life.augmented

ST25 Java SDK introduction (ST25 Software Development Kit)

ST25SDK in a nutshell (1/3)

Software library to be used in Java applications

What do I get from ST25 SDK?

Why shall I use it?

Multiplatform

- Can be run by any platform supporting JVM (Windows, Android, Linux, Mac)
- Some components can be re-used for iOS

Flexibility

- Abstraction for NFC Forum & ISO Commands
- Abstraction for Tags, and Readers
- RF commands isolated from UI interface
- Code can be reused in other Java applications
- Datasheet is optional. Tag features documented in API (Javadoc)

It accelerates and simplifies your development

ST25SDK in a nutshell (2/3)

What form does the ST25SDK take?

A public archive file (zip) containing:

- ST25SDK.jar Java library file
- API documentation (HTML Javadoc files)
- Native libraries for reader board (CR95HF, ST25R3911B-DISCO)
- Basic applications source code
  (Android and PC Java source code + executables)
- Helper classes

How do I get the latest ST25 SDK package?

http://st.com/st25sdk

What if I wish more documentation?

Data brief + User Manual on st.com

life.augmented

ST25SDK in a nutshell (3/3)

How is the ST25SDK licensed?

SLA0052 (Mix_MyLiberty) for the public version

SLA0085 for the version with TruST25 features

Included at the beginning of each source file

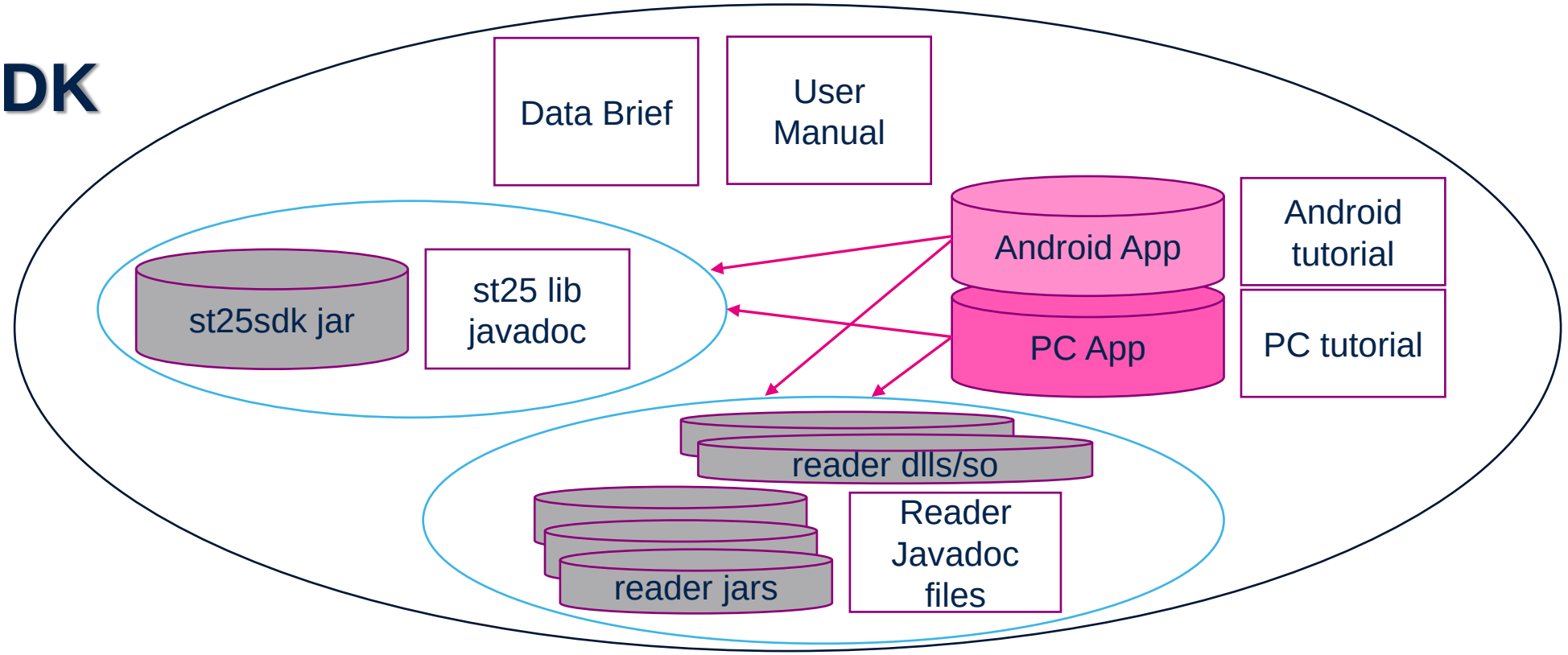
If I have any further questions?

https://community.st.com/community/st25-nfcrfid-tags-and-readers

life.augmented

ST25 SDK content

ST25 SDK



Application examples:

