life.augmented

ST25SDK iOS

ST25

Introduction

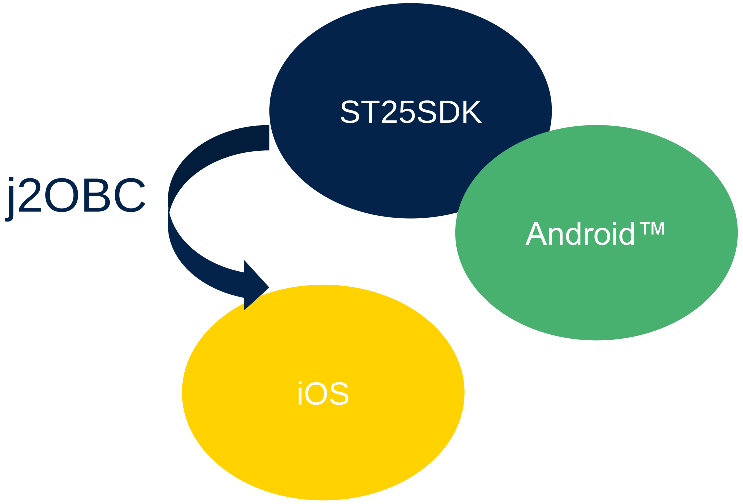
- ST25SDK is a software development kit providing a rich and comprehensive library to interact with ST25 tags and dynamic tags.
- ST25SDK contains all the necessary classes and abstractions to help the user in the development of mobile or desktop Java™ applications.

- ST25SDKiOS is the porting of the ST25SDK Java™ into iOS.
- This presentation will illustrate how we used the J2ObjC tool setup to achieve code sharing between our Android projects and iOS.
- It will show how we installed it and some sample code.

Why sharing code ?

- Initially, ST25SDK was written in Java™ for running applications on any platform supporting JVM (Windows®, Android™, Linux® and macOS®).
- Porting ST25SDK into iOS has many advantages :
  - Code is written once
  - Reducing maintenance
  - Similar behavior between iOS and on Android.

- Thanks to **J2OBJC** Tool for converting Java code into Objective-C code (iOS native language).

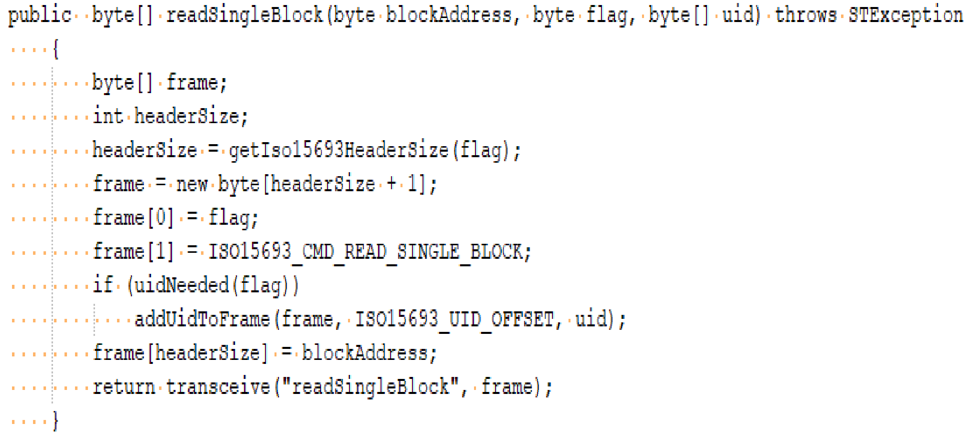


J2OBJC What is it ?

- It translates Java source code to Objective-C for the iOS (iPhone/iPad) platform.
- Open source google project : ref to https://developers.google.com/j2objc
- Source to Source compiler.
- You need java source code in input. Ex : ST25SDK
- On Android™ you run the java code.
- On iOS™ you run the translated Objective-C code.

