public class SmsftpLoggerPresenter extends SmsftpPresenter {
    private final Logger smsftpLogger;

    @Override
    public void receive() {} // disabled
    @Override
    public void insecureSendFile() {} // disabled
    @Override
    public void secureSendFile() {} // disabled
    @Override
    public void pair() {} // disabled

    public SmsftpLoggerPresenter(Logger smsftpLogger) {
        super(null); // we don't need the actual SmsftpFacade object here
        this.smsftpLogger = smsftpLogger;
    }

    @Override
    protected void onLogMessage(MessageType messageType, String logMessage) {
        // finally, log into the respective logger
        switch(messageType) {
            case FATAL: {
                this.smsftpLogger.fatal(logMessage);
                break;
            }
            case ERROR: {
                this.smsftpLogger.error(logMessage);
                break;
            }
            case WARN: {
                this.smsftpLogger.warn(logMessage);
                break;
            }
            case INFO: {
                this.smsftpLogger.info(logMessage);
                break;
            }
            case DEBUG: {
                this.smsftpLogger.debug(logMessage);
                break;
            }
            case TRACE: {
                this.smsftpLogger.trace(logMessage);
                break;
            }
            default: {
                // do nothing
                break;
            }
        }
    }
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/ConfigManager.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import org.yaml.snakeyaml.DumperOptions;
import org.yaml.snakeyaml.Yaml;
import org.apache.logging.log4j.Logger;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStream;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import java.io.IOException;
import java.io.FileNotFoundException;



public final class ConfigManager {
    public final static String ROOT_KEY = "appconfig";
    public final static String DOWNLOADLOC_KEY = "download-location";
    public final static String RETRY_ON_ERRORS_KEY = "retry-on-errors";
    public final static String COUNTRYCODE_KEY = "country-code";
    public final static String BACKENDTYPE_KEY = "backend-type";
    public final static String BACKENDSETTINGS_KEY = "backend-settings";

    public final static String BACKEND_DUMMY_VALUE = "dummy";
    public final static String BACKEND_GAMMUPSQL_VALUE = "gammu-psql";
    public final static String BACKEND_ANDROID_VALUE = "android";

    public final static String BACKEND_DUMMY_KEY = "backend-dummy";
    public final static String BACKEND_GAMMUPSQL_KEY = "backend-gammu-psql";
    public final static String BACKEND_ANDROID_KEY = "backend-android";

    public final static String BACKEND_DUMMY_HOST_KEY = "host";
    public final static String BACKEND_DUMMY_PORT_KEY = "port";
    public final static String BACKEND_DUMMY_LOCALMIN_KEY = "local-min";
    public final static String BACKEND_DUMMY_ROLE_KEY = "role";
    public final static String BACKEND_GAMMUPSQL_HOST_KEY = "database-host";
    public final static String BACKEND_GAMMUPSQL_PORT_KEY = "database-port";
    public final static String BACKEND_GAMMUPSQL_NAME_KEY = "database-name";
    public final static String BACKEND_GAMMUPSQL_USER_KEY = "database-user";
    public final static String BACKEND_GAMMUPSQL_PASS_KEY =
            "database-password";
    public final static String BACKEND_GAMMUPSQL_LOCALMIN_KEY =
            "local-min";
    public final static String BACKEND_ANDROID_HOST_KEY = "host";
    public final static String BACKEND_ANDROID_PORT_KEY = "port";

    private final static String CONFIG_FILE_NAME = "transmisms.yaml";

    private static Yaml yaml;
    private static Map<Object, Object> defaultConfig;
    public static Map<String, Object> defaultConfigRoot;

    static {
        DumperOptions options = new DumperOptions();
        options.setDefaultFlowStyle(DumperOptions.FlowStyle.BLOCK);
        ConfigManager.yaml = new Yaml(options);

        ConfigManager.defaultConfig =
                (Map<Object, Object>)(ConfigManager.yaml.load(
                ConfigManager.yaml.getClass().getClassLoader()
                .getResourceAsStream("transmisms-javafx.yaml")));

        ConfigManager.defaultConfigRoot =
                (Map<String, Object>)(ConfigManager.defaultConfig.get(
                ConfigManager.ROOT_KEY));
    }


    public static Map<String, Object> initAndReadConfig(Logger logger) {
        // verify config directory's and config's existence
        File dir = new File("conf");
        boolean dirExists = !dir.mkdir();
        boolean configExists = false;
        File f = new File(dir, ConfigManager.CONFIG_FILE_NAME);
        try {
            configExists = !f.createNewFile();
            logger.debug(configExists ? "Reading config file..." :
                    "Creating new config file...");
        }
        catch(IOException e) {
            logger.trace("got IOException!!!");
            e.printStackTrace();
        }

        Map<Object, Object> config =
                new HashMap<>(ConfigManager.defaultConfig);
        Map<String, Object> configRoot =
                (Map<String, Object>)(config.get(ConfigManager.ROOT_KEY));

        if(configExists) {
            // read config from file
            try {
                config = (Map<Object, Object>)(ConfigManager.yaml.load(
                        new FileInputStream(f)));
            }
            catch(FileNotFoundException e) {
            }
            // refresh configRoot since we used a new config
            Object cfRObj = config.get(ConfigManager.ROOT_KEY);
            if(cfRObj == null) {
                configExists = false;
            }
            else {
                configRoot = (Map<String, Object>)(cfRObj);
            }

        }
        else { // load from default yaml if not yet existing
            // we have loaded defaultConfig as config previously
            // so we won't do anything here
        }

        // fixes for some default values
        // fix blank download location to system-specific Downloads folder
        if(configRoot.get(ConfigManager.DOWNLOADLOC_KEY) == null) {
            configRoot.put(ConfigManager.DOWNLOADLOC_KEY,
                    System.getProperty("user.home") + File.separator +
                    "Downloads" + File.separator);
        }

        // write new config/fixes for config
        FileWriter fw = null;
        try {
            fw = new FileWriter(f);
            ConfigManager.yaml.dump(config, fw);
            fw.flush();
        }
        catch(IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
                fw.close();
            }
            catch(IOException e) {
                e.printStackTrace();
            }
        }

        return configRoot;
    }

    public static boolean writeConfig(Map<String, Object> configRoot) {
        Map<String, Object> config = new HashMap<>();
        config.put(ConfigManager.ROOT_KEY, new HashMap<>(configRoot));

        boolean isSuccessful = true;
        FileWriter fw = null;
        try {
            File f = new File(new File("conf"),
                    ConfigManager.CONFIG_FILE_NAME);
            fw = new FileWriter(f);
            ConfigManager.yaml.dump(config, fw);
            fw.flush();
        }
        catch(IOException e) {
            isSuccessful = false;
        }
        finally {
            if(fw != null) {
                try {
                    fw.close();
                }
                catch(IOException e) {
                    isSuccessful = false;
                }
            }
        }

        return isSuccessful;
    }

    public static<K, V> Map<K, V> mapDiff(Map<? extends K, ? extends V> l,
            Map<? extends K, ? extends V> r) {
        Map<K, V> diff = new HashMap<>();
        diff.putAll(l);
        diff.putAll(r);

        diff.entrySet().removeAll(l.size() <= r.size() ?
                l.entrySet() : r.entrySet());

        return diff;
    }
    public static Map mapDeepCopy(Map src) {
        return ConfigManager.mapDeepCopy(src, new HashMap<Object, Object>());
    }

    public static Map mapDeepCopy(Map src, Map dest) {
        Set<Map.Entry> entrySet = src.entrySet();
        for(Map.Entry entry : entrySet) {
            dest.put(entry.getKey(),
                    (entry.getValue() instanceof Map) ?
                    mapDeepCopy((Map<?, ?>)(entry.getValue())) :
                    entry.getValue());
        }
        return dest;
    }
}




---------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/ContactEditController.java
---------------------------------------------------------------

package com.transmisms.ui.javafx;

import javafx.beans.binding.Bindings;
import javafx.beans.binding.BooleanBinding;
import javafx.beans.property.SimpleMapProperty;
import javafx.collections.FXCollections;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.stage.WindowEvent;
import javafx.stage.Stage;

import java.util.HashMap;
import java.util.List;
import java.util.Map;



public class ContactEditController {
    @FXML
    private TextField contactNumber;
    @FXML
    private TextField contactName;
    @FXML
    private Label contactEditErrorText;
    @FXML
    private Label contactExistsErrorText;
    @FXML
    private Button saveButton;
    @FXML
    private Button pairButton;
    @FXML
    private Button unpairButton;
    @FXML
    private Button importButton;
    @FXML
    private Button exportButton;

    private Stage contactEditStage;

    private MainWindowController primaryController = null;

    private List<Map<String, Object>> contactOList = null;
    private SimpleMapProperty<String, Object> currentEntry = null;
    private Map<String, Object> entryMap = null;



    public void initialize(Stage contactEditStage,
            MainWindowController primaryController) {
        // set properties
        this.contactEditStage = contactEditStage;
        this.primaryController = primaryController;
        this.contactOList = primaryController.contactOList;
    }

    protected void prepareAndShowStage(Map<String, Object> currentEntry) {
        this.entryMap = currentEntry;
        this.currentEntry = new SimpleMapProperty<>(
                FXCollections.observableHashMap());
        if(currentEntry != null) {
            this.currentEntry.putAll(currentEntry);
        }

        // enable/disable controls as needed
        this.contactNumber.setDisable(currentEntry != null);
        this.pairButton.setDisable(currentEntry == null);
        this.importButton.setDisable(currentEntry == null);
        this.saveButton.setText((currentEntry != null) ?
                "Save" : "Add Contact");
        // bind related buttons
        this.saveButton.disableProperty().bind(
                this.contactNumber.textProperty().isEmpty());
        this.contactEditErrorText.visibleProperty().bind(
                this.contactNumber.textProperty().isEmpty());
        BooleanBinding entryIsPaired = new BooleanBinding() {
            { // this is an instance initializer
                super.bind(ContactEditController.this.currentEntry);
            }

            @Override
            protected boolean computeValue() {
                return ContactEditController.this.currentEntry.containsKey(
                        ContactManager.KEYS_KEY) &&
                        ContactEditController.this.currentEntry.get(
                        ContactManager.KEYS_KEY) != null;
            }
        };
        this.unpairButton.disableProperty().bind(entryIsPaired.not());
        this.exportButton.disableProperty().bind(entryIsPaired.not());

        // reset fields/load current entry
        if(currentEntry != null) {
            this.contactNumber.setText((String)(currentEntry.get(
                    ContactManager.MSISDN_KEY)));
            this.contactName.setText((String)(currentEntry.get(
                    ContactManager.NAME_KEY)));
        }
        else {
            this.contactName.setText("");
            this.contactNumber.setText("");
        }
        this.contactEditStage.showAndWait();
    }

    @FXML
    public void handleSaveButtonAction(ActionEvent event) {
        this.contactNumber.commitValue();
        this.contactName.commitValue();

        // try to normalize contactNumber first
        this.contactNumber.setText(Utils.normalizeMsisdn(
                this.contactNumber.getText(), "63"));

        if(this.entryMap == null) {
            // add to contact list
            this.contactOList.add(this.currentEntry);
            this.entryMap = this.currentEntry;
        }

        // set properties
        this.entryMap.put(
                ContactManager.MSISDN_KEY, this.contactNumber.getText());
        this.entryMap.put(
                ContactManager.NAME_KEY, this.contactName.getText());
        // and sort contact list again by name
        ContactManager.sortContactListByName(this.contactOList);

        // enable/disable controls as needed
        this.contactNumber.setDisable(true);
        this.pairButton.setDisable(false);
        this.importButton.setDisable(false);
        this.saveButton.setText("Save");

        // save on file ASAP
        ContactManager.writeConfig(ContactManager.convertToContactMap(
                    this.contactOList));
    }

    @FXML
    public void handlePairButtonAction(ActionEvent event) {
        // mark proper activeContact
        String peer = (String)this.currentEntry.get(ContactManager.MSISDN_KEY);
        this.primaryController.activeContact.set(peer);
        this.contactEditStage.close(); // cleanup
        // then pass this to the MainWindowController
        this.primaryController.handleExternalPairButtonAction(event);
    }

    @FXML
    public void handleUnpairButtonAction(ActionEvent event) {
        this.entryMap.remove(ContactManager.KEYS_KEY);
        // force sort contacts to apply changes
        ContactManager.sortContactListByName(this.contactOList);
        // save on file ASAP
        ContactManager.writeConfig(ContactManager.convertToContactMap(
                    this.contactOList));
        // enable/disable controls as needed
    }

    @FXML
    public void handleOnCloseRequest(WindowEvent event) {
        // unbind button properties first
        this.unpairButton.disableProperty().unbind();
        this.exportButton.disableProperty().unbind();
        this.saveButton.disableProperty().unbind();
        this.contactEditErrorText.visibleProperty().unbind();
        this.contactExistsErrorText.visibleProperty().unbind();
        // then reset our entry markers
        this.currentEntry = null;
        this.entryMap = null;
    }
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/ContactManager.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import org.yaml.snakeyaml.DumperOptions;
import org.yaml.snakeyaml.Yaml;
import org.apache.logging.log4j.Logger;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import java.io.IOException;
import java.io.FileNotFoundException;



public final class ContactManager {
    public final static String ROOT_KEY = "keystore";
    public final static String MSISDN_KEY = "number";
    public final static String NAME_KEY = "name";
    public final static String KEYS_KEY = "keys";
    public final static String HOST_PUB_ENC_KEY = "host-pub-enc";
    public final static String HOST_PRIV_ENC_KEY = "host-priv-enc";
    public final static String HOST_PUB_AUTH_KEY = "host-pub-auth";
    public final static String HOST_PRIV_AUTH_KEY = "host-priv-auth";
    public final static String PEER_PUB_ENC_KEY = "peer-pub-enc";
    public final static String PEER_PUB_AUTH_KEY = "peer-pub-auth";

    private final static String CONFIG_FILE_NAME = "contacts.yaml";

    private static Yaml yaml;
    private static Map<String, Object> defaultConfig;
    public static Map<String, Map<String, Object>> defaultConfigRoot;

    static {
        DumperOptions options = new DumperOptions();
        options.setDefaultFlowStyle(DumperOptions.FlowStyle.BLOCK);
        ContactManager.yaml = new Yaml(options);

        ContactManager.defaultConfig =
                (Map<String, Object>)(ContactManager.yaml.load(
                ContactManager.yaml.getClass().getClassLoader()
                .getResourceAsStream("transmisms-javafx-keystore.yaml")));

        ContactManager.defaultConfigRoot =
                (Map<String, Map<String, Object>>)
                (ContactManager.defaultConfig.get(
                ContactManager.ROOT_KEY));
    }

    public static Map<String, Map<String, Object>> initAndReadContacts(
            Logger logger) {
        // verify config directory's and config's existence
        File dir = new File("conf");
        boolean dirExists = !dir.mkdir();
        boolean configExists = false;
        File f = new File(dir, ContactManager.CONFIG_FILE_NAME);
        try {
            configExists = !f.createNewFile();
            logger.debug(configExists ? "Reading contacts file..." :
                    "Creating new contacts file...");
        }
        catch(IOException e) {
            logger.trace("got IOException!!!");
            e.printStackTrace();
        }

        Map<String, Object> config =
                new HashMap<>(ContactManager.defaultConfig);
        Map<String, Map<String, Object>> configRoot = null;

        if(ContactManager.defaultConfigRoot != null) {
            configRoot = new HashMap<>(ContactManager.defaultConfigRoot);
        }
        else {
            configRoot = new HashMap<>();
        }

        if(configExists) {
            // read config from file
            try {
                config = (Map<String, Object>)(ContactManager.yaml.load(
                        new FileInputStream(f)));
            }
            catch(FileNotFoundException e) {
            }

            // refresh configRoot since we used a new config
            Object cfRObj = config.get(ContactManager.ROOT_KEY);
            if(cfRObj == null) {
                configExists = false;
            }
            else {
                configRoot = (Map<String, Map<String, Object>>)(cfRObj);
            }

        }
        else { // load from default yaml if not yet existing
            // we have loaded defaultConfig as config previously
            // so we won't do anything here
        }

        // write new config/fixes for contacts
        FileWriter fw = null;
        try {
            fw = new FileWriter(f);
            ContactManager.yaml.dump(config, fw);
            fw.flush();
        }
        catch(IOException e) {
            e.printStackTrace();
        }
        finally {
            try {
                fw.close();
            }
            catch(IOException e) {
                e.printStackTrace();
            }
        }

        return configRoot;
    }

    public static boolean writeConfig(
            Map<String, Map<String, Object>> configRoot) {
        Map<String, Object> config = new HashMap<>();
        config.put(ContactManager.ROOT_KEY, new HashMap<>(configRoot));

        boolean isSuccessful = true;
        FileWriter fw = null;
        try {
            File f = new File(new File("conf"),
                    ContactManager.CONFIG_FILE_NAME);
            fw = new FileWriter(f);
            ContactManager.yaml.dump(config, fw);
            fw.flush();
        }
        catch(IOException e) {
            isSuccessful = false;
        }
        finally {
            if(fw != null) {
                try {
                    fw.close();
                }
                catch(IOException e) {
                    isSuccessful = false;
                }
            }
        }
        return isSuccessful;
    }

    //// utility functions here
    public static void sortContactListByName(
            List<Map<String, Object>> l) {
        Collections.sort(l, (m1, m2) -> {
            return ((String)(m1.get(
                ContactManager.NAME_KEY))).compareToIgnoreCase(
                ((String)(m2.get(ContactManager.NAME_KEY))));
        });
    }

    public static List<Map<String, Object>> convertToContactList(
            Map<String, Map<String, Object>> m) {
        List<Map<String, Object>> l = new ArrayList<>();
        m.forEach((k, v) -> {
            Map<String, Object> n = new HashMap<>();
            n.put(ContactManager.MSISDN_KEY, k);
            n.put(ContactManager.NAME_KEY, (String)(v.get(NAME_KEY)));
            n.put(ContactManager.KEYS_KEY, v.get(KEYS_KEY));
            l.add(n);
        });
        // sort created list first
        ContactManager.sortContactListByName(l);
        return l;
    }

    public static Map<String, Map<String, Object>> convertToContactMap(
            List<Map<String, Object>> l) {
        Map<String, Map<String, Object>> m = new HashMap<>();
        // loop for each element and reconstruct the map
        for(Map<String, Object> item : l) {
            Object number = item.get(ContactManager.MSISDN_KEY);
            Object name = item.get(ContactManager.NAME_KEY);

            // check for validity and skip if invalid
            if(!(name instanceof String && number instanceof String)) {
                continue;
            }

            Map<String, Object> n = new HashMap<>();
            n.put(ContactManager.NAME_KEY, (String)name);
            n.put(ContactManager.KEYS_KEY, item.get(ContactManager.KEYS_KEY));

            m.put((String)number, n);
        }
        return m;
    }
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/FingerprintDialog.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import javafx.scene.Parent;
import javafx.scene.control.ButtonType;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.text.Text;
import javafx.stage.StageStyle;

import javafx.fxml.FXMLLoader;

import java.io.IOException;



public class FingerprintDialog extends Dialog<ButtonType> {
    private final Label primaryLabel;
    private final Text secondaryLabel;
    private final Text emphasisLabel;



    public FingerprintDialog(String primaryText, String secondaryText,
            Image img) {
        // prepare and set the header
        FXMLLoader dialogTemplateLoader = new FXMLLoader(
                MainWindowController.class.getClassLoader()
                .getResource("dialog-fingerprint-template.fxml"));
        Parent header = null;
        try {
            header = (Parent)(dialogTemplateLoader.load());
        }
        catch(IOException e) {
            System.out.println(
                "Caught IOException when loading dialog header templates");
            e.printStackTrace();
        }
        // set css for the parent
        header.getStylesheets().add("dialog.css");
        // set icon for dialog
        ImageView iv = (ImageView)header.lookup(".dialogIcon");
        iv.setImage(img);
        // set necessary text properties
        this.primaryLabel = (Label)header.lookup(".dialogPrimaryLabel");
        this.secondaryLabel = (Text)header.lookup(".dialogSecondaryLabel");
        this.emphasisLabel = (Text)header.lookup(".dialogEmphasisLabel");
        primaryLabel.setText(primaryText);
        secondaryLabel.setText(secondaryText);
        // add the header to dialog pane
        this.getDialogPane().setHeader(header);

        // misc modifications for the dialog
        this.initStyle(StageStyle.UTILITY);
    }

    public void setEmphasisText(String text) {
        this.emphasisLabel.setText(text);
    }
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/Main.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;


import javafx.application.Application;
import javafx.application.Platform;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import javafx.stage.WindowEvent;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import java.lang.reflect.Method;
import java.util.Map;

import java.io.IOException;



public class Main extends Application {
    // initialize logging
    private final Logger mainLogger = LogManager.getLogger("com.transmisms");
    private final Logger protocolLogger =
            LogManager.getLogger("com.transmisms.smsftp.protocol");



    public class JavaFXUpdateMethodContainer {
        Method updateMessage;
        Method updateTitle;
        Method updateLongProgress;
        Method updateDoubleProgress;

        public JavaFXUpdateMethodContainer(Method updateMessage,
                Method updateTitle, Method updateLongProgress,
                Method updateDoubleProgress) {
            this.updateMessage = updateMessage;
            this.updateTitle = updateTitle;
            this.updateLongProgress = updateLongProgress;
            this.updateDoubleProgress = updateDoubleProgress;
        }
    }

    public interface TransmismsLogger {
        public void debug(CharSequence message);
        public void error(CharSequence message);
        public void fatal(CharSequence message);
        public void info(CharSequence message);
        public void trace(CharSequence message);
        public void warn(CharSequence message);
    }
public static void main(String[] args) {
    Application.launch(args);
}



// looking for public static void main()? here it is:
@Override
public void start(Stage primaryStage) {

    this.mainLogger.info("Starting transmisms-javafx v0.7...");

    // load and initialize yaml files on startup
    Map<String, Object> configRoot =
            ConfigManager.initAndReadConfig(this.mainLogger);
    Map<String, Map<String, Object>> contactsRoot =
            ContactManager.initAndReadContacts(this.mainLogger);

    this.mainLogger.trace("Entered UI Session loop");
    // set stylesheet
    setUserAgentStylesheet(STYLESHEET_MODENA);

    // initialize other stages
    Stage prefsStage = new Stage(StageStyle.UTILITY);
    Stage contactEditStage = new Stage(StageStyle.UTILITY);

    // get primaryLoader for JavaFx
    FXMLLoader primaryLoader = new FXMLLoader(getClass().getClassLoader().
            getResource("fxml-main.fxml"));
    FXMLLoader prefsLoader = new FXMLLoader(getClass().getClassLoader().
            getResource("prefs-window.fxml"));
    FXMLLoader contactEditLoader =
            new FXMLLoader(getClass().getClassLoader().
            getResource("contact-edit-window.fxml"));
    Parent primaryParent = null;
    Parent prefsParent = null;
    Parent contactEditParent = null;
    try {
        primaryParent = primaryLoader.load();
        prefsParent = prefsLoader.load();
        contactEditParent = contactEditLoader.load();
    }
    catch(IOException e) {
        this.mainLogger.error(e.getMessage());
        e.printStackTrace();
        Platform.exit(); // exit ASAP
    }

    // set css for the parents
    primaryParent.getStylesheets().add("main.css");
    prefsParent.getStylesheets().add("main.css");
    prefsParent.getStylesheets().add("prefs-window.css");
    contactEditParent.getStylesheets().add("main.css");
    contactEditParent.getStylesheets().add("contact-edit-window.css");

    // get controllers and initialize them
    PrefsController prefsController =
            (PrefsController)prefsLoader.getController();
    ContactEditController contactEditController =
            (ContactEditController)contactEditLoader.getController();
    MainWindowController primaryController =
            (MainWindowController)primaryLoader.getController();
    primaryController.initialize(configRoot, contactsRoot, prefsStage,
            contactEditController, this.mainLogger, this.protocolLogger);
    prefsController.initialize(configRoot, primaryController);
    contactEditController.initialize(contactEditStage,
            primaryController);

    // set close handler
    primaryStage.setOnCloseRequest((e) -> {
        mainLogger.trace("Main Window closed");
        primaryController.shutdown();
    });
    prefsStage.setOnCloseRequest((e) -> {
        prefsController.handleOnCloseRequest(e);
    });
    contactEditStage.setOnCloseRequest((e) -> {
        contactEditController.handleOnCloseRequest(e);
    });

    // set static sizes
    primaryStage.setMinWidth(300);
    primaryStage.setMinHeight(200);
    prefsStage.setMinWidth(300);
    prefsStage.setMinHeight(200);

    // set stage UI
    prefsStage.setTitle("Preferences");
        prefsStage.setScene(new Scene(prefsParent, 640, 520));
        prefsStage.initOwner(primaryStage);
        prefsStage.initModality(Modality.APPLICATION_MODAL);
        contactEditStage.setTitle("Contact Details");
        contactEditStage.setScene(new Scene(contactEditParent, 480, 300));
        contactEditStage.initOwner(primaryStage);
        contactEditStage.initModality(Modality.APPLICATION_MODAL);
        primaryStage.setTitle("TransmiSMS");
        primaryStage.setScene(new Scene(primaryParent, 640, 480));
        primaryStage.show(); // finally, show the stage

        this.mainLogger.trace(
                "Reached end of pseudo main(), JavaFX probably has taken over");
    }



    @Override
    public void stop() {
        this.mainLogger.trace("Exiting application...");
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/MainWindowController.java
--------------------------------------------------------------
package com.transmisms.ui.javafx;

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.core.protocol.Connection.ConnectionState;
import com.transmisms.core.util.crypto.BiometricFingerprint;
import com.transmisms.smsftp.protocol.SmsftpFacade;
import com.transmisms.smsftp.protocol.SmsftpPDUType;
import com.transmisms.smsftp.protocol.SmsftpPresenter;
import com.transmisms.smsftp.protocol.SmsftpSession;
import com.transmisms.smsftp.protocol.SmsftpSession.Role;
import com.transmisms.smsftp.protocol.SmsftpSession.SessionIntent;
import com.transmisms.smsftp.protocol.SmsftpSession.SessionState;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;
import com.transmisms.ui.javafx.smsservice.AndroidSmsService;
import com.transmisms.ui.javafx.smsservice.ExtendedSmsService;
import com.transmisms.ui.javafx.smsservice.ExtendedSmsService.ExtendedReason;
import com.transmisms.ui.javafx.smsservice.DummySmsService;
import com.transmisms.ui.javafx.smsservice.GammuSmsService;

import org.apache.logging.log4j.Logger;

import javafx.application.Platform;
import javafx.concurrent.Task;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.geometry.Side;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.control.Button;
import javafx.scene.control.ButtonBar.ButtonData;
import javafx.scene.control.ButtonType;
import javafx.scene.control.ContextMenu;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
import javafx.scene.control.MenuItem;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.TextArea;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javafx.stage.StageStyle;

import javafx.beans.binding.Bindings;
import javafx.beans.binding.BooleanBinding;
import javafx.beans.property.SimpleBooleanProperty;
import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;

import java.io.File;
import java.io.FilenameFilter;
import java.nio.file.Files;
```

```java
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;
import java.util.function.Consumer;

import com.transmisms.core.protocol.SmsServiceException;
import java.io.IOException;

import java.util.concurrent.ExecutionException;



public class MainWindowController {
    protected static class ContactListCell
            extends ListCell<Map<String, Object>> {
        private Node rootGraphic;
        private final MainWindowController mwc;

        public ContactListCell(MainWindowController mwc) {
            this.mwc = mwc;
        }

        @Override
        public void updateItem(Map<String, Object> item, boolean empty) {
            super.updateItem(item, empty);
            if(!empty && item != null) {
                try {
                    FXMLLoader lcLoader = new FXMLLoader(getClass()
                            .getClassLoader()
                            .getResource("contact-listcell-template.fxml"));
                    lcLoader.setController(this.mwc);
                    this.rootGraphic = (VBox)(lcLoader.load());
                }
                catch(IOException e) {
                    System.out.println(
                        "Caught IOException when loading list cell templates");
                    e.printStackTrace();
                }

                //// configure members' properties
                String msisdn = (String)(item.get(ContactManager.MSISDN_KEY));
                Label cname = (Label)(this.rootGraphic.lookup(".contactName"));
                cname.setText((String)(item.get(ContactManager.NAME_KEY)));
                Label cnum = (Label)(this.rootGraphic.lookup(".contactNum"));
                cnum.setText(msisdn);
                Label s = (Label)(this.rootGraphic.lookup(
                        ".contactStatus"));
                ProgressBar p = (ProgressBar)(this.rootGraphic.lookup(
                        ".operationProgress"));
                Button b = (Button)(this.rootGraphic.lookup(
                        ".cancelOperationButton"));
                // set id bindings
                s.idProperty().bind(Bindings.concat(msisdn, "StatusLabel"));
                p.idProperty().bind(Bindings.concat(msisdn, "ProgressBar"));
                b.idProperty().bind(Bindings.concat(msisdn, "CancelButton"));
                // set other bindings
                s.textProperty().bind(this.mwc.connectionStatus);
                p.progressProperty().bind(this.mwc.connectionProgress);

                // finally, set graphic
                this.setGraphic(this.rootGraphic);
            }
            else {
                // clear contents on empty
                this.setText(null);
                this.setGraphic(null);
            }
        }
    }

    private static class PresenterBase extends SmsftpPresenter {
        final private MainWindowController mwc;
        final protected SmsftpFacade smsftpFacade;
        private String statusCode = null;
        private String statusMessage = null;

        public PresenterBase(SmsftpFacade smsftpFacade,
                MainWindowController mwc) {
            super(smsftpFacade);
            this.mwc = mwc;
            this.smsftpFacade = smsftpFacade;
        }

        @Override
        public void onStatusUpdate(StatusType sType, SessionState sState,
                Object ... args) {
```

```java
if(sType == StatusType.STATE_CHANGED) {
    if(args == null || args.length < 1 ||
            !(args[0] instanceof SessionState)) {
        return; // exit ASAP
    }
    SessionState newState = (SessionState)(args[0]);
    if(sState.equals(newState)) {
        return; // exit on nothing changed condition
    }
    switch(newState) {
        case CONNECTED: {
            Platform.runLater(() -> {
                this.mwc.connectionProgress.set(-1);
                this.mwc.connectionStatus.set(
                        "Connected to peer");
            });
            break;
        }
        case INIT_RECEIVED:
        case INIT_SENT: {
            Platform.runLater(() ->
                    this.mwc.connectionStatus.set(
                    this.smsftpFacade
                    .smsftpSession.getEncryptionFlag() ?
                    "Exchanging connection and security details" :
                    "Exchanging connection details"));
            break;
        }
        case SENDING_DATA: {
            Platform.runLater(() ->
                    this.mwc.connectionStatus.set(
                    "Sending data"));
            break;
        }
        case RECEIVING_DATA: {
            Platform.runLater(() ->
                    this.mwc.connectionStatus.set(
                    "Receiving data"));
            break;
        }
        case INITIATOR_WAIT: {
            Platform.runLater(() ->
                    this.mwc.connectionStatus.set(
                    "Waiting for peer to respond"));
            break;
        }
        case INITIATOR_KEY_EXCHANGE:
        case RESPONDER_KEY_EXCHANGE: {
            Platform.runLater(() ->
                    this.mwc.connectionStatus.set(
                    "Exchanging keys with peer"));
            break;
        }
        case COMPLETED: {
            Platform.runLater(() ->
                    this.mwc.connectionStatus.set(
                    "Verified. Finalizing transfer"));
            break;
        }
        case COMPLETE_WAIT: {
            Platform.runLater(() -> {
                this.mwc.connectionProgress.set(-1);
                this.mwc.connectionStatus.set(
                        "Sending data; waiting for verification");
            });
            break;
        }
        case TERMINATING: {
            Platform.runLater(() -> {
                this.mwc.connectionProgress.set(-1);
                this.mwc.connectionStatus.set(
                        "Disconnecting");
            });
            break;
        }
        default: {
            return; // exit ASAP
        }
    }
}
else if(sType == StatusType.NOTICE) {
    if(this.mwc.currentFacade.smsftpSession
            .getSessionIntent().equals(
            SessionIntent.PAIR)) {
        if(sState.equals(SessionState.COMPLETED) ||
                sState.equals(SessionState.COMPLETE_WAIT)) {
            final SmsftpSession session =
                    this.mwc.currentFacade.smsftpSession;
            // extracted keys from session
            final byte[] hostPubEnc =
                session.localEncKeypair.getPublic().getEncoded();
```

```
final byte[] hostPrivEnc =
    session.localEncKeypair.getPrivate().getEncoded();
final byte[] hostPubAuth =
    session.localAuthKeypair.getPublic().getEncoded();
final byte[] hostPrivAuth =
    session.localAuthKeypair.getPrivate().getEncoded();
final byte[] peerPubEnc = session.peerEnckey;
final byte[] peerPubAuth = session.peerAuthkey;


// compute fingerprint
final byte[] localPubKeys =
        new byte[hostPubEnc.length+hostPubAuth.length];
System.arraycopy(hostPubEnc, 0, localPubKeys,
        0, hostPubEnc.length);
System.arraycopy(hostPubAuth, 0, localPubKeys,
        hostPubEnc.length, hostPubAuth.length);
final byte[] peerPubKeys =
        new byte[peerPubEnc.length+peerPubAuth.length];
System.arraycopy(peerPubEnc, 0, peerPubKeys,
        0, peerPubEnc.length);
System.arraycopy(peerPubAuth, 0, peerPubKeys,
        peerPubEnc.length, peerPubAuth.length);
final byte[] part1 =
        session.getRole().equals(Role.INITIATOR) ?
        localPubKeys : peerPubKeys;
final byte[] part2 =
        session.getRole().equals(Role.INITIATOR) ?
        peerPubKeys : localPubKeys;
final byte[] combined =
        new byte[part1.length+part2.length];
System.arraycopy(part1, 0, combined,
        0, part1.length);
System.arraycopy(part2, 0, combined,
        part1.length, part2.length);
final String bfp =
        BiometricFingerprint.getBiometricFingerprint(
        combined);

// prompt user for fingerprint verification/if keys
// should be saved
Platform.runLater(() -> {
    String aContact = this.mwc.activeContact.get();
    SimpleBooleanProperty saveKeys =
            new SimpleBooleanProperty(false);
    this.mwc.fpConfirmation.setEmphasisText(bfp);
    this.mwc.fpConfirmation.showAndWait()
            .filter(response -> response ==
            MainWindowController.dialogSavekeysButton)
            .ifPresent(response -> {
                    saveKeys.set(true);
            });
    // return asap on cancel
    if(!saveKeys.get()) {
        return;
    }

    // get the corresponding keymap for activeContact
    Map<String, byte[]> keyMap = null;
    for(Map<String, Object> m : this.mwc.contactOList) {
        Object currentMinO =
                m.get(ContactManager.MSISDN_KEY);
        if(currentMinO != null &&
                currentMinO instanceof String &&
                aContact.equals(
                currentMinO)) {
            keyMap = (Map<String, byte[]>)
                    m.get(ContactManager.KEYS_KEY);
            // create keyMap if not yet existing
            if(keyMap == null) {
                keyMap = new HashMap<>();
                m.put(ContactManager.KEYS_KEY, keyMap);
            }
            break; // exit the loop ASAP
        }
    }
    // fill up map with the necessary keys
    keyMap.put(ContactManager.HOST_PUB_ENC_KEY,
            hostPubEnc);
    keyMap.put(ContactManager.HOST_PRIV_ENC_KEY,
            hostPrivEnc);
    keyMap.put(ContactManager.HOST_PUB_AUTH_KEY,
            hostPubAuth);
    keyMap.put(ContactManager.HOST_PRIV_AUTH_KEY,
            hostPrivAuth);
    keyMap.put(ContactManager.PEER_PUB_ENC_KEY,
            peerPubEnc);
    keyMap.put(ContactManager.PEER_PUB_AUTH_KEY,
            peerPubAuth);
    // save on file ASAP
    ContactManager.writeConfig(
```

```
                            ContactManager.convertToContactMap(
                            this.mwc.contactOList));
                    this.mwc.refreshContactList = true;
                });
            }
        }
    }
}

@Override
public void onStatusUpdate(StatusType sType, ConnectionState oldState,
        Object ... args) {
    if(args == null || args.length < 1) {
        return; // exit ASAP
    }
    if(sType == StatusType.STATE_CHANGED &&
            args[0] instanceof ConnectionState) {
        ConnectionState newState = (ConnectionState)(args[0]);
        if(oldState == null || newState == null ||
                oldState.equals(newState)) {
            return; // exit ASAP
        }
        switch(newState) {
            case CLOSED: {
                Platform.runLater(() -> {
                    this.mwc.connectionStatus.set("Disconnected");
                });
                // then clean up afterwards
                this.mwc.cleanupTransmission();
                // force refresh by "sorting contact list"
                Platform.runLater(() -> {
                    if(this.mwc.refreshContactList) {
                        ContactManager.sortContactListByName(
                                this.mwc.contactOList);
                        this.mwc.refreshContactList = false;
                    }
                });
                break;
            }
            case ESTABLISHED: {
                Platform.runLater(() ->
                        this.mwc.connectionStatus.set(
                        "Connection established"));
                break;
            }
            case LISTENING: {
                Platform.runLater(() -> {
                    this.mwc.connectionProgress.set(-1);
                    this.mwc.connectionStatus.set(
                            "Waiting for connection requests");
                });
                break;
            }
            case REQUEST_RECEIVED: {
                Platform.runLater(() ->
                        this.mwc.connectionStatus.set(
                        "Received connection request"));
                break;
            }
            case REQUEST_SENT:
            case REQUEST_SENT_2:
            case REQUEST_SENT_3: {
                Platform.runLater(() -> {
                    this.mwc.connectionProgress.set(-1);
                    this.mwc.connectionStatus.set(
                            "Waiting for connection response");
                });
                break;
            }
            case CLOSE_WAIT: {
                Platform.runLater(() -> {
                    this.mwc.connectionProgress.set(-1);
                    this.mwc.connectionStatus.set(
                            "Disconnecting");
                    // handle non-success status codes for smsftp
                    if(this.statusCode != null &&
                            !this.statusCode.startsWith("0")) {
                        // set proper text via status code and msg
                        this.mwc.peerDisconnectNotification
                                .setSecondaryText(
                                "Peer disconnected the connection " +
                                "with the reason: " +
                                this.statusCode + " " +
                                this.statusMessage);
                        this.mwc.peerDisconnectNotification.show();
                    }
                });
                break;
            }
            case FIN_WAIT:
            case CLOSING_1:
```

```
            case CLOSING_2: {
                Platform.runLater(() -> {
                    this.mwc.connectionProgress.set(-1);
                        this.mwc.connectionStatus.set(
                        "Requesting disconnection");
                });
                break;
            }
            case TIME_WAIT: {
                Platform.runLater(() -> {
                    this.mwc.connectionProgress.set(-1);
                    this.mwc.connectionStatus.set(
                            "Finalizing disconnection");
                });
                break;
            }
            default: {
                return; // exit ASAP
            }
        }
    }
    else if(sType == StatusType.NOTICE) {
        if(args.length >= 2 && args[0] instanceof String &&
                args[1] instanceof String) {
            switch(oldState) {
                case CLOSE_WAIT: {
                    // populate status code and message
                    if(this.statusCode == null) {
                        this.statusCode = (String)args[0];
                        this.statusMessage = (String)args[1];
                    }
                    break;
                }
                default: {
                    return; // exit ASAP
                }
            }
        }
    }
}

@Override
protected void onUserPrompt(PromptType pType, String message) {
    switch(pType) {
        case CONNECTION_REQUEST: {
            break;
        }
        case SMS_SERVICE_ERROR: {
            Platform.runLater(() -> {
                this.mwc.errorNotification
                        .setPrimaryText(message);
                this.mwc.errorNotification
                        .setSecondaryText(message);
                // set as not ready (error state)
                this.mwc.smsServiceIsReady.set(false);
            });
            break;
        }
        case TIMEOUT:
        case PEER_ERROR_EXCEEDED_LIMIT: {
            Platform.runLater(() -> {
                if(pType.equals(PromptType.TIMEOUT)) {
                    this.mwc.connectionErrorNotification
                            .setPrimaryText("Connection Timed Out");
                    this.mwc.connectionErrorNotification
                            .setSecondaryText(
                            "Peer has not responded for a while. " +
                            "Do you want to retry connecting?");
                }
                else if(pType.equals(
                        PromptType.PEER_ERROR_EXCEEDED_LIMIT)) {
                    this.mwc.connectionErrorNotification
                            .setPrimaryText("Error Limit Exceeded");
                    this.mwc.connectionErrorNotification
                            .setSecondaryText(
                            "The application has noticed erratic " +
                            "behavior from peer. This might be an " +
                            "indication of a problematic network " +
                            "or a tampered connection. " +
                            "Do you still want to retry connecting?");
                }
                else {
                    assert false; // this should never happen
                    return; // exit ASAP
                }

                // don't prompt if just listening; just retry
                if(this.mwc.currentFacade.smsftpSession
                        .getSessionState().equals(
                        SessionState.CONNECTED) &&
                        this.mwc.currentFacade.smsftpSession
                        .getRole().equals(Role.RESPONDER)) {
                    this.mwc.currentFacade
                            .continueOperationAfterPrompt();
                    return;
                }
                // then prompt user afterwards
                this.mwc.connectionErrorNotification.showAndWait()
                        .ifPresent(response -> {
                    if(response ==
                            MainWindowController.dialogRetryButton) {
                        this.mwc.currentFacade
                                .continueOperationAfterPrompt();
                    }
                    else if(response ==
                            MainWindowController.dialogCancelButton) {
                        this.mwc.currentFacade
                                .cancelOperationAfterPrompt();
                    }
                    else { // NOTE: code should never reach here
                        assert false;
                    }
                });
            });
            break;
        }
        default: {
            return; // just do nothing and return ASAP
        }
    }
}

// we won't override onLogMessage since we already delegated the task
// to the LoggerPresenters
}

private static class ReceiverPresenter extends PresenterBase {
    final private MainWindowController mwc;

    public ReceiverPresenter(SmsftpFacade smsftpFacade,
            MainWindowController mwc) {
        super(smsftpFacade, mwc);
        this.mwc = mwc;
    }