Detect tag
Extract size information from **getSystemInfo** with **tag.getMemSizeInBytes()** call
Write NDEF message containing a URI record

ST25SDK library presentation

ST25SDK Java library

- The ST25SDK.jar library is central to the SDK
  - Include it inside an application to benefit from a lot of tag-related features
- ST25SDK library architecture is designed to ease NFC application development

Object oriented structure

- Tags have common properties
- Specific fields and methods for tags that have extra features or different from base tags
- We do not reinvent the wheel every time, just add or override fields and methods
- Tags inherit features based on tag hierarchy

Cross platforms/readers support

- Can be run on any platform supporting Java Runtime Environment
  - Windows, Android, Linux, macOS
  - iOS: some components can be re-used (ndef)

Cache tag information

- Useful when tag is out of RF field

life.augmented

How to benefit from st25sdk: a few examples

- Explicit API allows the application to read or write **NDEF formatted message** or **raw binary data** into each memory area of the tag.

```
st25DVTag.writeNdefMessage(MultiAreaInterface.AREA2, ndefMsgToWrite);
ndefMsgFromTag = st25DVTag.readNdefMessage(MultiAreaInterface.AREA1);

st25DVTag.writeBytes(offset, dataToWrite);
byte[] dataFromTag = st25DVTag.readBytes(offset, length);
```

- The read or write actions work no matter what the tag type is (Type5, Type4A…)
  - User doesn’t have to change the command. It is identical for all tag types.

- In case of error, an explicit **STException** is raised
  - The user application doesn’t have to interpret the code returned by the command..

- With Type5 tags, extended commands should be used above the first 8kbits.
  - The application doesn’t have to bother with specific read/write commands. The st25sdk manages it

Coding with and without SDK

- Imagine that we have a ST25DV64K tag configured with 4 memory areas
  - we want to write a NDEF message containing the URI **www.st.com** into Area4.
- Let's compare the (pseudo)code with and without SDK:

Without st25sdk library

```
Build a NDEF message:
•  Either with your own code
•  Or with some other code like Android's NDEF message
   (https://developer.android.com/reference/android/nfc/
   NdefMessage.html)

// Serialize the data:
byte[] ndefData = ndefMsg.serialize();

// Read 'EndA3' register to know the « End Of Area 3 »
and thus the beginning of Area4.
Send readConfig command: 22 a0 02 9e 11 e3 03 00 27 02
e0 09
00 bf

// Compute the Area4 offset from EndA3 register value

// Write the data to the relevant memory addresses and
with the relevant command:

Send extendedWriteSingleBlock command: 22 31 9e 11 e3 03
00 27 02 e0 02 06 03 00 d1 01

Send extendedWriteMultipleBlock command: 22 34 9e 11 e3
03 00 27 02 e0 03 06 02 00 07 55 01 73 74 2e 63 6f 6d fe
ff ff

Send extendedWriteSingleBlock command: 22 31 9e 11 e3 03
00 27 02 e0 02 06 03 0b d1 01
```

With st25sdk library

```
NDEFMsg ndefMsg = new NDEFMsg();

UriRecord uriRecord = new
UriRecord(NDEF_RTD_URI_ID_HTTP_WWW,
                                        "st.com");
ndefMsg.addRecord(uriRecord);

st25DVTag.writeNdefMessage(MultiAreaInterface.AREA4,
ndefMsg);
```

The same code also works for other tags with multiple Areas like M24SR or ST25TV!

It runs on Android and PC.

ST25SDK library architecture

API levels

- API stands for Application Programming Interface, i.e. the list of methods used by programs to access the ST25SDK library
- Two levels of API