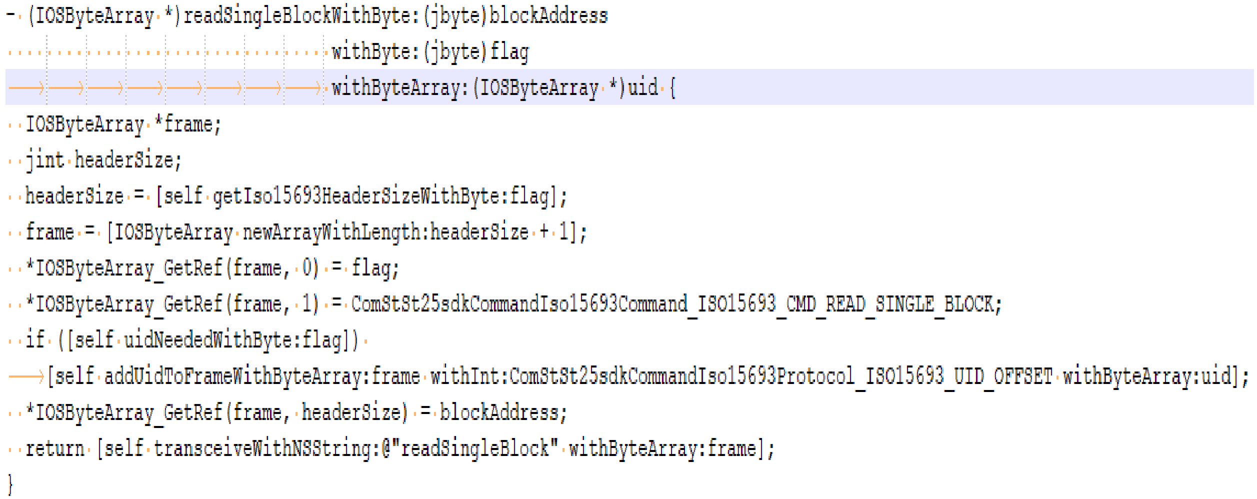
J2OBJC Example

- Here below, a short example of the Java ™ function `readSingleBlock` translated into J2OBJC :



```
public  byte[] readSingleBlock(byte blockAddress, byte flag, byte[] uid) throws STException
    {
        byte[] frame;
        int headerSize;
        headerSize = getIso15693HeaderSize(flag);
        frame = new byte[headerSize + 1];
        frame[0] = flag;
        frame[1] = ISO15693_CMD_READ_SINGLE_BLOCK;
        if (uidNeeded(flag))
            addUidToFrame(frame, ISO15693_UID_OFFSET, uid);
        frame[headerSize] = blockAddress;
        return transceive("readSingleBlock", frame);
    }
```

J2OBJC



```
- (IOSByteArray *)readSingleBlockWithByte:(jbyte)blockAddress
                                 withByte:(jbyte)flag
                                 withByteArray:(IOSByteArray *)uid {
  IOSByteArray *frame;
  jint headerSize;
  headerSize = [self getIso15693HeaderSizeWithByte:flag];
  frame = [IOSByteArray newArrayWithLength:headerSize + 1];
  *IOSByteArray_GetRef(frame, 0) = flag;
  *IOSByteArray_GetRef(frame, 1) = ComStSt25sdkCommandIso15693Command_ISO15693_CMD_READ_SINGLE_BLOCK;
  if ([self uidNeededWithByte:flag])
    [self addUidToFrameWithByteArray:frame withInt:ComStSt25sdkCommandIso15693Protocol_ISO15693_UID_OFFSET withByteArray:uid];
  *IOSByteArray_GetRef(frame, headerSize) = blockAddress;
  return [self transceiveWithNSString:@"readSingleBlock" withByteArray:frame];
}
```

life.augmented

J2OBJC Limitation

- Does not handle UI
- Forces Obj-C into project and bridging header between ObjC and Swift languages(Duh!).
- Java code can use only the translated objects of the JRE.
- Limited 3rd party java libraries.