    @Override
    protected void onUserPrompt(PromptType pType, String message) {
        switch(pType) {
            case CONNECTION_REQUEST: {
                // DATA_TRANSFER-specific variables
                boolean isEncrypted = false;
                byte[] encKey = null;
                byte[] authKey = null;
                // PAIR-specific variables

                // try to get sender info from contacts map
                // temporary container for keys, if needed
                Map<String, byte[]> itemKeys = null;
                final Map<String, Map<String, Object>> tempMap =
                        ContactManager.convertToContactMap(
                        this.mwc.contactOList);
                // extract contact, if existing from the contacts map
                final String peer = Utils.normalizeMsisdn(
                        this.mwc.smsService.getPeer(),
                        this.mwc.configRoot.get(
                        ConfigManager.COUNTRYCODE_KEY).toString());
                final Map<String, Object> tempItem = tempMap.get(peer);
                final boolean contactExists = (tempItem != null);
                if(contactExists) {
                    // check if keys are available
                    itemKeys = (Map<String, byte[]>)(tempItem.get(
                            ContactManager.KEYS_KEY));
                }
                final boolean paired = (itemKeys != null &&
                            itemKeys.containsKey(
                            ContactManager.PEER_PUB_ENC_KEY) &&
                            itemKeys.containsKey(
                            ContactManager.PEER_PUB_AUTH_KEY));

                // check intent and do appropriate actions needed
                if(this.smsftpFacade.smsftpSession
                        .getSessionIntent().equals(
                        SessionIntent.DATA_TRANSFER)) {
                    isEncrypted = this.smsftpFacade
                            .smsftpSession.getEncryptionFlag();
                    // do some sanity checks and autoreject on failure
                    if(isEncrypted && !paired) {
                    }

                    // then set the proper keys, if any
                    encKey = (isEncrypted && paired) ?
                            itemKeys.get(
                                ContactManager.PEER_PUB_ENC_KEY) :
                                null;
                        authKey = (isEncrypted && paired) ?
                                itemKeys.get(
                                ContactManager.PEER_PUB_AUTH_KEY) :
                                null;
                    }
                    else if(this.smsftpFacade.smsftpSession
                            .getSessionIntent().equals(
                            SessionIntent.PAIR)) {
                        // do nothing; not applicable
                    }
                    else {
                        assert false; // should never happen
                        throw new IllegalStateException(
                                "Invalid sessionIntent on request: " +
                                this.smsftpFacade.smsftpSession
                                .getSessionIntent());
                    }

                    // final variables to comply with lambda restrictions
                    final boolean fIsEncrypted = isEncrypted;
                    final byte[] fEncKey = encKey;
                    final byte[] fAuthKey = authKey;

                    Platform.runLater(() -> { // run on EDT
                        // customize acceptConfirmation before displaying
                        // customize with contact
                        this.mwc.acceptConfirmation.setSecondaryText(
                                (contactExists ?
                                (String)(tempItem.get(
                                ContactManager.NAME_KEY)) :
                                peer));
                        // customize according to intent
                        if(this.smsftpFacade.smsftpSession
                                .getSessionIntent().equals(
                                SessionIntent.DATA_TRANSFER)) {
                            this.mwc.acceptConfirmation.setPrimaryText(
                                    "Accept file transfer request?");
                            this.mwc.acceptConfirmation.appendSecondaryText(
                                    " is trying to send you a file named \"" +

this.smsftpFacade.smsftpSession.getFilename() +
                                    "\". " +
                                    (this.smsftpFacade
                                    .smsftpSession.getEncryptionFlag() ?
                                    "This will be transmitted over a " +
                                    "secure connection. " :
                                    "This will be transmitted over an" +
                                    " UNSECURE connection. ") +
                                    "Do you want to accept the request?");
                        }
                        else { // SessionIntent.PAIR
                            this.mwc.acceptConfirmation.setPrimaryText(
                                    "Accept pairing request?");
                            this.mwc.acceptConfirmation.appendSecondaryText(
                                    (paired ?
                                    " is trying to renew their pairing " +
                                    "with you. Please make sure to verify " +
                                    "their identity and their intent to " +
                                    "redo the pairing procedure. " :
                                    " is trying to pair with you. " +
                                    "This method is not as secure as " +
                                    "manually exchanging keys off-air, " +
                                    "however. ") +
                                    "Do you want to accept the request?");
                        }

                        // finally show acceptConfirmation dialog
                        this.mwc.acceptConfirmation.showAndWait()
                                .ifPresent(response -> {
                            // if user accepted the request
                            if(response ==
                                    MainWindowController.dialogAcceptButton) {
                                //  auto add contact if user accepts request
                                if(!contactExists && !fIsEncrypted) {
                                    // add 'unknown' contact first
                                    Map<String, Object> n =
                                        new HashMap<>();
                                    n.put(ContactManager.MSISDN_KEY, peer);
                                    // add a 'unique' "Unknown" entry
                                    String[] names = new String[
                                            this.mwc.contactOList.size()];
                                    for(int i = 0; i < names.length; i++) {
                                        names[i] = (String)
                                                (this.mwc.contactOList
                                                .get(i).get(
                                                ContactManager.NAME_KEY));
                                    }
                                    n.put(ContactManager.NAME_KEY,
                                            MainWindowController.returnUnique(
                                    "Unknown", names));
                            n.put(ContactManager.KEYS_KEY, null);
                            // add to map
                            this.mwc.contactOList.add(n);
                            // then save changes to file
                            ContactManager.writeConfig(ContactManager
                                    .convertToContactMap(
                                    this.mwc.contactOList));
                        }
                        this.mwc.isReceiving.set(true);
                        this.mwc.activeContact.set(peer);

                        // show connection controls later after
                        //      initialization of other components
                        Platform.runLater(() ->
                                this.mwc.showActiveConnectionControls()
                        );

                        // finally, accept the request
                        if(this.smsftpFacade.smsftpSession
                                .getSessionIntent().equals(
                                SessionIntent.DATA_TRANSFER)) {
                            this.smsftpFacade.acceptRequestAfterPrompt(
                                    fEncKey, fAuthKey);
                        }
                        else {
                            this.smsftpFacade.acceptRequestAfterPrompt(
                                    null, null);
                        }
                    }
                    else if(response ==
                            MainWindowController.dialogRejectButton) {
                        // stop facade asap
                        this.mwc.currentFacade.disconnect("1003");
                    }
                    else { // NOTE: code should never reach here
                        assert false;
                        return;
                    }
                });
            });
            break;
        }
        case TIMEOUT:
        case SMS_SERVICE_ERROR:
        case PEER_ERROR_EXCEEDED_LIMIT:
        default: {
            super.onUserPrompt(pType, message);
        }
    }
}

@Override
public void onStatusUpdate(StatusType sType, SessionState sState,
        Object ... args) {
    // check if event is still valid
    // also prevents 'dead' connections from sending status updates
    if(this.mwc.currentFacade == null ||
            !this.mwc.currentFacade.smsftpSession.connection
            .getConnectionState().equals(
            ConnectionState.ESTABLISHED)) {
        return;
    }

    // apply supermethod first
    super.onStatusUpdate(sType, sState, args);
    // apply our own changes, overriding previous changes, if any
    if(sType == StatusType.NOTICE) {
        // for receiving DATA
        if(sState.equals(SessionState.COMPLETED) &&
                this.mwc.currentFacade.smsftpSession.getSessionIntent()
                .equals(SessionIntent.DATA_TRANSFER)) {
            if(args != null && args.length >= 1) {
                if(args[0] == null) { // failure
                    Platform.runLater(() ->
                        this.mwc.ftpFailureNotification.show());
                }
                else if(args[0] instanceof byte[]) { // success
                    try {
                        String downloadPath =
                                (String)this.mwc.configRoot.get(
                                ConfigManager.DOWNLOADLOC_KEY);
                        Files.write(
                                Paths.get(downloadPath +
                                File.separator +
                                // returnUnique for automatic renaming
                                // of duplicates
                                MainWindowController.returnUnique(
                                this.mwc.currentFacade
                                .smsftpSession.getFilename(),
                                new File(downloadPath))),
                            (byte[])args[0]);
                    }
                    catch(IOException e) {
                    }
                    // finally, alert user
                    Platform.runLater(() ->
                        this.mwc.ftpRecvSuccessNotification.show());
                }
            }
        }
        else if(sState.equals(SessionState.RECEIVING_DATA)) {
            // no need to check args, we'll compute progress here

            // get skipped segnums, max, and current
            double segCount = this.mwc.currentFacade.smsftpSession
                    .getDataSegCount();
            double lastSeg = this.mwc.currentFacade.smsftpSession
                    .getLastDataSegNumReceived();
            double missingNos = this.mwc.currentFacade.smsftpSession
                    .getSkippedSegNumsLength();
            // lastSeg+2 to accomodate 0-based array and possibly
            // non-executed last call
            double progress = (lastSeg+2 - missingNos)/segCount;

            // compare old value to new value to prevent
            // unnecessary updates
            double curProgress = this.mwc.connectionProgress.get();
            if(progress != curProgress) {
                Platform.runLater(() -> {
                    this.mwc.connectionProgress.set(progress);
                    // override cases where > 100 percent misses
                    int progressPercent = (int)(progress*100);
                    progressPercent = (progressPercent > 100) ?
                            100 : progressPercent;
                    // set status with percentage
                    this.mwc.connectionStatus.set(
                            "Receiving data - " +
                            progressPercent + "%");
                });
            }
        }
    }
    else if(sType == StatusType.STATE_CHANGED) {
    }
}

@Override
public void onStatusUpdate(StatusType sType, ConnectionState oldState,
        Object ... args) {
    super.onStatusUpdate(sType, oldState, args);
}
}

private static class SenderPresenter extends PresenterBase {
    final private MainWindowController mwc;
    private int lastSegmentReq = -1;

    public SenderPresenter(SmsftpFacade smsftpFacade,
            MainWindowController mwc) {
        super(smsftpFacade, mwc);
        this.mwc = mwc;
    }

    @Override
    protected void onUserPrompt(PromptType pType, String message) {
        switch(pType) {
            case CONNECTION_REQUEST: {
                return; // do nothing; this should never happen
            }
            case TIMEOUT:
            case SMS_SERVICE_ERROR:
            case PEER_ERROR_EXCEEDED_LIMIT:
            default: {
                super.onUserPrompt(pType, message);
            }
        }
    }

    @Override
    public void onStatusUpdate(StatusType sType, SessionState sState,
            Object ... args) {
        super.onStatusUpdate(sType, sState, args);
        if(sType == StatusType.STATE_CHANGED) {
            if(args == null || args.length < 1 ||
                    !(args[0] instanceof SessionState)) {
                return; // exit ASAP
            }
            SessionState newState = (SessionState)(args[0]);
            if(sState.equals(newState)) {
                return; // exit on nothing changed condition
            }
            switch(newState) {
                case TERMINATING: {
                    if(this.mwc.currentFacade.smsftpSession
                            .getSessionIntent().equals(
                            SessionIntent.DATA_TRANSFER)) {
                        Platform.runLater(() ->
                                this.mwc.ftpSentSuccessNotification.show());
                        break;
                    }
                }
                default: {
                    // do nothing
                }
            }
        }
    }

    @Override
    public void onStatusUpdate(StatusType sType, ConnectionState oldState,
            Object ... args) {
        super.onStatusUpdate(sType, oldState, args);
        if(sType == StatusType.NOTICE) {
            SessionState sState =
                    this.mwc.currentFacade.smsftpSession.getSessionState();
            if(sState.equals(SessionState.SENDING_DATA) ||
                    sState.equals(SessionState.COMPLETE_WAIT)) {
                // extract segment for retransmission and
                // update lastSegmentReq when necessary
                PDUType t;
                int n;
                double missingNos = 0; // we'll leave this as zero for now
                if(args.length == 2 && args[0] instanceof PDUType &&
                        args[1] instanceof Integer &&
                        args[0] != null && args[1] != null) {
                    t = (PDUType)(args[0]);
                    n = (Integer)(args[1]);
                    n = (n > this.lastSegmentReq) ?
                            n : this.lastSegmentReq;
                    this.lastSegmentReq = n;
                }
                else {
                    return; // exit ASAP on failure
                }
                if(t.equals(SmsftpPDUType.SMSFTP_DATA)) {
                    // compute estimated progress, etc.
                    double segCount = this.mwc.currentFacade
                            .smsftpSession.getDataSize();
                    // this is just an estimate anyway
                    final double progress = (n - missingNos)/segCount;
                    // then modify related ui components
                    Platform.runLater(() -> {
                        this.mwc.connectionProgress.set(progress);
                        // override cases where > 100 percent misses
                        int progressPercent = (int)(progress*100);
                        progressPercent = (progressPercent > 100) ?
                                100 : progressPercent;
                        // set status with percentage
                        this.mwc.connectionStatus.set(
                                "Sending data - " +
                                progressPercent + "%");
                    });
                }
            }
        }
    }
}


// toolbar buttons
@FXML
private Button addContactButton;
@FXML
private Button editContactButton;
@FXML
private Button removeContactButton;
@FXML
private Button sendFileButton;
@FXML
private Button sendFileInsecurelyButton;
@FXML
private Button appMenuButton;
// infobars
@FXML
private Parent acceptConnInfobar;
@FXML
private Button toggleAcceptButton;
@FXML
private Parent errorInfobar;
// panes
@FXML
```

```
private HBox welcomePane;
@FXML
private ListView contactList;
// misc defined properties
@FXML
private String startRcvLblStr;
@FXML
private String stopRcvLblStr;
@FXML
private String startRcvBtnStr;
@FXML
private String stopRcvBtnStr;
// log-related controls
@FXML
private Button showHideButton;
@FXML
private TextArea logTarget;

final private ContextMenu appMenuCM = new ContextMenu();
private Stage prefsStage;
private ContactEditController contactEditController;
private StandardDialog removeContactConfirmation = null;
private StandardDialog peerDisconnectNotification = null;
private StandardDialog connectionErrorNotification = null;
private StandardDialog errorNotification = null;
private StandardDialog acceptConfirmation = null;
private StandardDialog disconnectConfirmation = null;
private StandardDialog ftpRecvSuccessNotification = null;
private StandardDialog ftpSentSuccessNotification = null;
private StandardDialog ftpFailureNotification = null;
private FingerprintDialog fpConfirmation = null;
private static final ButtonType dialogHideButton =
        new ButtonType("Hide", ButtonData.LEFT);
private static final ButtonType dialogCancelButton =
        new ButtonType("Disconnect", ButtonData.NO);
private static final ButtonType dialogRetryButton =
        new ButtonType("Retry", ButtonData.YES);
private static final ButtonType dialogDeleteButton =
        new ButtonType("Delete", ButtonData.OK_DONE);
private static final ButtonType dialogAcceptButton =
        new ButtonType("Accept", ButtonData.YES);
private static final ButtonType dialogRejectButton =
        new ButtonType("Reject", ButtonData.NO);
private static final ButtonType dialogSavekeysButton =
        new ButtonType("Save keys", ButtonData.YES);
private static final ButtonType dialogDisconnectButton =
        new ButtonType("Disconnect", ButtonData.OK_DONE);
private final FileChooser sendFileChooser = new FileChooser();

final private SimpleDoubleProperty connectionProgress =
        new SimpleDoubleProperty(-1);
final private SimpleStringProperty connectionStatus =
        new SimpleStringProperty("");
final private SimpleBooleanProperty isReceiving =
        new SimpleBooleanProperty(false);
final private SimpleBooleanProperty isAccepting =
        new SimpleBooleanProperty(true);
final private SimpleBooleanProperty isPostponingAccepting =
        new SimpleBooleanProperty(false);
final private SimpleBooleanProperty smsServiceIsReady =
        new SimpleBooleanProperty(false);

protected Map<String, Object> configRoot;
@FXML
protected ObservableList<Map<String, Object>> contactOList;
final protected SimpleStringProperty activeContact =
        new SimpleStringProperty(null);

private ExtendedSmsService smsService = null;
private String localMin = null;
private SmsftpFacade currentFacade = null;

private volatile CountDownLatch acCDL = null;
private volatile boolean refreshContactList = false;

private ExecutorService pool = Executors.newCachedThreadPool();
private Logger mainLogger;
private Logger protocolLogger;



/**
 * Initializes the controller for objects that are needed from
 * external sources as well as the properties of its members
 *
 */
public void initialize(Map<String, Object> configRoot,
        Map<String, Map<String, Object>> contactsRoot,
        Stage prefsStage, ContactEditController contactEditController,
        Logger mainLogger, Logger protocolLogger) {
    this.configRoot = ConfigManager.mapDeepCopy(configRoot);
```

```
this.prefsStage = prefsStage;
this.contactEditController = contactEditController;
this.mainLogger = mainLogger;
this.protocolLogger = protocolLogger;

// set properties for contactList
this.contactOList.addAll(
        ContactManager.convertToContactList(contactsRoot));
this.contactList.setItems(this.contactOList);
this.contactList.setCellFactory((lview) -> {
    ContactListCell lcell =
            new ContactListCell(this);
    return lcell;
});

// set bindings
this.acceptConnInfobar.visibleProperty().bind(
        this.smsServiceIsReady.and(this.activeContact.isNull()));
this.acceptConnInfobar.managedProperty().bind(
        this.smsServiceIsReady.and(this.activeContact.isNull()));
this.errorInfobar.visibleProperty().bind(
        this.smsServiceIsReady.not());
this.errorInfobar.managedProperty().bind(
        this.smsServiceIsReady.not());

this.welcomePane.visibleProperty().bind(
        Bindings.isEmpty(contactOList));
this.contactList.visibleProperty().bind(
        Bindings.isNotEmpty(contactOList));

ObservableList<Map<String, Object>> selectedItems =
        this.contactList.getSelectionModel().getSelectedItems();
BooleanBinding noContactSelection = Bindings.isEmpty(selectedItems);
BooleanBinding currentItemNotActiveContact =
        new BooleanBinding() {
            { // this is an instance initializer
                super.bind(selectedItems,
                        MainWindowController.this.activeContact);
            }

            @Override
            protected boolean computeValue() {
                // check for empty cases first
                return selectedItems.isEmpty() ||
                        (MainWindowController.this.activeContact.get()
                        != null &&
                // then check if selectedItem == activeContact
                        MainWindowController.this.activeContact.get()
                        .equals(selectedItems.get(0).get(
                        ContactManager.MSISDN_KEY)));
            }
        };
this.editContactButton.disableProperty().bind(
        noContactSelection.or(currentItemNotActiveContact));
this.removeContactButton.disableProperty().bind(
        noContactSelection.or(currentItemNotActiveContact));
this.sendFileInsecurelyButton.disableProperty().bind(
        noContactSelection.or(
        this.activeContact.isNotNull().or(
        this.smsServiceIsReady.not().or(
        Bindings.isNotNull(this.activeContact)))));
this.sendFileButton.disableProperty().bind(
        noContactSelection.or(
        this.activeContact.isNotNull().or(
        this.smsServiceIsReady.not().or(
        Bindings.isNotNull(this.activeContact)).or(
        new BooleanBinding() {
            { // this is an instance initializer
                super.bind(selectedItems);
            }

            @Override
            protected boolean computeValue() {
                return selectedItems.isEmpty() ||
                        (null == selectedItems.get(0).get(
                        ContactManager.KEYS_KEY));
            }
        }))));

//// initialize Dialogs
this.peerDisconnectNotification =
        new StandardDialog(
        "Peer Disconnected",
        "", // populated with custom status code and messages from
            // CLOSE_WAIT NOTICE
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-information.png"))
);
this.errorNotification =
        new StandardDialog(
        // NOTE: this is populated with custom messages from errors
        //          thrown by SmsService
        "",
        "",
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-error.png"))
);
this.connectionErrorNotification =
        new StandardDialog(
        // NOTE: this is populated with custom messages from errors
        //          thrown by SmsftpFacades
        "",
        "",
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-error.png"))
);
this.removeContactConfirmation = new StandardDialog(
        "Delete this contact?",
        "Deleting this contact will also delete the associated " +
        "pairing keys, if any. This action cannot be undone.",
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-warning.png"))
);
this.acceptConfirmation = new StandardDialog(
        "Accept file transfer request?",
        "", // NOTE: text here is generated dynamically
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-information.png"))
);
this.disconnectConfirmation = new StandardDialog(
        "Stop file transfer?",
        "You will lose all your progress for this session and the " +
        "connection will be terminated.",
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-warning.png"))
);
this.ftpRecvSuccessNotification = new StandardDialog(
        "File has been successfully received",
        "The file transfer has been completed and the received " +
        "data has been verified.",
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-information.png"))
);
this.ftpSentSuccessNotification = new StandardDialog(
        "File has been successfully sent",
        "The receiver has successfuly received the file and " +
        "verified the data.",
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-information.png"))
);
this.ftpFailureNotification = new StandardDialog(
        "File transfer failed",
        "The file transfer failed verification. This could " +
        "be caused by a faulty network connection " +
        "or a possible MITM attack from a third-party. Please " +
        "check your connection and try again.",
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-warning.png"))
);
this.fpConfirmation =  new FingerprintDialog(
        "Key exchange verification",
        "Use this to further verify that the keys exchanged " +
        "are identical. This step is optional.\n\n" +
        "Make sure that the words shown on the peer matches " +
        "the ones below:\n\n\n",
        new Image(
        this.getClass().getClassLoader().getResourceAsStream(
        "icons/dialog-password.png")));
// add buttons to dialogs
this.peerDisconnectNotification.getDialogPane().getButtonTypes()
        .add(ButtonType.OK);
this.errorNotification.getDialogPane().getButtonTypes()
        .add(MainWindowController.dialogHideButton);
this.errorNotification.getDialogPane().getButtonTypes()
        .add(MainWindowController.dialogCancelButton);
this.errorNotification.getDialogPane().getButtonTypes()
        .add(MainWindowController.dialogRetryButton);
this.connectionErrorNotification.getDialogPane().getButtonTypes()
        .add(MainWindowController.dialogCancelButton);
this.connectionErrorNotification.getDialogPane().getButtonTypes()
        .add(MainWindowController.dialogRetryButton);
this.removeContactConfirmation.getDialogPane().getButtonTypes()
        .add(MainWindowController.dialogDeleteButton);
    this.removeContactConfirmation.getDialogPane().getButtonTypes()
            .add(ButtonType.CANCEL);
    this.acceptConfirmation.getDialogPane().getButtonTypes()
            .add(MainWindowController.dialogAcceptButton);
    this.acceptConfirmation.getDialogPane().getButtonTypes()
            .add(MainWindowController.dialogRejectButton);
    this.disconnectConfirmation.getDialogPane().getButtonTypes()
            .add(MainWindowController.dialogDisconnectButton);
    this.disconnectConfirmation.getDialogPane().getButtonTypes()
            .add(ButtonType.CANCEL);
    this.ftpRecvSuccessNotification.getDialogPane().getButtonTypes()
            .add(ButtonType.OK);
    this.ftpSentSuccessNotification.getDialogPane().getButtonTypes()
            .add(ButtonType.OK);
    this.ftpFailureNotification.getDialogPane().getButtonTypes()
            .add(ButtonType.OK);
    this.fpConfirmation.getDialogPane().getButtonTypes()
            .add(MainWindowController.dialogSavekeysButton);
    this.fpConfirmation.getDialogPane().getButtonTypes()
            .add(ButtonType.CANCEL);

    // initialize FileChoosers
    this.sendFileChooser.setTitle("Select file to be sent");


    // task for tailing TextArea logTarget
    this.pool.submit(new Task<Void>() {
        @Override
        protected Void call() throws Exception {
            boolean logLoop = true;
            while(logLoop) {
                String logText = "";
                try { // try to retrieve log text
                    logText = QueueAppender.pop();
                }
                catch(InterruptedException e) {
                    logLoop = false; // stop looping
                    // and greedily consume all remaining messages
                    StringBuffer buff = new StringBuffer(logText);
                    String message = QueueAppender.poll();
                    while(message != null) {
                        buff.append(message);
                        message = QueueAppender.poll();
                    }
                    logText = buff.toString();
                }
                finally {
                    String logTextCopy = new String(logText);
                    logText = ""; // try to flush for sudden interruptions
                    Platform.runLater(() -> { // append and scroll to bottom
                        logTarget.appendText(logTextCopy);
                        logTarget.setScrollTop(Double.MAX_VALUE);
                    });
                }
            }
            return null; // for the sake that we return a "Void"
        }
    });

    // initialize ContextMenu for AppMenu
    MenuItem preferencesItem = new MenuItem("Preferences…");
    MenuItem aboutItem = new MenuItem("About");
    preferencesItem.setOnAction((e) -> {
        MainWindowController.this.prefsStage.show();
    });
    aboutItem.setOnAction((e) -> {
    });
    this.appMenuCM.getItems().add(preferencesItem);
    this.appMenuCM.getItems().add(aboutItem);

    // initialize current SmsService
    this.initializeSmsService();
    // start accepting connections
    if(this.smsServiceIsReady.get()) {
        this.startAcceptingConnections();
    }
}

/**
 * Shuts down sms service, and executor service
 */
public void shutdown() {
    // try to stop existing and any attempt to accept new connections
    if(this.isReceiving.get() || this.isAccepting.get()) {
        this.isReceiving.set(false);
        this.isAccepting.set(false);
        this.stopAcceptingConnections();
    }
    // try to stop running smsServices
    if(this.smsService != null) {
        this.smsService.stop();
    }
    // finally stop all other threads
    this.pool.shutdownNow();
}

@FXML
protected void handleSendFileButton(ActionEvent event) {
    final List<Map<String, Object>> selectedItems =
            this.contactList.getSelectionModel().getSelectedItems();

    this.handleSendAction(event, (sendFile) -> {
        String recvMsisdn = Utils.getIntFormat(this.activeContact.get(),
                this.configRoot.get(
                ConfigManager.COUNTRYCODE_KEY).toString());
        byte[] dataBytes = null;
        try {
            dataBytes = Files.readAllBytes(sendFile.toPath());
        }
        catch(IOException e) {
            // this covers SecurityException, OutOfMemoryError, and
            // IOException
            e.printStackTrace();
        }

        // initialize core objects
        Connection sConn = Connection.createInitiatorConnection(
                MainWindowController.this.localMin, recvMsisdn);
        SmsftpSession sSession =
                SmsftpSession.generateSenderDataSession(
                sConn, sendFile.getName(), dataBytes,
                ((Map<String, byte[]>)(selectedItems.get(0)
                .get(ContactManager.KEYS_KEY)))
                .get(ContactManager.HOST_PRIV_ENC_KEY),
                ((Map<String, byte[]>)(selectedItems.get(0)
                .get(ContactManager.KEYS_KEY)))
                .get(ContactManager.HOST_PRIV_AUTH_KEY));
        this.currentFacade = new SmsftpFacade(sSession,
                this.smsService);
        // add presenters
        this.currentFacade.addObserver(
                new SenderPresenter(this.currentFacade, this));
        this.currentFacade.addObserver(
                new SmsftpLoggerPresenter(this.protocolLogger));

        // start the transmission
        this.pool.submit(new Task<Void>() {
            @Override
            protected Void call() {
                MainWindowController.this.currentFacade.secureSendFile();
                return null; // for the sake that we return a "Void"
            }
        });
    });
}

@FXML
protected void handleSendFileInsecurelyButton(ActionEvent event) {
    // copied from handleSendFileButton
    final List<Map<String, Object>> selectedItems =
            this.contactList.getSelectionModel().getSelectedItems();

    this.handleSendAction(event, (sendFile) -> {
        String recvMsisdn = Utils.getIntFormat(this.activeContact.get(),
                this.configRoot.get(
                ConfigManager.COUNTRYCODE_KEY).toString());
        byte[] dataBytes = null;
        try {
            dataBytes = Files.readAllBytes(sendFile.toPath());
        }
        catch(IOException e) {
            // this covers SecurityException, OutOfMemoryError, and
            // IOException
            e.printStackTrace();
        }

        // initialize core objects
        Connection sConn = Connection.createInitiatorConnection(
                MainWindowController.this.localMin, recvMsisdn);
        SmsftpSession sSession =
                SmsftpSession.generateSenderDataSession(
                sConn, sendFile.getName(), dataBytes,
                null, null);
        this.currentFacade = new SmsftpFacade(sSession,
                this.smsService);
        // add presenters
        this.currentFacade.addObserver(
                new SenderPresenter(this.currentFacade, this));
        this.currentFacade.addObserver(
                new SmsftpLoggerPresenter(this.protocolLogger));

        // start the transmission
        this.pool.submit(new Task<Void>() {
            @Override
            protected Void call() {
                MainWindowController.this.currentFacade.insecureSendFile();
                return null; // for the sake that we return a "Void"
            }
        });
    });
}

private void handleSendAction(ActionEvent event, Consumer<File> c) {
    File sendFile = this.sendFileChooser.showOpenDialog(
        ((Node)event.getTarget()).getScene().getWindow());
    List<Map<String, Object>> selectedItems =
            this.contactList.getSelectionModel().getSelectedItems();

    if(sendFile != null && !selectedItems.isEmpty()) {
        Map<String, Object> item = selectedItems.get(0);
        if(item.get(ContactManager.MSISDN_KEY) == null) {
            return;
        }

        this.refreshServiceAndFacade(() -> {
            if(this.acCDL != null) {
                try {
                    this.acCDL.await();
                    this.acCDL = null;
                }
                catch(InterruptedException e) {
                    return;
                }
            }
            // setup necessary UI elements
            MainWindowController.this.activeContact.set(
                    (String)(item.get(ContactManager.MSISDN_KEY)));
            MainWindowController.this.showActiveConnectionControls();

            MainWindowController.this.smsService.setPeer(
                    Utils.getIntFormat(
                    MainWindowController.this.activeContact.get(),
                    MainWindowController.this.configRoot.get(
                    ConfigManager.COUNTRYCODE_KEY).toString()));
            c.accept(sendFile);
        });
    }
}

@FXML
protected void handleCancelButtonAction(ActionEvent event) {
    // prompt for user when cancelling first
    this.disconnectConfirmation.showAndWait()
            .filter(response -> response ==
            MainWindowController.dialogDisconnectButton)
            .ifPresent(response -> {
                this.currentFacade.disconnect("1002"); // stop facade asap
    });
}

@FXML
protected void handleToggleAcceptButtonAction(ActionEvent event) {
    // disable toggleAcceptButton first
    this.toggleAcceptButton.setDisable(true);
    if(this.isAccepting.get()) { // true -> false
        // stop accepting connections first
        this.stopAcceptingConnections();
        this.isAccepting.set(false);
        if(!this.isAccepting.get()) { // change if only successful
            this.acceptConnInfobar.getStyleClass().remove("infoInfobar");
            ((Label)(this.acceptConnInfobar.lookup(
                    ".infobarLabel"))).setText(this.startRcvLblStr);
            this.toggleAcceptButton.setText(this.startRcvBtnStr);
        }
    }
    else { // false -> true
        this.isAccepting.set(this.startAcceptingConnections());
        if(this.isAccepting.get()) { // change if only successful
            this.acceptConnInfobar.getStyleClass().add("infoInfobar");
            ((Label)(this.acceptConnInfobar.lookup(
                    ".infobarLabel"))).setText(this.stopRcvLblStr);
            this.toggleAcceptButton.setText(this.stopRcvBtnStr);
        }
    }
    // finally, reenable toggleAcceptButton
    this.toggleAcceptButton.setDisable(false);
}

@FXML
protected void handleShowErrorDetailsButtonAction(ActionEvent event) {
    this.errorNotification.showAndWait().filter(
            response -> response == MainWindowController.dialogRetryButton)
            // or dialogCancelButton or dialogHideButton
            .ifPresent(response -> {
            });
}

@FXML
protected void handleAddContactButtonAction(ActionEvent event) {
    this.contactEditController.prepareAndShowStage(null);
}

@FXML
protected void handleEditContactButtonAction(ActionEvent event) {
    // we assume that selectedItems is always not empty
    List<Map<String, Object>> selectedItems =
            this.contactList.getSelectionModel().getSelectedItems();

    this.contactEditController.prepareAndShowStage(selectedItems.get(0));
}

@FXML
protected void handleRemoveContactButtonAction(ActionEvent event) {
    List<Map<String, Object>> selectedItems =
            this.contactList.getSelectionModel().getSelectedItems();
    if(!selectedItems.isEmpty()) {
        this.removeContactConfirmation.showAndWait()
                .filter(response -> response ==
                MainWindowController.dialogDeleteButton)
                .ifPresent(response -> {
                    // remove contact from list
                    this.contactOList.remove(selectedItems.get(0));
                    // save on file ASAP
                    ContactManager.writeConfig(
                            ContactManager.convertToContactMap(
                            this.contactOList));
                });
    }
}

protected void handleExternalPairButtonAction(ActionEvent event) {
    // store activeContact temporarily to prevent resets caused
    // by stopAcceptingConnections()
    String tempAC = this.activeContact.get();
    String recvMsisdn = Utils.getIntFormat(this.activeContact.get(),
            this.configRoot.get(
            ConfigManager.COUNTRYCODE_KEY).toString());

    this.refreshServiceAndFacade(() -> {
        if(this.acCDL != null) {
            try {
                this.acCDL.await();
                this.acCDL = null;
            }
            catch(InterruptedException e) {
                return;
            }
        }

        // restore activeContact value we have stored earlier
        this.activeContact.set(tempAC);
        MainWindowController.this.smsService.setPeer(recvMsisdn);

        // initialize core objects
        Connection sConn = Connection.createInitiatorConnection(
                MainWindowController.this.localMin, recvMsisdn);
        SmsftpSession sSession =
                SmsftpSession.generateInitiatorPairSession(sConn);
        MainWindowController.this.currentFacade =
                new SmsftpFacade(sSession,
                MainWindowController.this.smsService);
        MainWindowController.this.currentFacade.addObserver(
                new SenderPresenter(
                MainWindowController.this.currentFacade,
                MainWindowController.this));
        MainWindowController.this.currentFacade.addObserver(
                new SmsftpLoggerPresenter(
                MainWindowController.this.protocolLogger));

        // show connection controls later after initialization of
        // other components
        MainWindowController.this.showActiveConnectionControls();

        // start pairing
        MainWindowController.this.pool.submit(new Task<Void>() {
            @Override
            protected Void call() {
                MainWindowController.this.currentFacade.pair();
                return null; // for the sake that we return a "Void"
            }
        });
    });
}

@FXML
protected void handleAppMenuButtonAction(ActionEvent event) {
    this.appMenuCM.show(appMenuButton, Side.LEFT,
            appMenuButton.getWidth(), 0);
}

@FXML
protected void handleShowHideButtonAction(ActionEvent event) {
    if(logTarget.isVisible()) {
        logTarget.getMinHeight();
        logTarget.setMinHeight(-10000);
        logTarget.setMaxHeight(-10000);
        showHideButton.setText("Show logs");
    }
    else {
        logTarget.setMinHeight(100);
        logTarget.setMaxHeight(Long.MAX_VALUE);
        showHideButton.setText("Hide logs");
    }
    logTarget.setVisible(!logTarget.isVisible());
}


//// utility functions
/**
 * Initializes or reinitializes this controller's SmsService. This also
 * stops the existing SmsService, if any, and starts the newly initialized
 * service.
 */
protected void initializeSmsService() {
    // stop current SmsService first
    if(this.smsService != null) {
        this.smsService.stop();
    }

    // determine which type of SmsService should be initialized
    String backendType = (String)(this.configRoot.get(
            ConfigManager.BACKENDTYPE_KEY));
    Map<String, Object> backendConfigRoot =
            (Map<String, Object>)((Map<String, Object>)
            this.configRoot.get(ConfigManager.BACKENDSETTINGS_KEY)).get(
            "backend-" + backendType);

    switch(backendType) { // initialize corresponding SmsService
        case ConfigManager.BACKEND_DUMMY_VALUE: {
            String host = (String)(backendConfigRoot.get(
                    ConfigManager.BACKEND_DUMMY_HOST_KEY));
            int port = (Integer)backendConfigRoot.get(
                    ConfigManager.BACKEND_DUMMY_PORT_KEY);
            this.localMin = (String)(backendConfigRoot.get(
                    ConfigManager.BACKEND_DUMMY_LOCALMIN_KEY));
            boolean isSender = "sender".equals(backendConfigRoot.get(
                    ConfigManager.BACKEND_DUMMY_ROLE_KEY));
            this.smsService = new DummySmsService(host, port, localMin,
                    isSender);
            break;
        }
        case ConfigManager.BACKEND_GAMMUPSQL_VALUE: {
            String host = (String)(backendConfigRoot.get(
                    ConfigManager.BACKEND_GAMMUPSQL_HOST_KEY));
            int port = (Integer)backendConfigRoot.get(
                    ConfigManager.BACKEND_GAMMUPSQL_PORT_KEY);
            String name = (String)(backendConfigRoot.get(
                    ConfigManager.BACKEND_GAMMUPSQL_NAME_KEY));
            String user = (String)(backendConfigRoot.get(
                    ConfigManager.BACKEND_GAMMUPSQL_USER_KEY));
            String pw = (String)(backendConfigRoot.get(
                    ConfigManager.BACKEND_GAMMUPSQL_PASS_KEY));
            this.localMin = (String)(backendConfigRoot.get(
                    ConfigManager.BACKEND_GAMMUPSQL_LOCALMIN_KEY));
            this.smsService = new GammuSmsService(
                    host, port, name, user, pw, this.localMin);
            break;
        }
        case ConfigManager.BACKEND_ANDROID_VALUE: {
            String host = (String)(backendConfigRoot.get(
                    ConfigManager.BACKEND_ANDROID_HOST_KEY));
            int port = (Integer)backendConfigRoot.get(
                    ConfigManager.BACKEND_ANDROID_PORT_KEY);
            this.smsService = new AndroidSmsService(host, port);
            break;
        }
        default:
            assert false;
            return; // do nothing and return ASAP
    }
    // set country code first
    this.smsService.setCountryCode(
            this.configRoot.get(ConfigManager.COUNTRYCODE_KEY).toString());
    // finally, start SmsService
    try {
        this.smsServiceIsReady.set(this.smsService.start());
        // set localMin via service if not yet configured on
        // this app's' preferences
        if(this.localMin == null) {
            this.localMin = this.smsService.getLocalMin();
        }
    }
    catch(SmsServiceException e) {
        // er must always be type of ExtendedReason
        ExtendedReason er = (ExtendedReason)e.getReason();
        // update error dialog box with details from the exception's
        // reason's properties
        this.errorNotification.setPrimaryText(er.getMessage());
        this.errorNotification.setSecondaryText(er.getDetails());
        this.smsServiceIsReady.set(false); // set as not ready
    }
}

protected boolean startAcceptingConnections() {
    if(!(this.smsServiceIsReady.get()) || currentFacade != null) {
        return false;
    }

    // initialize smsftp connection and session objects
    Connection recvConn =
            Connection.createResponderConnection(this.localMin);
    SmsftpSession recvSession =
            SmsftpSession.generateReceiverSession(recvConn);
    this.currentFacade = new SmsftpFacade(recvSession,
            this.smsService);
    // add presenters
    this.currentFacade.addObserver(
            new ReceiverPresenter(this.currentFacade, this));
    this.currentFacade.addObserver(
            new SmsftpLoggerPresenter(this.protocolLogger));

    // run the receive method in the background
    this.pool.submit(new Task<Void>() {
        @Override
        protected Void call() {
            MainWindowController.this.currentFacade.receive();
            return null; // for the sake that we return a "Void"
        }
    });

    return true;
}

protected boolean stopAcceptingConnections() {
    if(this.currentFacade != null) {
        // NOTE: 1002 - Cancelled by user
        this.currentFacade.disconnect("1002"); // stop facade asap
        return true; // finally just return true
    }
    return false; // return false on further attempts
}

private void refreshServiceAndFacade(Runnable newAction) {
    // preemptive checks
    if(!(this.currentFacade == null ^ this.isAccepting.get())) {
        return;
    }
    if(this.smsService != null) {
        this.acCDL = new CountDownLatch(1);
        // stop accepting connections and interrupt any incomplete
        // receiving procedures in progress, if any
        if(this.smsService.isRunning()) {
            if(this.isAccepting.get()) {
                this.stopAcceptingConnections();
                this.isPostponingAccepting.set(true);
            }
        }
        else {
        }
        newAction.run(); // run the new action (i.e. send file, pair)
    }
    else {
        return; // we cannot do anything with broken sms services
    }
}

protected synchronized void showActiveConnectionControls() {
    if(this.activeContact.get() != null) {
        String prefix = "#" + this.activeContact.get();
        this.contactList.lookup(prefix+"StatusLabel")
                .setVisible(true);
        this.contactList.lookup(prefix+"ProgressBar")
                .setVisible(true);
        this.contactList.lookup(prefix+"CancelButton")
                .setVisible(true);
    }
}

protected synchronized void hideActiveConnectionControls() {
    if(this.activeContact.get() != null) {
        String prefix = "#" + this.activeContact.get();
        this.contactList.lookup(prefix+"StatusLabel").setVisible(false);
        this.contactList.lookup(prefix+"ProgressBar").setVisible(false);
        this.contactList.lookup(prefix+"CancelButton").setVisible(false);
    }
}

protected void cleanupTransmission() {
    if(this.currentFacade != null) {
        this.currentFacade = null; // "destroy" currentFacade
        this.hideActiveConnectionControls(); // finally, some UI cleanups
        this.activeContact.set(null); // reset activeContact to null
        if(this.smsService != null) { // reset peer to null, if applicable
            this.smsService.setPeer(null);
        }

        if(this.acCDL != null) {
            this.acCDL.countDown();
        }

        // listen again if set to accept connections automatically
        if(this.isPostponingAccepting.get()) {
            // reset flag for use next time
            this.isPostponingAccepting.set(false);
        }
        else { // if not set to postpone accepting
            if(this.isAccepting.get()) {
                this.mainLogger.info("Listening for connections again");
                this.startAcceptingConnections();
            }
        }
    }
}


public static String returnUnique(String u, String ... a) {
    if(a == null) {
        return u;
    }
    for(String s : a) {
        if(s.equals(u)) { // found a conflict
            String[] parts = s.split(" ");
            if(parts.length >= 2) {
                // check if last part is numeric, and update i if needed
                try {
                    int j = Integer.parseInt(parts[parts.length-1]);
                    // then make i the bigger number
                    parts[parts.length-1] = Integer.toString(j+1);
                    return String.join(" ", parts);
                }
                catch(NumberFormatException e) {}
            }
            // just recurse appending " 2"
            return MainWindowController.returnUnique(u+" 2", a);
        }
    }
    return u;
}

public static String returnUnique(String u, File dir) {
    // we always assume dir is non-null
    String conflicts[] = dir.list(new FilenameFilter() {
        @Override
        public boolean accept(File dir, String name) {
            if(u.equals(name)) {
                return true;
            }
            return false;
        }
    });
    if(conflicts.length >= 1) {
        // split filename into its extension and non-extension parts
        String v = u;
        String[] parts = u.split("\\.");
        if(parts.length >= 2) {
            v = String.join(".",
                    Arrays.copyOf(parts, parts.length-1));
        }
        // try to append 2, 3, etc. and recurse again
        return returnUnique((returnUnique(v, v)) + (v.equals(u) ?
                "" : "."+parts[parts.length-1]), dir);
    }
    else {
        return u;
    }
}
```