Low level Command classes

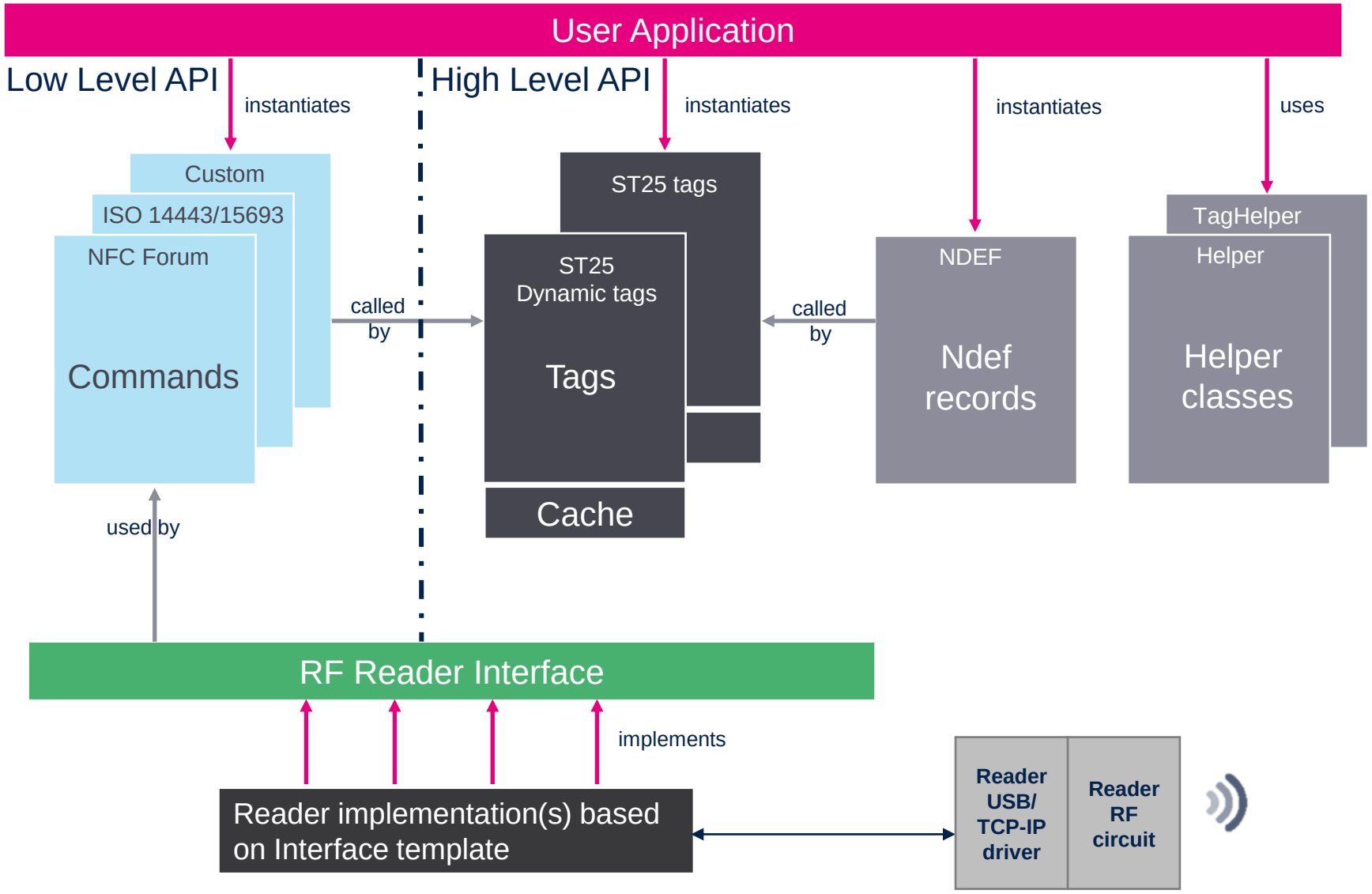
- Very close to standards and tag datasheets
- Hardly no parameter verification is done
- Used **by test applications** to check the behavior of the tag when sending RF commands with any parameter, including incorrect ones
- Used by **industrial/commercial applications** communicating with **several tags at once**

High level Tag and Helper classes

- Eases the life of the application developer
- Tag classes call Command classes after performing parameter checks to prevent incorrect use of the tag/SDK
- Used by mobile or industrial applications to communicate robustly with a single tag

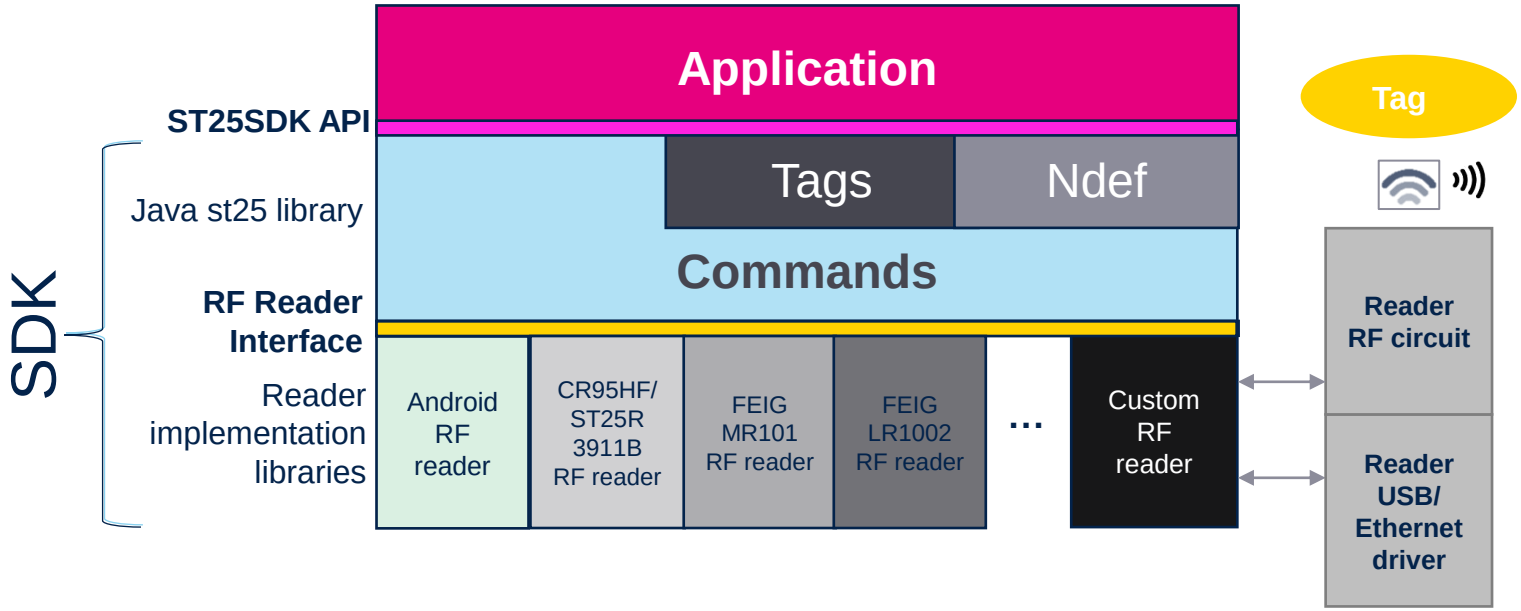
- One common interface for all readers: Android, ST discovery kits, FEIG readers are supported today

ST25SDK architecture overview



API layers

- Example with getMemSizeInBytes()



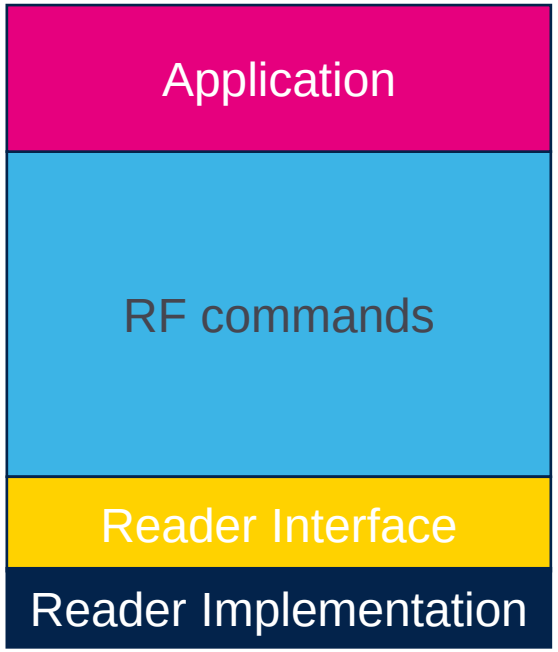
- **An application using the ST25SDK can:**
  - Instantiate a Command class object to access RF command methods OR
  - Instantiate a NFCTag object to access generic and tag-specific methods OR
  - Instantiate an Ndef class

- **The RF reader interface must**
  - Be used with a reader implementation, needed for Commands and Tags layers

Command API

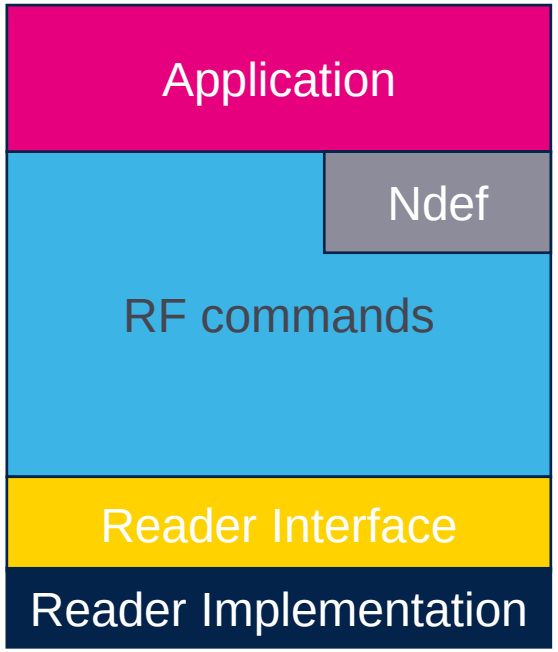
- **RF commands**
  - Low level RF commands, very close to standards and datasheets
  - Easy to work with multiple tags
  - Very little extra processing

- **Commands are separated in**
  - NDEF commands calling on unitary commands
  - Unitary RF commands corresponding to:
    - NFC Forum RF commands from specifications
    - Iso14443/15693 commands from specifications
    - Tag-specific commands from datasheet
    - Iso7816 commands from specifications