```
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/MappedStringConverter.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import javafx.beans.property.SimpleObjectProperty;
import javafx.util.StringConverter;

import java.util.Map;



/**
 * StringConverter implementation for custom mapped String values.
 */
public class MappedStringConverter extends StringConverter<String> {
    private final SimpleObjectProperty<Map<String, String>> lookupMap =
            new SimpleObjectProperty<>();



    @Override
    public String fromString(String value) {
        return value;
    }

    @Override
    public String toString(String object) {
        Map<String, String> m = this.lookupMap.get();
        if(m == null) {
            return "";
        }
        String str = m.get(object);
        if(str == null) {
            return "";
        }
        return str;
    }

    public Map<String, String> getLookupMap() {
        return this.lookupMap.get();
    }

    public void setLookupMap(Map<String, String> lookupMap) {
        this.lookupMap.set(lookupMap);
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/MsisdnStringConverter.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import javafx.util.converter.DefaultStringConverter;
import javafx.util.converter.LongStringConverter;



/**
 * StringConverter implementation for MSISDN values.
 */
public class MsisdnStringConverter extends DefaultStringConverter {
    private final LongStringConverter lsc = new LongStringConverter();

    @Override
    public String fromString(String str) {
        if(str == null) {
            return "";
        }
        return this.normalize(str);
    }

    @Override
    public String toString(String value) {
        if(value == null) {
            return "";
        }
        return value;
    }

    private String normalize(String str) {
        if(str.length() == 0) { // needed to accomodate blank strings
            return "";
        }
        // check for seemingly valid cases, but still need to fail
        // check for sole "+" cases
        if((str.length() == 1 && str.charAt(0) == '+') ||
                // check for "++..." cases
                (str.length() > 1 &&
                str.charAt(0) == str.charAt(1) &&
                str.charAt(0) == '+')) {
            throw new IllegalArgumentException();
        }
        // check if it is a parseable number
        Long l = this.lsc.fromString(str);
        if(l == null ||
                l.longValue() < 0) { // check for negative cases as well
            throw new IllegalArgumentException();
        }
        return str;
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/NonemptyStringConverter.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import javafx.util.converter.DefaultStringConverter;



/**
 * StringConverter implementation for non-emtpy String values.
 */
public class NonemptyStringConverter extends DefaultStringConverter {
    private String prevValue = "";



    @Override
    public String fromString(String value) {
        return super.fromString(value);
    }

    @Override
    public String toString(String object) {
        if(object != null && !(object.equals(""))) {
            this.prevValue = object;
            return super.toString(object);
        }
        return super.toString(this.prevValue);
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/PortStringConverter.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;


/**
 * StringConverter implementation for port number values.
 */
public class PortStringConverter extends PositiveIntegerStringConverter {
    private int prevValue = 0;



    @Override
    public Integer fromString(String value) {
        return this.normalize(super.fromString(value));
    }

    @Override
    public String toString(Integer value) {
        if(value == null) {
            return super.toString(this.normalize(this.prevValue));
        }
        return super.toString(this.normalize(value));
    }

    private int normalize(int val) {
        if(val >= 0 && val < 65536) {
            this.prevValue = val;
            return val;
        }
        else {
            return this.prevValue;
        }
    }
}
```

```
--------------------------------------------------------------
Filename:
src/main/java/com/transmisms/ui/javafx/PositiveIntegerStringConverter.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import javafx.util.converter.IntegerStringConverter;



/**
 * StringConverter implementation for positive Integer (and int primitive)
 * values.
 */
public class PositiveIntegerStringConverter extends IntegerStringConverter {
    @Override
    public Integer fromString(String value) {
        return Math.abs(super.fromString(value));
    }

    @Override
    public String toString(Integer value) {
        return super.toString(Math.abs(value));
    }
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/PrefsController.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import javafx.collections.ObservableMap;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.Node;
import javafx.scene.control.ComboBox;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.stage.DirectoryChooser;
import javafx.stage.Stage;
import javafx.stage.WindowEvent;

import java.io.File;
import java.util.HashMap;
import java.util.Map;

import java.io.IOException;



public class PrefsController {
    @FXML
    private ObservableMap<String, String> backendTypeConverterMap;

    @FXML
    private TextField downloadLocation;
    @FXML
    private ComboBox retryOnErrors;
    @FXML
    private TextField countryCode;
    @FXML
    private ComboBox backendType;
    @FXML
    private Label backendSettingsLabel;

    @FXML
    private TextField backendDummyHost;
    @FXML
    private TextField backendDummyPort;
    @FXML
    private TextField backendDummyLocalmin;
    @FXML
    private ComboBox backendDummyRole;
    @FXML
    private TextField backendAndroidHost;
    @FXML
    private TextField backendAndroidPort;
    @FXML
    private TextField backendGammuHost;
    @FXML
    private TextField backendGammuPort;
    @FXML
    private TextField backendGammuName;
    @FXML
    private TextField backendGammuUser;
    @FXML
    private TextField backendGammuPassword;
    @FXML
    private TextField backendGammuLocalmin;

    private final DirectoryChooser downloadLocationDirectoryChooser =
            new DirectoryChooser();

    private MainWindowController primaryController;
    private Map<String, Object> prevConfig;



    public void initialize(Map<String, Object> configRoot,
            MainWindowController primaryController) {
        // set properties
        this.primaryController = primaryController;

        // load config
        this.loadConfig(configRoot);

        // set listeners for controls

        // initialize misc properties
        this.downloadLocationDirectoryChooser.setTitle(
                "Save received files to");
        this.downloadLocation.end(); // move caret to end
        // set directory chooser's initialDirectory
        this.downloadLocationDirectoryChooser.setInitialDirectory(
                new File((String)(configRoot.get(
                ConfigManager.DOWNLOADLOC_KEY))));
        this.refreshBackendSettingsLabel(null);
    }

    public void loadConfig(Map<String, Object> configRoot) {
        this.prevConfig = new HashMap<>(configRoot);
        // NOTE: this assumes configRoot is complete and properly checked
        //       beforehand

        // root preferences
        this.downloadLocation.setText(
                (String)(configRoot.get(ConfigManager.DOWNLOADLOC_KEY)));
        this.retryOnErrors.setValue(
                (String)(configRoot.get(ConfigManager.RETRY_ON_ERRORS_KEY)));
        this.countryCode.setText(
                (String)(configRoot.get(ConfigManager.COUNTRYCODE_KEY)));
        this.backendType.setValue(
                (String)(configRoot.get(ConfigManager.BACKENDTYPE_KEY)));

        Map<String, Object> backendPrefsRoot =
                (Map<String, Object>)(configRoot.get(
                ConfigManager.BACKENDSETTINGS_KEY));
        Map<String, Object> backendDummyPrefsRoot =
                (Map<String, Object>)(backendPrefsRoot.get(
                ConfigManager.BACKEND_DUMMY_KEY));
        Map<String, Object> backendGammuPrefsRoot =
                (Map<String, Object>)(backendPrefsRoot.get(
                ConfigManager.BACKEND_GAMMUPSQL_KEY));
        Map<String, Object> backendAndroidPrefsRoot =
                (Map<String, Object>)(backendPrefsRoot.get(
                ConfigManager.BACKEND_ANDROID_KEY));

        // dummy backend settings
        this.backendDummyHost.setText(
                (String)(backendDummyPrefsRoot.get(
                ConfigManager.BACKEND_DUMMY_HOST_KEY)));
        this.backendDummyPort.setText(
                (String)(backendDummyPrefsRoot.get(
                ConfigManager.BACKEND_DUMMY_PORT_KEY)).toString());
        this.backendDummyLocalmin.setText(
                (String)(backendDummyPrefsRoot.get(
                ConfigManager.BACKEND_DUMMY_LOCALMIN_KEY)));
        this.backendDummyRole.setValue(
                (String)(backendDummyPrefsRoot.get(
                ConfigManager.BACKEND_DUMMY_ROLE_KEY)));

        // gammu backend settings
    this.backendGammuHost.setText(
            (String)(backendGammuPrefsRoot.get(
            ConfigManager.BACKEND_GAMMUPSQL_HOST_KEY)));
    this.backendGammuPort.setText(
            ((Integer)(backendGammuPrefsRoot.get(
            ConfigManager.BACKEND_GAMMUPSQL_PORT_KEY))).toString());
    this.backendGammuName.setText(
            (String)(backendGammuPrefsRoot.get(
            ConfigManager.BACKEND_GAMMUPSQL_NAME_KEY)));
    this.backendGammuUser.setText(
            (String)(backendGammuPrefsRoot.get(
            ConfigManager.BACKEND_GAMMUPSQL_USER_KEY)));
    this.backendGammuPassword.setText(
            (String)(backendGammuPrefsRoot.get(
            ConfigManager.BACKEND_GAMMUPSQL_PASS_KEY)));
    this.backendGammuLocalmin.setText(
            (String)(backendGammuPrefsRoot.get(
            ConfigManager.BACKEND_GAMMUPSQL_LOCALMIN_KEY)));

    // android backend settings
    this.backendAndroidHost.setText(
            (String)(backendAndroidPrefsRoot.get(
            ConfigManager.BACKEND_ANDROID_HOST_KEY)));
    this.backendAndroidPort.setText(
            ((Integer)(backendAndroidPrefsRoot.get(
            ConfigManager.BACKEND_ANDROID_PORT_KEY))).toString());
}

public void handleDownloadLocationSelectorButton(ActionEvent event) {
    File newDownloadLocDir =
            this.downloadLocationDirectoryChooser.showDialog(
            ((Node)event.getTarget()).getScene().getWindow());
            /*
            this.downloadLocationDirectoryChooser.showDialog(
            this.prefsStage);
            */
    if(newDownloadLocDir != null) {
        // change downloadLocation.text afterwards
        try {
            this.downloadLocation.setText(
                    newDownloadLocDir.getCanonicalPath());
            this.downloadLocationDirectoryChooser.setInitialDirectory(
                    null); // unset initialDirectory
            this.downloadLocation.end(); // move caret to end
        }
        catch(IOException e) {
            return; // exit ASAP on error
        }
    }
}

public void refreshBackendSettingsLabel(ActionEvent event) {
    this.backendSettingsLabel.setText(
            this.backendTypeConverterMap.get(this.backendType.getValue()) +
            " Backend Settings");
}

public void handleOnCloseRequest(WindowEvent event) {
    { // read changes and save back to config file
        Map<String, Object> configRoot =
                ConfigManager.mapDeepCopy(this.prevConfig);

        //// read changes and commit to configRoot
        configRoot.put(ConfigManager.DOWNLOADLOC_KEY,
                sanitizeAndGetText(this.downloadLocation));
        configRoot.put(ConfigManager.RETRY_ON_ERRORS_KEY,
                this.retryOnErrors.getValue());
        configRoot.put(ConfigManager.COUNTRYCODE_KEY,
                sanitizeAndGetText(this.countryCode));
        configRoot.put(ConfigManager.BACKENDTYPE_KEY,
                this.backendType.getValue());

        Map<String, Object> backendPrefsRoot =
                (Map<String, Object>)(configRoot.get(
                ConfigManager.BACKENDSETTINGS_KEY));
        Map<String, Object> backendDummyPrefsRoot =
                (Map<String, Object>)(backendPrefsRoot.get(
                ConfigManager.BACKEND_DUMMY_KEY));
        Map<String, Object> backendGammuPrefsRoot =
                (Map<String, Object>)(backendPrefsRoot.get(
                ConfigManager.BACKEND_GAMMUPSQL_KEY));
        Map<String, Object> backendAndroidPrefsRoot =
                (Map<String, Object>)(backendPrefsRoot.get(
                ConfigManager.BACKEND_ANDROID_KEY));

        // dummy backend settings
        backendDummyPrefsRoot.put(ConfigManager.BACKEND_DUMMY_HOST_KEY,
                sanitizeAndGetText(this.backendDummyHost));
        backendDummyPrefsRoot.put(ConfigManager.BACKEND_DUMMY_PORT_KEY,
                Integer.decode(sanitizeAndGetText(this.backendDummyPort)));
        backendDummyPrefsRoot.put(ConfigManager.BACKEND_DUMMY_LOCALMIN_KEY,
                sanitizeAndGetText(this.backendDummyLocalmin));
        backendDummyPrefsRoot.put(ConfigManager.BACKEND_DUMMY_ROLE_KEY,
                this.backendDummyRole.getValue());

        // gammu backend settings
        backendGammuPrefsRoot.put(ConfigManager.BACKEND_GAMMUPSQL_HOST_KEY,
                sanitizeAndGetText(this.backendGammuHost));
        backendGammuPrefsRoot.put(ConfigManager.BACKEND_GAMMUPSQL_PORT_KEY,
                Integer.decode(sanitizeAndGetText(this.backendGammuPort)));
        backendGammuPrefsRoot.put(ConfigManager.BACKEND_GAMMUPSQL_NAME_KEY,
                sanitizeAndGetText(this.backendGammuName));
        backendGammuPrefsRoot.put(ConfigManager.BACKEND_GAMMUPSQL_USER_KEY,
                sanitizeAndGetText(this.backendGammuUser));
        backendGammuPrefsRoot.put(ConfigManager.BACKEND_GAMMUPSQL_PASS_KEY,
                sanitizeAndGetText(this.backendGammuPassword));
        backendGammuPrefsRoot.put(
                ConfigManager.BACKEND_GAMMUPSQL_LOCALMIN_KEY,
                sanitizeAndGetText(this.backendGammuLocalmin));

        // android backend settings
        backendAndroidPrefsRoot.put(ConfigManager.BACKEND_ANDROID_HOST_KEY,
                sanitizeAndGetText(this.backendAndroidHost));
        backendAndroidPrefsRoot.put(ConfigManager.BACKEND_ANDROID_PORT_KEY,
                Integer.decode(
                sanitizeAndGetText(this.backendAndroidPort)));


        // finally write to file and save curConfig
        ConfigManager.writeConfig(configRoot); // write to file
        // compare changes and apply changes ASAP if needed
        boolean smsServiceChanged = !this.prevConfig.get(
                ConfigManager.BACKENDTYPE_KEY).equals(
                configRoot.get(ConfigManager.BACKENDTYPE_KEY));
        if(!smsServiceChanged) {
            // check for changes in the currently selected
            // SmsService's settings Map
            String backendKey = "backend-" +
                    configRoot.get(ConfigManager.BACKENDTYPE_KEY);
            Map<String, Object> oldConf =
                    (Map<String, Object>)(((Map<String, Object>)
                    this.prevConfig.get(
                    ConfigManager.BACKENDSETTINGS_KEY)).get(backendKey));
            Map<String, Object> newConf =
                    (Map<String, Object>)(((Map<String, Object>)
                    configRoot.get(
                    ConfigManager.BACKENDSETTINGS_KEY)).get(backendKey));
            Map<String, Object> diff =
                    ConfigManager.mapDiff(oldConf, newConf);
            smsServiceChanged = (diff.size() > 0);
        }
        // save everything to prevConfig
        this.prevConfig = configRoot;
        // finally, copy changes over primaryController
        this.primaryController.configRoot = ConfigManager.mapDeepCopy(
                configRoot, this.primaryController.configRoot);

        // reinitialize and restart service if relevant values are changed
        if(smsServiceChanged) {
            this.primaryController.initializeSmsService();
        }
    }
}

//// utility functions
private static String sanitizeAndGetText(TextField t) {
    t.commitValue(); // reformat text first
    return t.getText();
}
}
```

```
--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/QueueAppender.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;


import org.apache.logging.log4j.core.Filter;
import org.apache.logging.log4j.core.Layout;
import org.apache.logging.log4j.core.LogEvent;
import org.apache.logging.log4j.core.appender.AbstractAppender;
import org.apache.logging.log4j.core.appender.AppenderLoggingException;
import org.apache.logging.log4j.core.config.plugins.Plugin;
import org.apache.logging.log4j.core.config.plugins.PluginAttribute;
import org.apache.logging.log4j.core.config.plugins.PluginElement;
import org.apache.logging.log4j.core.config.plugins.PluginFactory;
import org.apache.logging.log4j.core.layout.PatternLayout;
import java.io.Serializable;
import java.util.Queue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;



@Plugin(name="QueueAppender", category="Core", elementType="appender",
        printObject=true)
public class QueueAppender extends AbstractAppender {

    private static BlockingQueue<String> queue =
            new LinkedBlockingQueue<String>();

    public static String pop() throws InterruptedException {
        return QueueAppender.queue.take();
    }

    public static String poll() throws InterruptedException {
        return QueueAppender.queue.poll(0l, TimeUnit.MILLISECONDS);
    }

    private QueueAppender(String name, Filter filter,
            Layout<? extends Serializable> layout,
            final boolean ignoreExceptions) {
        super(name, filter, layout, ignoreExceptions);
    }

    @Override
    public void append(LogEvent event) {
        try {
            QueueAppender.queue.offer(new String(this.getLayout().toByteArray(
                    event)), 1000l, TimeUnit.MILLISECONDS);
        }
        catch(InterruptedException e) {
            return; // do nothing and exit ASAP
        }
        catch(Exception ex) {
            if (!ignoreExceptions()) {
                throw new AppenderLoggingException(ex);
            }
        }
    }

    @PluginFactory
    public static QueueAppender createAppender(
            @PluginAttribute("name") String name,
            @PluginElement("Filter") final Filter filter,
            @PluginElement("Layout") Layout<? extends Serializable> layout,
            @PluginAttribute("ignoreExceptions") boolean ignoreExceptions,
            @PluginAttribute("otherAttribute") String otherAttribute) {
        // we basically do what other implementations of
        // @PluginFactory createAppender() do: some checks and behave as

        // a normal factory method
        if (name == null) {
            LOGGER.error("No name provided for QueueAppender");
            return null;
        }
        if (layout == null) {
            layout = PatternLayout.createDefaultLayout();
        }
        return new QueueAppender(name, filter, layout, ignoreExceptions);
    }

    @PluginFactory
    public static QueueAppender createAppender(
            @PluginAttribute("name") String name,
            @PluginElement("Filter") final Filter filter,
            @PluginElement("Layout") Layout<? extends Serializable> layout,
            @PluginAttribute("otherAttribute") String otherAttribute) {
        return QueueAppender.createAppender(name, filter, layout, true,
                otherAttribute);
    }

}



--------------------------------------------------------------
Filename:
src/main/java/com/transmisms/ui/javafx/smsservice/AndroidSmsService.java
--------------------------------------------------------------

package com.transmisms.ui.javafx.smsservice;

import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.core.protocol.SmsService;
import com.google.gson.Gson;
import org.zeromq.ZMQ;
import org.zeromq.ZMQ.Context;
import org.zeromq.ZMQ.Poller;
import org.zeromq.ZMQ.Socket;

import java.util.Date;
import java.util.Queue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicReference;

import com.transmisms.core.protocol.SmsServiceException;
import zmq.ZError;



public class AndroidSmsService extends ExtendedSmsService {
    public enum Reason implements SmsServiceException.Reason<Reason>,
           ExtendedSmsService.ExtendedReason {
        IOERROR ("Problem connecting to device",
          "Please check device connectivity and try again."),
        NOPONG ("Device is not responding",
          "Please check the device and try again.");



        private final String message;
        private final String details;


        // default access control since enums are restricted
        Reason(String message, String details) {
            this.message = message;
            this.details = details;
        }


        @Override
        public Reason valueOf() {
            return valueOf(this.name());
        }

        @Override
        public final String getMessage() {
            return this.message;
        }
        @Override
        public final String getDetails() {
            return this.details;
        }
    }



    /**
     * Copied from <code>com.transimsms.androidcompanion.ClientAction</code>.
     * This must be always up to date to the Transmisms Android Companion's
     * ClientAction class.
     */
    public static class ClientAction {
        final public String action;

        // send
        final public String recipient;
        final public String message;

        // ping
        // msisdn
        // NOTHING FOLLOWS

        public ClientAction() {
            this.action = null;
            this.recipient = null;
            this.message = null;
        }

        // send
        public ClientAction(String recipient, String message) {
            this.action = "send";
            this.recipient = recipient;
            this.message = message;
        }

        // msisdn
        public ClientAction(String s) {
            this.action = "msisdn";
            this.recipient = null;
```

```java
        this.message = null;
    }

    // ping
    public ClientAction(char c) {
        this.action = "ping";
        this.recipient = null;
        this.message = null;
    }
}

/**
 * Copied from <code>com.transimsms.androidcompanion.ServerAction</code>.
 * This must be always up to date to the Transmisms Android Companion's
 * ServerAction class.
 */
public static class ServerAction {
    final public String action;

    // recv
    final public String sender;
    final public String message;
    final public Long timestamp;

    // error
    final public String code;
    final public String[] alternativecodes; // optional

    // pong
    // NOTHING FOLLOWS

    public ServerAction() {
        this.action = null;
        this.code = null;
        this.alternativecodes = null;
        this.sender = null;
        this.message = null;
        this.timestamp = null;
    }

    // recv
    public ServerAction(String sender,
            String message, long timestamp) {
        this.action = "recv";
        this.sender = sender;
        this.message = message;
        this.timestamp = timestamp;
        this.code = null;
        this.alternativecodes = null;
    }

    // error
    public ServerAction(String code, String[] alternativecodes) {
        this.action = "error";
        this.code = code;
        this.alternativecodes = alternativecodes;
        this.sender = null;
        this.message = null;
        this.timestamp = null;
    }

    // msisdn
    public ServerAction(String msisdn) {
        this.action = "msisdn";
        this.message = msisdn;
        // no fields
        this.code = null;
        this.alternativecodes = null;
        this.sender = null;
        this.timestamp = null;
    }

    // pong
    public ServerAction(char c) {
        this.action = "pong";
        // no fields
        this.code = null;
        this.alternativecodes = null;
        this.sender = null;
        this.message = null;
        this.timestamp = null;
    }
}



public final static int PING_TIMEOUT = 1000;

private final Gson gson = new Gson();
private final String zmqAddress;
private volatile boolean isStarted = false;
private String peerMin = null;
private String localMin = null;

private final Queue<String> commandQueue =
        new ConcurrentLinkedQueue<String>();
private final Queue<String> sendQueue =
        new ConcurrentLinkedQueue<String>();
private final BlockingQueue<SmsEntry> recvQueue =
        new LinkedBlockingQueue<SmsEntry>();
private final AtomicReference<SmsServiceException> currentException =
        new AtomicReference<>(null);

private CountDownLatch cdl = new CountDownLatch(1);



public AndroidSmsService(String host, int port) {
    this.zmqAddress = "tcp://" + host + ":" + port;
}


@Override
public String getLocalMin() {
    this.sendMsisdnRequest();
    if(this.cdl != null) {
        try {
            this.cdl.await();
        }
        catch(InterruptedException e) {} // do nothing on interruption
        this.cdl = null;
    }
    return this.localMin;
}

@Override
public void setPeer(String peerMin) {
    super.setPeer(peerMin);
    this.peerMin = peerMin;
}

@Override
public String getPeer() {
    return this.peerMin;
}

@Override
protected void sendMessageToBackend(String message) {
    if(this.isStarted && this.peerMin != null) {
        this.sendQueue.offer(message);
    }
}

@Override
public SmsEntry pollMessageFromBackend(long timeout, TimeUnit unit)
        throws SmsServiceException, InterruptedException {
    // throw exceptions generated from our polling loop, if any
    SmsServiceException e = this.currentException.get();
    if(e != null) {
        this.currentException.set(null); // clear exception
        throw e;
    }
    // for normal cases
    if(this.isStarted) {
        return this.recvQueue.poll(timeout, unit);
    }
    return null;
}

@Override
public boolean start() throws SmsServiceException {
    super.start();
    if(!this.isStarted) {
        final CountDownLatch cdl = new CountDownLatch(1);
        AtomicReference<SmsServiceException> ses = new AtomicReference<>();
        // start a separate thread to consume and produce entries
        // this also includes the closing procedures
        (new Thread() {
            @Override
            public void run() {
                // setup necessary connections
                final Context zmqContext = ZMQ.context(1);
                final Socket zmqSocket = zmqContext.socket(ZMQ.PAIR);
                // set linger to stop connection ASAP
                zmqSocket.setLinger(0);
                // connect or bind depending on the role
                zmqSocket.connect(AndroidSmsService.this.zmqAddress);

                AndroidSmsService.this.isStarted = false;
                // NOTE: all messages prior to 'starting' this service
                //       are DISCARDED
                // PING receiver and wait for PONG
String jsonStr = AndroidSmsService.this.gson.toJson(
        new ClientAction('p')); // PING ClientAction
zmqSocket.send(jsonStr);

// attempt getting a PONG response for LIMIT times
int LIMIT = 3;
for(int i = 0; i < LIMIT; i++) {
    Poller plr = zmqContext.poller();
    plr.register(zmqSocket, Poller.POLLIN);
    try {
        plr.poll(PING_TIMEOUT); // wait for PING_TIMEOUT ms
    }
    catch(ZError.IOException e) {
        System.out.println("got ZError.IOException " +
                "on connectivity test: " + e);
        ses.set(new SmsServiceException(
                Reason.IOERROR));
    }
    if(plr.pollin(0)) {
        String s = zmqSocket.recvStr(0);
        ServerAction sAction =
                AndroidSmsService.this.gson.fromJson(s,
                ServerAction.class);
        if(sAction != null && sAction.action != null &&
            sAction.action.equals("pong")) {
            plr.close(); // preemptive close before exiting
            AndroidSmsService.this.isStarted = true;
            break; // exit ASAP
        }
        // otherwise, discard received data and try again
        else {
            // 'infinite' retries/exhaustion of entries
            i--;
        }
    }
    plr.close(); // because we need more close()
}
if(!AndroidSmsService.this.isStarted &&
        ses.get() == null) {
    ses.set(new SmsServiceException(Reason.NOPONG));
}
cdl.countDown(); // ready for examining our isStarted flag

// loop until isStarted flag is revoked
while(AndroidSmsService.this.isStarted) {
    // send all queued commands
    for(String message =
            AndroidSmsService.this.commandQueue.poll();
            message != null;
            message =
            AndroidSmsService.this.commandQueue.poll()) {
        zmqSocket.send(message);
    }
    // send all pending messages
    for(String message =
            AndroidSmsService.this.sendQueue.poll();
            message != null; message =
            AndroidSmsService.this.sendQueue.poll()) {
        ClientAction cAction = new ClientAction(
                AndroidSmsService.this.peerMin, message);
        String str = AndroidSmsService.this.gson.toJson(
                cAction, ClientAction.class);
        zmqSocket.send(str);
    }
    // poll for the time being
    Poller plr = zmqContext.poller();
    plr.register(zmqSocket, Poller.POLLIN);
    try {
        plr.poll(100); // poll for 100ms
    }
    catch(ZError.IOException e) {
        System.out.println("got ZError.IOException: " + e);
        AndroidSmsService.this.stop(); // stop ASAP
        // then set currentException to be thrown on
        // pollMessage()/pollMessageFromBackend() later
        AndroidSmsService.this.currentException.set(
                new SmsServiceException(Reason.IOERROR));
        break;
    }
    if(plr.pollin(0)) {
        String str = zmqSocket.recvStr(0);
        ServerAction sAction =
                AndroidSmsService.this.gson.fromJson(str,
                ServerAction.class);
        // check for errors first
        if(sAction.action == null) {
            continue; // do nothing and loop back
        }
        if(sAction.action.equals("recv")) {
            AndroidSmsService.this.recvQueue.offer(
                    new SmsEntry(sAction.sender,
                    sAction.message,
                    new Date(sAction.timestamp)));
        }
        else if(sAction.action.equals("msisdn")) {
            AndroidSmsService.this.localMin =
                    sAction.message;
            if(AndroidSmsService.this.cdl != null) {
                AndroidSmsService.this.cdl.countDown();
            }
        }
        else if(sAction.action.equals("error")) {
        }
    }
    plr.close();
}

// finally, close everything afterwards
try {
    if(zmqSocket != null) {
        zmqSocket.close();
    }
}
catch(ZError.IOException e) {
    System.out.println("got ZError.IOException on " +
            "closing zmqSocket: " + e);
}
try {
    if(!zmqContext.isTerminated()) {
        zmqContext.term();
    }
}
catch(ZError.IOException e) {
    System.out.println("got ZError.IOException on " +
            "closing zmqContext: " + e);
}
// set other variables as well
AndroidSmsService.this.isStarted = false;
        }
    }).start();
    try {
        cdl.await();
    }
    catch(InterruptedException e) {
        return false;
    }
    if(!this.isStarted) {
        // do cleanup and throw exception
        this.sendQueue.clear();
        this.recvQueue.clear();
        throw ses.get();
    }
    return this.isStarted;
    }
    return false;
}

@Override
public boolean stop() {
    super.stop();
    if(this.isStarted) {
        this.isStarted = false;
        // clear sendQueue and recvQueue
        this.sendQueue.clear();
        this.recvQueue.clear();
    }
    return false;
}

@Override
public boolean isRunning() {
    return this.isStarted;
}

private void sendMsisdnRequest() {
    if(this.localMin == null) {
        ClientAction cAction = new ClientAction("msisdn");
        String str = this.gson.toJson(cAction, ClientAction.class);
        this.commandQueue.offer(str);
    }
}
}

------------------------------------------------------------
Filename:
src/main/java/com/transmisms/ui/javafx/smsservice/ControlPrioritySmsService.java
------------------------------------------------------------

package com.transmisms.ui.javafx.smsservice;
```

```java
import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.core.protocol.SmsService;
import com.transmisms.core.protocol.SmsServiceException;
import com.transmisms.core.util.commons.FIFOEntry;

import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.PriorityBlockingQueue;
import java.util.concurrent.TimeUnit;



public abstract class ControlPrioritySmsService implements SmsService {
    private static class StringWrapper implements Comparable<StringWrapper> {
        final public String value;

        public StringWrapper(String value) {
            this.value = value;
        }

        @Override
        public int compareTo(StringWrapper o) {
            // NOTE: this assumes that o will never be null
            return StringWrapper.getWeight(o.value) -
                    StringWrapper.getWeight(this.value);
        }

        @Override
        public boolean equals(Object o) {
            if(o == null || !(o instanceof StringWrapper)) {
                return false;
            }
            else {
                return 0 == this.compareTo((StringWrapper)o);
            }
        }

        public static int getWeight(String s) {
            // NOTE: this assumes that s will never be null
            // NOTE: this is based on static initialization of PDU class
            if(s.toLowerCase().startsWith("no")) {
                return 50;
            }
            else if(s.startsWith("?transmismsv")) {
                return 40;
            }
            else if(s.startsWith("f") || s.startsWith("F")) {
                return 40;
            }
            else if(s.startsWith("r")) {
                return 30;
            }
            else if(s.startsWith("a") || s.startsWith("b") ||
                    s.startsWith("c")) {
                return 20;
            }
            else if(s.startsWith("d") || s.startsWith("i") ||
                    s.startsWith("j") || s.startsWith("e") ||
                    s.startsWith("E")) {
                return 10;
            }
            return 0;
        }
    }



    final private int MAX_BUFFER_SIZE = 5;

    final private int loopTimeout;
    final private PriorityBlockingQueue<FIFOEntry<StringWrapper>> pbq =
            new PriorityBlockingQueue<>();
    final private Queue<String> lpq = new ConcurrentLinkedQueue<>();
    protected boolean isStarted = false;
    private Thread sendLoopThread = null;



    protected abstract void sendMessageToBackend(String message);

    public abstract String getLocalMin();

    public abstract void setPeer(String peerMin);

    public abstract String getPeer();



    public ControlPrioritySmsService(int loopTimeout) {
        this.loopTimeout = loopTimeout;
    }

    public ControlPrioritySmsService() {
        this.loopTimeout = 5000;
    }



    @Override
    public final void sendMessage(String message) {
        if(this.isStarted) {
            if(StringWrapper.getWeight(message) > 10) {
                this.pbq.offer(new FIFOEntry(new StringWrapper(message)));
            }
            else {
                this.lpq.offer(message);
            }
        }
    }

    @Override
    public synchronized boolean start() throws SmsServiceException {
        if(!this.isStarted) {
            this.isStarted = true;
            // send message on a loop on a separate thread
            this.sendLoopThread = new Thread(() -> {
                while(this.isStarted) {
                    try {
                        FIFOEntry<StringWrapper> entry = this.pbq.poll(
                                this.loopTimeout, TimeUnit.MILLISECONDS);
                        if(entry != null) {
                            String message = entry.getEntry().value;
                            this.sendMessageToBackend(message);
                            for(int i = 0; i < this.MAX_BUFFER_SIZE-1; i++) {
                                entry = this.pbq.poll();
                                // break when we run out of entries
                                if(entry == null) {
                                    break;
                                }
                                message = entry.getEntry().value;
                                this.sendMessageToBackend(message);
                            }
                        }
                        else {
                            String message = this.lpq.poll();
                            if(message != null) {
                                this.sendMessageToBackend(message);
                            }
                        }
                    }
                    catch(InterruptedException e) {
                        break;
                    }
                }
            });
            this.sendLoopThread.setDaemon(true);
            this.sendLoopThread.start();
            return true;
        }
        return false;
    }

    @Override
    public synchronized boolean stop() {
        if(this.isStarted) {
            this.isStarted = false;
            this.sendLoopThread.interrupt();
            this.pbq.clear();
            return true;
        }
        return false;
    }

    @Override
    public abstract SmsEntry pollMessage(long timeout, TimeUnit unit)
            throws SmsServiceException, InterruptedException;

    @Override
    public abstract boolean isRunning();
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/smsservice/DummySmsService.java
--------------------------------------------------------------

package com.transmisms.ui.javafx.smsservice;

import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.core.protocol.SmsService;

import org.zeromq.ZMQ;
import org.zeromq.ZMQ.Context;
import org.zeromq.ZMQ.Poller;
import org.zeromq.ZMQ.Socket;

import java.util.Date;
import java.util.Queue;
import java.util.concurrent.BlockingQueue;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.atomic.AtomicReference;

import com.transmisms.core.protocol.SmsServiceException;
import zmq.ZError;



public class DummySmsService extends ExtendedSmsService {
    public enum Reason implements SmsServiceException.Reason<Reason>,
           ExtendedSmsService.ExtendedReason {
        IOERROR ("Problem connecting to Dummy device",
          "Please check device connectivity and try again."),
        NOPONG ("Dummy device is not responding",
          "Please check the device and try again.");



        private final String message;
        private final String details;


        // default access control since enums are restricted
        Reason(String message, String details) {
            this.message = message;
            this.details = details;
        }


        @Override
        public Reason valueOf() {
            return valueOf(this.name());
        }

        @Override
        public final String getMessage() {
            return this.message;
        }
        @Override
        public final String getDetails() {
            return this.details;
        }
    }



    public final static int PING_TIMEOUT = 1000;

    private final String zmqAddress;
    private final boolean isSender;
    private volatile boolean isStarted = false;
    private String peerMin = null;
    private String localMin = null;

    private final Queue<String> sendQueue =
            new ConcurrentLinkedQueue<String>();
    private final BlockingQueue<SmsEntry> recvQueue =
            new LinkedBlockingQueue<SmsEntry>();
    private final AtomicReference<SmsServiceException> currentException =
            new AtomicReference<>(null);



    public DummySmsService(String host, int port, String localMin,
            boolean isSender) {
        super(100); // set loopTimeout as 100ms
        this.zmqAddress = "tcp://" + host + ":" + port;
        this.localMin = localMin;
        this.isSender = isSender;
    }



    @Override
    public String getLocalMin() {
        return this.localMin;
    }
```

```
    @Override
    public void setPeer(String peerMin) {
        super.setPeer(peerMin);
        this.peerMin = peerMin;
    }

    @Override
    public String getPeer() {
        return this.peerMin;
    }

    @Override
    protected void sendMessageToBackend(String message) {
        if(this.isStarted && this.peerMin != null) {
            this.sendQueue.offer(this.getLocalMin() + ":" + message);
        }
    }

    @Override
    public SmsEntry pollMessageFromBackend(long timeout, TimeUnit unit)
            throws SmsServiceException, InterruptedException {
        // throw exceptions generated from our polling loop, if any
        SmsServiceException e = this.currentException.get();
        if(e != null) {
            this.currentException.set(null); // clear exception
            throw e;
        }
        // for normal cases
        if(this.isStarted) {
            return this.recvQueue.poll(timeout, unit);
        }
        return null; // return null if not yet started or on failures
    }

    @Override
    public boolean start() throws SmsServiceException {
        super.start();
        if(!this.isStarted) {
            final CountDownLatch cdl = new CountDownLatch(1);
            AtomicReference<SmsServiceException> ses = new AtomicReference<>(
                    new SmsServiceException(Reason.NOPONG));
            // start a separate thread to consume and produce entries
            // this also includes the closing procedures
            (new Thread() {
                @Override
                public void run() {
                    // setup necessary connections
                    final Context zmqContext = ZMQ.context(1);
                    final Socket zmqSocket = zmqContext.socket(ZMQ.PAIR);
                    // set linger to stop connection ASAP
                    zmqSocket.setLinger(0);
                    // connect or bind depending on the role
                    if(DummySmsService.this.isSender) {
                        zmqSocket.connect(DummySmsService.this.zmqAddress);
                    }
                    else {
                        zmqSocket.bind(DummySmsService.this.zmqAddress);
                    }

                    DummySmsService.this.isStarted = false;
                    // NOTE: all messages prior to 'starting' this service
                    //       are DISCARDED
                    if(DummySmsService.this.isSender) {
                        // PING receiver and wait for PONG
                        zmqSocket.send("PING");

                        // attempt getting a PONG response for LIMIT times
                        int LIMIT = 3;
                        for(int i = 0; i < LIMIT; i++) {
                            Poller plr = zmqContext.poller();
                            plr.register(zmqSocket, Poller.POLLIN);
                            try {
                                plr.poll(PING_TIMEOUT); // wait for
PING_TIMEOUT ms
                            }
                            catch(ZError.IOException e) {
                                System.out.println("got ZError.IOException " +
                                        "on connectivity test: " + e);
                                ses.set(new SmsServiceException(
                                        Reason.IOERROR));
                            }
                            if(plr.pollin(0)) {
                                String raw = zmqSocket.recvStr(0);
                                if(raw.equals("PONG")) {
                                    plr.close(); // preemptive close before
exiting
                                    DummySmsService.this.isStarted = true;
                                    break; // exit ASAP
                                }
                                // otherwise, discard received data and try
again
```

```
                    }
                    plr.close(); // because we need more close()
                }
            }
            cdl.countDown(); // ready for examining our isStarted flag

            // loop until isStarted flag is revoked
            while(DummySmsService.this.isStarted) {
                // send all pending messages
                for(String message =
                        DummySmsService.this.sendQueue.poll();
                        message != null; message =
                        DummySmsService.this.sendQueue.poll()) {
                    zmqSocket.send(message);
                }
                // poll for the time being
                Poller plr = zmqContext.poller();
                plr.register(zmqSocket, Poller.POLLIN);
                try {
                    plr.poll(100); // poll for 100ms
                }
                catch(ZError.IOException e) {
                    System.out.println("got ZError.IOException: " + e);
                    DummySmsService.this.stop(); // stop ASAP
                    // then set currentException to be thrown on
                    // pollMessage()/pollMessageFromBackend() later
                    DummySmsService.this.currentException.set(
                            new SmsServiceException(Reason.IOERROR));
                    break;
                }
                if(plr.pollin(0)) {
                    // process PING/PONG requests first
                    String raw = zmqSocket.recvStr(0);
                    if(raw.equals("PING")) { // reply with PONG
                        zmqSocket.send("PONG");
                        continue;
                    }
                    else if(raw.equals("PONG")) {
                        // just ignore and try again
                        continue;
                    }
                    String[] parts = raw.split(":", 2);
                    String sender = parts[0];
                    String msg = parts[1];
                    plr.close();
                    DummySmsService.this.recvQueue.offer(
                            new SmsEntry(sender, msg, new Date()));
                }
                plr.close();
            }

            // finally, close everything afterwards
            try {
                if(zmqSocket != null) {
                    zmqSocket.close();
                }
            }
            catch(ZError.IOException e) {
                System.out.println("got ZError.IOException on " +
                        "closing zmqSocket: " + e);
            }
            try {
                if(!zmqContext.isTerminated()) {
                    zmqContext.term();
                }
            }
            catch(ZError.IOException e) {
                System.out.println("got ZError.IOException on " +
                        "closing zmqContext: " + e);
            }
            // set other variables as well
            DummySmsService.this.isStarted = false;
        }
    }).start();
    try {
        cdl.await();
    }
    catch(InterruptedException e) {
        return false;
    }
    if(!this.isStarted) {
        // do cleanup and throw exception
        this.sendQueue.clear();
        this.recvQueue.clear();
        throw ses.get();
    }
    return this.isStarted;
}
return false;
}

    @Override
    public boolean stop() {
        super.stop();
        if(this.isStarted) {
            this.isStarted = false;
            // clear sendQueue and recvQueue
            this.sendQueue.clear();
            this.recvQueue.clear();
        }
        return false;
    }

    @Override
    public boolean isRunning() {
        return this.isStarted;
    }
}
```