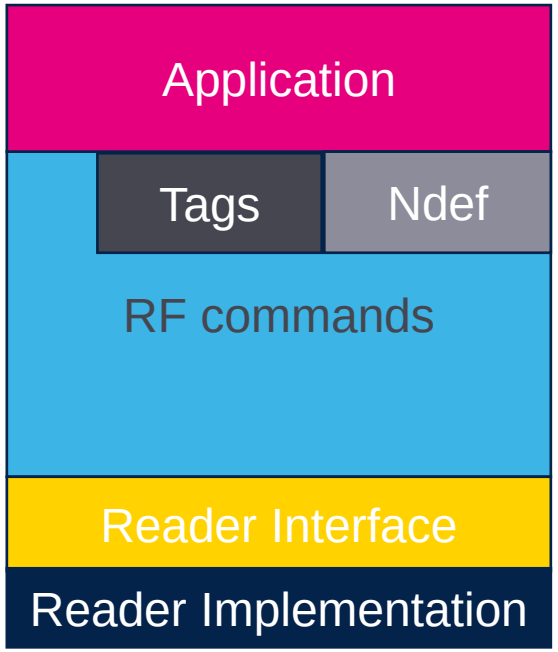
NDEF API

- The **NDEF API** allows
  - To parse NDEF from raw bytes
  - To create/edit/read/write NDEF with well known records
  - To manage multiple records
- It is used by ST25 tags but can also be used to address multiple tags simultaneously



Tag API

- The **Tag API** is the highest level of API to work with a single tag
  - It contains the supported RF commands
  - High level commands are also available, such as readBlocks() that will manage reading through areas, using readMultiple commands when supported, etc.
- Inheritance hierarchy:
  - ST25DVTag ➔ STType5MultiAreaTag ➔ STType5Tag ➔ Type5Tag ➔ NFCTag
- Tags are implemented in the application based on the product identified using a **SDK helper** class



TagHelper classes

- **TagHelper**
  - Identifies a tag from its UID

```
NFCTag recognizedNFCTag;
NfcTagTypes tagType = readerInterface.decodeTagType(uid);
ProductID productName;
if (tagType == NfcTagTypes.NFC_TAG_TYPE_V) {
    productName = TagHelper.identifyTypeVProduct(readerInterface, uid);
} else if (tagType == NfcTagTypes.NFC_TAG_TYPE_4A) {
    productName = TagHelper.identifyType4Product(readerInterface, uid);
} else {
    productName = TagHelper.identifyProduct(readerInterface, uid);
}
switch (productName) {
    case PRODUCT_ST_ST25DV02K_W:
        recognizedNFCTag = new ST25DVWTag(readerInterface, uid);
        break;
…
    case PRODUCT_ST_ST25TA02K_D:
        recognizedNFCTag = new ST25TA02KDTag(readerInterface, uid);
        break;
}
```

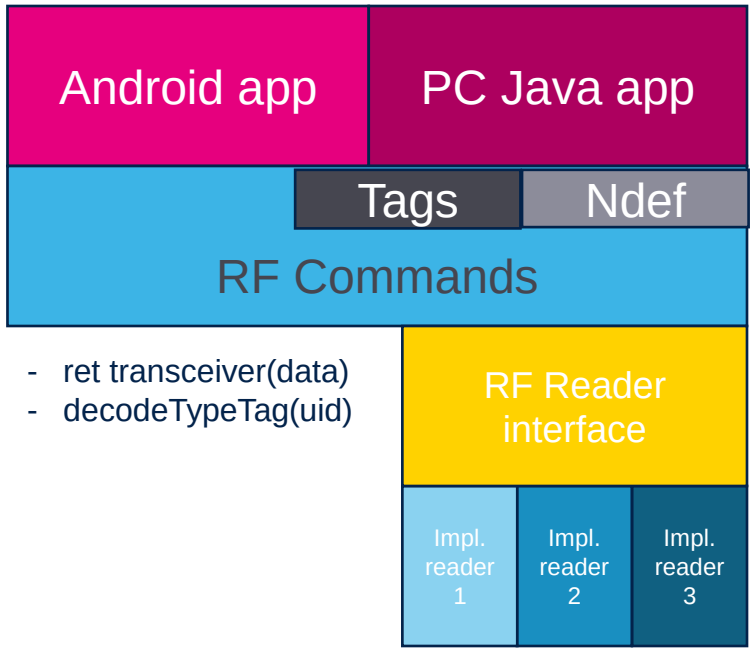
Helper classes

- **Helper**
  - Methods for data conversions

```
byte[] uid = iso15693Cmd.inventory()[0];  // First element from inventory
String uidString = Helper.convertByteArrayToHexString(uid);
```

RF reader interface

- The RF reader interface can be seen as the porting layer of the ST25SDK
  - Each reader must implement the SDK's readerInterface methods
- Some examples of implementation are delivered with the SDK:
  - Android
  - CR95HF board
  - ST25R3911B-DISCO, ST25R3916-DISCO
  - FEIG ELECTRONIC's PR101, MR102, LR1002, CPR30



Code examples

- Basic code example for CR95HF/ST25R3911B-DISCO:
  - Error handling and import statements were removed for clarity

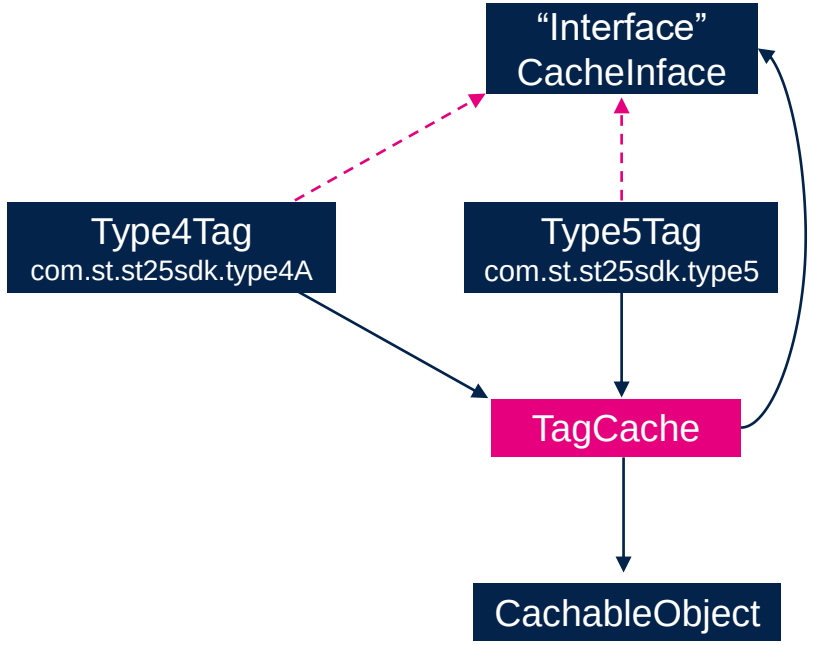
```
// Create new cr95/3911b reader object from native library
STReader myReader = new STReader();
myReader.connect();

// Retrieve low level RF interface for that reader
RFReaderInterface readerInterface = myReader.getTransceiveInterface();

Iso15693Command iso15693Cmd = new Iso15693Command(readerInterface, null);
byte[] uid = iso15693Cmd.inventory()[0];  // First element from inventory

Type5Tag myTag = new Type5Tag(readerInterface, uid);
myTag.getMemSizeInBytes();
```

Tag cache

- An internal cache system allows to keep tag information when the tag is not in the reader field
  - Especially useful for smartphone application to keep using tag data once the tag is removed from the RF field (ex. read the data on the smartphone screen after a Tap action)

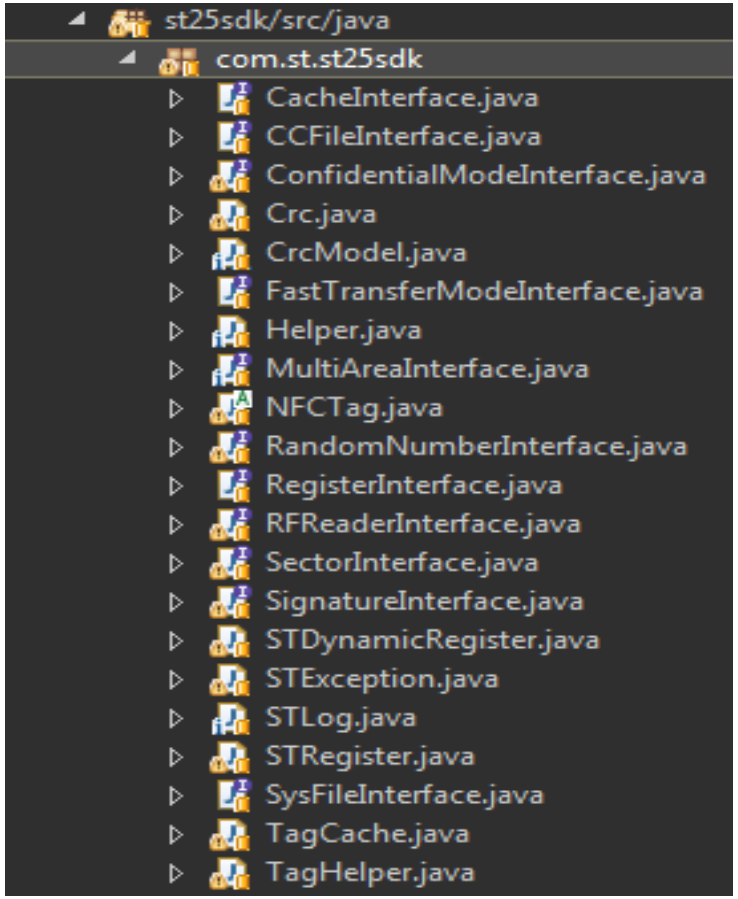
- A Tag is made of several internal objects like NDEF File, System File, CC File, Registers…etc.
  - The information can be read once (by issuing a RF command). If the same information is requested later on by the application, the SDK will check if it is present in cache and valid. If yes, it will return it without executing another RF command.