```
--------------------------------------------------------------
Filename:
src/main/java/com/transmisms/ui/javafx/smsservice/ExtendedSmsService.java
--------------------------------------------------------------

package com.transmisms.ui.javafx.smsservice;

import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.ui.javafx.Utils;

import java.util.concurrent.TimeUnit;

import com.transmisms.core.protocol.SmsServiceException;


public abstract class ExtendedSmsService extends ControlPrioritySmsService {
    public interface ExtendedReason {
        public String getMessage();
        public String getDetails();
    }

    private String countryCode = null;
    private String normalizedPeer = null;


    public ExtendedSmsService() {}

    public ExtendedSmsService(int loopTimeout) {
        super(loopTimeout);
    }



    public final SmsEntry pollMessage(long timeout, TimeUnit unit)
            throws SmsServiceException, InterruptedException {
        SmsEntry entry = this.pollMessageFromBackend(timeout, unit);
        // set as the peer if there's still no peer set previously
        if(entry != null && this.getPeer() == null) {
            if(this.countryCode == null) {
                this.setPeer(entry.getSender());
            }
            else {
                this.setPeer(Utils.getIntFormat(entry.getSender(),
                        this.countryCode));
            }
        }
        if(entry != null && this.getPeer() != null) {
            // return null on non-peer senders
            if(!entry.getSender().endsWith(this.normalizedPeer)) {
                return null;
            }
        }
        return entry;
    }

    public abstract SmsEntry pollMessageFromBackend(long timeout,
            TimeUnit unit)
            throws SmsServiceException, InterruptedException;

    public void setCountryCode(String countryCode) {
        this.countryCode = countryCode;
    }

    public void setPeer(String peer) {
        if(peer != null) {
            this.normalizedPeer = Utils.normalizeMsisdn(peer,
                    this.countryCode);
        }
        else {
            this.normalizedPeer = null;
        }
    }
}



---------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/smsservice/GammuSmsService.java
---------------------------------------------------------------

package com.transmisms.ui.javafx.smsservice;

import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.core.protocol.SmsService;

import java.sql.Connection;
import java.sql.Timestamp;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Future;
import java.util.concurrent.TimeUnit;

import com.transmisms.core.protocol.SmsServiceException;
import java.sql.SQLException;
import java.util.concurrent.CancellationException;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.TimeoutException;



public class GammuSmsService extends ExtendedSmsService {
    public enum Reason implements SmsServiceException.Reason<Reason>,
           ExtendedSmsService.ExtendedReason {
        NOCONN ("Problem connecting to database backend",
          "Please check database connectivity and try again."),
        NODEVICE ("Device not found",
          "Please check the device and try again.");



        private final String message;
        private final String details;


        // default access control since enums are restricted
        Reason(String message, String details) {
            this.message = message;
            this.details = details;
        }


        @Override
        public Reason valueOf() {
            return valueOf(this.name());
        }

        @Override
        public final String getMessage() {
            return this.message;
        }
        @Override
        public final String getDetails() {
            return this.details;
        }
    }




    private static final String messageQueryOrderStr =
            " ORDER BY \"UpdatedInDB\", \"ID\" ASC";
    // NOTE: This filtering only works on mobile-terminated numbers
    //       (i.e. phones). This might be good on CPs with customized suffix
    //       numbers, but not on the rest.
    private static final String messageQueryFilteredStr =
            "SELECT * FROM inbox WHERE \"ID\" > ? AND " +
            "\"UpdatedInDB\" > ? AND \"SenderNumber\" LIKE ?" +
            messageQueryOrderStr;
    private static final String messageQueryUnfilteredStr =
            "SELECT * FROM inbox WHERE \"ID\" > ? AND " +
            "\"UpdatedInDB\" > ?" + messageQueryOrderStr;
    private static final String statusQueryStr =
            "SELECT * FROM phones WHERE \"TimeOut\" > " +
            "(CURRENT_TIMESTAMP - INTERVAL '2 MINUTE')";
    private static final String insertQueryStr =
            "INSERT INTO outbox (\"TextDecoded\", \"DestinationNumber\", " +
            "\"CreatorID\", \"Priority\") VALUES (?, ?, ?, ?)";
    private static final String getCurtimestampQueryStr =
            "SELECT CURRENT_TIMESTAMP AT TIME ZONE current_setting('TimeZone')";
    public final static int PING_TIMEOUT = 3000;

    private boolean isStarted = false;
    private Timestamp startTime;
    private String peerMin = null;
    private int lastMessageId = -1;
    private final String dbHost;
    private final int dbPort;
    private final String dbName;
    private final String dbUser;
    private final String dbPassword;
    private final String localMin;

    private Connection dbConnection = null;
    private PreparedStatement messageQuery = null;
    private PreparedStatement statusQuery = null;
    private PreparedStatement insertQuery = null;
    private PreparedStatement getCurtimestampQuery = null;

    private final Queue<SmsEntry> entryQueue = new ConcurrentLinkedQueue<>();
    private final ExecutorService executor =
            Executors.newSingleThreadExecutor();
    private Future<?> queryFuture = null;



    public GammuSmsService(String dbHost, int dbPort, String dbName,
            String dbUser, String dbPassword, String localMin) {
        super(3000); // set loopTimeout as 3000ms
        this.dbHost = dbHost;
        this.dbPort = dbPort;
        this.dbName = dbName;
        this.dbUser = dbUser;
        this.dbPassword = dbPassword;
        this.localMin = localMin;
    }



    public String getLocalMin() {
        return this.localMin;
    }

    @Override
    public void setPeer(String peerMin) {
        super.setPeer(peerMin);
        this.peerMin = peerMin;
        try {
            if(peerMin != null) {
                this.messageQuery = this.dbConnection.prepareStatement(
                        GammuSmsService.messageQueryFilteredStr);
            }
            else {
                this.messageQuery = this.dbConnection.prepareStatement(
                        GammuSmsService.messageQueryUnfilteredStr);
            }
        }
        catch(SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public String getPeer() {
        return this.peerMin;
    }

    @Override
    protected synchronized void sendMessageToBackend(String message) {
        if(this.isStarted && this.getPeer() != null) {
            try {
                this.insertQuery.setString(1, message);
                this.insertQuery.setString(2, this.getPeer());
                // NOTE: 'default' is used only on single-modem setups. This
                //       cannot be used on Gammu databases for use with
                //       multiple modems
                this.insertQuery.setString(3, "default");
                this.insertQuery.setInt(4,
                        ControlPrioritySmsService.StringWrapper.getWeight(
                        message));
                // we won't get the return value of the statement below
                this.insertQuery.executeUpdate();
            }
            catch(SQLException e) {
```

```
            e.printStackTrace();
            return;
        }
    }
}

@Override
public SmsEntry pollMessageFromBackend(long timeout, TimeUnit unit)
        throws SmsServiceException, InterruptedException {
    if(this.isStarted) {
        SmsEntry entry = null;
        // try to get a pending entry first
        entry = this.entryQueue.poll();
        if(entry != null) {
            return entry;
        }

        // create a separate task for querying
        this.queryFuture = this.executor.submit(() -> {
            while(!Thread.interrupted()) {
                ResultSet rs = null;
                boolean hasResults = false;
                String sender = null;
                String message = null;
                Timestamp timestamp = null;
                int messageId = -1;
                try {
                    this.messageQuery.setInt(1, this.lastMessageId);
                    this.messageQuery.setTimestamp(2, this.startTime);
                    if(this.getPeer() != null) {
                        this.messageQuery.setString(3, "%"+this.getPeer());
                    }

                    rs = this.messageQuery.executeQuery();
                    hasResults = rs.next();
                    if(hasResults) {
                        sender = rs.getString("SenderNumber");
                        message = rs.getString("TextDecoded");
                        timestamp = rs.getTimestamp("UpdatedInDB");
                        messageId = rs.getInt("ID");
                        // update lastMessageId for next queries
                        this.lastMessageId = messageId;
                    }
                }
                catch(SQLException e) {
                    e.printStackTrace();
                    break; // exit on errors
                }
                finally {
                    // clean up first
                    try {
                        if(rs != null) {
                            rs.close();
                        }
                    }
                    catch(SQLException e) {
                        e.printStackTrace();
                        break; // exit on errors
                    }

                    // return if there are results, loop again otherwise
                    if(hasResults) {
                        // then return a single entry
                        this.entryQueue.offer(new SmsEntry(
                                    sender, message, timestamp));
                        return null;
                    }
                    else {
                        try { // just sleep for a while and try again
                            Thread.sleep(1000); // sleep for a second
                        }
                        catch(InterruptedException e) {
                            break; // exit on break
                        }
                    }
                }
            }
            return null; // just return null on timeout/failure
        });

        try {
            // execute and wait for results until timeout
            this.queryFuture.get(timeout, unit);
            entry = this.entryQueue.poll();
        }
        catch(ExecutionException e) {
            if(e != null && e.getCause() != null) {
                e.getCause().printStackTrace();
            }
        }
        catch(TimeoutException e) {
            if(e != null && e.getCause() != null) {
                e.getCause().printStackTrace();
            }
        }
        catch(CancellationException e) {
            if(e != null && e.getCause() != null) {
                e.getCause().printStackTrace();
            }
        }
        finally {
            this.queryFuture = null;
        }
        return entry;
    }
    return null; // just return null otherwise (or on timeout/error)
}

@Override
public synchronized boolean start() throws SmsServiceException {
    super.start();
    if(!this.isStarted) {
        // start the connection
        String jdbcUrl = "jdbc:postgresql://" + this.dbHost + ":"
                + this.dbPort + "/" + this.dbName;
        ResultSet rs = null;
        try {
            // prepare connection
            this.dbConnection = DriverManager.getConnection(
                    jdbcUrl, this.dbUser, this.dbPassword);
            // prepare PreparedStatements
            this.messageQuery = this.dbConnection.prepareStatement(
                    GammuSmsService.messageQueryUnfilteredStr);
            this.statusQuery = this.dbConnection.prepareStatement(
                    GammuSmsService.statusQueryStr);
            this.insertQuery = this.dbConnection.prepareStatement(
                    GammuSmsService.insertQueryStr);
            this.getCurtimestampQuery = this.dbConnection.prepareStatement(
                    GammuSmsService.getCurtimestampQueryStr);

            // get CURRENT_TIMESTAMP and use it for querying messages
            rs = this.getCurtimestampQuery.executeQuery();
            if(rs.next()) {
                this.startTime = rs.getTimestamp(1);
            }

            // attempt checking for connected 'phones' within the
            // last 2 minutes for LIMIT times
            int LIMIT = 3;
            for(int i = 0; i < LIMIT; i++) {
                rs = this.statusQuery.executeQuery();
                if(rs.next()) { // got a 'phone'
                    this.isStarted = true;
                    break;
                }
                // sleep for a while before retrying
                try {
                    Thread.sleep(PING_TIMEOUT);
                }
                catch(InterruptedException e) {
                    break;
                }
            }
            if(!this.isStarted) {
                this.stop(); // stop the service ASAP
                throw new SmsServiceException(Reason.NODEVICE);
            }
        }
        catch(SQLException e) {
            this.stop(); // stop the service ASAP
            throw new SmsServiceException(Reason.NOCONN);
        }
        finally { // try to close everything
            try {
                if(rs != null) {
                    rs.close();
                }
            }
            catch(SQLException e) {
                e.printStackTrace();
            }

            // finally, set the isStarted flag to true
        }
        return this.isStarted;
    }
    else {
        return false;
    }
}

@Override
```

```
    public boolean stop() {
        super.stop();
        if(this.isStarted) {
            // stop the execution of current queries
            if(this.queryFuture != null) {
                this.queryFuture.cancel(true);
            }
            try { // close the connection and other related objects
                if(this.dbConnection != null) {
                    this.dbConnection.close();
                }
                if(this.messageQuery != null) {
                    this.messageQuery.close();
                }
                if(this.statusQuery != null) {
                    this.statusQuery.close();
                }
                if(this.insertQuery != null) {
                    this.insertQuery.close();
                }
            }
            catch(SQLException e) {
                e.printStackTrace();
            }
            finally {
                // then let JVM GC the remnants
                this.dbConnection = null;
                this.messageQuery = null;
                this.statusQuery = null;
                this.insertQuery = null;
                this.getCurtimestampQuery = null;
                // finally, set the isStarted flag to false
                this.isStarted = true;
            }
            return true;
        }
        else {
            return false;
        }
    }

    @Override
    public boolean isRunning() {
        return this.isStarted;
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/StandardDialog.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;

import javafx.scene.Parent;
import javafx.scene.control.ButtonType;
import javafx.scene.control.Dialog;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.text.Text;
import javafx.stage.StageStyle;

import javafx.fxml.FXMLLoader;

import java.io.IOException;



public class StandardDialog extends Dialog<ButtonType> {
    private final Label primaryLabel;
    private final Text secondaryLabel;



    public StandardDialog(String primaryText, String secondaryText,
            Image img) {
        // prepare and set the header
        FXMLLoader dialogTemplateLoader = new FXMLLoader(
                MainWindowController.class.getClassLoader()
                .getResource("dialog-template.fxml"));
        Parent header = null;
        try {
            header = (Parent)(dialogTemplateLoader.load());
        }
        catch(IOException e) {
            //mainLogger.error(
            System.out.println(
                "Caught IOException when loading dialog header templates");
                e.printStackTrace();
        }
        // set css for the parent
        header.getStylesheets().add("dialog.css");
        // set icon for dialog
        ImageView iv = (ImageView)header.lookup(".dialogIcon");
        iv.setImage(img);
        // set necessary text properties
        this.primaryLabel = (Label)header.lookup(".dialogPrimaryLabel");
        this.secondaryLabel = (Text)header.lookup(".dialogSecondaryLabel");
        primaryLabel.setText(primaryText);
        secondaryLabel.setText(secondaryText);
        // add the header to dialog pane
        this.getDialogPane().setHeader(header);

        // misc modifications for the dialog
        this.initStyle(StageStyle.UTILITY);
    }


    public void setPrimaryText(String text) {
        this.primaryLabel.setText(text);
    }

    public void setSecondaryText(String text) {
        this.secondaryLabel.setText(text);
    }

    public void appendSecondaryText(String text) {
        this.secondaryLabel.setText(this.secondaryLabel.getText() + text);
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/ui/javafx/Utils.java
--------------------------------------------------------------

package com.transmisms.ui.javafx;



public class Utils {
    public static final int MIN_MIN_LENGTH = 10;

    /**
     * Normalizes MSISDN and other numbers into 10-digit MINs
     *
     * @param msisdn number to be normalized into 10-digit MINs
     */
    public static String normalizeMsisdn(String msisdn, String countryCode) {
        // if carrier-provided access code, return ASAP
        if(msisdn == null || msisdn.length() < MIN_MIN_LENGTH) {
            return msisdn;
        }

        String min = msisdn;
        // remove leading '+', if any
        if(min.startsWith("+")) {
            min = min.substring(1);
        }
        // remove leading zeroes
        min = min.replaceFirst("^0+(?!$)", "");
        if(min.startsWith(countryCode)) {
            min = min.substring(countryCode.length());
        }
        return min;
    }

    /**
     * Converts mobile numbers into international formats
     */
    public static String getIntFormat(String min, String countryCode) {
        // if carrier-provided access code, return ASAP
        if(min == null || min.length() < MIN_MIN_LENGTH) {
            return min;
        }

        String intNumber = min;
        // try to normalize first
        intNumber = Utils.normalizeMsisdn(min, countryCode);
        // then append the country code
        intNumber = "+"+countryCode+intNumber;

        return intNumber;
    }
}
```

```
--------------------------------------------------------------
Filename: src/main/resources/contact-edit-window.css
--------------------------------------------------------------

#contactEditErrorText, #contactExistsErrorText {
    -fx-text-fill: red;
}

#pairingSettingsLabel {
    -fx-font-weight: bold;
}
```

```
--------------------------------------------------------------
Filename: src/main/resources/contact-edit-window.fxml
--------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<?import com.transmisms.ui.javafx.MsisdnStringConverter?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<HBox fx:controller="com.transmisms.ui.javafx.ContactEditController"
        xmlns:fx="http://javafx.com/fxml"
        alignment="CENTER">
    <padding><Insets top="12" right="12" bottom="12" left="12" /></padding>

    <VBox spacing="24">
        <GridPane fx:id="contactEditGrid" hgap="12" vgap="12">
            <gridLinesVisible>false</gridLinesVisible>

            <Label text="Name:"
                    GridPane.halignment="RIGHT"
                    GridPane.rowIndex="0" GridPane.columnIndex="0" />
            <TextField fx:id="contactName"
                    GridPane.columnSpan="1"
                    GridPane.rowIndex="0" GridPane.columnIndex="1" />
            <Label text="Mobile Number:"
                    GridPane.halignment="RIGHT"
                    GridPane.rowIndex="1" GridPane.columnIndex="0" />
            <TextField fx:id="contactNumber"
                    disable="false"
                    GridPane.columnSpan="1"
                    GridPane.rowIndex="1" GridPane.columnIndex="1">
                <textFormatter><TextFormatter>
                    <valueConverter>
                        <MsisdnStringConverter />
                    </valueConverter>
                </TextFormatter></textFormatter>
            </TextField>
            <Label fx:id="contactExistsErrorText"
                    GridPane.columnSpan="2"
                    GridPane.rowIndex="2" GridPane.columnIndex="0"
                    visible="false"
                    text="Mobile number already exists" />
            <Label fx:id="contactEditErrorText"
                    GridPane.columnSpan="2"
                    GridPane.rowIndex="2" GridPane.columnIndex="0"
                    visible="true"
                    text="Invalid/Empty mobile number" />
            <Separator
                    GridPane.columnSpan="2"
                    GridPane.rowIndex="3" GridPane.columnIndex="0" />
            <Label fx:id="pairingSettingsLabel"
                    text="Pairing"
                    GridPane.columnSpan="2"
                    GridPane.rowIndex="4" GridPane.columnIndex="0" />
            <HBox alignment="CENTER"
                    GridPane.columnSpan="2"
                    GridPane.rowIndex="5" GridPane.columnIndex="0">
                <Button text="Pair/Renew pairing" fx:id="pairButton"
                        onAction="#handlePairButtonAction"
                        disable="true" />
                <Button text="Unpair" fx:id="unpairButton"
                        onAction="#handleUnpairButtonAction"
                        disable="true" />
                <Button text="Import keys" fx:id="importButton"
                        disable="true" />
                <Button text="Export keys" fx:id="exportButton"
                        disable="true" />
            </HBox>
            <Separator
                    GridPane.columnSpan="2"
                    GridPane.rowIndex="6" GridPane.columnIndex="0" />
            <HBox alignment="CENTER"
                    GridPane.columnSpan="2"
                    GridPane.rowIndex="7" GridPane.columnIndex="0">
                <Button text="Save" fx:id="saveButton"
                        onAction="#handleSaveButtonAction" />
            </HBox>
        </GridPane>
    </VBox>
</HBox>
```

```
--------------------------------------------------------------
Filename: src/main/resources/contact-listcell-template.fxml
--------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<?import org.kordamp.ikonli.javafx.FontIcon?>


<VBox xmlns:fx="http://javafx.com/fxml">
    <BorderPane>
        <center>
            <Label styleClass="contactName" maxWidth="Infinity"
                    text="" />
        </center>
        <right>
            <Label styleClass="contactStatus" maxHeight="Infinity"
                    visible="false"
                    text="" />
        </right>
    </BorderPane>
    <Label styleClass="contactNum" text=""/>
    <BorderPane>
        <center>
            <ProgressBar styleClass="operationProgress" maxWidth="Infinity"
                    visible="false"
                    progress="-1" />
        </center>
        <right>
            <Button visible="false" styleClass="cancelOperationButton"
                    onAction="#handleCancelButtonAction">
                <graphic>
                    <FontIcon />
                </graphic>
            </Button>
        </right>
    </BorderPane>
</VBox>
```

```
--------------------------------------------------------------
Filename: src/main/resources/dialog-fingerprint-template.fxml
--------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.image.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>


<BorderPane
        maxWidth="480" prefWidth="480"
        xmlns:fx="http://javafx.com/fxml">

    <padding><Insets top="24" right="12" bottom="0" left="12" /></padding>

    <left>
        <ImageView styleClass="dialogIcon">
            <BorderPane.margin>
                <Insets top="0" right="12" bottom="0" left="0" />
            </BorderPane.margin>
        </ImageView>
    </left>
    <center>
        <VBox>
            <Label VBox.vgrow="ALWAYS" styleClass="dialogPrimaryLabel" />
            <Text styleClass="dialogSecondaryLabel" wrappingWidth="420" />
            <Text VBox.vgrow="ALWAYS" styleClass="dialogEmphasisLabel"
                    wrappingWidth="420" />
        </VBox>
    </center>
</BorderPane>
```

```
--------------------------------------------------------------
Filename: src/main/resources/dialog-template.fxml
--------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.image.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>



<BorderPane
        maxWidth="360" prefWidth="360"
        xmlns:fx="http://javafx.com/fxml">

    <padding><Insets top="24" right="12" bottom="0" left="12" /></padding>

    <left>
        <ImageView styleClass="dialogIcon">
            <BorderPane.margin>
                <Insets top="0" right="12" bottom="0" left="0" />
            </BorderPane.margin>
        </ImageView>
    </left>
    <center>
        <VBox>
            <Label VBox.vgrow="ALWAYS" styleClass="dialogPrimaryLabel" />
            <Text styleClass="dialogSecondaryLabel" wrappingWidth="300" />
        </VBox>
    </center>
</BorderPane>
```

```
--------------------------------------------------------------
Filename: src/main/resources/dialog.css
--------------------------------------------------------------

.dialogPrimaryLabel {
    -fx-wrap-text: true;
    -fx-font-size: 14px;
    -fx-font-weight: bold;
    -fx-padding: 0 0 8px 0;
}

.dialogSecondaryLabel {
    -fx-wrap-text: true;
    -fx-font-size: 12px;
}

.dialogEmphasisLabel {
    -fx-wrap-text: true;
    -fx-font-size: 18px;
    -fx-text-alignment: center;
}
```

```
--------------------------------------------------------------
Filename: src/main/resources/fxml-main.fxml
--------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.collections.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<?import org.kordamp.ikonli.javafx.FontIcon?>
<?import org.kordamp.ikonli.javafx.StackedFontIcon?>


<BorderPane fx:controller="com.transmisms.ui.javafx.MainWindowController"
        xmlns:fx="http://javafx.com/fxml">
    <fx:define>
        <FXCollections fx:factory="observableArrayList"
                fx:id="contactOList" />
        <java.lang.String fx:id="startRcvBtnStr"
                fx:value="Start accepting" />
        <java.lang.String fx:id="stopRcvBtnStr"
                fx:value="Stop accepting" />
        <java.lang.String fx:id="startRcvLblStr"
                fx:value="Not accepting connection requests" />
        <java.lang.String fx:id="stopRcvLblStr"
                fx:value="Accepting new connections" />
    </fx:define>

    <padding><Insets top="0" right="0" bottom="0" left="0" /></padding>

    <top>
        <VBox>
            <ToolBar>
                <Button fx:id="addContactButton" styleClass="first"
                        onAction="#handleAddContactButtonAction">
                    <graphic><FontIcon /></graphic>
                    <tooltip><Tooltip text="Add contact" /></tooltip>
                </Button>
                <Button fx:id="editContactButton"
                        onAction="#handleEditContactButtonAction">
                    <graphic><FontIcon /></graphic>
                    <tooltip><Tooltip text="Edit contact" /></tooltip>
                </Button>
                <Button fx:id="removeContactButton"
                        onAction="#handleRemoveContactButtonAction">
                    <graphic><FontIcon /></graphic>
                    <tooltip><Tooltip text="Delete contact" /></tooltip>
                </Button>
                <Separator orientation="vertical" />
                <Button fx:id="sendFileButton"
                        onAction="#handleSendFileButton">
                    <graphic><FontIcon /></graphic>
                    <tooltip><Tooltip text="Send file securely" /></tooltip>
                </Button>
                <Button fx:id="sendFileInsecurelyButton"
                        onAction="#handleSendFileInsecurelyButton">
                    <graphic><FontIcon /></graphic>
                    <tooltip><Tooltip text="Send file unsecurely" /></tooltip>
                </Button>
                <Region HBox.hgrow="ALWAYS" />
                <Button fx:id="appMenuButton" styleClass="last"
                        onAction="#handleAppMenuButtonAction">
                    <graphic><FontIcon /></graphic>
                    <tooltip><Tooltip text="More..." /></tooltip>
                </Button>
            </ToolBar>
            <BorderPane fx:id="acceptConnInfobar"
                    styleClass="infobar,infoInfobar"
                    visible="true" managed="true">
                <center>
                    <Label styleClass="infobarLabel" maxWidth="Infinity"
                            text="Accepting new connections" />
                </center>
                <right>
                    <HBox styleClass="infobuttonContainer">
                        <Button fx:id="toggleAcceptButton"
                                text="Stop accepting"
                                onAction="#handleToggleAcceptButtonAction" />
                    </HBox>
                </right>
            </BorderPane>
            <BorderPane fx:id="errorInfobar" styleClass="infobar"
                    visible="false" managed="false">
                <center>
                    <Label styleClass="infobarLabel" maxWidth="Infinity"
                            text="Encountered problems with Sms Service" />
                </center>
                <right>
                    <HBox styleClass="infobuttonContainer">
                        <Button fx:id="showErrorDetailsButton" text="Details"
                            onAction="#handleShowErrorDetailsButtonAction"
                            />
                    </HBox>
                </right>
            </BorderPane>
        </VBox>
    </top>

    <center>
        <SplitPane orientation="vertical">
```

```
            <StackPane>
                <fx:include fx:id="welcomePane" source="welcome-screen.fxml" />
                <ListView fx:id="contactList" />
            </StackPane>

            <TextArea fx:id="logTarget"
                    editable="false"
                    visible="false"
                    minHeight="-Infinity"
                    maxHeight="-Infinity"
                    prefHeight="-Infinity"
                    />
        </SplitPane>
    </center>

    <bottom>
        <HBox spacing="10" alignment="center_right">
            <Button fx:id="showHideButton" text="Show logs"
                    onAction="#handleShowHideButtonAction" />
        </HBox>
    </bottom>
</BorderPane>
```

```
--------------------------------------------------------------
Filename: src/main/resources/log4j2.yaml
--------------------------------------------------------------

Configuration:
  status: warn
  packages: "com.transmisms.ui.javafx"

  Appenders:
    Console:
      name: Console
      target: SYSTEM_OUT
      PatternLayout:
        Pattern: "%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level %logger{36} - %msg%n"
    QueueAppender:
      name: QueueAppender
      PatternLayout:
        Pattern: "%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level - %msg%n"


  Loggers:
    Root:
      level: warn
      AppenderRef:
        - ref: Console
    Logger:
      name: "com.transmisms.smsftp.protocol"
      level: info
      AppenderRef:
        - ref: QueueAppender
```

```
--------------------------------------------------------------
Filename: src/main/resources/main.css
--------------------------------------------------------------

.root {
    /* custom colors defined here */
    warning-yellow: #ffe16b; /* elementary OS Banana 300 */
    info-blue: #64baff; /* elementary OS Blueberry 300 */
    error-red: #ed5353; /* elementary OS Strawberry 300 */

    text-black: #0c0c0d; /* Firefox Photon Grey 90 */
    text-grey: #737373; /* Firefox Photon Grey 50 */
    text-black-90: #0c0c0de6; /* 90% Firefox Photon Grey 90 */
    text-grey-90: #737373e6; /* 90% Firefox Photon Grey 50 */

    button-black: text-black;
    button-border-black: #999999;
    button-pressed-bg: #eeeeee;
}

.label {
    -fx-font-smoothing-type: gray;
    -fx-text-fill: text-black-90;
}

.list-view:focused .cell:selected .label, .menu-item:focused .label,
.cell:focused:selected .choice-box .label {
    -fx-text-fill: white;
}

.tool-bar .button {
    -fx-padding: 5px;
    -fx-background-color: transparent;
    -fx-border-color: transparent;
    -fx-border-radius: 3px;
}

.tool-bar .button:disabled > .graphic {
    -fx-opacity: 0.25;
}

.tool-bar .button:hover {
    -fx-background-color: white;
    -fx-background-radius: 2px;
    -fx-border-color: button-border-black;
}

.tool-bar .button:pressed {
    -fx-background-color: button-pressed-bg;
    -fx-border-color: button-border-black;
}

.tool-bar .ikonli-font-icon {
    -fx-icon-size: 24px;
    -fx-icon-color: button-black;
    -fx-opacity: 0.7;
}

#addContactButton .ikonli-font-icon {
    -fx-icon-code: "mdi-account-plus";
}

#editContactButton .ikonli-font-icon {
    -fx-icon-code: "mdi-account-settings";
}

#removeContactButton .ikonli-font-icon {
    -fx-icon-code: "mdi-account-remove";
}

#sendFileButton .ikonli-font-icon {
    -fx-icon-code: "mdi-file-send";
}

#sendFileInsecurelyButton .ikonli-font-icon {
    -fx-icon-code: "gmi-no-encryption";
}

#appMenuButton .ikonli-font-icon {
    -fx-icon-code: "mdi-menu";
}

#contactList .cell {
    -fx-padding: 6;
}

.contactName {
    -fx-alignment: center-left;
    -fx-padding: 0 0 0 6; /* set left padding to 6 */
}

.contactNum {
    -fx-alignment: center-left;
    -fx-padding: 0 0 0 6; /* set left padding to 6 */
}

.contactStatus {
    -fx-alignment: center-right;
    -fx-padding: 0 12 0 0; /* set right padding to 12 */
}

.contactNum, .contactStatus {
    -fx-font-size: 11;
    -fx-opacity: 0.5;
}

.cell:selected .contactNum, .cell:selected .contactStatus {
    -fx-opacity: 0.9;
}

.cancelOperationButton {
    -fx-background-color: transparent;
}

.cancelOperationButton .ikonli-font-icon {
    -fx-icon-code: "gmi-cancel";
    -fx-icon-size: 16px;
    -fx-icon-color: black;
    -fx-opacity: 0.5;
}

.cancelOperationButton .ikonli-font-icon:hover {
    -fx-opacity: 0.6;
}

.cancelOperationButton .ikonli-font-icon:pressed {
    -fx-opacity: 0.8;
}

.progress-bar {
    -fx-padding: 0 0 0 6; /* set left padding to 6 */
}

.progress-bar > .bar {
    -fx-background-insets: 1 1 2 1;
    -fx-padding: 0.20em;
}

#logTarget {
    -fx-font-family: "Courier", monospace;
}

.infobar {
    -fx-padding: 6 12 6 12;
}

.infoInfobar {
    -fx-background-color: info-blue;
}

.infobarLabel {
    -fx-alignment: center-left;
    -fx-padding: 0 0 0 6; /* set left padding to 6 */
}

.infobuttonContainer {
    -fx-alignment: center-right;
}

.infobuttonContainer .button {
    -fx-alignment: center-right;
}

#errorInfobar {
    -fx-background-color: error-red;
}

#welcomePaneRoot {
    -fx-spacing: 20px;
}

#welcomeActions {
    -fx-alignment: center-left;
}

#welcomeTitle, #welcomeSubtitle {
    -fx-alignment: center;
}

#welcomeTitle {
    -fx-font-size: 22px;
    -fx-text-fill: text-black;
}

#welcomeSubtitle {
    -fx-text-fill: text-grey;
}

#welcomeAddContactButton, #welcomeWaitContactButton {
    -fx-padding: 0px;
    -fx-background-color: transparent;
}

#welcomeAddContactButton .ikonli-font-icon, #welcomeWaitContactButton
.ikonli-font-icon {
    -fx-icon-size: 36px;
    -fx-icon-color: button-black;
    -fx-opacity: 0.7;
}

#welcomeAddContactButton .ikonli-font-icon {
    -fx-icon-code: "mdi-account-plus";
}

#welcomeWaitContactButton .ikonli-font-icon {
    -fx-icon-code: "gmi-more-horiz";
}

.welcomeActionContainer {
    -fx-alignment: center-left;
}

.welcomeAction {
    -fx-text-fill: text-black;
}

.welcomeActionDesc {
    -fx-text-fill: text-grey;
}



------------------------------------------------------------
Filename: src/main/resources/prefs-window.css
------------------------------------------------------------

#prefsGrid .label {
    /*fx-alignment: right;*/
}

#backendSettingsLabel {
    /*-fx-font-size: 22px;*/
    -fx-font-weight: bold;
}

#prefsErrorText {
    -fx-text-fill: red;
}




------------------------------------------------------------
Filename: src/main/resources/prefs-window.fxml
------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<?import com.transmisms.ui.javafx.NonemptyStringConverter?>
<?import com.transmisms.ui.javafx.MappedStringConverter?>
<?import com.transmisms.ui.javafx.PortStringConverter?>

<?import java.lang.String?>

<?import java.net.*?>
<?import javafx.collections.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<?import org.kordamp.ikonli.javafx.FontIcon?>
<?import org.kordamp.ikonli.javafx.StackedFontIcon?>



<HBox fx:controller="com.transmisms.ui.javafx.PrefsController"
        xmlns:fx="http://javafx.com/fxml"
        alignment="CENTER">
    <fx:define>
        <MappedStringConverter fx:id="backendTypeConverter">
            <lookupMap>
                <FXCollections fx:factory="observableHashMap"
                        fx:id="backendTypeConverterMap"
                        dummy="Dummy"
                        gammu-psql="Gammu via PostgreSQL"
                        android="Android Companion" />
            </lookupMap>
        </MappedStringConverter>
        <MappedStringConverter fx:id="retryOnErrorsConverter">
            <lookupMap>
                <FXCollections fx:factory="observableHashMap"
                        prompt="Prompt"
                        always="Always"
                        never="Never" />
            </lookupMap>
        </MappedStringConverter>
    </fx:define>


    <padding><Insets top="12" right="12" bottom="12" left="12" /></padding>

    <VBox spacing="24">
        <GridPane fx:id="prefsGrid" hgap="12" vgap="12">
            <gridLinesVisible>false</gridLinesVisible>

            <Label text="Save received files to:"
                    GridPane.halignment="RIGHT"
                    GridPane.rowIndex="0" GridPane.columnIndex="0" />
            <TextField fx:id="downloadLocation"
                    editable="false"
                    GridPane.columnSpan="1"
                    GridPane.rowIndex="0" GridPane.columnIndex="1" />
```

```
<Button text="…" fx:id="downloadLocationSelectorButton"
        onAction="#handleDownloadLocationSelectorButton"
        GridPane.columnSpan="1"
        GridPane.rowIndex="0" GridPane.columnIndex="2" />
<Label text="Retry on errors:"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="1" GridPane.columnIndex="0" />
<ComboBox fx:id="retryOnErrors"
        converter="$retryOnErrorsConverter"
        value="prompt"
        GridPane.columnSpan="1"
        GridPane.rowIndex="1" GridPane.columnIndex="1">
    <items>
        <FXCollections fx:factory="observableArrayList">
            <String fx:value="prompt" />
            <String fx:value="always" />
            <String fx:value="never" />
        </FXCollections>
    </items>
</ComboBox>
<Label text="Country Code:"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="2" GridPane.columnIndex="0" />
<TextField fx:id="countryCode"
        GridPane.columnSpan="1"
        GridPane.rowIndex="2" GridPane.columnIndex="1">
    <textFormatter><TextFormatter>
        <valueConverter><PortStringConverter /></valueConverter>
    </TextFormatter></textFormatter>
</TextField>
<Label text="Sms Backend Used:"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="3" GridPane.columnIndex="0" />
<ComboBox fx:id="backendType"
        converter="$backendTypeConverter"
        value="android"
        onAction="#refreshBackendSettingsLabel"
        GridPane.columnSpan="1"
        GridPane.rowIndex="3" GridPane.columnIndex="1">
    <items>
        <FXCollections fx:factory="observableArrayList">
            <String fx:value="dummy" />
            <String fx:value="gammu-psql" />
            <String fx:value="android" />
        </FXCollections>
    </items>
</ComboBox>

<Separator
        GridPane.columnSpan="3"
        GridPane.rowIndex="4" GridPane.columnIndex="0" />

<Label fx:id="backendSettingsLabel"
        text="Backend Settings"
        GridPane.columnSpan="3"
        GridPane.rowIndex="5" GridPane.columnIndex="0" />

<!-- dummy backend settings -->
<Label text="Host:"
        visible="${backendType.value == 'dummy'}"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="6" GridPane.columnIndex="0" />
<TextField fx:id="backendDummyHost"
        text="localhost"
        promptText="Hostname or IP address"
        visible="${backendType.value == 'dummy'}"
        GridPane.columnSpan="1"
        GridPane.rowIndex="6" GridPane.columnIndex="1">
    <textFormatter><TextFormatter>
        <valueConverter>
            <NonemptyStringConverter />
        </valueConverter>
    </TextFormatter></textFormatter>
</TextField>
<Label text="Port:"
        visible="${backendType.value == 'dummy'}"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="7" GridPane.columnIndex="0" />
<TextField fx:id="backendDummyPort"
        text="8767"
        visible="${backendType.value == 'dummy'}"
        GridPane.columnSpan="1"
        GridPane.rowIndex="7" GridPane.columnIndex="1">
    <textFormatter><TextFormatter>
        <valueConverter><PortStringConverter /></valueConverter>
    </TextFormatter></textFormatter>
</TextField>
<Label text="Local MIN:"
        visible="${backendType.value == 'dummy'}"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="8" GridPane.columnIndex="0" />
<TextField fx:id="backendDummyLocalmin"
        text="9121234567"
        promptText="10-digit mobile number"
        visible="${backendType.value == 'dummy'}"
        GridPane.columnSpan="1"
        GridPane.rowIndex="8" GridPane.columnIndex="1">
    <textFormatter><TextFormatter>
        <valueConverter>
            <NonemptyStringConverter />
        </valueConverter>
    </TextFormatter></textFormatter>
</TextField>
<Label text="Role:"
        visible="${backendType.value == 'dummy'}"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="9" GridPane.columnIndex="0" />
<ComboBox fx:id="backendDummyRole"
        visible="${backendType.value == 'dummy'}"
        value="sender"
        GridPane.columnSpan="1"
        GridPane.rowIndex="9" GridPane.columnIndex="1">
    <items>
        <FXCollections fx:factory="observableArrayList">
            <String fx:value="sender" />
            <String fx:value="receiver" />
        </FXCollections>
    </items>
</ComboBox>

<!-- gammu backend settings -->
<Label text="Database Host:"
        visible="${backendType.value == 'gammu-psql'}"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="6" GridPane.columnIndex="0" />
<TextField fx:id="backendGammuHost"
        text="localhost"
        promptText="Hostname or IP address"
        visible="${backendType.value == 'gammu-psql'}"
        GridPane.columnSpan="1"
        GridPane.rowIndex="6" GridPane.columnIndex="1">
    <textFormatter><TextFormatter>
        <valueConverter>
            <NonemptyStringConverter />
        </valueConverter>
    </TextFormatter></textFormatter>
</TextField>
<Label text="Database Port:"
        visible="${backendType.value == 'gammu-psql'}"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="7" GridPane.columnIndex="0" />
<TextField fx:id="backendGammuPort"
        text="5432"
        visible="${backendType.value == 'gammu-psql'}"
        GridPane.columnSpan="1"
        GridPane.rowIndex="7" GridPane.columnIndex="1">
    <textFormatter><TextFormatter>
        <valueConverter><PortStringConverter /></valueConverter>
    </TextFormatter></textFormatter>
</TextField>
<Label text="Database Name:"
        visible="${backendType.value == 'gammu-psql'}"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="8" GridPane.columnIndex="0" />
<TextField fx:id="backendGammuName"
        text="gammu"
        visible="${backendType.value == 'gammu-psql'}"
        GridPane.columnSpan="1"
        GridPane.rowIndex="8" GridPane.columnIndex="1">
    <textFormatter><TextFormatter>
        <valueConverter>
            <NonemptyStringConverter />
        </valueConverter>
    </TextFormatter></textFormatter>
</TextField>
<Label text="Database User:"
        visible="${backendType.value == 'gammu-psql'}"
        GridPane.halignment="RIGHT"
        GridPane.rowIndex="9" GridPane.columnIndex="0" />
<TextField fx:id="backendGammuUser"
        text="gammu"
        visible="${backendType.value == 'gammu-psql'}"
        GridPane.columnSpan="1"
        GridPane.rowIndex="9" GridPane.columnIndex="1">
    <textFormatter><TextFormatter>
        <valueConverter>
            <NonemptyStringConverter />
        </valueConverter>
    </TextFormatter></textFormatter>
</TextField>
<Label text="Database Password:"
        visible="${backendType.value == 'gammu-psql'}"
```

```
                GridPane.halignment="RIGHT"
                GridPane.rowIndex="10" GridPane.columnIndex="0" />
        <TextField fx:id="backendGammuPassword"
                text="gammu"
                visible="${backendType.value == 'gammu-psql'}"
                GridPane.columnSpan="1"
                GridPane.rowIndex="10" GridPane.columnIndex="1" />
        <Label text="Local MIN:"
                visible="${backendType.value == 'gammu-psql'}"
                GridPane.halignment="RIGHT"
                GridPane.rowIndex="11" GridPane.columnIndex="0" />
        <TextField fx:id="backendGammuLocalmin"
                text="9121234567"
                promptText="10-digit mobile number"
                visible="${backendType.value == 'gammu-psql'}"
                GridPane.columnSpan="1"
                GridPane.rowIndex="11" GridPane.columnIndex="1">
            <textFormatter><TextFormatter>
                <valueConverter>
                    <NonemptyStringConverter />
                </valueConverter>
            </TextFormatter></textFormatter>
        </TextField>

        <!-- android backend settings -->
        <Label text="Host:"
                visible="${backendType.value == 'android'}"
                GridPane.halignment="RIGHT"
                GridPane.rowIndex="6" GridPane.columnIndex="0" />
        <TextField fx:id="backendAndroidHost"
                text="192.168.254.102"
                promptText="Hostname or IP address"
                visible="${backendType.value == 'android'}"
                GridPane.columnSpan="1"
                GridPane.rowIndex="6" GridPane.columnIndex="1">
            <textFormatter><TextFormatter>
                <valueConverter>
                    <NonemptyStringConverter />
                </valueConverter>
            </TextFormatter></textFormatter>
        </TextField>
        <Label text="Port:"
                visible="${backendType.value == 'android'}"
                GridPane.halignment="RIGHT"
                GridPane.rowIndex="7" GridPane.columnIndex="0" />
        <TextField fx:id="backendAndroidPort"
                text="8767"
                visible="${backendType.value == 'android'}"
                GridPane.columnSpan="1"
                GridPane.rowIndex="7" GridPane.columnIndex="1">
            <textFormatter><TextFormatter>
                <valueConverter><PortStringConverter /></valueConverter>
            </TextFormatter></textFormatter>
        </TextField>
    </GridPane>
    <Label fx:id="prefsErrorText"
            visible="false"
            text="The fields marked red are invalid" />
  </VBox>
</HBox>



--------------------------------------------------------------
Filename: src/main/resources/transmisms-javafx-keystore.yaml
--------------------------------------------------------------

---
# NOTE: for insecure contacts, just leave security info blank
keystore:
  # sample:
  # 'contact-msisdn': # numeric without leading zeroes,
  #                     country code, etc.
  #   name: 'contact-name'
  #   keys: # just null/empty for insecure contacts
  #     host-pub-enc  : >-
  #       alignedbase64encodedstringthatcanbeconverted
  #       tonativebytearrayonjava====================
  #     host-priv-enc : >-
  #     host-pub-auth : >-
  #     host-priv-auth: >-
  #     peer-pub-enc  : >-
  #     peer-pub-auth : >-
...