- Cache can be enabled/disabled



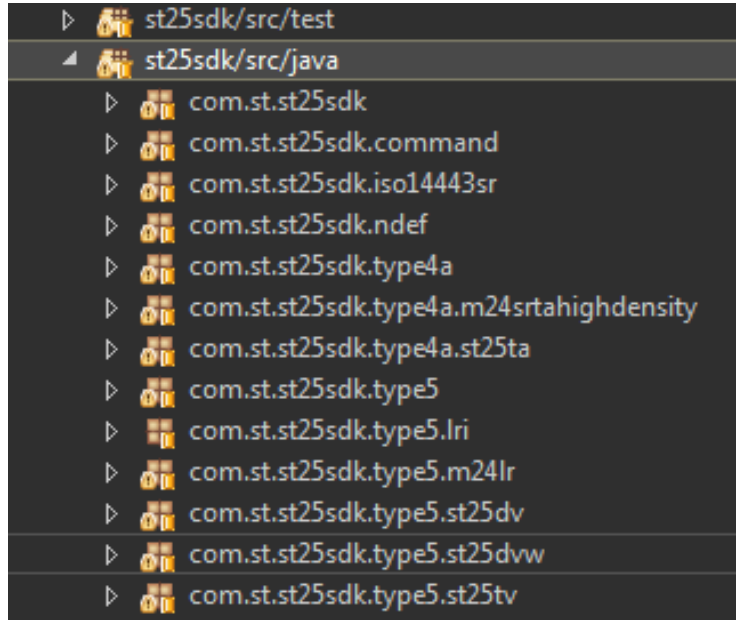
ST25SDK design

SDK code organization

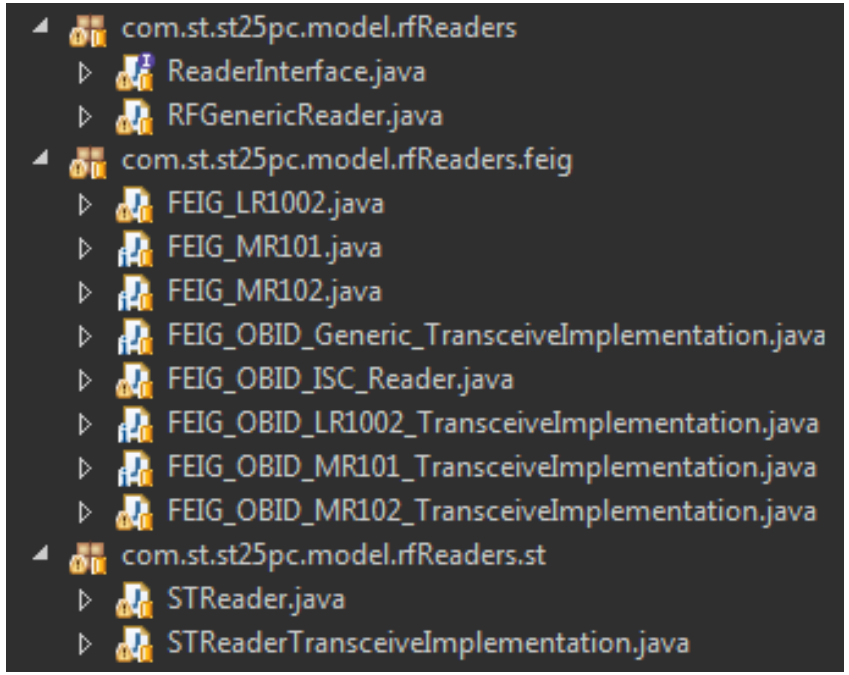


ST25SDK contains several Java packages

- ST25SDK root folder contains classes:
  - Helper
  - TagHelper
- Sub packages:
  - Commands
    - st25sdk.command
    - st25sdk.ndef
  - Tags
    - iso14443sr
    - st25sdk.type4a
    - st25sdk.type5



Reader implementation



- **For PC app**, support is provided for
  - ST Readers
    - ST25R3911B-DISCO
    - ST25R3916-DISCO
    - CR95HF
  - FEIG ELETRONIC
    - MR102
    - LR1002
    - CPR30
- **RF reader interface** implementations are delivered inside SDK in com.st.st25pc
  - Java classes + Java Native Interface in C/C++ to interact with .dll/.so
- **For Android**, native **NFC API** is used