--------------------------------------------------------------
Filename: src/main/resources/transmisms-javafx.yaml
--------------------------------------------------------------

---
appconfig:
  download-location : # should be autodetected at startup if empty
  retry-on-errors   : !!str 'prompt'
  country-code      : !!str 63
  backend-type      : !!str 'dummy'
  backend-settings:
    backend-dummy:
      host: !!str 'localhost'
      port: !!int 8767
      local-min: !!str '9127654321'
      role: !!str 'sender'
    backend-gammu-psql:
      database-host     : !!str 'localhost'
      database-port     : !!int 5432
      database-name     : !!str 'gammu'
      database-user     : !!str 'gammu'
      database-password : !!str 'gammu'
      local-min         : !!str '9127654321'
    backend-android:
      host: !!str 192.168.254.102
      port: !!int 8767
...



--------------------------------------------------------------
Filename: src/main/resources/welcome-screen.fxml
--------------------------------------------------------------

<?xml version="1.0" encoding="UTF-8"?>

<?import java.net.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.text.*?>

<?import org.kordamp.ikonli.javafx.FontIcon?>
<?import org.kordamp.ikonli.javafx.StackedFontIcon?>



<HBox alignment="CENTER" xmlns:fx="http://javafx.com/fxml">
    <padding>
        <Insets top="36" right="12" bottom="12" left="12" />
    </padding>
    <VBox fx:id="welcomePaneRoot">
        <VBox alignment="CENTER">
            <Label fx:id="welcomeTitle"
                    text="Add contacts" />
            <Label fx:id="welcomeSubtitle"
                    text="You don't have any contacts yet." />
        </VBox>
        <GridPane fx:id="welcomeActions"
                hgap="10" vgap="10">
            <gridLinesVisible>false</gridLinesVisible>

            <Button fx:id="welcomeAddContactButton"
                    GridPane.rowIndex="0" GridPane.columnIndex="0">
                <graphic><FontIcon /></graphic>
            </Button>
            <VBox styleClass="welcomeActionContainer"
                GridPane.rowIndex="0" GridPane.columnIndex="1">
                <Label styleClass="welcomeAction"
                        text="Add" />
                <Label styleClass="welcomeActionDesc"
                        text="Manually add a new contact." />
            </VBox>

            <Button fx:id="welcomeWaitContactButton"
                    GridPane.rowIndex="1" GridPane.columnIndex="0">
                <graphic><FontIcon /></graphic>
            </Button>
            <VBox styleClass="welcomeActionContainer"
                GridPane.rowIndex="1" GridPane.columnIndex="1">
                <Label styleClass="welcomeAction"
                    text="Wait" />
                <Label styleClass="welcomeActionDesc"
                    text="Wait for a pairing or file transfer request." />
            </VBox>
        </GridPane>
    </VBox>
</HBox>



--------------------------------------------------------------
```

```
Filename: src/test/integration/java/AdversarialConnectionTest.java
--------------------------------------------------------------

/*
// NOTE: just to be sure we have BouncyCastle as a provider
//Security.addProvider(new BouncyCastleProvider());
*/
package com.transmisms.test.integration;

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.Session;
import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;

import org.apache.logging.log4j.LogManager;


import static org.testng.Assert.assertNotEquals;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;

import org.testng.annotations.Test;



public class AdversarialConnectionTest {

    //// final Object for repeated use across tests
    private static final String sendMin = "09121234567";
    private static final String recvMin = "09129876543";



    @Test
    public void previousSessionPdusReceivedTest() {
        class ThrowbackSmsService extends DummySmsService  {
            final static String oldMessage = "?transmismsv0.7.0\n" +
                "9>#N£W!>W(N'L+GR4Yo)\n" +
                "last/nullprotocol/0001/OK; Completed without error\n" +
                "\n" +
                "To stop receiving messages, reply with NO";

            final int limit = 1;
            int count = 0;

            public ThrowbackSmsService(String localMin) {
                super(localMin);
            }

            public ThrowbackSmsService(String localMin, DummySmsService peer) {
                super(localMin, peer);
            }

            @Override
            public void sendMessage(String message) {
                // don't send any retransmission messages for this test
                if(message.charAt(0) == 'r') {
                    return;
                }

                boolean shouldResendOldMessage = this.count == 0;

                if(shouldResendOldMessage) {
                    System.out.println("Resending old message...");
                    this.peer.insertSmsEntry(
                            new SmsEntry(this.localMin, this.oldMessage));
                }
                this.peer.insertSmsEntry(
                        new SmsEntry(this.localMin, message));

                // comment out this block for filtering against DATA PDUs:
                if(message.charAt(0) != 'd') {
                    System.out.println("Sendto "
                            + this.localMin + ": " + message);
                }

                this.count++;
                if(this.count >= this.limit) {
                    this.count = 0;
                }
            }
        }

        // initialize sms services
        DummySmsService sendService = new ThrowbackSmsService(sendMin);
        DummySmsService recvService = new ThrowbackSmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);
```

```
        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn = Connection.createResponderConnection(recvMin);
        Session sendSession = NullSession.generateSenderSession(sendConn);
        Session recvSession = NullSession.generateReceiverSession(recvConn);
        NullFacade sendFacade =
                new NullFacade(sendSession, sendService);
        NullFacade recvFacade =
                new NullFacade(recvSession, recvService);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("PPT-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("PPT-recv")));
        sendFacade.addObserver(new AllYesPresenter(sendFacade));
        recvFacade.addObserver(new AllYesPresenter(recvFacade));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.connect();
        };
        Runnable recvRunnable = () -> {
            recvFacade.listen();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, 20000,
                "previousSessionPdusReceivedTest");

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("PPT").info(
                "END of previousSessionPdusReceivedTest");
    }

}



--------------------------------------------------------------
Filename: src/test/integration/java/AllYesPresenter.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.CoreProtocolFacade;


public class AllYesPresenter extends NullPresenter {
    private CoreProtocolFacade coreProtocolFacade;

    public AllYesPresenter(CoreProtocolFacade coreProtocolFacade) {
        super(coreProtocolFacade);
        this.coreProtocolFacade = coreProtocolFacade;
    }

    @Override
    protected void onTimeout() {
        // default: cancel immediately after timeout
        this.coreProtocolFacade.continueOperationAfterPrompt();
    }
}



--------------------------------------------------------------
Filename: src/test/integration/java/AllYesSmsftpPresenter.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.Presenter.PromptType;
import com.transmisms.smsftp.protocol.SmsftpFacade;



public class AllYesSmsftpPresenter extends NullSmsftpPresenter {
    public AllYesSmsftpPresenter(SmsftpFacade smsftpFacade, byte[] enckey,
            byte[] authkey) {
        super(smsftpFacade, enckey, authkey);
    }

    public AllYesSmsftpPresenter(SmsftpFacade smsftpFacade) {
        super(smsftpFacade);
    }
    @Override
    protected void onUserPrompt(PromptType pType, String message) {
        if(pType == PromptType.TIMEOUT) {
            this.smsftpFacade.continueOperationAfterPrompt();
        }
        else {
            super.onUserPrompt(pType, message);
        }
    }
}




--------------------------------------------------------------
Filename: src/test/integration/java/BlackholeOutgoingSmsService.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.SmsServiceException;



public class BlackholeOutgoingSmsService extends DummySmsService {

    public BlackholeOutgoingSmsService(String localMin) {
        super(localMin);
    }

    public BlackholeOutgoingSmsService(String localMin, DummySmsService peer) {
        super(localMin, peer);
    }

    @Override
    public void sendMessage(String message) {
        // do nothing; discard all messages to be sent

        // NOTE: you can comment this out for debugging:
        //System.out.println("Nosend: " + this.localMin + ": " + message);
        // or this code block for filtering against DATA PDUs:
    }
}




--------------------------------------------------------------
Filename: src/test/integration/java/ConnectionTest.java
--------------------------------------------------------------

/*
// NOTE: just to be sure we have BouncyCastle as a provider
//Security.addProvider(new BouncyCastleProvider());
*/
package com.transmisms.test.integration;

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.Session;
import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;

import org.apache.logging.log4j.LogManager;


import static org.testng.Assert.assertNotEquals;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;

import org.testng.annotations.Test;



public class ConnectionTest {

    //// final Object for repeated use across tests
    private static final String sendMin = "09121234567";
    private static final String recvMin = "09129876543";



    @Test
    public void connectDisconnectTest() {
        // initialize sms services
        DummySmsService sendService = new DummySmsService(sendMin);
        DummySmsService recvService = new DummySmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn = Connection.createResponderConnection(recvMin);
        Session sendSession = NullSession.generateSenderSession(sendConn);
        Session recvSession = NullSession.generateReceiverSession(recvConn);
        NullFacade sendFacade =
                new NullFacade(sendSession, sendService);
        NullFacade recvFacade =
                new NullFacade(recvSession, recvService);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("CDT-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("CDT-recv")));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.connect();
        };
        Runnable recvRunnable = () -> {
            recvFacade.listen();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, 10000,
                "connectDisconnectTest");

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("CDT").info("END of connectDisconnectTest");
    }

    @Test
    public void timeoutShouldTriggerReactionTest() {
        // initialize sms services
        DummySmsService sendService = new DummySmsService(sendMin);
        DummySmsService recvService =
                new BlackholeOutgoingSmsService(recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn =
                Connection.createResponderConnection(recvMin);
        Session sendSession = NullSession.generateSenderSession(sendConn);
        Session recvSession = NullSession.generateReceiverSession(recvConn);
        NullFacade sendFacade =
                new NullFacade(sendSession, sendService);
        NullFacade recvFacade =
                new NullFacade(recvSession, recvService);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("TOT-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("TOT-recv")));
        sendFacade.addObserver(new NullPresenter(sendFacade));
        recvFacade.addObserver(new NullPresenter(recvFacade));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.connect();
        };
        Runnable recvRunnable = () -> {
            recvFacade.listen();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, 8000,
                "timeoutShouldTriggerReactionTest");

        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("TOT")
                .info("END of timeoutShouldTriggerReactionTest");
    }

    @Test
    public void userContinueTimeoutShouldContinue() {
        class UnreliableSmsService extends DummySmsService {
            final int limit = 3;
            int count = 0;
            public UnreliableSmsService(String localMin) {
                super(localMin);
            }

            public UnreliableSmsService(String localMin, DummySmsService peer) {
                super(localMin, peer);
            }

            @Override
            public void sendMessage(String message) {
                // don't send any retransmission messages for this test
                if(message.charAt(0) == 'r') {
                    return;
                }

                boolean isReliable = this.count != 0;

                if(isReliable) {
                    this.peer.insertSmsEntry(
                            new SmsEntry(this.localMin, message));
                }

                // comment out this block for filtering against DATA PDUs:
                /*
                if(message.charAt(0) != 'd') {
                    System.out.println((isReliable ? "Sendto: " :
                            "Nosend: ") + this.localMin + ": " + message);
                }
                */

                this.count++;
                if(this.count >= this.limit) {
                    this.count = 0;
                }
            }
        }

        // initialize sms services
        DummySmsService sendService = new UnreliableSmsService(sendMin);
        DummySmsService recvService =
                new UnreliableSmsService(recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn =
                Connection.createResponderConnection(recvMin);
        Session sendSession = NullSession.generateSenderSession(sendConn);
        Session recvSession = NullSession.generateReceiverSession(recvConn);
        NullFacade sendFacade =
                new NullFacade(sendSession, sendService);
        NullFacade recvFacade =
                new NullFacade(recvSession, recvService);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("TCT-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("TCT-recv")));
        sendFacade.addObserver(new AllYesPresenter(sendFacade));
        recvFacade.addObserver(new AllYesPresenter(recvFacade));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.connect();
        };
        Runnable recvRunnable = () -> {
            recvFacade.listen();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, 30000,
                "userContinueTimeoutShouldContinue");

        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("TCT")
                .info("END of userContinueTimeoutShouldContinue");
    }
}



--------------------------------------------------------------
Filename: src/test/integration/java/DelayedConnectionTest.java
--------------------------------------------------------------

/*
// NOTE: just to be sure we have BouncyCastle as a provider
//Security.addProvider(new BouncyCastleProvider());
*/
package com.transmisms.test.integration;
import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.Session;
import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;

import org.apache.logging.log4j.LogManager;


import static org.testng.Assert.assertNotEquals;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;

import org.testng.annotations.Test;



public class DelayedConnectionTest {

    //// final Object for repeated use across tests
    private static final String sendMin = "09121234567";
    private static final String recvMin = "09129876543";



    @Test
    public void delayedDeliveryTest() {
        class SlowSmsService extends DummySmsService  {

            public SlowSmsService(String localMin) {
                super(localMin);
            }

            public SlowSmsService(String localMin, DummySmsService peer) {
                super(localMin, peer);
            }

            @Override
            public void sendMessage(String message) {
                Thread t = new Thread(() -> {
                    try {
                        Thread.sleep(2000);
                    }
                    catch(InterruptedException e) {
                        return; // exit ASAP
                    }
                    this.peer.insertSmsEntry(
                            new SmsEntry(this.localMin, message));
                });
                t.start();
            }
        }

        // initialize sms services
        DummySmsService sendService = new SlowSmsService(sendMin);
        DummySmsService recvService = new SlowSmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn = Connection.createResponderConnection(recvMin);
        Session sendSession = NullSession.generateSenderSession(sendConn);
        Session recvSession = NullSession.generateReceiverSession(recvConn);
        NullFacade sendFacade =
                new NullFacade(sendSession, sendService);
        NullFacade recvFacade =
                new NullFacade(recvSession, recvService);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("DDT-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("DDT-recv")));
        sendFacade.addObserver(new AllYesPresenter(sendFacade));
        recvFacade.addObserver(new AllYesPresenter(recvFacade));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.connect();
        };
        Runnable recvRunnable = () -> {
            recvFacade.listen();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, 20000,
                "delayedDeliveryTest");
        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("CDT").info("END of delayedDeliveryTest");
    }

}
```

```
--------------------------------------------------------------
Filename: src/test/integration/java/DummySmsService.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.core.protocol.SmsService;
import com.transmisms.core.protocol.SmsServiceException;

import java.util.concurrent.BlockingQueue;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.TimeUnit;



public class DummySmsService implements SmsService {
    protected DummySmsService peer = null;
    protected String localMin;
    private boolean fakeStatus = false;

    protected BlockingQueue<SmsEntry> bq;



    public DummySmsService(String localMin) {
        this(localMin, null);
    }

    public DummySmsService(String localMin, DummySmsService peer) {
        this.localMin = localMin;
        this.setPeer(peer);
        this.bq = new LinkedBlockingQueue<>();
    }



    public void setPeer(DummySmsService peer) {
        if(this.peer == null) {
            this.peer = peer;
        }
    }

    protected void insertSmsEntry(SmsEntry entry) {
        // NOTE: no checking for overflow since it is assumed that it should
        //       never happen on our use case
        this.bq.offer(entry);
    }

    @Override
    public boolean isRunning() {
        return this.fakeStatus;
    }

    @Override
    public boolean start() {
        this.fakeStatus = true;
        return true;
    }

    @Override
    public boolean stop() {
        this.fakeStatus = false;
        return true;
    }

    @Override
    public void sendMessage(String message) {
        // NOTE: you can comment this out for debugging:
        //System.out.println("Sendto: " + this.localMin + ": " + message);
        // or this code block for filtering against DATA PDUs:
        this.peer.insertSmsEntry(new SmsEntry(this.localMin, message));
    }

    @Override
    public SmsEntry pollMessage(long timeout, TimeUnit unit)
            throws SmsServiceException, InterruptedException {
        return this.bq.poll(timeout, unit);
    }
}
```

```
--------------------------------------------------------------
Filename: src/test/integration/java/DuplicateConnectionTest.java
--------------------------------------------------------------

/*
// NOTE: just to be sure we have BouncyCastle as a provider
//Security.addProvider(new BouncyCastleProvider());
*/
package com.transmisms.test.integration;

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.Session;
import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;

import org.apache.logging.log4j.LogManager;


import static org.testng.Assert.assertNotEquals;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;

import org.testng.annotations.Test;



public class DuplicateConnectionTest {

    //// final Object for repeated use across tests
    private static final String sendMin = "09121234567";
    private static final String recvMin = "09129876543";



    @Test
    public void duplicateDeliveryTest() {
        class DoubleSmsService extends DummySmsService  {

            public DoubleSmsService(String localMin) {
                super(localMin);
            }

            public DoubleSmsService(String localMin, DummySmsService peer) {
                super(localMin, peer);
            }

            @Override
            public void sendMessage(String message) {
                this.peer.insertSmsEntry(
                        new SmsEntry(this.localMin, message));
                // then send the message a while later
                Thread t = new Thread(() -> {
                    try {
                        Thread.sleep(500);
                    }
                    catch(InterruptedException e) {
                        return; // exit ASAP
                    }
                    this.peer.insertSmsEntry(
                            new SmsEntry(this.localMin, message));
                });
                t.start();
            }
        }

        // initialize sms services
        DummySmsService sendService = new DoubleSmsService(sendMin);
        DummySmsService recvService = new DoubleSmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn = Connection.createResponderConnection(recvMin);
        Session sendSession = NullSession.generateSenderSession(sendConn);
        Session recvSession = NullSession.generateReceiverSession(recvConn);
        NullFacade sendFacade =
                new NullFacade(sendSession, sendService);
        NullFacade recvFacade =
                new NullFacade(recvSession, recvService);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("2XT-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("2XT-recv")));
        sendFacade.addObserver(new AllYesPresenter(sendFacade));
        recvFacade.addObserver(new AllYesPresenter(recvFacade));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.connect();
        };
        Runnable recvRunnable = () -> {
            recvFacade.listen();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, 20000,
                "duplicateDeliveryTest");

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        Utils.sleepSilently(10); // sleep for a while to "synchronize" stdout
        LogManager.getLogger("CDT").info("END of duplicateDeliveryTest");
    }
}



--------------------------------------------------------------
Filename: src/test/integration/java/ErraticConnectionTest.java
--------------------------------------------------------------

/*
// NOTE: just to be sure we have BouncyCastle as a provider
//Security.addProvider(new BouncyCastleProvider());
*/
package com.transmisms.test.integration;

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.Session;
import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;

import org.apache.logging.log4j.LogManager;


import static org.testng.Assert.assertNotEquals;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;

import org.testng.annotations.Test;



public class ErraticConnectionTest {

    //// final Object for repeated use across tests
    private static final String sendMin = "09121234567";
    private static final String recvMin = "09129876543";



    @Test
    public void corruptedDeliveryTest() {
        class ErraticSmsService extends DummySmsService  {
            final int limit = 3;
            int count = 0;

            public ErraticSmsService(String localMin) {
                super(localMin);
            }

            public ErraticSmsService(String localMin, DummySmsService peer) {
                super(localMin, peer);
            }

            @Override
            public void sendMessage(String msg) {
                // don't send any retransmission messages for this test
                if(msg.charAt(0) == 'r') {
                    return;
                }

                boolean isReliable = this.count != 0;

                String message = msg;
                if(!isReliable) {
                    message = "f" + msg;
                }
                this.peer.insertSmsEntry(
                        new SmsEntry(this.localMin, message));

                // comment out this block for filtering against DATA PDUs:
                /*
                if(message.charAt(0) != 'd') {
                    System.out.println("Sendto "
                            + this.localMin + ": " + message);
                }
                */

                this.count++;
                if(this.count >= this.limit) {
                    this.count = 0;
                }
            }
        }

        // initialize sms services
        DummySmsService sendService = new ErraticSmsService(sendMin);
        DummySmsService recvService = new ErraticSmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn = Connection.createResponderConnection(recvMin);
        Session sendSession = NullSession.generateSenderSession(sendConn);
        Session recvSession = NullSession.generateReceiverSession(recvConn);
        NullFacade sendFacade =
                new NullFacade(sendSession, sendService);
        NullFacade recvFacade =
                new NullFacade(recvSession, recvService);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("CDT-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("CDT-recv")));
        sendFacade.addObserver(new AllYesPresenter(sendFacade));
        recvFacade.addObserver(new AllYesPresenter(recvFacade));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.connect();
        };
        Runnable recvRunnable = () -> {
            recvFacade.listen();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, 20000,
                "corruptedDeliveryTest");

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("CDT").info("END of corruptedDeliveryTest");
    }

}



--------------------------------------------------------------
Filename: src/test/integration/java/NullFacade.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.BinaryPDUDecoder;
import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.CoreProtocolFacade;
import com.transmisms.core.protocol.Session;
import com.transmisms.core.protocol.SmsService;
import com.transmisms.core.protocol.TextBasedPDUDecoder;



public class NullFacade extends CoreProtocolFacade {
    private final String SUBPROTOCOL_NAME = "nullprotocol";

    private static final String USER_ACCEPT_STATUS_CODE = "0002";
    private static final String COMPLETED_STATUS_CODE = "0001";

    private static final String[] knownProtocols = { "nullprotocol" };
    public NullFacade(Session session, SmsService smsService) {
        this(session, smsService, 1);
    }

    public NullFacade(Session session, SmsService smsService,
            int maxPduLifetime) {
        super(session, BinaryPDUDecoder.getInstance(),
                new TextBasedPDUDecoder(NullFacade.knownProtocols){},
                smsService, maxPduLifetime);
    }

    @Override
    public String getSubprotocolName() {
        return this.SUBPROTOCOL_NAME;
    }

    @Override
    protected String[] processReqAndGenRepPayload(Object[] requestData) {
        // just accept all incoming connections regardless of the data's
        // contents
        String[] retArr = { NullFacade.USER_ACCEPT_STATUS_CODE };
        return retArr;
    }

    @Override
    protected String generateHeadPayload() {
        return ""; // nothing is returned (results in "head/nullprotocol/")
    }

    @Override
    protected void processHeadPayload(Object[] headData) {
        // do nothing; we don't process anything here
    }

    @Override
    protected void onEstablished() {
        // if initiator, just disconnect
        if(this.coreSession.connection.role ==
                Connection.Role.INITIATOR) {
            this.disconnect(COMPLETED_STATUS_CODE);
        } // otherwise, do nothing and wait
    }

    public void listen() {
        super.listen();
    }

    public void connect() {
        super.connect("");
    }

}



--------------------------------------------------------------
Filename: src/test/integration/java/NullPresenter.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.CoreProtocolFacade;
import com.transmisms.core.protocol.Presenter;
import com.transmisms.core.protocol.Presenter.MessageType;
import com.transmisms.core.protocol.Presenter.PromptType;

import java.util.Observable;



public class NullPresenter extends Presenter {
    private CoreProtocolFacade coreProtocolFacade;

    public NullPresenter(CoreProtocolFacade coreProtocolFacade) {
        //super(coreProtocolFacade);
        this.coreProtocolFacade = coreProtocolFacade;
    }

    @Override
    public final void update(Observable o, Object arg) {
        // NOTE: copied from SmsftpPresenter.update()
        // check for unwanted cases
        if(arg == null || !(arg instanceof Object[])) {
            return; // do nothing
        }
        Object[] args = (Object[])arg;
        if(args.length <= 1 || !(args[0] instanceof MessageType)) {
            return; // do nothing
        }
        this.processMessage(o, args);
    }

    private void processMessage(Observable o, Object[] args) {
        // exit asap if message type is not applicable to this
        if(!(args.length == 2 && args[1] instanceof PromptType)) {
            return;
        }

        // args[0] is guaranteed instanceof MessageType
        MessageType messageType = (MessageType)args[0];
        PromptType promptType = (PromptType)args[1];

        switch(messageType) {
            // NOTE: copied from SmsftpPresenter.processMessage()
            // check for unwanted cases
            case PROMPT: {
                this.onUserPrompt(promptType, null);
                break;
            }
            default: {
                // do nothing
                break;
            }
        }
    }

    protected void onUserPrompt(PromptType promptType, String message) {
        switch(promptType) {
            case TIMEOUT: {
                // act using a background task to simulate user input
                Thread t = new Thread() {
                    @Override
                    public void run() {
                        try {
                            Thread.sleep(500); // sleep for a while
                        }
                        catch(InterruptedException e) {
                            // just maintain interrupt status and continue
                            Thread.currentThread().interrupt();
                        }
                        NullPresenter.this.onTimeout();
                    }
                };
                t.start();
                break;
            }
            default: {
                // do nothing
                break;
            }
        }
    }

    protected void onTimeout() {
        // default: cancel immediately after timeout
        this.coreProtocolFacade.cancelOperationAfterPrompt();
    }
}



--------------------------------------------------------------
Filename: src/test/integration/java/NullSession.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.Session;

import java.util.UUID;



public class NullSession extends Session {

    public NullSession(Connection connection, UUID sessionId) {
        super(connection, sessionId);
    }



    public static NullSession generateSenderSession(Connection connection) {
        return new NullSession(connection, UUID.randomUUID());
    }

    public static NullSession generateReceiverSession(Connection connection) {
        return new NullSession(connection, null);
    }
}




------------------------------------------------------------
Filename: src/test/integration/java/NullSmsftpPresenter.java
------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.Presenter.PromptType;
import com.transmisms.smsftp.protocol.SmsftpFacade;
import com.transmisms.smsftp.protocol.SmsftpPresenter;
import com.transmisms.smsftp.protocol.SmsftpSession;

import java.util.Observable;



public class NullSmsftpPresenter extends SmsftpPresenter {
    protected final SmsftpFacade smsftpFacade;
    private final SmsftpSession session;
    private final byte[] enckey;
    private final byte[] authkey;



    public NullSmsftpPresenter(SmsftpFacade smsftpFacade, byte[] enckey,
            byte[] authkey) {
        super(smsftpFacade);
        this.smsftpFacade = smsftpFacade;
        this.session = this.smsftpFacade.getSmsftpSession();
        this.enckey = enckey;
        this.authkey = authkey;
    }

    public NullSmsftpPresenter(SmsftpFacade smsftpFacade) {
        this(smsftpFacade, null, null);
    }



    protected void onLogMessage(MessageType mType, String message) {
        // do nothing; we delegate logging to other Presenters
    }

    protected void onStatusUpdate() {
    }

    protected void onUserPrompt(PromptType pType, String message) {
        switch(pType) {
            case CONNECTION_REQUEST: {
                // if responder
                if(this.session.getRole() == SmsftpSession.Role.RESPONDER) {
                    if(this.session.getEncryptionFlag()) { // if encrypted
                    }
                    // accept using a background task to simulate user input
                    Thread t = new Thread() {
                        @Override
                        public void run() {
                            try { // sleep for a while
                                Thread.sleep(500);
                            }
                            catch(InterruptedException e) {
                                // just maintain interrupt status and
                                // continue
                                Thread.currentThread().interrupt();
                            }
                            // finally accept the request
                            NullSmsftpPresenter.this.smsftpFacade
                                .acceptRequestAfterPrompt(
                                NullSmsftpPresenter.this.enckey,
                                NullSmsftpPresenter.this.authkey);
                        }
                    };
                    t.start();
                }
                // else, do nothing (we are the ones who should send requests)
                break;
            }
            case TIMEOUT: {
                break;
            }
            case SMS_SERVICE_ERROR: {
                break;
            }
            case PEER_ERROR_EXCEEDED_LIMIT: {
                break;
            }
            default: {
                return; // just do nothing and return ASAP
            }
        }
    }
}


------------------------------------------------------------
Filename: src/test/integration/java/OutOfOrderConnectionTest.java
------------------------------------------------------------

/*
// NOTE: just to be sure we have BouncyCastle as a provider
//Security.addProvider(new BouncyCastleProvider());
*/
package com.transmisms.test.integration;

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.Session;
import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;

import java.util.Deque;
import java.util.concurrent.LinkedBlockingDeque;

import org.apache.logging.log4j.LogManager;


import static org.testng.Assert.assertNotEquals;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;

import org.testng.annotations.Test;



public class OutOfOrderConnectionTest {

    //// final Object for repeated use across tests
    private static final String sendMin = "09121234567";
    private static final String recvMin = "09129876543";



    @Test
    public void outOfOrderDeliveryTest() {
        class OutOfOrderSmsService extends DummySmsService  {
            final int limit = 3;
            boolean getHead = false;

            final Deque<String> d = new LinkedBlockingDeque<>();

            public OutOfOrderSmsService(String localMin) {
                super(localMin);
            }

            public OutOfOrderSmsService(String localMin, DummySmsService peer) {
                super(localMin, peer);
            }

            @Override
            public void sendMessage(String message) {
                // just send retransmission PDUs normally
                if(message.charAt(0) != 'r') {
                    /*
                    System.out.println("Sendto: " + this.localMin +
                            ": " + s);
                    */
                    this.peer.insertSmsEntry(
                            new SmsEntry(this.localMin, message));
                    return; // exit ASAP
                }

                this.d.add(message);
                if(this.d.size() > this.limit) {

                    // send something else out of order
                    String s = this.getHead ?
                            this.d.pollFirst() : this.d.pollLast();
                    if(s != null) {
                        // comment out this block for debugging
                        /*
                        if(message.charAt(0) != 'd') {
                            System.out.println("Sendto: " + this.localMin +
                                    ": " + s);
                    }
                    */
                    this.peer.insertSmsEntry(
                            new SmsEntry(this.localMin, s));
                }
                this.getHead = !this.getHead; // flip getHead state
            }
        }
    }

    // initialize sms services
    DummySmsService sendService = new OutOfOrderSmsService(sendMin);
    DummySmsService recvService = new OutOfOrderSmsService(
            recvMin, sendService);
    sendService.setPeer(recvService);

    // initialize Core objects
    Connection sendConn =
            Connection.createInitiatorConnection(sendMin, recvMin);
    Connection recvConn = Connection.createResponderConnection(recvMin);
    Session sendSession = NullSession.generateSenderSession(sendConn);
    Session recvSession = NullSession.generateReceiverSession(recvConn);
    NullFacade sendFacade =
            new NullFacade(sendSession, sendService);
    NullFacade recvFacade =
            new NullFacade(recvSession, recvService);
    sendFacade.addObserver(
            new SmsftpLoggerPresenter(LogManager.getLogger("O3T-send")));
    recvFacade.addObserver(
            new SmsftpLoggerPresenter(LogManager.getLogger("O3T-recv")));
    sendFacade.addObserver(new AllYesPresenter(sendFacade));
    recvFacade.addObserver(new AllYesPresenter(recvFacade));

    // initialize, run, and join Runnables
    Runnable sendRunnable = () -> {
        sendFacade.connect();
    };
    Runnable recvRunnable = () -> {
        recvFacade.listen();
    };
    Utils.runAndJoin(sendRunnable, recvRunnable, 30000,
            "outOfOrderDeliveryTest");

    assertEquals(Connection.ConnectionState.CLOSED,
            sendConn.getConnectionState());
    assertEquals(Connection.ConnectionState.CLOSED,
            recvConn.getConnectionState());

    Utils.sleepSilently(10); // sleep for a while to "synchronize" stdout
    LogManager.getLogger("CDT").info("END of outOfOrderDeliveryTest");
  }
}



--------------------------------------------------------------
Filename: src/test/integration/java/SmsftpConnectionTest.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.Connection;
import com.transmisms.smsftp.protocol.SmsftpFacade;
import com.transmisms.smsftp.protocol.SmsftpSession;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;

import com.transmisms.core.util.crypto.AES;
import com.transmisms.core.util.crypto.RSA;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Arrays;
import org.apache.logging.log4j.LogManager;


import static org.testng.Assert.assertNotEquals;
import static org.testng.Assert.assertNotNull;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;

import org.testng.annotations.Test;



public class SmsftpConnectionTest {
    //// final Objects for repeated use across tests
    private final String sendMin = "09121234567";
    private final String recvMin = "09129876543";
    private final String filename = "test.txt";

    private final String testStr = "test string";
    private final byte[] dataBytes = testStr.getBytes();

    private static final int FTC_TIMEOUT = 8000;
    private static final int PRSC_TIMEOUT = 24000;


    @Test
    public void xFileTransferShouldComplete() {
        // initialize sms services
        DummySmsService sendService = new DummySmsService(sendMin);
        DummySmsService recvService = new DummySmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn =
                Connection.createResponderConnection(recvMin);
        SmsftpSession sendSession = SmsftpSession.generateSenderDataSession(
                sendConn, filename, dataBytes, null, null);
        SmsftpSession recvSession = SmsftpSession.generateReceiverSession(
                recvConn);

        SmsftpFacade sendFacade = new SmsftpFacade(sendSession, sendService,
                1);
        SmsftpFacade recvFacade = new SmsftpFacade(recvSession, recvService,
                1);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("XFTC-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("XFTC-recv")));
        sendFacade.addObserver(new NullSmsftpPresenter(sendFacade));
        recvFacade.addObserver(new NullSmsftpPresenter(recvFacade));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.insecureSendFile();
        };
        Runnable recvRunnable = () -> {
            recvFacade.receive();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, FTC_TIMEOUT,
                "xFileTransferShouldCompleteTest");

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        assertEquals(testStr, new String(recvFacade.getDataBytes()));

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("XFTC")
                .info("END of xFileTransferShouldCompleteTest");
    }

    @Test
    public void eFileTransferShouldComplete() {
        // populate with proper encryption keys
        ByteArrayOutputStream ePrivOs = new ByteArrayOutputStream();
        ByteArrayOutputStream ePubOs = new ByteArrayOutputStream();
        ByteArrayOutputStream aPrivOs = new ByteArrayOutputStream();
        ByteArrayOutputStream aPubOs = new ByteArrayOutputStream();
        try {
            RSA.generateKeys(ePrivOs, ePubOs);
            RSA.generateKeys(aPrivOs, aPubOs);
        }
        catch(IOException e) {
            fail("got IOException: " + e);
        }
        byte[] pubEnckey = ePubOs.toByteArray();
        byte[] privEnckey = ePrivOs.toByteArray();
        byte[] pubAuthkey = aPubOs.toByteArray();
        byte[] privAuthkey = aPrivOs.toByteArray();

        // initialize sms services
        DummySmsService sendService = new DummySmsService(sendMin);
        DummySmsService recvService = new DummySmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn =
                Connection.createResponderConnection(recvMin);
        SmsftpSession sendSession = SmsftpSession.generateSenderDataSession(
                sendConn, filename, dataBytes, privEnckey, privAuthkey);
        SmsftpSession recvSession = SmsftpSession.generateReceiverSession(
                recvConn);

        SmsftpFacade sendFacade = new SmsftpFacade(sendSession, sendService,
                1);
        SmsftpFacade recvFacade = new SmsftpFacade(recvSession, recvService,
                1);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("EFTC-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("EFTC-recv")));
        sendFacade.addObserver(new NullSmsftpPresenter(sendFacade));
        recvFacade.addObserver(new NullSmsftpPresenter(recvFacade, pubEnckey,
                pubAuthkey));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.secureSendFile();
        };
        Runnable recvRunnable = () -> {
            recvFacade.receive();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, FTC_TIMEOUT,
                "eFileTransferShouldCompleteTest");

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        assertEquals(testStr, new String(recvFacade.getDataBytes()));

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("EFTC")
                .info("END of eFileTransferShouldCompleteTest");
    }

    @Test
    public void pairingShouldComplete() {
        // initialize sms services
        DummySmsService sendService = new DummySmsService(sendMin);
        DummySmsService recvService = new DummySmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn =
                Connection.createResponderConnection(recvMin);
        SmsftpSession sendSession =
                SmsftpSession.generateInitiatorPairSession(sendConn);
        SmsftpSession recvSession = SmsftpSession.generateReceiverSession(
                recvConn);

        SmsftpFacade sendFacade = new SmsftpFacade(sendSession, sendService,
                1);
        SmsftpFacade recvFacade = new SmsftpFacade(recvSession, recvService,
                1);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("PRSC-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("PRSC-recv")));
        sendFacade.addObserver(new AllYesSmsftpPresenter(sendFacade));
        recvFacade.addObserver(new AllYesSmsftpPresenter(recvFacade, null,
                null));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.pair();
        };
        Runnable recvRunnable = () -> {
            recvFacade.receive();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, PRSC_TIMEOUT,
                "pairingShouldCompleteTest");

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        // verify keys
        assertNotNull(sendSession.peerEnckey);
        assertNotNull(sendSession.peerAuthkey);
        assertNotNull(recvSession.peerEnckey);
        assertNotNull(recvSession.peerAuthkey);
        assertNotEquals(0, sendSession.peerEnckey.length);
        assertNotEquals(0, sendSession.peerAuthkey.length);
        assertNotEquals(0, recvSession.peerEnckey.length);
        assertNotEquals(0, recvSession.peerAuthkey.length);
        assertTrue(Arrays.equals(sendSession.peerEnckey,
                recvSession.localEncKeypair.getPublic().getEncoded()));
        assertTrue(Arrays.equals(sendSession.peerAuthkey,
                recvSession.localAuthKeypair.getPublic().getEncoded()));
        assertTrue(Arrays.equals(recvSession.peerEnckey,
                sendSession.localEncKeypair.getPublic().getEncoded()));
        assertTrue(Arrays.equals(recvSession.peerAuthkey,
                sendSession.localAuthKeypair.getPublic().getEncoded()));

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("PRSC")
                .info("END of pairingShouldCompleteTest");
    }
}
```

```
--------------------------------------------------------------
Filename: src/test/integration/java/SmsftpLargeFileTest.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import com.transmisms.core.protocol.Connection;
import com.transmisms.smsftp.protocol.SmsftpFacade;
import com.transmisms.smsftp.protocol.SmsftpSession;
import com.transmisms.smsftp.temputil.SmsftpLoggerPresenter;

import com.transmisms.core.util.crypto.AES;
import com.transmisms.core.util.crypto.RSA;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import org.apache.logging.log4j.LogManager;
import org.apache.commons.io.IOUtils;


import static org.testng.Assert.assertNotEquals;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;

import org.testng.annotations.Test;


public class SmsftpLargeFileTest {
    //// final Object for repeated use across tests
    private final String sendMin = "09121234567";
    private final String recvMin = "09129876543";
    private final String filename = "test.txt";

    private static byte[] dataBytes = null;

    private static final int LTC_TIMEOUT = 40000;

    static {
        try {
            dataBytes = IOUtils.toByteArray(
                    ClassLoader.getSystemResourceAsStream("5MB.zip"));
        }
        catch(IOException e) {
            assert false;
        }
    }



    @Test
    public void xLargeFileTransferShouldComplete() {
        // initialize sms services
        DummySmsService sendService = new DummySmsService(sendMin);
        DummySmsService recvService = new DummySmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn =
                Connection.createResponderConnection(recvMin);
        SmsftpSession sendSession = SmsftpSession.generateSenderDataSession(
                sendConn, filename, dataBytes, null, null);
        SmsftpSession recvSession = SmsftpSession.generateReceiverSession(
                recvConn);

        SmsftpFacade sendFacade = new SmsftpFacade(sendSession, sendService,
                4);
        SmsftpFacade recvFacade = new SmsftpFacade(recvSession, recvService,
                4);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("XLTC-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("XLTC-recv")));
        sendFacade.addObserver(new NullSmsftpPresenter(sendFacade));
        recvFacade.addObserver(new NullSmsftpPresenter(recvFacade));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.insecureSendFile();
        };
        Runnable recvRunnable = () -> {
            recvFacade.receive();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, LTC_TIMEOUT,
                "xLargeFileTransferShouldCompleteTest");

        // check if data received bytes' and original bytes' hashes match
        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("SHA-1");
        }
        catch(NoSuchAlgorithmException e) {
            fail("Problems initializing native SHA-1 digest");
        }
        md.update(dataBytes);
        byte[] srcHash = md.digest();
        md.update(recvFacade.getDataBytes());
        byte[] rcvHash = md.digest();
        assertTrue(Arrays.equals(srcHash, rcvHash));

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("XLTC")
                .info("END of xLargeFileTransferShouldCompleteTest");
    }

    @Test
    public void eLargeFileTransferShouldComplete() {
        // populate with proper encryption keys
        ByteArrayOutputStream ePrivOs = new ByteArrayOutputStream();
        ByteArrayOutputStream ePubOs = new ByteArrayOutputStream();
        ByteArrayOutputStream aPrivOs = new ByteArrayOutputStream();
        ByteArrayOutputStream aPubOs = new ByteArrayOutputStream();
        try {
            RSA.generateKeys(ePrivOs, ePubOs);
            RSA.generateKeys(aPrivOs, aPubOs);
        }
        catch(IOException e) {
            fail("got IOException: " + e);
        }
        byte[] pubEnckey = ePubOs.toByteArray();
        byte[] privEnckey = ePrivOs.toByteArray();
        byte[] pubAuthkey = aPubOs.toByteArray();
        byte[] privAuthkey = aPrivOs.toByteArray();