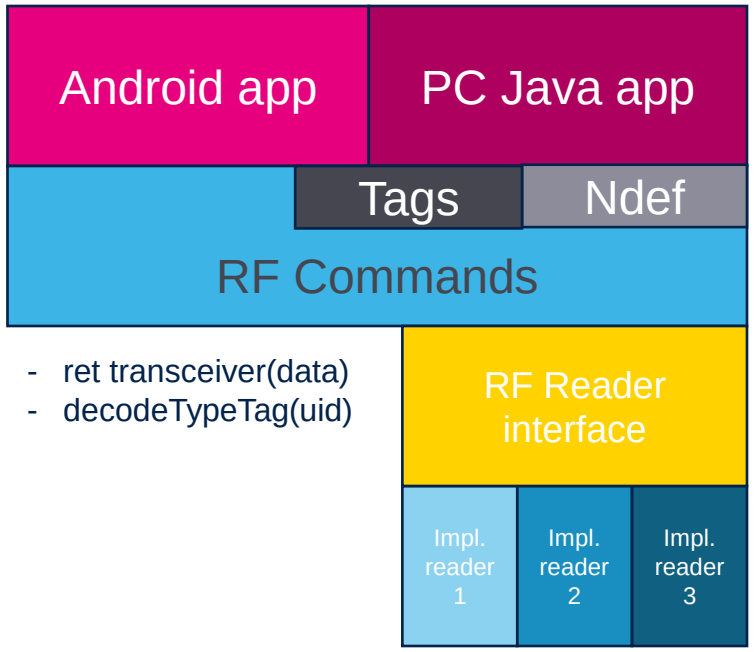
Reader support

Reader support

- ST25R3911B-DISCO and ST25R3916-DISCO
  - The RF reader interface implementation communicates via JNI classes to C/C++ code
- **CR95HF** (CR95HF Board in M24LR-DISCOVERY)
  - The RF reader interface implementation is done through JNI classes C/C++
  - Build with gcc on Windows and Linux with usb-hid backend
- **Industrial FEIG ELECTRONIC readers**
  - The RF reader interface implementation is done in java with proprietary classes delivered by partner FEIG ELECTRONIC

RF reader interface

- Some examples of implementation are delivered with the SDK compatible with:
  - Android
  - CR95HF / ST25R3911B-DISCO / S25R3916-DISCO
- All those examples were written by ST
  - customers can support other readers through the JNI mechanism
- Source code of st25sdk.jar not delivered in SDK



Android / iOS Support

Android / iOS support

- **Android**
  - The RF reader interface implementation is directly plugged on Android SDK APIs (thin wrapper)

- **iOS**
  - A tool named j2objc to convert the ST25SDK Java code to Objective C:
    - « J2ObjC is an open-source command-line tool from Google that translates Java source code to Objective-C for the iOS (iPhone/iPad) platform. »
  - Ndef package of st25sdk library is re-used for the ST25 iOS application (NFC Tap)

Android specific: TagDiscovery

- The ST25SDK Zip file contains a helper class called **TagDiscovery**
  - located in integration/android/helper/TagDiscovery.java

- It is very useful for Android development to instantiate a class corresponding to the tag that was tapped
  - When you tap an NFC tag in Android environment, the application receives a notification called intent
  - This intent contains a tag object that can be used to communicate with the tag.
  - This tag is an object defined by Android. It contains some information about the tag (UID, type, memory size…) and you can use it to perform some standard actions like reading or writing NDEF messages.

- The **TagDiscovery** class allows to convert the tag, as defined by Android, into a tag recognized by the ST25SDK.
- The **new tag object** (recognized by the ST25SDK) can then be used to perform any action, including calling **ST proprietary commands**.

How to use the TagDiscovery

- Here is a code snippet showing how to use the **TagDiscovery** class to obtain an object used to execute commands:

```
@Override
public void onResume() {
        super.onResume();

        Intent intent = getIntent();
        if(intent == null) {
            return;
        }

        Tag androidTag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
        if (androidTag != null) {
            // Perform tag discovery in an asynchronous task
            // onTagDiscoveryCompleted() will be called when the discovery is completed.
            new TagDiscovery(this).execute(androidTag);
        }
}

@Override
public void onTagDiscoveryCompleted(NFCTag nfcTag, TagHelper.ProductID productId) {
        // You can save the nfcTag object and start using it

        // Example:
        NDEFMsg ndefMsg = nfcTag.readNdefMessage();
}
```

- The **TagDiscovery** is performed in an asynchronous task
  - onTagDiscoveryCompleted() is notified when the discovery is finished
- The '**nfcTag**' object can then be used to execute some ST25SDK commands.

life.augmented

Solutions for NFC / RFID tags & readers

ST25

ST25 SIMPLY MORE CONNECTED

Thank you

life.augmented