        // initialize sms services
        DummySmsService sendService = new DummySmsService(sendMin);
        DummySmsService recvService = new DummySmsService(
                recvMin, sendService);
        sendService.setPeer(recvService);

        // initialize Core objects
        Connection sendConn =
                Connection.createInitiatorConnection(sendMin, recvMin);
        Connection recvConn =
                Connection.createResponderConnection(recvMin);
        SmsftpSession sendSession = SmsftpSession.generateSenderDataSession(
                sendConn, filename, dataBytes, privEnckey, privAuthkey);
        SmsftpSession recvSession = SmsftpSession.generateReceiverSession(
                recvConn);

        SmsftpFacade sendFacade = new SmsftpFacade(sendSession, sendService,
                4);
        SmsftpFacade recvFacade = new SmsftpFacade(recvSession, recvService,
                4);
        sendFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("ELTC-send")));
        recvFacade.addObserver(
                new SmsftpLoggerPresenter(LogManager.getLogger("ELTC-recv")));
        sendFacade.addObserver(new NullSmsftpPresenter(sendFacade));
        recvFacade.addObserver(new NullSmsftpPresenter(recvFacade, pubEnckey,
                pubAuthkey));

        // initialize, run, and join Runnables
        Runnable sendRunnable = () -> {
            sendFacade.secureSendFile();
        };
        Runnable recvRunnable = () -> {
            recvFacade.receive();
        };
        Utils.runAndJoin(sendRunnable, recvRunnable, LTC_TIMEOUT,
                "eLargeFileTransferShouldCompleteTest");

        // check if data received bytes' and original bytes' hashes match
        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("SHA-1");
        }
        catch(NoSuchAlgorithmException e) {
            fail("Problems initializing native SHA-1 digest");
        }
        md.update(dataBytes);
        byte[] srcHash = md.digest();
        md.update(recvFacade.getDataBytes());
        byte[] rcvHash = md.digest();
        assertTrue(Arrays.equals(srcHash, rcvHash));

        assertEquals(Connection.ConnectionState.CLOSED,
                sendConn.getConnectionState());
        assertEquals(Connection.ConnectionState.CLOSED,
                recvConn.getConnectionState());

        // sleep for a while to "synchronize" stdout
        Utils.sleepSilently(10);
        LogManager.getLogger("ELTC")
                .info("END of eLargeFileTransferShouldCompleteTest");
    }
}



--------------------------------------------------------------
Filename: src/test/integration/java/Utils.java
--------------------------------------------------------------

package com.transmisms.test.integration;

import java.util.concurrent.Executors;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.TimeUnit;

import static org.testng.AssertJUnit.fail;



/**
 * Utility class to contain helper methods for use by integration tests
 */
public class Utils {

    /**
     * Runs a pair of Runnables in background and waits for both at most
     * <code>timeoutMillis<code> milliseconds for the Runnables to complete.
     * A timeout of <code>0</code> means to wait forever.
     *
     * @param r1 one of the Runnables to be ran in the background
     * @param r2 one of the Runnables to be ran in the background
     * @param timeoutMillis time to wait in milliseconds
     * @param testName name of the test currently being run
     */
    public static void runAndJoin(Runnable r1, Runnable r2, long timeoutMillis,
            String testName) {
        ExecutorService es = Executors.newFixedThreadPool(2);
        es.submit(r1);
        es.submit(r2);
        es.shutdown();
        try {
            if(timeoutMillis == 0) {
                // for "infinite" wait
                es.awaitTermination(Long.MAX_VALUE, TimeUnit.HOURS);
            }
            else {
                es.awaitTermination(timeoutMillis, TimeUnit.MILLISECONDS);
            }
        }
        catch(InterruptedException e) {
            System.out.println("INTERRUPTION on " + testName + "!");
            fail("Test interrupted");
        }
    }

    /**
     * Generates encryption and authentication key pairs for use with
     * transmisms smsftp integration tests
     *
     * @return a four-member byte[][] containing keys in byte[] format in the
     *          order: public encryption key, private encryption key,
     *          public authentication key, and private authentication key
     */
    public static byte[][] generateEncAuthKeyPairs() {
        return null;
    }

    /**
     * Causes the currently executing thread to sleep for a specified number
     * of milliseconds, silently failing if interrupted.
     *
     * @param millis the length of time to sleep in milliseconds
     *
     * @see Thread
     */
    public static void sleepSilently(int millis) {
        try {
            Thread.sleep(10);
        }
        catch(InterruptedException e) {
            // do nothing
        }
    }
}
```

```
--------------------------------------------------------------
Filename: src/test/resources/log4j2-test.yaml
--------------------------------------------------------------

Configuration:
  status: warn

  Appenders:
    Console:
      name: Console
      target: SYSTEM_OUT
      PatternLayout:
        Pattern: "%d{HH:mm:ss.SSS} %-5level %logger{36} - %msg%n"

  Loggers:
    Root:
      level: trace
      AppenderRef:
        - ref: Console
```

```
--------------------------------------------------------------
Filename: src/test/staging/java/AndroidCompanionTest.java
--------------------------------------------------------------

package com.transmisms.test.staging;

import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.core.protocol.SmsServiceException;
import com.transmisms.ui.javafx.smsservice.AndroidSmsService;

import static org.testng.Assert.assertNotEquals;
import static org.testng.Assert.assertNotNull;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;
import org.testng.annotations.Test;

import java.util.concurrent.TimeUnit;



public class AndroidCompanionTest {
    public final static String TEST_HOST = "192.168.254.181";
    public final static int TEST_PORT = 8767;

    public final static String TEST_MSG = "help";
    public final static String TEST_RECIPIENT = "8080";
    public final static String OWN_MSISDN = "+639754314753";

    // ERROR TEST CASES
    @Test(expectedExceptions = SmsServiceException.class, timeOut = 7000,
            priority = 1)
    public void nonExistentServerShouldThrowSmsServiceErrorTest()
            throws SmsServiceException {
        System.out.println("Starting
nonExistentServerShouldThrowSmsServiceErrorTest");
        AndroidSmsService service =
                new AndroidSmsService("localhost", 65535);
        service.start();
        // we don't need to cleanup for non-existent connections
        System.out.println("End of
nonExistentServerShouldThrowSmsServiceErrorTest");
    }


    // NORMAL TEST CASES
    @Test(timeOut = 7000, priority = 2)
    public void startServiceTest()
            throws SmsServiceException, InterruptedException {
        System.out.println("Starting startServiceTest");
        Thread.sleep(1000); // sleep to avoid flooding the server with tests
        AndroidSmsService service =
                new AndroidSmsService(TEST_HOST, TEST_PORT);
        service.start();
        // cleanup
        service.stop();
        System.out.println("End of startServiceTest");
    }

    @Test(timeOut = 4000, priority = 3)
    public void msisdnTest()
            throws SmsServiceException, InterruptedException {
        System.out.println("Starting msisdnTest");
        Thread.sleep(1000); // sleep to avoid flooding the server with tests
        AndroidSmsService service =
                new AndroidSmsService(TEST_HOST, TEST_PORT);
        service.start();
        String msisdn = service.getLocalMin();
        assertEquals(OWN_MSISDN, msisdn);
        // cleanup
        service.stop();
        System.out.println("End of msisdnTest");
    }

    @Test(timeOut = 34000, priority = 4)
    public void sendRecvMessageTest()
            throws SmsServiceException, InterruptedException {
        System.out.println("Starting sendMessageTest");
        Thread.sleep(1000); // sleep to avoid flooding the server with tests
        AndroidSmsService service =
                new AndroidSmsService(TEST_HOST, TEST_PORT);
        service.start();

        // send message
        service.setPeer(TEST_RECIPIENT);
        service.sendMessage(TEST_MSG);

        Thread.sleep(1000); // sleep for a while to let the message be sent
        // try to receive any message
        service.setPeer(null);
        SmsEntry msg = null;
        while(msg == null) {
            msg = service.pollMessage(10, TimeUnit.SECONDS);
        }
        assertNotNull(msg);

        // cleanup
        service.stop();
        System.out.println("End of sendMessageTest");
    }
}
```

```
--------------------------------------------------------------
Filename: src/test/staging/java/GammuTest.java
--------------------------------------------------------------

package com.transmisms.test.staging;

import com.transmisms.core.protocol.SmsEntry;
import com.transmisms.core.protocol.SmsServiceException;
import com.transmisms.ui.javafx.smsservice.GammuSmsService;

import static org.testng.Assert.assertNotEquals;
import static org.testng.Assert.assertNotNull;
import static org.testng.AssertJUnit.assertEquals;
import static org.testng.AssertJUnit.assertTrue;
import static org.testng.AssertJUnit.assertFalse;
import static org.testng.AssertJUnit.fail;
import org.testng.annotations.Test;

import java.util.concurrent.TimeUnit;



public class GammuTest {
    public final static String TEST_HOST        = "localhost";
    public final static int TEST_PORT           = 5432;
    public final static String TEST_DB          = "smsd";
    public final static String TEST_USER        = "smsd";
    public final static String TEST_PASS        = "";
    public final static String TEST_MSISDN      = "9127654321";

    public final static String TEST_MSG         = "help";
    public final static String TEST_RECIPIENT   = "8080";



    // ERROR TEST CASES
    @Test(expectedExceptions = SmsServiceException.class, timeOut = 10000,
            priority = 1)
    public void nonExistentServerShouldThrowSmsServiceError()
            throws SmsServiceException {
        System.out.println("Starting
nonExistentServerShouldThrowSmsServiceErrorTest");
        GammuSmsService service =
                new GammuSmsService("localhost", 65535, TEST_DB, TEST_USER,
                TEST_PASS, TEST_MSISDN);
        service.start();
        // we don't need to cleanup for non-existent connections
        System.out.println("End of
nonExistentServerShouldThrowSmsServiceErrorTest");
    }


    // NORMAL TEST CASES
    @Test(timeOut = 10000, priority = 2)
    public void startServiceTest() throws SmsServiceException {
        System.out.println("Starting startServiceTest");
        GammuSmsService service =
                new GammuSmsService(TEST_HOST, TEST_PORT, TEST_DB, TEST_USER,
                TEST_PASS, TEST_MSISDN);
        service.start();
        // cleanup
        service.stop();
        System.out.println("End of startServiceTest");
    }

    @Test(timeOut = 34000, priority = 3)
    public void sendRecvMessageTest()
            throws SmsServiceException, InterruptedException {
        System.out.println("Starting sendMessageTest");
        GammuSmsService service =
                new GammuSmsService(TEST_HOST, TEST_PORT, TEST_DB, TEST_USER,
                TEST_PASS, TEST_MSISDN);
        service.start();

        // send message
        service.setPeer(TEST_RECIPIENT);
        service.sendMessage(TEST_MSG);

        Thread.sleep(1000); // sleep for a while to let the message be sent
        // try to receive any message
        service.setPeer(null);
        SmsEntry msg = null;
        while(msg == null) {
            msg = service.pollMessage(10, TimeUnit.SECONDS);
        }
        assertNotNull(msg);

        // cleanup
        service.stop();
        System.out.println("End of sendMessageTest");
    }
}



-------------------------------------------------------------
Filename: src/test/unit/java/AllTestsSuite.java
-------------------------------------------------------------

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;
import org.junit.experimental.categories.Categories;
import org.junit.experimental.categories.Categories.IncludeCategory;
import org.junit.experimental.categories.Categories.ExcludeCategory;

@IncludeCategory({ RegularTests.class, ExceptionTests.class })
@RunWith(Categories.class)
@SuiteClasses({ CoreProtocolTest.class,
                SmsftpTest.class,
                DecoderTest.class,
                BinaryDecoderTest.class,
                TextBasedDecoderTest.class,
                VersionTest.class,
                SessionTest.class,
                SegmentManagerTest.class,
                Base85Test.class,
                RSAPEMImportExportTest.class,
                BiometricFingerprintTest.class})
public class AllTestsSuite {}



-------------------------------------------------------------
Filename: src/test/unit/java/Base85Test.java
-------------------------------------------------------------

import com.transmisms.core.util.codec.Base85;

import java.util.Arrays;
import java.io.IOException;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.fail;

import org.junit.experimental.categories.Category;

import org.junit.Test;



public class Base85Test {
    @Category(RegularTests.class)
    @Test
    public void whiteSpaceShouldBeIrrelevant() throws IOException {
        // encoded string: &Ct>3]-''~
        byte[] magicNumbersA = { 16, -42, 99, 29, -69, 32, 56 };

        assertTrue(Arrays.equals(Base85.decode("&Ct>3]-''~"),
                Base85.decode("&Ct>3] -''~")));
        assertTrue(Arrays.equals(Base85.decode("&Ct>3]-''~"),
                Base85.decode("&Ct >3]\n-''~")));
        assertTrue(Arrays.equals(Base85.decode("&Ct>3]-''~"),
                Base85.decode(" & Ct  >3]\n -\n  ''~")));
    }

    @Category(RegularTests.class)
    @Test
    public void suffixesShouldBeIrrelevant() throws IOException {
        // checks for various inputs with combinations of suffixes
        //      a. ~
        //      b. ~>
        //      c. none

        /*
         * The following are the magic numbers behind our strings
         *   poT]46!?V86oONZ~ : { -8, -83, -110, 47, 65, 90, 66, -29, 68, 54,
-16, 35 };
         *   1h$7?@fPD#J$]~   : { 52, 97, -6, -30, 98, -1, -31, 104, 127, -76 };
         *   DX14]rT.k~       : { 110, -21, 83, 57, -3, -27, 109 };
         */
        assertTrue(Arrays.equals(Base85.decode("poT]46!?V86oONZ~"),
                Base85.decode("poT]46!?V86oONZ~>")));
        assertTrue(Arrays.equals(Base85.decode("poT]46!?V86oONZ~"),
                Base85.decode("poT]46!?V86oONZ")));

        assertTrue(Arrays.equals(Base85.decode("1h$7?@fPD#J$]~"),
                Base85.decode("1h$7?@fPD#J$]~>")));
        assertTrue(Arrays.equals(Base85.decode("1h$7?@fPD#J$]~"),
                Base85.decode("1h$7?@fPD#J$]")));

        assertTrue(Arrays.equals(Base85.decode("DX14]rT.k~"),
                Base85.decode("DX14]rT.k~>")));
        assertTrue(Arrays.equals(Base85.decode("DX14]rT.k~"),
                Base85.decode("DX14]rT.k")));
    }

    @Category(RegularTests.class)
    @Test
    public void nonGSM7CharactersShouldBeReplaced() throws IOException {
        assertTrue(Arrays.equals(Base85.decode("[pxu%"),
                                 Base85.decode("vpxu%"))); // [ -> v
        assertTrue(Arrays.equals(Base85.decode("\\NIHB"),
                                 Base85.decode("wNIHB"))); // \ -> w
        assertTrue(Arrays.equals(Base85.decode("P]L+<"),
                                 Base85.decode("PxL+<"))); // ] -> x
        assertTrue(Arrays.equals(Base85.decode("&^0ui"),
                                 Base85.decode("&y0ui"))); // ^ -> y
        assertTrue(Arrays.equals(Base85.decode("_k`!l"),
                                 Base85.decode("_k£!l"))); // ` -> £
    }

    @Category(RegularTests.class)
    @Test
    public void byteInputShouldReturnExpectedEncodedString()
            throws IOException {
        // Magic numbers and their corresponding encoded forms
        // WVIdN3D91@"[r~    if encoded
        byte[] magicNumbersA = { -87, -7, 35, -69, 57, 75, -7, -96, 5, 69 };
        // KbX_\:-"WM[7iKR~  if encoded
        byte[] magicNumbersB = { -123, 21, 4, 71, 78, 57, -106, -72, -75, 76,
54, -93 };
        // %o;>OX#'>~        if encoded
        byte[] magicNumbersC = { 15, 79, -26, -109, -85, 51, -13 };

        assertEquals("WVIdN3D91@\"vr",    Base85.encode(magicNumbersA));
        assertEquals("KbX_w:-\"WMv7iKR",  Base85.encode(magicNumbersB));
        assertEquals("%o;>OX#'>",         Base85.encode(magicNumbersC));
    }
}
```

```
--------------------------------------------------------------
Filename: src/test/unit/java/BinaryDecoderTest.java
--------------------------------------------------------------
```

```
// NOTE: For code readability, assertEquals() tests' arguments  here are on
//       reverse order (expected <-> was)
import com.transmisms.core.protocol.PDU;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.core.protocol.CorePDUType;
import com.transmisms.core.protocol.PDUMalformedException;
import com.transmisms.core.protocol.InvalidCRCException;
import com.transmisms.smsftp.protocol.SmsftpBinaryPduDecoder;
import com.transmisms.smsftp.protocol.SmsftpPDUType;

import java.util.Arrays;
import java.util.HashSet;
import java.util.Set;
import java.util.UUID;
import java.nio.ByteBuffer;


import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.fail;

import org.junit.experimental.categories.Category;
import org.junit.Test;



public class BinaryDecoderTest {
    // ERROR TEST CASES

    // General
    @Category(ExceptionTests.class)
    @Test(expected = NullPointerException.class)
    public void nullShouldThrowNullPointerException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(null);
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void emptyShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("");
        //                                                  PCr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void tooShortShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("1");
        //                                                  PCr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void invalidFirstCharShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("X8f");
        //                                                  PCr
    }

    @Category(ExceptionTests.class)
    @Test(expected = InvalidCRCException.class)
    public void invalidCRCShouldThrowInvalidCRCException()
            throws PDUMalformedException, InvalidCRCException  {
        // CRC of "r" should be 0x59
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("r58");
        //                                                  PCr
    }


    // CORE_FIN
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void coreFinShortSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "f12345678901234567 89db");
        //       P-Sessionid----------StatCr
    }

    @Category(ExceptionTests.class)
    @Test
    public void coreFinTruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        try {
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "fFX7%V4B>2:M~12>%W*h=050076");
            //       P-Sessionid----------StatCr
            fail("expected PDUMalformedException on truncated Session Id " +
                    "on SMSFTP_FIN: FX7%V4B>2:M~12>%W*h=");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // with Additional Message field
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "fFX7%V4B>2:M~12>%W*h=0500This is a test messagef9");
            //       P-Sessionid----------StatAdditional*Message****Cr
            fail("expected PDUMalformedException on truncated Session Id " +
                    "on SMSFTP_FIN: FX7%V4B>2:M~12>%W*h=");
        }
        catch(PDUMalformedException e) { } // do nothing
    }

    @Category(ExceptionTests.class)
    @Test
    public void coreFinMalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        try {
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "fFX7%V4B>2¿Mf12>%W*h=050068");
            //       P-Sessionid----------StatCr
            fail("expected PDUMalformedException on malformed Session Id " +
                    "on SMSFTP_FIN: FX7%V4B>2¿Mf12>%W*h=");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // with Additional Message field
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "fFX7%V4B>2¿Mf12>%W*h=0500This is a test messageb7");
            //       P-Sessionid----------StatAdditional*Message****Cr
            fail("expected PDUMalformedException on malformed Session Id " +
                    "on SMSFTP_FIN: FX7%V4B>2¿Mf12>%W*h=");
        }
        catch(PDUMalformedException e) { } // do nothing
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void coreFinShortStatusCodeShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "fFX7%V4B>2:Mf12>%W*h=ERRc5");
        //       P-Sessionid----------StatCr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void coreFinMalformedStatusCodeShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "fFX7%V4B>2:Mf12>%W*h=ERRRec");
        //       P-Sessionid----------StatCr
    }

    // CORE_FIN+ACK
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void coreFinackShortSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "F123456789012345678940");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void coreFinackTruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "FFX7%V4B>2:M~12>%W*h=41");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void coreFinackMalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "FFX7%V4B>2¿Mf12>%W*h=8d");
        //       P-Sessionid----------Cr
    }

    // CORE_RETRANSMISSION
    /*
     * NOTE: there might not need to have PDUMalformedException here since
     *       this is just a retransmission PDU; we must parse as much as we
     *       can and ignore the rest
     */


    // SMSFTP_INIT
    //// SMSFTP_INIT - General
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpInitGeneralEmptyShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("a20");
        //                                                PRCr
    }

    //// SMSFTP_INIT - Unencrypted connections
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpInitNoencShortSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "a123456789012345678891f");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpInitNoencTruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "aFX7%V4B>2:M~12>%W*h=db");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpInitNoencMalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "aFX7%V4B>2¿Mf12>%W*h=42");
        //       P-Sessionid----------Cr
    }

    //// SMSFTP_INIT - Encrypted connections
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpInitEnc1TruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"a01FX7%V4B>2:M~12>%W*h=12345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123345d
f");
        //
PSi-Sessionid----------Encrypted-aes-key-segment-part-1-------------------------
------------------------------------------------------------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpInitEnc1MalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"a01FX7%V4B>2¿Mf12>%W*h=12345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123345c
8");
        //
PSi-Sessionid----------Encrypted-aes-key-segment-part-1-------------------------
------------------------------------------------------------Cr
    }


    // SMSFTP_META
    //// SMSFTP_META - General
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpMetaGeneralEmptyShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("b29");
        //                                                PRCr <Invalid>
    }

    //// SMSFTP_META - Unencrypted connections
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpMetaNoencTruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "b00FX7%V4B>2:M~12>%W*h=XXCs8W*1234567890123456789012345 2d");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpMetaNoencMalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "b00FX7%V4B>2¿Mf12>%W*h=XXCs8W*12345678901234567890123 4571");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
    }

    // commented out because this might not make sense on padded "small"
    // segment counts
    /*
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpMetaNoencTruncatedSegmentCountShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "b001234567890123456 7890XXCs8~*12345678901234567890 12345??");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
    }
    */

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpMetaNoencMalformedSegmentCountShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "b001234567890123456 7890XXCs¿W*12345678901234567890 1234584");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
    }

    @Category(ExceptionTests.class)
    @Test
    public void smsftpMetaNoencInvalidCecFlagsShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        try { // XXZ
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b001234567890123456 7890XXZs8W*12345678901234567890 1234536");
            //       PSi-Sessionid----------CecSgctSha1---------------------Cr
            fail("expected PDUMalformedException on invalid CEC: XXZ");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // XZX
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b001234567890123456 7890XZXs8W*12345678901234567890 1234571");
```

```
            //       PSi-Sessionid----------CecSgctSha1---------------------Cr
            fail("expected PDUMalformedException on invalid CEC: XZX");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // CXX
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b001234567890123456789 0CXXs8W*1234567890123456789012345c7");
            //       PSi-Sessionid----------CecSgctSha1---------------------Cr
            fail("expected PDUMalformedException on invalid CEC: CXX");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // XEX
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0012345678901234567890XEXs8W*1234567890123456789012345d6");
            //       PSi-Sessionid----------CecSgctSha1---------------------Cr
            fail("expected PDUMalformedException on invalid CEC: XEX");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // XEC
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0012345678901234567890XECs8W*12345678901234567890123450 8");
            //       PSi-Sessionid----------CecSgctSha1---------------------Cr
            fail("expected PDUMalformedException on invalid CEC: XEC");
        }
        catch(PDUMalformedException e) { } // do nothing
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpMetaNoencShortSha1ShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "b0012345678901234567890XXCs8W*12345675a");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpMetaNoencLongSha1ShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0012345678901234567890XXCs8W*123456789012345678901234567890c1");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpMetaNoencTruncatedSha1ShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "b0012345678901234567890XXCs8W*12345678901234567~9012345c3");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpMetaNoencMalformedSha1ShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "b0012345678901234567890XXCs8W*123456¿890123456789012345e8");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
    }

    //// SMSFTP_META - Encrypted connections
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpMetaEnc1TruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b01123456789~1234567890CECs8W*1234567890123456789012345678901234567890123456789
01234567890123456789012345678901234567890123456789012345678901234567890123456782
e");
        //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpMetaEnc1MalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b01123456789012345¿7890CECs8W*1234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345678a
f");
        //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
    }

    @Category(ExceptionTests.class)
    @Test
    public void smsftpMetaEnc1InvalidCecFlagsShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        try { // XEZ
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0112345678901234567890XEZs8W*1234567890123456789012345678901234567890123456789
01234567890123456789012345678901234567890123456789012345678901234567890123456781
f");
            //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid CEC: XEZ");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // XZX
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0112345678901234567890XZXs8W*1234567890123456789012345678901234567890123456789
01234567890123456789012345678901234567890123456789012345678901234567890123456786
7");
            //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid CEC: XZX");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // ZEX
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0112345678901234567890ZEXs8W*1234567890123456789012345678901234567890123456789
01234567890123456789012345678901234567890123456789012345678901234567890123456786
b");
            //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid CEC: ZEX");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // CXC
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0112345678901234567890CXCs8W*1234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345678e
e");
            //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid CEC: CXC");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // XXX
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0112345678901234567890XXXs8W*1234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345678d
e");
            //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid CEC: XXX");
        }
        catch(PDUMalformedException e) { } // do nothing
    }


    // commented out because this might not make sense on padded "small"
// segment counts
/*
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpMetaEnc1TruncatedSegmentCountShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0112345678901234567890XXCs~W*1234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345678?
?");
        //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
}
```

```
    */

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpMetaEnc1MalformedSegmentCountShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"b011234567890123456789 0XXC¿8W*123456789012345678901234567890123456789012345678 9
012345678901234567890123456789012345678901234567890123456789012345678901234567 80
7");
        //
PSi-Sessionid----------CecSgctRsa-signature-segment----------------------------
-----------------------------------------------------------Cr
    }


    // SMSFTP_READY
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpReadyShortSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "c1234567890123456789 47");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpReadyTruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "c12345678901234~6789000");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpReadyMalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "c1234567890123456¿89038");
        //       P-Sessionid----------Cr
    }


    // SMSFTP_END
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpEndShortSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "e1234567890123456789af");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpEndTruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "e12345678901234~6789096");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpEndMalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "e1234567890123456¿890d3");
        //       P-Sessionid----------Cr
    }


    // SMSFTP_END_WAIT
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpEndwaitShortSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "E1234567890123456789 34");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpEndwaitTruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
```

```
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "E12345678901234~678905e");
        //       P-Sessionid----------Cr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
smsftpEndwaitMalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "E1234567890123456¿890a5");
        //       P-Sessionid----------Cr
    }


    // SMSFTP_DATA
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpDataShortSegmentIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("ds8*50");
        //                                                  PSgidCr
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpDataMalformedSegmentIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("ds8¿*c9");
        //                                                  PSgidCr
    }

    // Reason for commenting out: should be impossible to happen + requires
    // more processing
    // - SMSFTP_DATA, SMSFTP_INIT, and SMSFTP_META, SMSFTP_INITIATOR_IDENTITY,
    //   and SMSFTP_RESPONDER_IDENTITY suffers from the same dilemma
    // - for SMSFTP_META.noenc, SHA1 can be converted and checked for
    //   byte[].length
    /*
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void smsftpDataMalformedBodyShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        SmsftpBinaryPduDecoder.getInstance().decodeBinary("ds8W*¿df");
        //                                                  PSgidBCr
    }
    */



    // NORMAL TEST CASES
    // CORE_FIN
    @Category(RegularTests.class)
    @Test
    public void coreFinValuesShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        { // without Additional Message field
            // this is UUID 752500ef3c564de38b6f2d9f0e6d2e72
            PDU finPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "fFX7%V4B>2:Mf12>%W*h=000172");
            //       P-Sessionid----------StatCr

            assertEquals(finPDU.getPduType(), CorePDUType.CORE_FIN);
            Object[] data = finPDU.getData();
            assertEquals(data.length, 2);
            // Session Id
            assertTrue(data[0] instanceof UUID);
            assertEquals((UUID)data[0], hexStringsToUUID(
                    "752500ef3c564de3", "8b6f2d9f0e6d2e72"));
            // Status Code
            assertTrue(data[1] instanceof String);
            assertTrue("0001".equals((String)(data[1])));
        }
        { // with Additional Message field
            PDU finPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "fFX7%V4B>2:Mf12>%W*h=0001This is a test messageb6");
            //       P-Sessionid----------StatAdditional*Message****Cr
            assertEquals(finPDU.getPduType(), CorePDUType.CORE_FIN);
            Object[] data = finPDU.getData();
            assertTrue(data.length == 3);
            // Session Id
            assertTrue(data[0] instanceof UUID);
            assertEquals((UUID)data[0], hexStringsToUUID(
                    "752500ef3c564de3", "8b6f2d9f0e6d2e72"));
            // Status Code
            assertTrue(data[1] instanceof String);
            assertTrue("0001".equals((String)(data[1])));
            // Additional Message
            assertTrue(data[2] instanceof String);
            assertTrue("This is a test message".equals((String)(data[2])));
    }
}


// CORE_FIN+ACK
@Category(RegularTests.class)
@Test
public void coreFinackValuesShouldBeReturned()
        throws PDUMalformedException, InvalidCRCException  {
    // this is UUID 752500ef3c564de38b6f2d9f0e6d2e72
    PDU finackPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
            "FFX7%V4B>2:Mf12>%W*h=05");
    //       P-Sessionid----------Cr
    assertEquals(finackPDU.getPduType(), CorePDUType.CORE_FINACK);
    Object[] data = finackPDU.getData();
    assertTrue(data.length == 1);
    // Session Id
    assertTrue(data[0] instanceof UUID);
    assertEquals((UUID)data[0], hexStringsToUUID(
                "752500ef3c564de3", "8b6f2d9f0e6d2e72"));
}


// CORE_RETRANSMISSION
@Category(RegularTests.class)
@Test
public void coreRetransmissionValuesShouldBeReturned()
        throws PDUMalformedException, InvalidCRCException  {
    { // empty
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "r59");
        //       PCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 0);
    }
    { // single hex
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "ra01de");
        //       PPSiCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 2);
        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], SmsftpPDUType.SMSFTP_INIT);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 1);
    }
    { // single base85
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "rd000083");
        //       PPSgidCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 2);
        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 3095042);
    }
    { // multi hex
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "ra03b01c00e3");
        //       PPSiPSiPSiCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 6);

        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], SmsftpPDUType.SMSFTP_INIT);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 3);

        assertTrue(data[2] instanceof PDUType);
        assertEquals(data[2], SmsftpPDUType.SMSFTP_META);
        assertTrue(data[3] instanceof Integer);
        assertEquals(((Integer)data[3]).intValue(), 1);

        assertTrue(data[4] instanceof PDUType);
        assertEquals(data[4], SmsftpPDUType.SMSFTP_READY);
        assertTrue(data[5] instanceof Integer);
        assertEquals(((Integer)data[5]).intValue(), 0);
    }
    { // multi base85
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "rd0000d0001dtdrl75");
        //       PPSgidPSgidPSgidCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 6);
        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 3095042);

        assertTrue(data[2] instanceof PDUType);
        assertEquals(data[2], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[3] instanceof Integer);
        assertEquals(((Integer)data[3]).intValue(), 3095043);

        assertTrue(data[4] instanceof PDUType);
        assertEquals(data[4], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[5] instanceof Integer);
        assertEquals(((Integer)data[5]).intValue(), 310244);
    }
    { // multi mixed
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "ra03d0001dtdrlb00f00dkana75");
        //       PPSiPSgidPSgidPSiPSiPSgidCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 12);

        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], SmsftpPDUType.SMSFTP_INIT);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 3);

        assertTrue(data[2] instanceof PDUType);
        assertEquals(data[2], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[3] instanceof Integer);
        assertEquals(((Integer)data[3]).intValue(), 3095043);

        assertTrue(data[4] instanceof PDUType);
        assertEquals(data[4], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[5] instanceof Integer);
        assertEquals(((Integer)data[5]).intValue(), 310244);

        assertTrue(data[6] instanceof PDUType);
        assertEquals(data[6], SmsftpPDUType.SMSFTP_META);
        assertTrue(data[7] instanceof Integer);
        assertEquals(((Integer)data[7]).intValue(), 0);

        assertTrue(data[8] instanceof PDUType);
        assertEquals(data[8], CorePDUType.CORE_FIN);
        assertTrue(data[9] instanceof Integer);
        assertEquals(((Integer)data[9]).intValue(), 0);

        assertTrue(data[10] instanceof PDUType);
        assertEquals(data[10], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[11] instanceof Integer);
        assertEquals(((Integer)data[11]).intValue(), 15244969);
    }
}


@Category(RegularTests.class)
@Test
public void coreRetransmissionImperfectValuesShouldBeReturned()
        throws PDUMalformedException, InvalidCRCException  {
    { // last segment cut short
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "ra01f00dwat36");
        //       PPSiPSiPSgidCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 4);

        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], SmsftpPDUType.SMSFTP_INIT);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 1);

        assertTrue(data[2] instanceof PDUType);
        assertEquals(data[2], CorePDUType.CORE_FIN);
        assertTrue(data[3] instanceof Integer);
        assertEquals(((Integer)data[3]).intValue(), 0);
        // remaining is unreadable
    }
    { // invalid PDU prefix
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "ra03x01c0060");
        //       PPSiPSiPSiCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 4);

        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], SmsftpPDUType.SMSFTP_INIT);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 3);
        // 'x' PDU is unknown -> continue

        assertTrue(data[2] instanceof PDUType);
        assertEquals(data[2], SmsftpPDUType.SMSFTP_READY);
        assertTrue(data[3] instanceof Integer);
        assertEquals(((Integer)data[3]).intValue(), 0);
    }
    { // invalid hex (format)
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "rf00ixxa00e3");
        //       PPSiPSiPSiCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 4);

        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], CorePDUType.CORE_FIN);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 0);

        // 'xx' hex is invalid

        assertTrue(data[2] instanceof PDUType);
        assertEquals(data[2], SmsftpPDUType.SMSFTP_INIT);
        assertTrue(data[3] instanceof Integer);
        assertEquals(((Integer)data[3]).intValue(), 0);
    }
    { // invalid segment id (format)
        PDU retPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "ra01ds8W¿dtdrlb00f00dkanaca");
        //       PPSiPSgidPSgidPSiPSiPSgidCr
        assertEquals(retPDU.getPduType(), CorePDUType.CORE_RETRANSMISSION);
        Object[] data = retPDU.getData();
        assertEquals(data.length, 10);

        assertTrue(data[0] instanceof PDUType);
        assertEquals(data[0], SmsftpPDUType.SMSFTP_INIT);
        assertTrue(data[1] instanceof Integer);
        assertEquals(((Integer)data[1]).intValue(), 1);

        // 's8W¿' segment id is invalid

        assertTrue(data[2] instanceof PDUType);
        assertEquals(data[2], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[3] instanceof Integer);
        assertEquals(((Integer)data[3]).intValue(), 310244);

        assertTrue(data[4] instanceof PDUType);
        assertEquals(data[4], SmsftpPDUType.SMSFTP_META);
        assertTrue(data[5] instanceof Integer);
        assertEquals(((Integer)data[5]).intValue(), 0);

        assertTrue(data[6] instanceof PDUType);
        assertEquals(data[6], CorePDUType.CORE_FIN);
        assertTrue(data[7] instanceof Integer);
        assertEquals(((Integer)data[7]).intValue(), 0);

        assertTrue(data[8] instanceof PDUType);
        assertEquals(data[8], SmsftpPDUType.SMSFTP_DATA);
        assertTrue(data[9] instanceof Integer);
        assertEquals(((Integer)data[9]).intValue(), 15244969);
    }
    // NOTE: for invalid segment id for specific PDU (i.e. '4' for 'i'
    //       PDU, it should be simply ignored by the protocol
}


// SMSFTP_INIT
@Category(RegularTests.class)
@Test
public void smsftpInitNoencValuesShouldBeReturned()
        throws PDUMalformedException, InvalidCRCException  {
    PDU initPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
            "a0012345678901234567890c1");
    //       PSi-Sessionid----------Cr
    assertEquals(initPDU.getPduType(), SmsftpPDUType.SMSFTP_INIT);
    Object[] data = initPDU.getData();
    assertEquals(data.length, 2);
    assertTrue(data[0] instanceof Integer);
    assertEquals(data[0], 0);
    // Session Id
    assertTrue(data[1] instanceof UUID);
    assertEquals((UUID)data[1], hexStringsToUUID(
                "32699b3242279c09", "32699b3242279c09"));
}

@Category(RegularTests.class)
@Test
public void smsftpInitEnc1ValuesShouldBeReturned()
        throws PDUMalformedException, InvalidCRCException  {
    PDU initPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"a011234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123458
2");
        //
PSi-Sessionid----------Encrypted-aes-key-segment-part-1-------------------------
--------------------------------------------------------------Cr
        assertEquals(initPDU.getPduType(), SmsftpPDUType.SMSFTP_INIT);
        Object[] data = initPDU.getData();
        assertEquals(data.length, 3);
        assertTrue(data[0] instanceof Integer);
        assertEquals(data[0], 1);
        // Session Id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
        // Encrypted AES Segment
        assertTrue(data[2] instanceof String);
        assertEquals((String)data[2],
"123456789012345678901234567890123456789012345678901234567890123456789012345678 9
0123456789012345678901234567890123456789012345");
    }

    @Category(RegularTests.class)
    @Test
    public void smsftpInitEnc2and3ValuesShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        { // part 2
            PDU initPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"a021234567890123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123456789012345f
3");
            //
PSi-Encrypted-aes-key-segment-part-2--------------------------------------------
----------------------------------------------------Cr
            assertEquals(initPDU.getPduType(), SmsftpPDUType.SMSFTP_INIT);
            Object[] data = initPDU.getData();
            assertEquals(data.length, 3);
            assertTrue(data[0] instanceof Integer);
            assertEquals(data[0], 2);
            // Session Id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                        "32699b3242279c09", "32699b3242279c09"));
            // Encrypted AES Segment
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345");
        }
        { // part 3
            PDU initPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"a031234567890123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123456789012345d
c");
            //
PSi-Encrypted-aes-key-segment-part-3--------------------------------------------
----------------------------------------------------Cr
            assertEquals(initPDU.getPduType(), SmsftpPDUType.SMSFTP_INIT);
            Object[] data = initPDU.getData();
            assertEquals(data.length, 3);
            assertTrue(data[0] instanceof Integer);
            assertEquals(data[0], 3);
            // Session Id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                        "32699b3242279c09", "32699b3242279c09"));
            // Encrypted AES Segment
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345");
        }
    }


    // SMSFTP_META
    @Category(RegularTests.class)
    @Test
    public void smsftpMetaNoencValuesShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        PDU metaPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "b0012345678901234567890XXCs8W*s8W-!s8W-!s8W-!s8W-!s8W-!67");
        //       PSi-Sessionid----------CecSgctSha1---------------------Cr
        assertEquals(metaPDU.getPduType(), SmsftpPDUType.SMSFTP_META);
        Object[] data = metaPDU.getData();
        assertEquals(data.length, 5);
        assertTrue(data[0] instanceof Integer);
        assertEquals(data[0], 0);
        // Session Id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
        // CEC
        assertTrue(data[2] instanceof String);
        assertEquals("XXC", data[2]);
        // Segment Count
        assertTrue(data[3] instanceof Integer);
        assertEquals((Integer)data[3], new Integer(16777215));
        // SHA1 Checksum
        assertTrue(data[4] instanceof byte[]);
        assertTrue(Arrays.equals((byte[])data[4],
                    javax.xml.bind.DatatypeConverter.parseHexBinary(
                    "ffffffffffffffffffffffffffffffffffffffff")));
    }

    @Category(RegularTests.class)
    @Test
    public void smsftpMetaEnc1ValuesShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        // Segment count is 16777215 (or 'ffffff')
        PDU metaPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0112345678901234567890CECs8W*123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456787
d");
        //
PSi-Sessionid----------CecSgctRsa-signature-segment-------------------
-------------------------------------------------------------Cr
        assertEquals(metaPDU.getPduType(), SmsftpPDUType.SMSFTP_META);
        Object[] data = metaPDU.getData();
        assertEquals(data.length, 5);
        assertTrue(data[0] instanceof Integer);
        assertEquals(data[0], 1);
        // Session id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
        // CEC
        assertTrue(data[2] instanceof String);
        assertEquals(data[2], "CEC");
        // Segment count
        assertTrue(data[3] instanceof Integer);
        assertEquals((Integer)data[3], new Integer(16777215));
        // RSA Signature Segment
        assertTrue(data[4] instanceof String);
        assertEquals((String)data[4],
"1234567890123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345678");
    }

    @Category(RegularTests.class)
    @Test
    public void smsftpMetaEnc2and3RsaSigSegmentShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        { // part 2
            PDU metaPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0212345678901234567890123456789012345678901234567890123456789012345 6
7890123456789012345678901234567890123456789012345678901234567890123454
4");
            //
PSi-Rsa-signature-segment---------------------------------------------
---------------------------------------------------Cr
            assertEquals(metaPDU.getPduType(), SmsftpPDUType.SMSFTP_META);
            Object[] data = metaPDU.getData();
            assertEquals(data.length, 3);
            assertTrue(data[0] instanceof Integer);
            assertEquals(data[0], 2);
            // Session id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                        "32699b3242279c09", "32699b3242279c09"));
            // RSA Signature segment
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2],
"1234567890123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345");
        }
        { // part 3
            PDU metaPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0312345678901234567890123456789012345678901234567890123456789012345 6
7890123456789012345678901234567890123456789012345678901234567890123456
b");
            //
PSi-Rsa-signature-segment---------------------------------------------
---------------------------------------------------Cr
            assertEquals(metaPDU.getPduType(), SmsftpPDUType.SMSFTP_META);
            Object[] data = metaPDU.getData();
```

```
            assertEquals(data.length, 3);
            assertTrue(data[0] instanceof Integer);
            assertEquals(data[0], 3);
            // Session id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                        "32699b3242279c09", "32699b3242279c09"));
            // RSA Signature segment
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2],
"1234567890123456789012345678901234567890123456789012345678901234567890
0123456789012345678901234567890123456789012345");
        }
    }


    // SMSFTP_READY
    @Category(RegularTests.class)
    @Test
    public void smsftpReadyValuesShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        PDU readyPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                "c12345678901234567890 42");
        //       P-Sessionid- --------Cr
        assertEquals(readyPDU.getPduType(), SmsftpPDUType.SMSFTP_READY);
        Object[] data = readyPDU.getData();
        assertEquals(data.length, 1);
        // Session id
        assertTrue(data[0] instanceof UUID);
        assertEquals((UUID)data[0], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
    }


    // SMSFTP_DATA
    @Category(RegularTests.class)
    @Test
    public void smsftpDataValuesShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        { // emtpy Data (this is still perfectly acceptable :) )
            // Segment Id in hex: "ffffff"
            PDU dataPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "ds8W*2a");
            //       PSgidCr
            assertEquals(dataPDU.getPduType(), SmsftpPDUType.SMSFTP_DATA);
            Object[] data = dataPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 16777215);
        }
        { // with Data
            // Segment Id in hex: "ffffff"
            PDU dataPDU =
                    SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "ds8W*Sample Bodycb");
            //       PSgid-Data------Cr
            assertEquals(dataPDU.getPduType(), SmsftpPDUType.SMSFTP_DATA);
            Object[] data = dataPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 16777215);
            // Data
            assertTrue(data[1] instanceof String);
            assertEquals(data[1], "Sample Body");
        }
    }


    // SMSFTP_INITIATOR_IDENTITY
    @Category(RegularTests.class)
    @Test
    public void smsftpInitiatorIdentityValuesShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        { // part 1
            PDU iidPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"i0112345678901234567890123456789012345678901234567890123456789012345 6
78901234567890123456789012345678901234567890123456789012345678901234 5e
3");
            //
PSi-Rsa-public-key----------------------------------------------------
------------------------------------------------Cr
            assertEquals(iidPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY);
            Object[] data = iidPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 1);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
        { // part 2
            PDU iidPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"i021234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123459
2");
            //
PSi-Rsa-public-key--------------------------------------------------------------
----------------------------------------------------Cr
            assertEquals(iidPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY);
            Object[] data = iidPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 2);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
        { // part 3
            PDU iidPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"i031234567890123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123456789012345b
d");
            //
PSi-Rsa-public-key--------------------------------------------------------------
----------------------------------------------------Cr
            assertEquals(iidPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY);
            Object[] data = iidPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 3);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
        { // part 4
            PDU iidPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"i041234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123457
0");
            //
PSi-Rsa-public-key--------------------------------------------------------------
----------------------------------------------------Cr
            assertEquals(iidPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY);
            Object[] data = iidPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 4);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
        { // part 5
            PDU iidPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"i051234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123455
f");
            //
PSi-Rsa-public-key--------------------------------------------------------------
----------------------------------------------------Cr
            assertEquals(iidPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_INITIATOR_IDENTITY);
            Object[] data = iidPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 5);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
    }

    // SMSFTP_RESPONDER_IDENTITY
    @Category(RegularTests.class)
    @Test
    public void smsftpResponderIdentityValuesShouldBeReturned()
            throws PDUMalformedException, InvalidCRCException  {
        { // part 1
            PDU ridPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"j011234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123455
4");
            //
PSi-Rsa-public-key--------------------------------------------------------------
----------------------------------------------------Cr
            assertEquals(ridPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY);
            Object[] data = ridPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 1);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
        { // part 2
            PDU ridPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"j021234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123452
5");
            //
PSi-Rsa-public-key--------------------------------------------------------------
----------------------------------------------------Cr
            assertEquals(ridPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY);
            Object[] data = ridPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 2);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
        { // part 3
            PDU ridPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"j031234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123450
a");
            //
PSi-Rsa-public-key--------------------------------------------------------------
----------------------------------------------------Cr
            assertEquals(ridPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY);
            Object[] data = ridPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 3);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
        { // part 4
            PDU ridPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"j041234567890123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123456789012345c
7");
            //
PSi-Rsa-public-key--------------------------------------------------------------
----------------------------------------------------Cr
            assertEquals(ridPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY);
            Object[] data = ridPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 4);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
01234567890123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
        { // part 5
            PDU ridPDU = SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"j051234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123456
8");
            //
PSi-Rsa-public-key-----------------------------------------------------------
--------------------------------------------------Cr
            assertEquals(ridPDU.getPduType(),
                    SmsftpPDUType.SMSFTP_RESPONDER_IDENTITY);
            Object[] data = ridPDU.getData();
            assertEquals(data.length, 2);
            // Segment Id
            assertTrue(data[0] instanceof Integer);
            assertEquals(((Integer)data[0]).intValue(), 5);
            // RSA Public Key
            assertTrue(data[1] instanceof String);
            assertEquals((String)data[1],
"1234567890123456789012345678901234567890123456789012345678901234567890123456789
01234567890123456789012345678901234567890123456789012345678901234567890123456789012345");
        }
    }


    // Multi-tests
    //// Includes the following:
    //// - SMSFTP_INIT
    //// - SMSFTP_META
    //// - SMSFTP_INITIATOR_IDENTITY
    //// - SMSFTP_RESPONDER_IDENTITY
    //// - SMSFTP_DATA
    @Category(ExceptionTests.class)
    @Test
    public void smsftpMultiInvalidSegmentIdShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException  {
        //// SMSFTP_INIT
        try { // 04
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"a041234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123451
1");
            //
PSi-Sessionid----------Encrypted-aes-key-segment-part-1----------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_INIT: 04");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // 0X
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"a0X1234567890123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123456789012345d
8");
            //
PSi-Sessionid----------Encrypted-aes-key-segment-part-1----------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_INIT: 0X");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // G0
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"aG01234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123450
2");
            //
PSi-Sessionid----------Encrypted-aes-key-segment-part-1----------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_INIT: G0");
        }
        catch(PDUMalformedException e) { } // do nothing

        //// SMSFTP_META
        try { // 04
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0412345678901234567890CECs8W*1234567890123456789012345678901234567890123456789
012345678901234567890123456789012345678901234567890123456789012345678901234567e
e");
            //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_META: 04");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // 0X
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"b0X12345678901234567890CECs8W*1234567890123456789012345678901234567890123456789
01234567890123456789012345678901234567890123456789012345678901234567890123456782
7");
            //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_META: 0X");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // G0
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"bG012345678901234567890CECs8W*1234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789012345678f
d");
            //
PSi-Sessionid----------CecSgctRsa-signature-segment-----------------------------
-----------------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_META: G0");
        }
        catch(PDUMalformedException e) { } // do nothing

        //// SMSFTP_INITIATOR_IDENTITY
        try { // 00
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"i001234567890123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123456789012345c
c");
            //
PSi-RSA-public-key-------------------------------------------------------------
--------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_INITIATOR_IDENTITY: 00");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // 06
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"i061234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123452
e");
            //
PSi-RSA-public-key-------------------------------------------------------------
--------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_INITIATOR_IDENTITY: 06");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // 0X
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"i0X1234567890123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123456789012345b
9");
            //
PSi-RSA-public-key-------------------------------------------------------------
--------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_INITIATOR_IDENTITY: 0X");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // G0
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"iG01234567890123456789012345678901234567890123456789012345678901234567890123456
78901234567890123456789012345678901234567890123456789012345678901234567890123456
3");
            //
PSi-RSA-public-key-------------------------------------------------------------
--------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_INITIATOR_IDENTITY: G0");
        }
        catch(PDUMalformedException e) { } // do nothing

        //// SMSFTP_RESPONDER_IDENTITY
        try { // 00
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(
"j00123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123457
b");
          //
PSi-RSA-public-key------------------------------------------------------------
--------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_RESPONDER_IDENTITY: 00");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // 06
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"j06123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123459
9");
          //
PSi-RSA-public-key------------------------------------------------------------
--------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_RESPONDER_IDENTITY: 06");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // 0X
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"j0X123456789012345678901234567890123456789012345678901234567890123456
7890123456789012345678901234567890123456789012345678901234567890123450
e");
          //
PSi-RSA-public-key------------------------------------------------------------
--------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_RESPONDER_IDENTITY: 0X");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // G0
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(

"jG0123456789012345678901234567890123456789012345678901234567890123456
789012345678901234567890123456789012345678901234567890123456789012345d
4");
          //
PSi-RSA-public-key------------------------------------------------------------
--------------------------------------------------Cr
            fail("expected PDUMalformedException on invalid Segment Id " +
                    "on SMSFTP_RESPONDER_IDENTITY: G0");
        }
        catch(PDUMalformedException e) { } // do nothing

        //// SMSFTP_DATA
        try { // malformed
            SmsftpBinaryPduDecoder.getInstance().decodeBinary(
                    "ds8W¿Sample Body03");
            //       PSgid-Data------Cr
            fail("expected PDUMalformedException on malformed Segment Id " +
                    "on SMSFTP_RESPONDER_IDENTITY: s8W¿");
        }
        catch(PDUMalformedException e) { } // do nothing
    }

    /**
     *
     *
     */
    private UUID hexStringsToUUID(String msb, String lsb) {
        ByteBuffer msbBuffer = ByteBuffer.allocate(Long.BYTES).put(
                javax.xml.bind.DatatypeConverter.parseHexBinary(msb));
        ByteBuffer lsbBuffer = ByteBuffer.allocate(Long.BYTES).put(
                javax.xml.bind.DatatypeConverter.parseHexBinary(lsb));
        msbBuffer.flip(); // required for reading
        lsbBuffer.flip(); // required for reading
        return new UUID(msbBuffer.getLong(), lsbBuffer.getLong());
    }
}



-------------------------------------------------------------
Filename: src/test/unit/java/BiometricFingerprintTest.java
-------------------------------------------------------------

import com.transmisms.core.util.crypto.BiometricFingerprint;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.fail;
import org.junit.experimental.categories.Category;

import org.junit.Test;



public class BiometricFingerprintTest {
    @Category(RegularTests.class)
    @Test
    public void fingerprintShouldBeReturned() {
        byte[] testInput1 = {};
        String testResult1 = "surrender classroom pandemic tycoon finicky " +
                             "glitter disable ancient component edict " +
                             "rebellion uncut Montana facial borderline " +
                             "peachy pharmacy stormy amusement Algol";
        assertEquals(testResult1,
                BiometricFingerprint.getBiometricFingerprint(testInput1));

        byte[] testInput2 = {0x46, 0x75, 0x63, 0x6b, 0x0a};
        String testResult2 = "speculate bluebird millionaire spearhead " +
                             "graduate offload hurricane music repellent " +
                             "backfield Hamilton tissue combustion " +
                             "sweatband Atlantic merit pedigree Geiger " +
                             "Apollo quiver";
        assertEquals(testResult2,
                BiometricFingerprint.getBiometricFingerprint(testInput2));
    }

    @Category(ExceptionTests.class)
    @Test(expected = NullPointerException.class)
    public void nullShouldThrowNullPointerException() {
        BiometricFingerprint.getBiometricFingerprint(null);
    }
}




-------------------------------------------------------------
Filename: src/test/unit/java/CoreProtocolTest.java
-------------------------------------------------------------

import com.transmisms.core.protocol.CoreLookupTable;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;

import org.junit.experimental.categories.Category;

import org.junit.Test;



public class CoreProtocolTest {

    @Category(RegularTests.class)
    @Test
    public void coreLookupTableHealthCheck() {
        CoreLookupTable clt = new CoreLookupTable();
    }
}




-------------------------------------------------------------
Filename: src/test/unit/java/DecoderTest.java
-------------------------------------------------------------

import com.transmisms.core.protocol.Connection;
import com.transmisms.core.protocol.CoreProtocolFacade;
import com.transmisms.core.protocol.CorePDUType;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.smsftp.protocol.SmsftpFacade;
import com.transmisms.smsftp.protocol.SmsftpSession;

import com.transmisms.core.protocol.PDUMalformedException;
import com.transmisms.core.protocol.InvalidCRCException;

import java.util.HashSet;
import java.util.Set;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.fail;
import org.junit.experimental.categories.Category;

import org.junit.Test;


public class DecoderTest {
    private static CoreProtocolFacade generateTestFacade() {
        return new SmsftpFacade(SmsftpSession.generateInitiatorPairSession(
                Connection.createResponderConnection(null)), null);
    }

    // ERROR TEST CASES
    @Category(ExceptionTests.class)
    @Test(expected = NullPointerException.class)
    public void nullShouldThrowNullPointerException()
            throws PDUMalformedException, InvalidCRCException {
        CoreProtocolFacade f = DecoderTest.generateTestFacade();
        f.decode(null);
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void emptyShouldThrowPDUMalformedException()
            throws PDUMalformedException, InvalidCRCException {
        CoreProtocolFacade f = DecoderTest.generateTestFacade();
        f.decode("");
    }


    // NORMAL TEST CASES
    // HUMAN_REJECT
    @Category(RegularTests.class)
    @Test
    public void noShouldReturnHumanRejectPdu()
            throws PDUMalformedException, InvalidCRCException {
        CoreProtocolFacade f = DecoderTest.generateTestFacade();
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode("NO").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode("no").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode("No").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode("nO").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode(" NO ").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode("NO ").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode(" NO").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode("NO\n").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT,
                f.decode("NO\n\n").getPduType());
        assertEquals(CorePDUType.HUMAN_REJECT, f.decode("NO \n").getPduType());
    }
}



--------------------------------------------------------------
Filename: src/test/unit/java/ExceptionTests.java
--------------------------------------------------------------

public interface ExceptionTests {}




--------------------------------------------------------------
Filename: src/test/unit/java/RegularTests.java
--------------------------------------------------------------

public interface RegularTests {}




--------------------------------------------------------------
Filename: src/test/unit/java/RSAPEMImportExportTest.java
--------------------------------------------------------------

import com.transmisms.core.util.crypto.RSA;
import com.transmisms.core.util.crypto.RSADecoderException;

import org.apache.commons.codec.binary.Base64;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Arrays;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.fail;
import org.junit.experimental.categories.Category;

import org.junit.Test;


public class RSAPEMImportExportTest {

    @Category(RegularTests.class)
    @Test
    public void x509ShouldBeReturned() throws RSADecoderException {
        byte[] pemFileByteArray1 = ("-----BEGIN PUBLIC KEY-----\n" +
        "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1SgRBuihq+k9vWwuGBlE\n" +
        "rB4XntzDtTlE53wL48F/Re49BrX8X2cczneLwGLDSTj34w0AZ2zHK9Ue/NmpLtrv\n" +
        "sxVaFzhFOLZ7o9kcI/ncbNktlyrNmyeMDL0tZI84lvTWfcxf7pFxaXotef+y5RIm\n" +
        "RYMOI5kv00dSgujHhM/ToUar2/bITKVVo7bhsA1hgat7YeuD7EJBPb6n061IPNtN\n" +
        "Ew0TonCp82kqg+bYKOssTXcxMsuOBGvp+ZwNkneY7b2IzF7ZtDs746FBNF9qLI0B\n" +
        "nqBdUA0n9JdPJCltIYaMmUIgtcv9tXjbpY1Y3FAIh4/Mg4J7KoGWIRoXXGrcjiW/\n" +
        "7QIDAQAB\n" +
        "-----END PUBLIC KEY-----").getBytes();

        byte[] pemFileByteArray2 = ("-----BEGIN PUBLIC KEY-----\n" +
        "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA3wgs9G9FWccRDQd9d++n\n" +
        "OT6HGsrHDzdyw63HoCQMtMm2mibDAWAV5m9bi7KmFY6wralYmPADpzp2s0O+CFb6\n" +
        "1X/H7Ftm9En79mJZEy0BPbNd+8+/AFd0v1q3Jzx8wOhm92+OyICGXheKMAXDx944\n" +
        "lgAAFPxyYOVwnP+Xu2B/odjniQNr0NM0ZsJRGcsIrKfOzuVBiXAMtECR9eFGVIuB\n" +
        "1U2xNzrptjaPXQef6ETWNuCejaN/3DmmZXOfZRB/cWIhrnFCspi1yYDI0NJpUmS5\n" +
        "THShYpF4hjbwrETbXW75CNXSa0KK38YvPaspprR6ja0a4I3+Gt20hwdw2afHW2Kg\n" +
        "QQIDAQAB\n" +
        "-----END PUBLIC KEY-----").getBytes();

        byte[] expectedX509Format1 = Base64.decodeBase64(
                "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1SgRBuihq+k9vW" +
                "wuGBlErB4XntzDtTlE53wL48F/Re49BrX8X2cczneLwGLDSTj34w0AZ2zH" +
                "K9Ue/NmpLtrvsxVaFzhFOLZ7o9kcI/ncbNktlyrNmyeMDL0tZI84lvTWfc" +
                "xf7pFxaXotef+y5RImRYMOI5kv00dSgujHhM/ToUar2/bITKVVo7bhsA1h" +
                "gat7YeuD7EJBPb6n061IPNtNEw0TonCp82kqg+bYKOssTXcxMsuOBGvp+Z" +
                "wNkneY7b2IzF7ZtDs746FBNF9qLI0BnqBdUA0n9JdPJCltIYaMmUIgtcv9" +
                "tXjbpY1Y3FAIh4/Mg4J7KoGWIRoXXGrcjiW/7QIDAQAB");
        byte[] expectedX509Format2 = Base64.decodeBase64(
                "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA3wgs9G9FWccRDQ" +
                "d9d++nOT6HGsrHDzdyw63HoCQMtMm2mibDAWAV5m9bi7KmFY6wralYmPAD" +
                "pzp2s0O+CFb61X/H7Ftm9En79mJZEy0BPbNd+8+/AFd0v1q3Jzx8wOhm92" +
                "+OyICGXheKMAXDx944lgAAFPxyYOVwnP+Xu2B/odjniQNr0NM0ZsJRGcsI" +
                "rKfOzuVBiXAMtECR9eFGVIuB1U2xNzrptjaPXQef6ETWNuCejaN/3DmmZX" +
                "OfZRB/cWIhrnFCspi1yYDI0NJpUmS5THShYpF4hjbwrETbXW75CNXSa0KK" +
                "38YvPaspprR6ja0a4I3+Gt20hwdw2afHW2KgQQIDAQAB");

        assertTrue(Arrays.equals(expectedX509Format1,
                RSA.readPublicKeyFromPEM(new ByteArrayInputStream(
                pemFileByteArray1))));
        assertTrue(Arrays.equals(expectedX509Format2,
                RSA.readPublicKeyFromPEM(new ByteArrayInputStream(
                pemFileByteArray2))));
    }

    @Category(RegularTests.class)
    @Test
    public void pemShouldBeWritten() throws IOException {
        String expectedPEMString1 = "-----BEGIN PUBLIC KEY-----\n" +
        "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1SgRBuihq+k9vWwuGBlE\n" +
        "rB4XntzDtTlE53wL48F/Re49BrX8X2cczneLwGLDSTj34w0AZ2zHK9Ue/NmpLtrv\n" +
        "sxVaFzhFOLZ7o9kcI/ncbNktlyrNmyeMDL0tZI84lvTWfcxf7pFxaXotef+y5RIm\n" +
        "RYMOI5kv00dSgujHhM/ToUar2/bITKVVo7bhsA1hgat7YeuD7EJBPb6n061IPNtN\n" +
        "Ew0TonCp82kqg+bYKOssTXcxMsuOBGvp+ZwNkneY7b2IzF7ZtDs746FBNF9qLI0B\n" +
        "nqBdUA0n9JdPJCltIYaMmUIgtcv9tXjbpY1Y3FAIh4/Mg4J7KoGWIRoXXGrcjiW/\n" +
        "7QIDAQAB\n" +
        "-----END PUBLIC KEY-----";

        String expectedPEMString2 = "-----BEGIN PUBLIC KEY-----\n" +
        "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA3wgs9G9FWccRDQd9d++n\n" +
        "OT6HGsrHDzdyw63HoCQMtMm2mibDAWAV5m9bi7KmFY6wralYmPADpzp2s0O+CFb6\n" +
        "1X/H7Ftm9En79mJZEy0BPbNd+8+/AFd0v1q3Jzx8wOhm92+OyICGXheKMAXDx944\n" +
        "lgAAFPxyYOVwnP+Xu2B/odjniQNr0NM0ZsJRGcsIrKfOzuVBiXAMtECR9eFGVIuB\n" +
        "1U2xNzrptjaPXQef6ETWNuCejaN/3DmmZXOfZRB/cWIhrnFCspi1yYDI0NJpUmS5\n" +
        "THShYpF4hjbwrETbXW75CNXSa0KK38YvPaspprR6ja0a4I3+Gt20hwdw2afHW2Kg\n" +
        "QQIDAQAB\n" +
        "-----END PUBLIC KEY-----";
        byte[] x509Format1 = Base64.decodeBase64(
                "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1SgRBuihq+k9vW" +
                "wuGBlErB4XntzDtTlE53wL48F/Re49BrX8X2cczneLwGLDSTj34w0AZ2zH" +
                "K9Ue/NmpLtrvsxVaFzhFOLZ7o9kcI/ncbNktlyrNmyeMDL0tZI84lvTWfc" +
                "xf7pFxaXotef+y5RImRYMOI5kv00dSgujHhM/ToUar2/bITKVVo7bhsA1h" +
                "gat7YeuD7EJBPb6n061IPNtNEw0TonCp82kqg+bYKOssTXcxMsuOBGvp+Z" +
                "wNkneY7b2IzF7ZtDs746FBNF9qLI0BnqBdUA0n9JdPJCltIYaMmUIgtcv9" +
                "tXjbpY1Y3FAIh4/Mg4J7KoGWIRoXXGrcjiW/7QIDAQAB");
        byte[] x509Format2 = Base64.decodeBase64(
                "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA3wgs9G9FWccRDQ" +
                "d9d++nOT6HGsrHDzdyw63HoCQMtMm2mibDAWAV5m9bi7KmFY6wralYmPAD" +
                "pzp2s0O+CFb61X/H7Ftm9En79mJZEy0BPbNd+8+/AFd0v1q3Jzx8wOhm92" +
            "+OyICGXheKMAXDx944lgAAFPxyYOVwnP+Xu2B/odjniQNr0NM0ZsJRGcsI" +
            "rKfOzuVBiXAMtECR9eFGVIuB1U2xNzrptjaPXQef6ETWNuCejaN/3DmmZX" +
            "OfZRB/cWIhrnFCspi1yYDI0NJpUmS5THShYpF4hjbwrETbXW75CNXSa0KK" +
            "38YvPaspprR6ja0a4I3+Gt20hwdw2afHW2KgQQIDAQAB");
    {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        String expectedPEMString = expectedPEMString1;
        byte[] x509Format = x509Format1;

        RSA.writePublicKeyToPEM(x509Format, baos);

        // uses default charset (for just the test)
        String PEMString = baos.toString();
        assertEquals(expectedPEMString, PEMString.trim());
    }
    {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        String expectedPEMString = expectedPEMString2;
        byte[] x509Format = x509Format2;

        RSA.writePublicKeyToPEM(x509Format, baos);

        // uses default charset (for just the test)
        String PEMString = baos.toString();
        assertEquals(expectedPEMString, PEMString.trim());
    }
}

/*
 * NOTE:
 * Exception classes:
 * - IOException
 * - NoSuchAlgorithmException
 * - InvalidKeySpecException
 */

/*
 * Misc cases:
 * - empty
 * - null
 * - header/footer problems
 *   - missing header/footer
 *   - wrong header+footer
 *   - mismatched header and footer
 */

/*
 * Reading:
 * - PemReader.readPemObject()
 *   - IOException
 *   - bc:DecoderException
 */

/*
 * Writing:
 * - PemWriter.writeObject()
 *   - IOException
 */

@Category(ExceptionTests.class)
@Test(expected = RSADecoderException.class)
public void emptyFileShouldThrowRSADecoderException()
        throws RSADecoderException {
    byte[] pemFileByteArray = ("").getBytes();
    RSA.readPublicKeyFromPEM(new ByteArrayInputStream(pemFileByteArray));
}

@Category(ExceptionTests.class)
@Test(expected = RSADecoderException.class)
public void headlessFileShouldThrowRSADecoderException()
        throws RSADecoderException {
    byte[] pemFileByteArray = (
    "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1SgRBuihq+k9vWwuGBlE\n" +
    "rB4XntzDtTlE53wL48F/Re49BrX8X2ccz neLwGLDSTj34w0AZ2zHK9Ue/NmpLtrv\n" +
    "sxVaFzhFOLZ7o9kcI/ncbNktlyrNmyeMDL0tZI84lvTWfcxf7pFxaXotef+y5RIm\n" +
    "RYMOI5kv00dSgujHhM/ToUar2/bITKVVo7bhsA1hgat7YeuD7EJBPb6n061IPNtN\n" +
    "Ew0TonCp82kqg+bYKOssTXcxMsuOBGvp+ZwNkneY7b2IzF7ZtDs746FBNF9qLI0B\n" +
    "nqBdUA0n9JdPJCltIYaMmUIgtcv9tXjbpY1Y3FAIh4/Mg4J7KoGWIRoXXGrcjiW/\n" +
    "7QIDAQAB").getBytes();
    RSA.readPublicKeyFromPEM(new ByteArrayInputStream(pemFileByteArray));
}

@Category(ExceptionTests.class)
@Test(expected = RSADecoderException.class)
public void missingFooterShouldThrowRSADecoderException()
        throws RSADecoderException {
    byte[] pemFileByteArray = ("-----BEGIN PUBLIC KEY-----\n" +
    "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1SgRBuihq+k9vWwuGBlE\n" +
    "rB4XntzDtTlE53wL48F/Re49BrX8X2cczneLwGLDSTj34w0AZ2zHK9Ue/NmpLtrv\n" +
    "sxVaFzhFOLZ7o9kcI/ncbNktlyrNmyeMDL0tZI84lvTWfcxf7pFxaXotef+y5RIm\n" +
    "RYMOI5kv00dSgujHhM/ToUar2/bITKVVo7bhsA1hgat7YeuD7EJBPb6n061IPNtN\n" +
        "Ew0TonCp82kqg+bYKOssTXcxMsuOBGvp+ZwNkneY7b2IzF7ZtDs746FBNF9qLI0B\n" +
        "nqBdUA0n9JdPJCltIYaMmUIgtcv9tXjbpY1Y3FAIh4/Mg4J7KoGWIRoXXGrcjiW/\n" +
        "7QIDAQAB").getBytes();
        RSA.readPublicKeyFromPEM(new ByteArrayInputStream(pemFileByteArray));
    }

    @Category(ExceptionTests.class)
    @Test(expected = RSADecoderException.class)
    public void mismatchedPEMHeadersShouldThrowRSADecoderException()
            throws RSADecoderException {
        byte[] pemFileByteArray = ("-----BEGIN PUBLIC KEY-----\n" +
        "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA1SgRBuihq+k9vWwuGBlE\n" +
        "rB4XntzDtTlE53wL48F/Re49BrX8X2cczneLwGLDSTj34w0AZ2zHK9Ue/NmpLtrv\n" +
        "sxVaFzhFOLZ7o9kcI/ncbNktlyrNmyeMDL0tZI84lvTWfcxf7pFxaXotef+y5RIm\n" +
        "RYMOI5kv00dSgujHhM/ToUar2/bITKVVo7bhsA1hgat7YeuD7EJBPb6n061IPNtN\n" +
        "Ew0TonCp82kqg+bYKOssTXcxMsuOBGvp+ZwNkneY7b2IzF7ZtDs746FBNF9qLI0B\n" +
        "nqBdUA0n9JdPJCltIYaMmUIgtcv9tXjbpY1Y3FAIh4/Mg4J7KoGWIRoXXGrcjiW/\n" +
        "7QIDAQAB\n" +
        "-----END PUBIC KEY-----").getBytes();
        RSA.readPublicKeyFromPEM(new ByteArrayInputStream(pemFileByteArray));
    }

    @Category(ExceptionTests.class)
    @Test(expected = RSADecoderException.class)
    public void misalignedPEMHeadersShouldThrowRSADecoderException()
            throws RSADecoderException {
        byte[] pemFileByteArray = ("NOISE\n-----BEGIN PUBLIC KEY-----\n\n" +
            "-----END PUBLIC KEY-----").getBytes();
        RSA.readPublicKeyFromPEM(new ByteArrayInputStream(pemFileByteArray));
    }
}
```

```
---------------------------------------------------------------
Filename: src/test/unit/java/SegmentManagerTest.java
---------------------------------------------------------------

import com.transmisms.core.protocol.SegmentManager;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.fail;

import org.junit.experimental.categories.Category;

import org.junit.Test;


public class SegmentManagerTest {
    @Category(RegularTests.class)
    @Test
    public void orderedUpdatesShouldComplete() {
        { // single-part
            SegmentManager sm = new SegmentManager(1);
            sm.updateSegment("test", 0);
            assertEquals(sm.getCompletedString(), "test");
        }
        { // multi-part
            SegmentManager sm = new SegmentManager(4);
            sm.updateSegment("t", 0);
            sm.updateSegment("e", 1);
            sm.updateSegment("s", 2);
            sm.updateSegment("t", 3);
            assertEquals(sm.getCompletedString(), "test");
        }
    }

    @Category(RegularTests.class)
    @Test
    public void unorderedUpdatesShouldComplete() {
        {
            SegmentManager sm = new SegmentManager(4);
            sm.updateSegment("e", 1);
            sm.updateSegment("s", 2);
            sm.updateSegment("t", 3);
            sm.updateSegment("t", 0);
            assertEquals(sm.getCompletedString(), "test");
        }
        {
            SegmentManager sm = new SegmentManager(4);
            sm.updateSegment("e", 1);
            sm.updateSegment("s", 2);
            sm.updateSegment("t", 0);
            sm.updateSegment("t", 3);
            assertEquals(sm.getCompletedString(), "test");
        }
    {
        SegmentManager sm = new SegmentManager(4);
        sm.updateSegment("t", 3);
        sm.updateSegment("s", 2);
        sm.updateSegment("e", 1);
        sm.updateSegment("t", 0);
        assertEquals(sm.getCompletedString(), "test");
    }
    { // abc P0
        SegmentManager sm = new SegmentManager(3);
        sm.updateSegment("a", 0);
        sm.updateSegment("b", 1);
        sm.updateSegment("c", 2);
        assertEquals(sm.getCompletedString(), "abc");
    }
    { // abc P1
        SegmentManager sm = new SegmentManager(3);
        sm.updateSegment("a", 0);
        sm.updateSegment("c", 2);
        sm.updateSegment("b", 1);
        assertEquals(sm.getCompletedString(), "abc");
    }
    { // abc P2
        SegmentManager sm = new SegmentManager(3);
        sm.updateSegment("b", 1);
        sm.updateSegment("a", 0);
        sm.updateSegment("c", 2);
        assertEquals(sm.getCompletedString(), "abc");
    }
    { // abc P3
        SegmentManager sm = new SegmentManager(3);
        sm.updateSegment("c", 2);
        sm.updateSegment("b", 1);
        sm.updateSegment("a", 0);
        assertEquals(sm.getCompletedString(), "abc");
    }
    { // abc P4
        SegmentManager sm = new SegmentManager(3);
        sm.updateSegment("b", 1);
        sm.updateSegment("c", 2);
        sm.updateSegment("a", 0);
        assertEquals(sm.getCompletedString(), "abc");
    }
    { // abc P5
        SegmentManager sm = new SegmentManager(3);
        sm.updateSegment("c", 2);
        sm.updateSegment("a", 0);
        sm.updateSegment("b", 1);
        assertEquals(sm.getCompletedString(), "abc");
    }
}

@Category(RegularTests.class)
@Test
public void duplicateUpdatesShouldComplete() {
    {
        SegmentManager sm = new SegmentManager(4);
        sm.updateSegment("t", 0);
        sm.updateSegment("t", 0);
        sm.updateSegment("e", 1);
        sm.updateSegment("e", 1);
        sm.updateSegment("s", 2);
        sm.updateSegment("s", 2);
        sm.updateSegment("t", 3);
        sm.updateSegment("t", 3);
        assertEquals(sm.getCompletedString(), "test");
    }
    {
        SegmentManager sm = new SegmentManager(4);
        sm.updateSegment("t", 3);
        sm.updateSegment("t", 3);
        sm.updateSegment("s", 2);
        sm.updateSegment("s", 2);
        sm.updateSegment("e", 1);
        sm.updateSegment("e", 1);
        sm.updateSegment("t", 0);
        sm.updateSegment("t", 0);
        assertEquals(sm.getCompletedString(), "test");
    }
    {
        SegmentManager sm = new SegmentManager(4);
        sm.updateSegment("t", 3);
        sm.updateSegment("s", 2);
        sm.updateSegment("e", 1);
        sm.updateSegment("t", 3);
        sm.updateSegment("s", 2);
        sm.updateSegment("e", 1);
        sm.updateSegment("t", 0);
        assertEquals(sm.getCompletedString(), "test");
    }
}
```

```
}
```

```
-------------------------------------------------------------
Filename: src/test/unit/java/SessionTest.java
-------------------------------------------------------------

import com.transmisms.core.protocol.Session;

import java.util.Arrays;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;

import org.junit.experimental.categories.Category;

import org.junit.Test;

public class SessionTest {

    // constructor tests

    // getSessionId() tests


    // initializeData(int) tests
}
```

```
-------------------------------------------------------------
Filename: src/test/unit/java/SmsftpTest.java
-------------------------------------------------------------

import com.transmisms.smsftp.protocol.SmsftpLookupTable;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;

import org.junit.experimental.categories.Category;

import org.junit.Test;



public class SmsftpTest {

    @Category(RegularTests.class)
    @Test
    public void smsftpLookupTableHealthCheck() {
        SmsftpLookupTable slt = new SmsftpLookupTable();
    }
}
```

```
-------------------------------------------------------------
Filename: src/test/unit/java/TextBasedDecoderTest.java
-------------------------------------------------------------

import com.transmisms.core.protocol.CorePDUType;
import com.transmisms.core.protocol.PDU;
import com.transmisms.core.protocol.PDUType;
import com.transmisms.core.protocol.PDUMalformedException;
import com.transmisms.core.protocol.Version;
import com.transmisms.smsftp.protocol.SmsftpTextBasedPduDecoder;

import java.util.UUID;
import java.nio.ByteBuffer;

import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.fail;

import org.junit.experimental.categories.Category;
import org.junit.Test;
```

```
public class TextBasedDecoderTest {
    /**
     *
     *
     */
    private UUID hexStringsToUUID(String msb, String lsb) {
        ByteBuffer msbBuffer = ByteBuffer.allocate(Long.BYTES).put(
                javax.xml.bind.DatatypeConverter.parseHexBinary(msb));
        ByteBuffer lsbBuffer = ByteBuffer.allocate(Long.BYTES).put(
                javax.xml.bind.DatatypeConverter.parseHexBinary(lsb));
        msbBuffer.flip(); // required for reading
        lsbBuffer.flip(); // required for reading
        return new UUID(msbBuffer.getLong(), lsbBuffer.getLong());
    }




    // ERROR TEST CASES
    @Category(ExceptionTests.class)
    @Test
    public void generalMalformedVersionShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        try { // version is a tuple
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7\n" +
                    "12345678901234567890\n" +
                    "head/smsftp/enc\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException on malformed Version: " +
                    "0.7");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // version is a quadruple
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0.1\n" +
                    "12345678901234567890\n" +
                    "head/smsftp/enc\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException on malformed Version: " +
                    "0.7.0.1");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // non-numeric version "number"
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsvx.7.0\n" +
                    "12345678901234567890\n" +
                    "head/smsftp/enc\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException on malformed Version: " +
                    "x.7.0");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // negative version number
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.-1\n" +
                    "12345678901234567890\n" +
                    "head/smsftp/enc\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException on malformed Version: " +
                    "0.7.-1");
        }
        catch(PDUMalformedException e) {} // do nothing
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void generalShortSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "1234567890123456789\n" +
                "head/smsftp/enc\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void generalTruncatedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "123456789~1234567890\n" +
                "head/smsftp/enc\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void generalMalformedSessionIdShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "1234567890123¿567890\n" +
                "head/smsftp/enc\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void generalInvalidHeaderShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?otr10\n" +
                "12345678901234567890\n" +
                "head/smsftp/enc\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
        fail("expected PDUMalformedException on invalid Header: " +
                "?otr10");
    }

    @Category(ExceptionTests.class)
    @Test
    public void generalPayloadTooLongShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        try { // CORE_CONNECTION_REQUEST/send
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "req/smsftp/+639121234567/send/magic.txt/blah\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with long Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // CORE_CONNECTION_REQUEST/pair
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "req/smsftp/+639121234567/pair/blah\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with long Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // CORE_CONNECTION_RESPONSE
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "rep/smsftp/0002/Accepted by user/blah\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with long Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // CORE_HEAD
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "head/smsftp/enc/blah\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with long Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // CORE_LAST_ACK
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "last/smsftp/0001/Completed without error/blah\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with long Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
    }

    @Category(ExceptionTests.class)
    @Test
    public void generalPayloadTooShortShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        try { // CORE_CONNECTION_REQUEST/send
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "req/smsftp/+639121234567/send\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with short Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
        // CORE_CONNECTION_REQUEST/pair doesn't make sense here
        try { // CORE_CONNECTION_RESPONSE
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "rep/smsftp\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with short Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // CORE_HEAD
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "head/smsftp\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with short Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
        try { // CORE_LAST_ACK
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "last/smsftp\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException with short Payload");
        }
        catch(PDUMalformedException e) {} // do nothing
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void generalInvalidPayloadTypeShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "data/smsftp/3dff01a9c072\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
        fail("expected PDUMalformedException on invalid Payload Type: " +
                "data");
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void generalInvalidProtocolNameShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "rep/smsssh/22/root@10.11.102.193\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
    }


    // CORE_CONNECTION_REQUEST
    @Category(ExceptionTests.class)
    @Test
    public void
coreConnectionRequestInvalidMsisdnShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        try { // non digits
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "req/smsftp/+63912A234567/send/testfile.zip\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException on invalid MSISDN/MIN: " +
                    "+63912A234567");
        }
        catch(PDUMalformedException e) { } // do nothing
        try { // 'plus' sign in the middle of msisdn/min
            SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "req/smsftp/0912123+567/pair\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            fail("expected PDUMalformedException on invalid MSISDN/MIN: " +
                    "0912123+567");
        }
        catch(PDUMalformedException e) { } // do nothing
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
coreConnectionRequestInvalidIntentShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "req/smsftp/+639121234567/post\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
    }

    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
coreConnectionRequestInvalidFilenameShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "req/smsftp/+639121234567/send/test*file.zip\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
    }


    // CORE_CONNECTION_RESPONSE
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void
coreConnectionResponseInvalidStatusCodeShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "rep/smsftp/9999/Accepted by user\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
    }


    // CORE_HEAD
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void coreHeadInvalidEncFieldShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "head/smsftp/INVALID/Some sort of message\n" +
                "\n" +
                "This is an extra message");
    }


    // CORE_LAST_ACK
    @Category(ExceptionTests.class)
    @Test(expected = PDUMalformedException.class)
    public void coreLastAckInvalidStatusCodeShouldThrowPDUMalformedException()
            throws PDUMalformedException {
        SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "last/smsftp/XXXX/Completed without error\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
    }



    // NORMAL TEST CASES

    // CORE_CONNECTION_REQUEST
    @Category(RegularTests.class)
    @Test
    public void coreConnectionRequestValuesShouldBeReturned()
            throws PDUMalformedException {
        { // send
            PDU connReqPDU =
                    SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "req/smsftp/+639871234567/send/testfile.zip\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            assertEquals(connReqPDU.getPduType(),
                    CorePDUType.CORE_CONNECTION_REQUEST);
            Object[] data = connReqPDU.getData();
            assertEquals(data.length, 7);
            // Protocol Version
            assertTrue(data[0] instanceof Version);
            assertEquals((Version)data[0], new Version(0, 7, 0));
            // Session Id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
            // Transmisms Protocol
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2], "smsftp");
            // Receiver MSISDN/MIN
            assertTrue(data[3] instanceof String);
            assertEquals((String)data[3], "+639871234567");
            // Intent
            assertTrue(data[4] instanceof String);
            assertEquals((String)data[4], "send");
            // Filename
            assertTrue(data[5] instanceof String);
            assertEquals((String)data[5], "testfile.zip");
            // Additional Message
            assertTrue(data[6] instanceof String);
            assertEquals((String)data[6], "This is an extra message");
        }
        { // send 2
            PDU connReqPDU =
                    SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +

"req/smsftp/+639871234567/send/abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVW
XYZ0123456789 .-#$%&'()+,i=@_\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message too");
            assertEquals(connReqPDU.getPduType(),
                    CorePDUType.CORE_CONNECTION_REQUEST);
            Object[] data = connReqPDU.getData();
            assertEquals(data.length, 7);
            // Protocol Version
            assertTrue(data[0] instanceof Version);
            assertEquals((Version)data[0], new Version(0, 7, 0));
            // Session Id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
            // Transmisms Protocol
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2], "smsftp");
            // Receiver MSISDN/MIN
            assertTrue(data[3] instanceof String);
            assertEquals((String)data[3], "+639871234567");
            // Intent
            assertTrue(data[4] instanceof String);
            assertEquals((String)data[4], "send");
            // Filename
            assertTrue(data[5] instanceof String);
            assertEquals((String)data[5],

"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
.-#$%&'()+,i=@_");
            // Additional Message
            assertTrue(data[6] instanceof String);
            assertEquals((String)data[6], "This is an extra message too");
        }
        { // pair
            PDU connReqPDU =
                    SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "req/smsftp/09871234567/pair\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            assertEquals(connReqPDU.getPduType(),
                    CorePDUType.CORE_CONNECTION_REQUEST);
            Object[] data = connReqPDU.getData();
            assertEquals(data.length, 6);
            // Protocol Version
            assertTrue(data[0] instanceof Version);
            assertEquals((Version)data[0], new Version(0, 7, 0));
            // Session Id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
            // Transmisms Protocol
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2], "smsftp");
            // Receiver MSISDN/MIN
            assertTrue(data[3] instanceof String);
            assertEquals((String)data[3], "09871234567");
            // Intent
            assertTrue(data[4] instanceof String);
            assertEquals((String)data[4], "pair");
            // Additional Message
            assertTrue(data[5] instanceof String);
            assertEquals((String)data[5], "This is an extra message");
        }
    }


    // CORE_CONNECTION_RESPONSE
    @Category(RegularTests.class)
    @Test
    public void coreConnectionResponseValuesShouldBeReturned()
            throws PDUMalformedException {
        { // with status message
            PDU connResPDU =
                    SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "rep/smsftp/0002/Accepted by user\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            assertEquals(connResPDU.getPduType(),
                    CorePDUType.CORE_CONNECTION_RESPONSE);
            Object[] data = connResPDU.getData();
            assertEquals(data.length, 6);
            // Protocol Version
            assertTrue(data[0] instanceof Version);
            assertEquals((Version)data[0], new Version(0, 7, 0));
            // Session Id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
            // Transmisms Protocol
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2], "smsftp");
            // Status Code
            assertTrue(data[3] instanceof String);
            assertEquals((String)data[3], "0002");
            // Status Message
            assertTrue(data[4] instanceof String);
            assertEquals((String)data[4], "Accepted by user");
            // Additional Message
            assertTrue(data[5] instanceof String);
            assertEquals((String)data[5], "This is an extra message");
        }
        { // missing status message
            PDU connResPDU =
                    SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "rep/smsftp/0002\n" +
                    "\n" + // extra \n for additional message
                    "This is an extra message");
            assertEquals(connResPDU.getPduType(),
                    CorePDUType.CORE_CONNECTION_RESPONSE);
            Object[] data = connResPDU.getData();
            assertEquals(data.length, 6);
            // Protocol Version
            assertTrue(data[0] instanceof Version);
            assertEquals((Version)data[0], new Version(0, 7, 0));
            // Session Id
            assertTrue(data[1] instanceof UUID);
            assertEquals((UUID)data[1], hexStringsToUUID(
                    "32699b3242279c09", "32699b3242279c09"));
            // Transmisms Protocol
            assertTrue(data[2] instanceof String);
            assertEquals((String)data[2], "smsftp");
            // Status Code
            assertTrue(data[3] instanceof String);
            assertEquals((String)data[3], "0002");
            // Status Message
            assertNull(data[4]);
            // Additional Message
            assertTrue(data[5] instanceof String);
            assertEquals((String)data[5], "This is an extra message");
        }
        { // without additional message
            PDU connResPDU =
                    SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                    "?transmismsv0.7.0\n" +
                    "12345678901234567890\n" +
                    "rep/smsftp/0002/Accepted by user");
            assertEquals(connResPDU.getPduType(),
                    CorePDUType.CORE_CONNECTION_RESPONSE);
            Object[] data = connResPDU.getData();
            assertEquals(data.length, 6);
        // Protocol Version
        assertTrue(data[0] instanceof Version);
        assertEquals((Version)data[0], new Version(0, 7, 0));
        // Session Id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                "32699b3242279c09", "32699b3242279c09"));
        // Transmisms Protocol
        assertTrue(data[2] instanceof String);
        assertEquals((String)data[2], "smsftp");
        // Status Code
        assertTrue(data[3] instanceof String);
        assertEquals((String)data[3], "0002");
        // Status Message
        assertTrue(data[4] instanceof String);
        assertEquals((String)data[4], "Accepted by user");
        // Additional Message field is missing
        assertNull(data[5]);
    }
}


// CORE_HEAD
@Category(RegularTests.class)
@Test
public void coreHeadValuesShouldBeReturned()
        throws PDUMalformedException {
    { // with additional message
        PDU headPDU =
                SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "head/smsftp/enc\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
        assertEquals(headPDU.getPduType(), CorePDUType.CORE_HEAD);
        Object[] data = headPDU.getData();
        assertEquals(data.length, 5);
        // Protocol Version
        assertTrue(data[0] instanceof Version);
        assertEquals((Version)data[0], new Version(0, 7, 0));
        // Session Id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                "32699b3242279c09", "32699b3242279c09"));
        // Transmisms Protocol
        assertTrue(data[2] instanceof String);
        assertEquals((String)data[2], "smsftp");
        // Encryption
        assertTrue(data[3] instanceof String);
        assertEquals((String)data[3], "enc");
        // Additional Message
        assertTrue(data[4] instanceof String);
        assertEquals((String)data[4], "This is an extra message");
    }
    { // without additional message
        PDU headPDU =
                SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "head/smsftp/noenc");
        assertEquals(headPDU.getPduType(), CorePDUType.CORE_HEAD);
        Object[] data = headPDU.getData();
        assertEquals(data.length, 5);
        // Protocol Version
        assertTrue(data[0] instanceof Version);
        assertEquals((Version)data[0], new Version(0, 7, 0));
        // Session Id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                "32699b3242279c09", "32699b3242279c09"));
        // Transmisms Protocol
        assertTrue(data[2] instanceof String);
        assertEquals((String)data[2], "smsftp");
        // Encryption
        assertTrue(data[3] instanceof String);
        assertEquals((String)data[3], "noenc");
        // Additional Message field is missing
        assertNull(data[4]);
    }
}


// CORE_LAST_ACK
@Category(RegularTests.class)
@Test
public void coreLastAckValuesShouldBeReturned()
        throws PDUMalformedException {
    { // with status message
        PDU lastAckPDU =
                SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "last/smsftp/0001/Completed without error\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
        assertEquals(lastAckPDU.getPduType(), CorePDUType.CORE_LAST_ACK);
        Object[] data = lastAckPDU.getData();
        assertEquals(data.length, 6);
        // Protocol Version
        assertTrue(data[0] instanceof Version);
        assertEquals((Version)data[0], new Version(0, 7, 0));
        // Session Id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                "32699b3242279c09", "32699b3242279c09"));
        // Transmisms Protocol
        assertTrue(data[2] instanceof String);
        assertEquals((String)data[2], "smsftp");
        // Status Code
        assertTrue(data[3] instanceof String);
        assertEquals((String)data[3], "0001");
        // Status Message
        assertTrue(data[4] instanceof String);
        assertEquals((String)data[4], "Completed without error");
        // Additional Message
        assertTrue(data[5] instanceof String);
        assertEquals((String)data[5], "This is an extra message");
    }
    { // missing status message
        PDU lastAckPDU =
                SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "last/smsftp/0001\n" +
                "\n" + // extra \n for additional message
                "This is an extra message");
        assertEquals(lastAckPDU.getPduType(), CorePDUType.CORE_LAST_ACK);
        Object[] data = lastAckPDU.getData();
        assertEquals(data.length, 6);
        // Protocol Version
        assertTrue(data[0] instanceof Version);
        assertEquals((Version)data[0], new Version(0, 7, 0));
        // Session Id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                "32699b3242279c09", "32699b3242279c09"));
        // Transmisms Protocol
        assertTrue(data[2] instanceof String);
        assertEquals((String)data[2], "smsftp");
        // Status Code
        assertTrue(data[3] instanceof String);
        assertEquals((String)data[3], "0001");
        // Status Message
        assertNull(data[4]);
        // Additional Message
        assertTrue(data[5] instanceof String);
        assertEquals((String)data[5], "This is an extra message");
    }
    { // without additional message
        PDU lastAckPDU =
                SmsftpTextBasedPduDecoder.getInstance().decodeTextBased(
                "?transmismsv0.7.0\n" +
                "12345678901234567890\n" +
                "last/smsftp/0001/Completed without error");
        assertEquals(lastAckPDU.getPduType(), CorePDUType.CORE_LAST_ACK);
        Object[] data = lastAckPDU.getData();
        assertEquals(data.length, 6);
        // Protocol Version
        assertTrue(data[0] instanceof Version);
        assertEquals((Version)data[0], new Version(0, 7, 0));
        // Session Id
        assertTrue(data[1] instanceof UUID);
        assertEquals((UUID)data[1], hexStringsToUUID(
                "32699b3242279c09", "32699b3242279c09"));
        // Transmisms Protocol
        assertTrue(data[2] instanceof String);
        assertEquals((String)data[2], "smsftp");
        // Status Code
        assertTrue(data[3] instanceof String);
        assertEquals((String)data[3], "0001");
        // Status Message
        assertTrue(data[4] instanceof String);
        assertEquals((String)data[4], "Completed without error");
        // Additional Message field is missing
        assertNull(data[5]);
    }
}
}
```

```
--------------------------------------------------------------
Filename: src/test/unit/java/VersionTest.java
--------------------------------------------------------------

import com.transmisms.core.protocol.Version;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotEquals;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;

import org.junit.experimental.categories.Category;

import org.junit.Test;



public class VersionTest {

    // no, we won't check for a.hashCode() == b.hashCode() == a.equals(b)
    // but we "followed" the java.lang.Object contract anyway

    @Category(RegularTests.class)
    @Test
    public void sameVersionShouldBeEquivalent() {
        Version tester1 = new Version(0, 0, 0);
        Version tester2 = new Version(0, 0, 0);
        Version tester3 = new Version(0, 0, 0);

        // symmetric test of equality
        assertEquals(tester2, tester1);
        assertEquals(0, tester2.compareTo(tester1));

        // transitive test of equality
        assertEquals(tester1, tester2);
        assertEquals(0, tester1.compareTo(tester2));
        assertEquals(tester2, tester3);
        assertEquals(0, tester2.compareTo(tester3));
        assertEquals(tester1, tester3);
        assertEquals(0, tester1.compareTo(tester3));
    }

    @Category(RegularTests.class)
    @Test
    public void diffVersionShouldNotBeEquivalent() {
        Version tester1 = new Version(0, 0, 0);
        Version tester2 = new Version(0, 0, 0);
        Version tester4 = new Version(3, 7, 2);

        // tests for inequality
        assertNotEquals(tester1, tester4);
        assertNotEquals(0, tester1.compareTo(tester4));

        // other misc tests
        assertNotEquals(tester4, tester1);
        assertNotEquals(0, tester4.compareTo(tester1));
        assertNotEquals(tester2, tester4);
        assertNotEquals(0, tester2.compareTo(tester4));
    }


    @Category(RegularTests.class)
    @Test
    public void newerVersionShouldBeMore() {
        Version tester1 = new Version(6, 8, 9);
        Version tester2 = new Version(6, 8, 8);
        Version tester3 = new Version(6, 7, 9);
        Version tester4 = new Version(5, 8, 9);

        assertTrue(tester1.compareTo(tester2) > 0);
        assertTrue(tester1.compareTo(tester3) > 0);
        assertTrue(tester1.compareTo(tester4) > 0);

        assertTrue(tester2.compareTo(tester3) > 0);
    }

    @Category(RegularTests.class)
    @Test
    public void olderVersionShouldBeLess() {
        Version tester1 = new Version(6, 8, 2);
        Version tester2 = new Version(6, 8, 3);
        Version tester3 = new Version(6, 9, 2);
        Version tester4 = new Version(7, 8, 2);

        assertTrue(tester1.compareTo(tester2) < 0);
        assertTrue(tester1.compareTo(tester3) < 0);
        assertTrue(tester1.compareTo(tester4) < 0);

        assertTrue(tester2.compareTo(tester3) < 0);
    }

    @Category(ExceptionTests.class)
    @Test(expected = IllegalArgumentException.class)
    public void negativeParam1ShouldThrowIllegalArgumentException() {
        Version tester = new Version(-3, 7, 5);
    }

    @Category(ExceptionTests.class)
    @Test(expected = IllegalArgumentException.class)
    public void negativeParam2ShouldThrowIllegalArgumentException() {
        Version tester = new Version(3, -7, 5);
    }

    @Category(ExceptionTests.class)
    @Test(expected = IllegalArgumentException.class)
    public void negativeParam3ShouldThrowIllegalArgumentException() {
        Version tester = new Version(3, 7, -5);
    }
}
```

Android Companion App:

```
--------------------------------------------------------------
Filename: build.gradle
--------------------------------------------------------------

// required to use plugin 'com.android.application'
buildscript {
    repositories {
        jcenter()
        mavenCentral()
        mavenLocal()
        flatDir {
            name 'local'
            dirs '../repos'
        }
    }

    dependencies {
        classpath "com.android.tools.build:gradle:2.2.0"
    }
}

// apply plugin: 'java' // add java tasks
apply plugin: 'com.android.application' // add android tasks
apply plugin: 'eclipse' // create eclipse project

// java jar configuration (manifest.mf in jar)
sourceCompatibility = '1.7'
version = '0.9'

android {
    compileSdkVersion 23
    buildToolsVersion "23.0.3"

    defaultConfig {
        minSdkVersion 19
        targetSdkVersion 23
    }
}

repositories {
    jcenter() // because we love https and supersets
    mavenCentral()
    mavenLocal()
    flatDir {
        name 'local'
        dirs '../repos'
    }
}

dependencies {
    //compile group: 'com.transmisms.smsftp', name: 'smsftp-protocol', version: '0.9'
    compile group: 'org.zeromq', name: 'jeromq', version: '0.4.3'
    compile group: 'com.google.code.gson', name: 'gson', version: '2.8.1'

    // for style backwards compatiblity
    compile 'com.android.support:appcompat-v7:23.4.0'
    compile 'com.android.support:cardview-v7:23.4.0'
    compile 'com.android.support:recyclerview-v7:23.4.0'
    compile 'com.android.support.constraint:constraint-layout:1.0.2'
    compile 'com.android.support:design:23.4.0'

    testCompile group: 'junit', name: 'junit', version: '4.12'
    testCompile group: 'org.mockito', name: 'mockito-core', version: '1.10.19'
}

uploadArchives {
    repositories {
        flatDir {
            name 'local'
            dirs '../repos'
       }
    }
}



--------------------------------------------------------------
Filename: src/main/AndroidManifest.xml
--------------------------------------------------------------

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.transmisms.androidcompanion"
    android:versionCode="0"
    android:versionName="0.0.1" >
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.SEND_SMS" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />

    <application android:label="@string/app_name"
            android:theme="@style/AppTheme">
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name"
            android:configChanges="orientation|screenSize|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".SetupActivity"
            android:label="Companion setup"
            android:theme="@style/SetupTheme"
            android:configChanges="orientation|screenSize|keyboardHidden">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        <service android:name=".CompanionService"
                 android:exported="false" />
    </application>
</manifest>



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/androidcompanion/ClientAction.java
--------------------------------------------------------------

package com.transmisms.androidcompanion;

public class ClientAction {
    final public String action;

    // send
    final public String recipient;
    final public String message;

    // ping
    // msisdn
    // NOTHING FOLLOWS

    public ClientAction() {
        this.action = null;
        this.recipient = null;
        this.message = null;
    }
}



--------------------------------------------------------------
Filename: src/main/java/com/transmisms/androidcompanion/CompanionService.java
--------------------------------------------------------------
package com.transmisms.androidcompanion;

import com.google.gson.Gson;
import org.zeromq.ZMQ;
import org.zeromq.ZMQ.Context;
import org.zeromq.ZMQ.Poller;
import org.zeromq.ZMQ.Socket;

import android.app.PendingIntent;
import android.app.Notification;
import android.app.Notification.Builder;
import android.app.Service;
import android.content.Intent;
import android.content.IntentFilter;
import android.content.SharedPreferences;
import android.os.Binder;
import android.os.Handler;
import android.os.IBinder;
import android.preference.PreferenceManager;
import android.telephony.SmsManager;
import android.widget.Toast;

import org.zeromq.ZMQException;
```

```java
import zmq.ZError;


public class CompanionService extends Service {
    public class CompanionBinder extends Binder {
        CompanionService getService() {
            return CompanionService.this;
        }
    }

    private static final int ONGOING_NOTIFICATION_ID = 1;

    private String msisdn;

    private Gson gson = new Gson();

    private CompanionBinder mBinder = new CompanionBinder();
    private Handler mHandler;
    private Thread mThread;
    private boolean mIsRunning = false;
    private SmsReceiver smsReceiver = new SmsReceiver();



    @Override
    public void onCreate() {
        super.onCreate();
        this.mHandler = new Handler();

        // create PendingIntent to start MainActivity from the notification
        Intent notificationIntent = new Intent(this, MainActivity.class);
        notificationIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP |
                Intent.FLAG_ACTIVITY_SINGLE_TOP);
        PendingIntent pendingIntent = PendingIntent.getActivity(this, 0,
                notificationIntent, PendingIntent.FLAG_UPDATE_CURRENT);
        Notification notification = new Notification.Builder(this)
                .setContentTitle("Transmisms Companion")
                .setContentText("Tap to go to the app")
                .setSmallIcon(R.drawable.ic_import_export_white_24dp)
                .setContentIntent(pendingIntent)
                .setTicker("Transmisms companion service is running")
                .build();
        // start service as a foreground service
        startForeground(ONGOING_NOTIFICATION_ID, notification);
    }

    @Override
    public IBinder onBind(Intent intent) {
        //return null;
        return this.mBinder;
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        super.onStartCommand(intent, flags, startId);
        if(this.mThread == null) {
            // load previously loaded settings, if any
            SharedPreferences sPrefs =
                    PreferenceManager.getDefaultSharedPreferences(this);
            final String savedPortValue = sPrefs.getString(
                    MainActivity.PORT_PREF_KEY, "8767");
            this.getMsisdn(); // try to get msisdn from config asap
            this.mThread = new Thread() {
                @Override
                public void run() {
                    // set necessary flags
                    CompanionService.this.mIsRunning = true;

                    // register receiver
                    IntentFilter filter = new IntentFilter();
                    filter.addAction(
                            android.provider.Telephony.Sms.Intents
                            .SMS_RECEIVED_ACTION);
                    filter.setPriority(999); // set to slightly high
                    CompanionService.this.registerReceiver(
                            CompanionService.this.smsReceiver, filter);

                    // initialize zmq connection
                    String addr = "0.0.0.0:" + savedPortValue;
                    Context ctx = ZMQ.context(1);
                    Socket sck = ctx.socket(ZMQ.PAIR);
                    sck.setLinger(0);
                    sck.bind("tcp://" + addr);
                    // serve until stopped
                    while(CompanionService.this.mIsRunning &&
                            !Thread.currentThread().isInterrupted()) {
                        // set zmq poller
                        Poller plr = ctx.poller();
                        plr.register(sck, Poller.POLLIN);
                        try {
                            plr.poll(100);
                            if(plr.pollin(0)) {
                                String str = sck.recvStr(0);
                                ClientAction cAction =
                                        CompanionService.this.gson.fromJson(
                                        str, ClientAction.class);
                                if(cAction != null && cAction.action != null &&
                                        cAction.action.equals("ping")) {
                                    // send pong back
                                    String jsonStr = CompanionService
                                        .this.gson.toJson(
                                        new ServerAction('p'));
                                    sck.send(jsonStr);
                                }
                                else if(cAction.action.equals("send") &&
                                        cAction.recipient != null &&
                                        cAction.message != null) {
                                    // send sms
                                    SmsManager.getDefault().sendTextMessage(
                                            cAction.recipient, null,
                                            cAction.message, null, null);
                                }
                                else if(cAction.action.equals("msisdn")) {
                                    // create and send the msisdn
                                    // server action object as json
                                    String jsonStr = CompanionService
                                        .this.gson.toJson(
                                        new ServerAction(
                                        CompanionService.this.getMsisdn()));
                                    sck.send(jsonStr);
                                }
                            }
                            // exhaust queue then proceed
                            String jsonStr =
                                    CompanionService.this.smsReceiver.pop();
                            while(jsonStr != null) {
                                sck.send(jsonStr);
                                jsonStr = CompanionService
                                        .this.smsReceiver.pop();
                            }
                        }
                        catch(ZMQException e) {
                            // exit loop on zmq exception
                            break;
                        }
                        plr.close();
                    }
                    // cleanup for zmq
                    if(sck != null) {
                        try {
                            sck.close();
                        }
                        catch(ZError.IOException e) {
                            // do nothing if already "closed"
                        }
                    }
                    if(ctx != null && !ctx.isTerminated()) {
                        try {
                            // NOTE: This was purposively commented out due
                            //       to some Android version's implementation
                            //       of libcore causing problems with nio
                            //       methods invoked by explicitly term() or
                            //       destroying zeromq Contexts. These
                            //       Contexts will be eventually destroyed
                            //       outside the main loop after exiting this
                            //       thread
                            //ctx.term();
                        }
                        catch(ZError.IOException e) {
                            // do nothing if already "closed"
                        }
                    }

                    // cleanup when the service exits
                    CompanionService.this.mIsRunning = false;
                    CompanionService.this.stopSelf();
                    mHandler.post(new Runnable() {
                        @Override
                        public void run() {
                            Toast.makeText(CompanionService.this,
                                "Service stopped", Toast.LENGTH_SHORT).show();
                        }
                    });
                }
            };
            this.mThread.start();
        }

        // we want this service to continue running until it is explicitly
        // stopped
        return Service.START_STICKY;
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        this.mIsRunning = false; // set the stop flag
        this.mThread = null; // and cleanup the thread references
        this.unregisterReceiver(this.smsReceiver);
    }

    public boolean isRunning() {
        return this.mIsRunning;
    }

    public String getMsisdn() {
        // get value from config, if not accessed yet by this service
        if(this.msisdn == null) {
            SharedPreferences sPrefs =
                    PreferenceManager
                    .getDefaultSharedPreferences(
                    CompanionService.this);
            String savedMsisdnValue =
                    sPrefs.getString(
                    MainActivity.MSISDN_PREF_KEY
                    , null);
            CompanionService.this.msisdn =
                    savedMsisdnValue;
        }
        return this.msisdn;
    }
}




--------------------------------------------------------------
Filename: src/main/java/com/transmisms/androidcompanion/MainActivity.java
--------------------------------------------------------------

package com.transmisms.androidcompanion;

import com.transmisms.androidcompanion.CompanionService.CompanionBinder;

import android.Manifest;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.os.IBinder;
import android.preference.PreferenceManager;
import android.support.design.widget.FloatingActionButton;
import android.support.design.widget.Snackbar;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;

import java.util.HashMap;
import java.util.Map;



public class MainActivity extends AppCompatActivity {
    public final static String MSISDN_PREF_KEY = "msisdn";
    public final static String PORT_PREF_KEY = "port";

    public final static int READ_SMS_REQUEST_CODE = 0;
    public final static int SEND_SMS_REQUEST_CODE = 1;
    public final static int RECEIVE_SMS_REQUEST_CODE = 2;
    public final static int READ_PHONE_STATE_CODE = 3;
    private final static Map<String, Integer> permissionCodeMap =
            new HashMap<>();

    private FloatingActionButton mFab;
    private CompanionService mService;

    static {
        MainActivity.permissionCodeMap.put(Manifest.permission.READ_SMS,
                READ_SMS_REQUEST_CODE);
        MainActivity.permissionCodeMap.put(Manifest.permission.SEND_SMS,
                SEND_SMS_REQUEST_CODE);
        MainActivity.permissionCodeMap.put(Manifest.permission.RECEIVE_SMS,
                RECEIVE_SMS_REQUEST_CODE);
    }


    private ServiceConnection mServiceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName className,
                IBinder service) {
            CompanionBinder cb = (CompanionBinder)service;
            MainActivity.this.mService = cb.getService();
            MainActivity.this.setFabImageAsStart(true);
        }

        @Override
        public void onServiceDisconnected(ComponentName className) {
            MainActivity.this.mService = null;
        }
    };



    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        SharedPreferences sPrefs =
                PreferenceManager.getDefaultSharedPreferences(this);
        String savedMsisdn =
                sPrefs.getString(MainActivity.MSISDN_PREF_KEY, null);
        String savedPort =
                sPrefs.getString(MainActivity.PORT_PREF_KEY, null);
        if(savedMsisdn != null && savedPort != null) {
            this.setContentView(R.layout.main_layout);
            Toolbar appBar = (Toolbar)(this.findViewById(R.id.mainAppBar));
            this.setSupportActionBar(appBar);
            this.mFab = (FloatingActionButton)(this.findViewById(
                    R.id.floatingActionButton));
            this.attemptStartCompanionService();
        }
        else {
            this.onSetupMenuItemClick(null);
        }
    }

    @Override
    public void onStart() {
        super.onStart();
        TextView textView = (TextView) findViewById(R.id.text_view);
        textView.setText("Main world!");
    }

    @Override
    public void onResume() {
        super.onResume();
        // change FAB's icon depending on the CompanionService's state
        this.setFabImageAsStart(this.isServiceRunning());
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // unbind the service but don't stop it from running
        if(this.mService != null) {
            this.unbindService(this.mServiceConnection);
            this.mService = null;
        }
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.main_menu, menu);
        return true;
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
            String permissions[], int[] grantResults) {
        switch(requestCode) {
            case READ_SMS_REQUEST_CODE:
            case SEND_SMS_REQUEST_CODE:
            case RECEIVE_SMS_REQUEST_CODE: {
                if(grantResults.length > 0 &&
                        grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    // start service immediately
                    this.startCompanionService();
                    Toast.makeText(this, "Service started",
                            Toast.LENGTH_SHORT).show();
                }
                else { // permission denied
                Snackbar.make(
                        this.findViewById(R.id.main_layout),
                        "Service cannot be started. " +
                        "Allow permission requested and try again.",
                        Snackbar.LENGTH_LONG).show();
            }
            break;
        }
        default: {
            // do nothing
        }
    }
}

// handler when the setup menu item is clicked
public void onSetupMenuItemClick(MenuItem m) {
    if(this.isServiceRunning()) {
        this.stopCompanionService();
    }
    Intent setupIntent = new Intent(this, SetupActivity.class);
    this.startActivity(setupIntent);
    this.finish();
}

// handler when the FAB is clicked
public void onFabButtonClick(View v) {
    boolean isServiceStarted = this.isServiceRunning();
    if(isServiceStarted) {
        this.stopCompanionService();
    }
    else {
        this.attemptStartCompanionService();
    }
    this.setFabImageAsStart(!isServiceStarted);
}

// attempts to start service with permission checks beforehand
public void attemptStartCompanionService() {
    // check for permissions
    boolean permissionGranted = true;
    String newReqPermission = null;
    for(String permission : MainActivity.permissionCodeMap.keySet()) {
        permissionGranted = permissionGranted &&
                (ContextCompat.checkSelfPermission(this, permission) ==
                PackageManager.PERMISSION_GRANTED);
        if(!permissionGranted) {
            newReqPermission = permission;
        }
    }
    // permission denied
    if(!permissionGranted && newReqPermission != null) {
        ActivityCompat.requestPermissions(this, new String[]{
                newReqPermission},
                MainActivity.permissionCodeMap.get(newReqPermission));
    }
    else { // permission granted
        this.startCompanionService(); // just start the service ASAP
        Toast.makeText(this, "Service started", Toast.LENGTH_SHORT).show();
    }
}

//// utility functions
private void setFabImageAsStart(boolean serviceStarted) {
    if(serviceStarted) { // service started
        // set icon as stop
        this.mFab.setImageDrawable(ContextCompat.getDrawable(this,
                R.drawable.ic_stop_white_24dp));
    }
    else { // service stopped
        // set icon as start
        this.mFab.setImageDrawable(ContextCompat.getDrawable(this,
                R.drawable.ic_play_arrow_white_24dp));
    }
}

private boolean isServiceRunning() {
    return this.mService != null && this.mService.isRunning() == true;
}

private void startCompanionService() {
    Intent intent = new Intent(this, CompanionService.class);
    this.startService(intent);
    this.bindService(intent, this.mServiceConnection,
            Context.BIND_AUTO_CREATE);
}

private void stopCompanionService() {
    if(this.mService != null) {
        this.unbindService(this.mServiceConnection);
        Intent intent = new Intent(this, CompanionService.class);
        this.stopService(intent);
            this.mService = null;
        }
    }
}
```

```
-------------------------------------------------------------
Filename: src/main/java/com/transmisms/androidcompanion/ServerAction.java
-------------------------------------------------------------

package com.transmisms.androidcompanion;

public class ServerAction {
    final public String action;

    // recv
    final public String sender;
    final public String message;
    final public Long timestamp;

    // error
    final public String code;
    final public String[] alternativecodes; // optional

    // pong
    // NOTHING FOLLOWS

    public ServerAction() {
        this.action = null;
        this.code = null;
        this.alternativecodes = null;
        this.sender = null;
        this.message = null;
        this.timestamp = null;
    }

    // recv
    public ServerAction(String sender,
            String message, long timestamp) {
        this.action = "recv";
        this.sender = sender;
        this.message = message;
        this.timestamp = timestamp;
        this.code = null;
        this.alternativecodes = null;
    }

    // error
    public ServerAction(String code, String[] alternativecodes) {
        this.action = "error";
        this.code = code;
        this.alternativecodes = alternativecodes;
        this.sender = null;
        this.message = null;
        this.timestamp = null;
    }

    // msisdn
    public ServerAction(String msisdn) {
        this.action = "msisdn";
        this.message = msisdn;
        // no fields
        this.code = null;
        this.alternativecodes = null;
        this.sender = null;
        this.timestamp = null;
    }

    // pong
    public ServerAction(char c) {
        this.action = "pong";
        // no fields
        this.code = null;
        this.alternativecodes = null;
        this.sender = null;
        this.message = null;
        this.timestamp = null;
    }
}
```

```
-------------------------------------------------------------
Filename: src/main/java/com/transmisms/androidcompanion/SetupActivity.java
-------------------------------------------------------------

package com.transmisms.androidcompanion;
```

```
import android.Manifest;
import android.app.AlertDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.preference.PreferenceManager;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.telephony.TelephonyManager;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;



public class SetupActivity extends AppCompatActivity {
    private EditText msisdnField;
    private Button finishButton;
    private String bindPort = "8767";



    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // initialize views an widgets
        setContentView(R.layout.setup_layout);
        Toolbar appBar = (Toolbar)(this.findViewById(R.id.setup_appbar));
        this.setSupportActionBar(appBar);
        // "bind" setup_finish_button's enable property to
        // setup_msisdn_field's !emptiness
        this.finishButton = (Button)(this.findViewById(
                R.id.setup_finish_button));
        this.msisdnField = (EditText)(this.findViewById(
                R.id.setup_msisdn_field));
        this.msisdnField.addTextChangedListener(new TextWatcher() {
            @Override
            public void afterTextChanged(Editable s) {
                SetupActivity.this.finishButton.setEnabled(s.length() != 0);
            }
            @Override
            public void beforeTextChanged(CharSequence s, int start,
                    int count, int after) {}
            @Override
            public void onTextChanged(CharSequence s, int start, int before,
                    int count) {}
        });
        // load previously loaded settings, if any
        SharedPreferences sPrefs =
                PreferenceManager.getDefaultSharedPreferences(this);
        String savedMsisdnValue = sPrefs.getString(
                MainActivity.MSISDN_PREF_KEY, null);
        String savedPortValue = sPrefs.getString(
                MainActivity.PORT_PREF_KEY, null);
        if(savedMsisdnValue != null) {
            this.msisdnField.setText(savedMsisdnValue);
        }
        if(savedPortValue != null) {
            this.bindPort = savedPortValue;
        }
        // try to autodetect msisdn asap
        this.onDetectMsisdnButtonClicked(
                this.findViewById(android.R.id.content));
    }

    public void onDetectMsisdnButtonClicked(View v) {
        // check for permissions first
        if(ContextCompat.checkSelfPermission(
                this, Manifest.permission.READ_PHONE_STATE) ==
                PackageManager.PERMISSION_GRANTED) { // granted
            this.detectMsisdn();
        }
        else { // denied
            // try to request permissions
            ActivityCompat.requestPermissions(this, new String[]{
                    Manifest.permission.READ_PHONE_STATE},
                    MainActivity.READ_PHONE_STATE_CODE);
        }
    }

    public void detectMsisdn() {
        // get msisdn
        TelephonyManager tMgr = (TelephonyManager)(this.getSystemService(
                Context.TELEPHONY_SERVICE));
        String msisdn = tMgr.getLine1Number().trim();
        // check for null, empty, or invalid msisdns
        if(msisdn != null && msisdn.length() != 0 && !msisdn.contains("?")) {
            // put msisdn on appropriate text fields
            this.msisdnField.setText(msisdn);
        }
        else { // probably we got null or "???"
            // ask user to input number manually
            Toast.makeText(this, "Could not detect this device's number. " +
                    "Please enter the number manually.",
                    Toast.LENGTH_LONG).show();
        }
    }

    public void onFinishButtonClick(View v) {
        SharedPreferences sPrefs =
                PreferenceManager.getDefaultSharedPreferences(this);
        // save settings
        SharedPreferences.Editor sPrefsEditor = sPrefs.edit();
        sPrefsEditor.putString(MainActivity.MSISDN_PREF_KEY,
                this.msisdnField.getText().toString());
        sPrefsEditor.putString(MainActivity.PORT_PREF_KEY,
                this.bindPort);
        sPrefsEditor.apply();

        // then start the main activity afterwards
        Intent mainIntent = new Intent(this, MainActivity.class);
        this.startActivity(mainIntent);
        this.finish();
    }

    public void onPortCustButtonClick(View v) {
        // initialize the portPicker builder dialog
        AlertDialog.Builder builder = new AlertDialog.Builder(this);
        View viewInflated = LayoutInflater.from(this).inflate(
                R.layout.port_input_layout,
                (ViewGroup)(this.findViewById(android.R.id.content)), false);
        builder.setView(viewInflated);
        final EditText portInput =
                (EditText)(viewInflated.findViewById(R.id.port_field));
        portInput.setText(this.bindPort);
        builder.setMessage("Enter desired port number (1024-65535):");
        builder.setPositiveButton("Save",
                new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int w) {
                dialog.dismiss();
                SetupActivity.this.bindPort = String.valueOf(Integer.parseInt(
                        portInput.getText().toString()));
            }
        });
        builder.setNegativeButton("Cancel",
                new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int w) {
                dialog.cancel();
            }
        });
        // set listeners for portInput to change the confirmation (Save)
        // button's enabled property
        final AlertDialog portPicker = builder.create();
        portInput.addTextChangedListener(new TextWatcher() {
            @Override
            public void afterTextChanged(Editable s) {
                boolean enableButton = (s.length() != 0);
                if(enableButton) {
                    int i = Integer.parseInt(s.toString());
                    enableButton = (i > 1024 && i < 65536);
                }
                portPicker.getButton(
                        DialogInterface.BUTTON_POSITIVE).setEnabled(
                        enableButton);
            }
            @Override
            public void beforeTextChanged(CharSequence s, int start,
                    int count, int after) {}
            @Override
            public void onTextChanged(CharSequence s, int start, int before,
                    int count) {}
        });
        // finally show portPicker
        portPicker.show();
    }

    @Override
    public void onRequestPermissionsResult(int requestCode,
            String permissions[], int[] grantResults) {
```

```
        switch(requestCode) {
            case MainActivity.READ_PHONE_STATE_CODE: {
                if(grantResults.length > 0 &&
                        grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                    detectMsisdn();
                }
                else {
                    // notify user about insufficient permissions
                    Toast.makeText(this, "Cannot autodetect number due to " +
                            "insufficient permissions",
                            Toast.LENGTH_SHORT).show();
                }
            }
            default: {
                // do nothing
            }
        }
    }
}
```

```
-------------------------------------------------------------
Filename: src/main/java/com/transmisms/androidcompanion/SmsReceiver.java
-------------------------------------------------------------

package com.transmisms.androidcompanion;

import com.google.gson.Gson;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.gsm.SmsMessage;
import android.util.Log;
import java.util.Date;
import java.util.Queue;
import java.util.concurrent.ConcurrentLinkedQueue;

public class SmsReceiver extends BroadcastReceiver {

    private Gson gson = new Gson();
    private Queue<String> msgQueue = new ConcurrentLinkedQueue<>();

    public String pop() {
        return this.msgQueue.poll();
    }

    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle bundle = intent.getExtras();
        if(bundle != null) {
            Object[] pduArr = (Object[])bundle.get("pdus");
            SmsMessage[] msgArr = new SmsMessage[pduArr.length];

            // NOTE: this receiver does not support multipart messages;
            //       received multipart messages' parts will be just treated
            //       as individual messages
            for(int i = 0; i < msgArr.length; i++) {
                msgArr[i] = SmsMessage.createFromPdu((byte[])pduArr[i]);
                String sender = msgArr[i].getOriginatingAddress();
                String msgBody = msgArr[i].getMessageBody().toString();
                long timestamp = msgArr[i].getTimestampMillis();
                String jsonStr = gson.toJson(new ServerAction(sender, msgBody,
                        timestamp));
                Log.d(SmsReceiver.class.getName(), "Sending to peer: "
                        + jsonStr);
                // finally send via zmq (through msgQueue)
                this.msgQueue.offer(jsonStr);
            }
            // starting from API 19(Kitkat) we cannot  prevent other receivers
            // from receiving the same broadcast/intent
        }
    }
}
```

```
-------------------------------------------------------------
Filename: src/main/res/layout/main_layout.xml
-------------------------------------------------------------

<?xml version="1.0" encoding="utf-8"?>

<android.support.design.widget.CoordinatorLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:fitsSystemWindows="true"
    android:noHistory="true"
    android:launchMode="singleTop"
    tools:context="com.rootbearlabs.androidtest1.MainActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/mainAppBar"
        android:layout_width="match_parent"
        android:layout_height="?attr/actionBarSize"
        android:elevation="4dp"
        android:background="?attr/colorPrimary"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        app:popupTheme="@style/ThemeOverlay.AppCompat.Light"

        android:layout_gravity="top"
        />

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

        android:layout_gravity="center"
        />

    <android.support.design.widget.FloatingActionButton
        android:id="@+id/floatingActionButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginRight="24dp"
        android:layout_marginBottom="24dp"
        android:src="@drawable/ic_play_arrow_white_24dp"
        android:clickable="true"
        android:onClick="onFabButtonClick"

        app:fabSize="normal"
        app:elevation="6dp"
        app:pressedTranslationZ="6dp"
        app:borderWidth="0dp"

        android:layout_gravity="end|bottom"
        />


</android.support.design.widget.CoordinatorLayout>
```

```
-------------------------------------------------------------
Filename: src/main/res/layout/port_input_layout.xml
-------------------------------------------------------------

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/setup_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:fitsSystemWindows="true"
    >

    <EditText
        android:id="@+id/port_field"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:layout_marginStart="24dp"
        android:layout_marginEnd="24dp"
        android:inputType="number"
        android:text=""
        android:hint=""
        />
</android.support.constraint.ConstraintLayout>
```

```
-------------------------------------------------------------
Filename: src/main/res/layout/setup_layout.xml
-------------------------------------------------------------

<?xml version="1.0" encoding="utf-8"?>
```

```
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/setup_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:fitsSystemWindows="true"
    android:noHistory="true"
    android:launchMode="singleTop"
    tools:context="com.rootbearlabs.androidtest1.SetupActivity">

    <android.support.v7.widget.Toolbar
        android:id="@+id/setup_appbar"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:minHeight="?android:attr/actionBarSize"
        android:elevation="4dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintDimensionRatio="H,16:9"
        app:titleMarginStart="24dp"
        app:titleMarginBottom="16dp"
        android:background="?attr/colorPrimary"
        android:theme="@style/ThemeOverlay.AppCompat.Dark.ActionBar"
        android:gravity="bottom"
        app:titleTextAppearance="@style/LargeTitleText"
        />

    <TextView
        android:id="@+id/setup_text_1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="24dp"
        android:layout_marginEnd="24dp"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/setup_appbar"
        android:textSize="16sp"
        android:text="Your peers will connect to you using your phone number."
        />

    <EditText
        android:id="@+id/setup_msisdn_field"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="24dp"
        android:layout_marginEnd="24dp"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toBottomOf="@id/setup_text_1"
        android:inputType="phone"
        android:textSize="24sp"
        android:text=""
        android:hint="Your phone number"
        />

    <Button
        android:id="@+id/setup_detectmsisdn_button"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/setup_msisdn_field"
        style="@style/Widget.AppCompat.Button.Borderless.Colored"
        android:onClick="onDetectMsisdnButtonClicked"
        android:text="DETECT PHONE NUMBER"
        />

    <Button
        android:id="@+id/setup_finish_button"
        android:enabled="false"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        style="@style/Widget.AppCompat.Button.Borderless.Colored"
        android:drawableEnd="@drawable/ic_chevron_end"
        android:drawableTint="@color/colorAccent"
        android:text="FINISH"
        android:onClick="onFinishButtonClick"
        />

    <View
        android:id="@+id/setup_finish_divider"
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintBottom_toTopOf="@id/setup_finish_button"
        android:background="?android:attr/listDivider"
        />

    <Button
        android:id="@+id/setup_portcustomize_button"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintBottom_toTopOf="@id/setup_finish_divider"
        style="@style/Widget.AppCompat.Button.Borderless.Colored"
        android:text="Customize port bindings"
        android:onClick="onPortCustButtonClick"
        />
</android.support.constraint.ConstraintLayout>



----------------------------------------------------------------
Filename: src/main/res/menu/main_menu.xml
----------------------------------------------------------------

<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_setup"
        android:title="Setup device"
        app:showAsAction="never"
        android:onClick="onSetupMenuItemClick"
    />
    <item
        android:id="@+id/action_about"
        android:title="About"
        app:showAsAction="never"
    />
</menu>



----------------------------------------------------------------
Filename: src/main/res/values/colors.xml
----------------------------------------------------------------

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="blue">#1976D2</color> <!-- blue 700 -->
    <color name="darkBlue">#004BA0</color> <!-- blue 700 (dark) -->
    <color name="indigo">#5C6BC0</color> <!-- indigo 400 -->
    <color name="white">#FFFFFF</color>

    <color name="colorPrimary">@color/blue</color>
    <color name="colorPrimaryDark">@color/darkBlue</color>
    <color name="colorPrimaryText">@color/white</color>
    <color name="colorAccent">@color/indigo</color>
    <color name="colorNavigationBar">@color/blue</color>
</resources>



----------------------------------------------------------------
Filename: src/main/res/values/strings.xml
----------------------------------------------------------------

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Transmisms Companion</string>
</resources>
```

```
-------------------------------------------------------------
Filename: src/main/res/values/styles.xml
-------------------------------------------------------------

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
        <item name="windowNoTitle">true</item>
        <item name="windowActionBar">true</item>
    </style>
    <style name="SetupTheme" parent="AppTheme" />
    <style name="LargeTitleText"
            parent="TextAppearance.Widget.AppCompat.Toolbar.Title">
        <item name="android:textSize">24sp</item>
    </style>
</resources>



/*
 * You can also request a copy of this source from the library or the
 * laboratory manager/assistant. I have also archived this in case both
 * copies are not available. It's not recommended to copy source from this
 * document since binary dependencies such as images are not included.
 */
```

Acknowledgements

For the mouths that are deprived

that their hunger be satisfied,

For the worker who toils

that his burden be relieved,

For the benevolent master

and their followers to succeed,

And for the hopeful who kept on waiting

that their wishes be fulfilled.