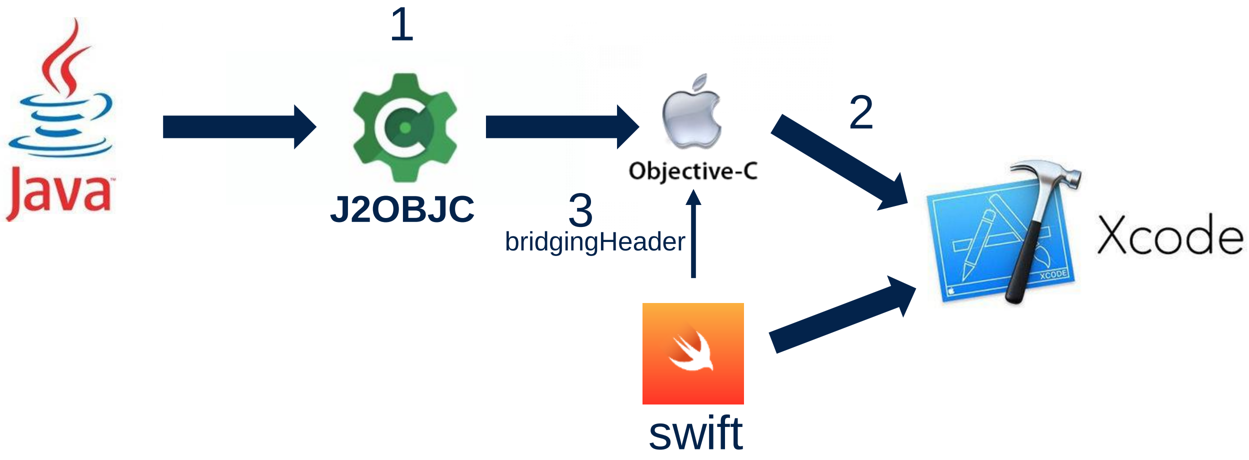
J2OBJC Requirements

- iOS Developement setup : Mac + XCode
- Java™ JDK1.8 or higher version.
- JRE translated into OBJ-C :
  - https://github.com/google/j2objc/releases

life.augmented

ST25SDK iOS

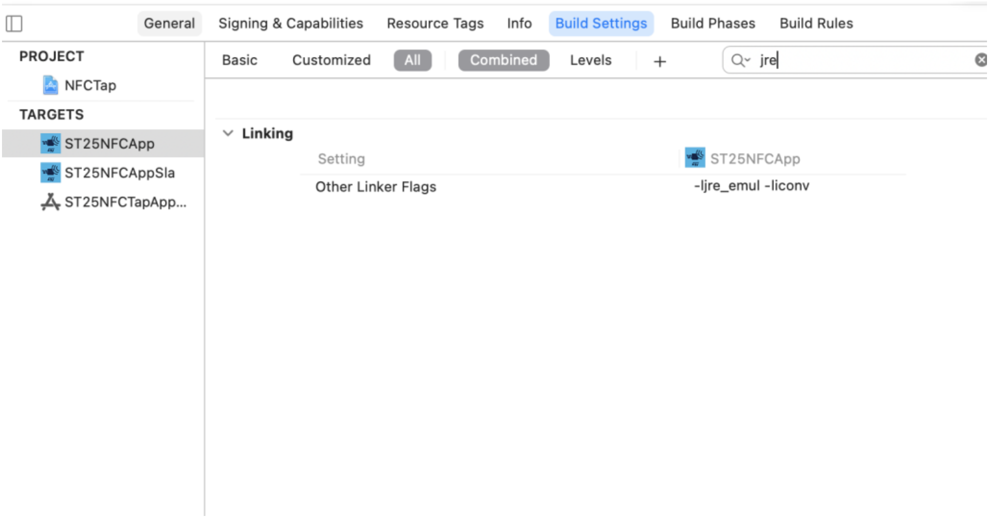
- The porting of the ST25SDK into ST25SDKiOS was done in three phases :
    1. We run the J2ObjC command line tool to convert the whole ST25SDK Java into Objective-C.
    2. Then, we exported every ObjC files into our Xcode project.
    3. We used a bridging header in Xcode in order to add Objective-C files to our existing Swift app.



Setting up Xcode 1/3

- Open XCode with iOS NFC Tap project.
  - Ref to https://www.st.com/content/st_com/en/products/embedded-software/st25-nfc-rfid-software/stsw-st25ios001.html

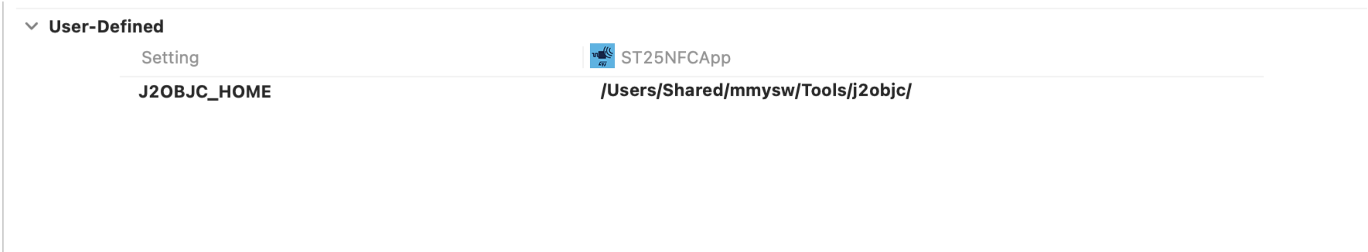
- Linking the JRE:
  - Navigate to the *Build Settings* tab and search for *Other Linker Flags.*
  - add: *-ljre_emul –liconv*. This will link the JRE emulation library.



Other Linker Flags -ljre_emul -liconv

Setting up Xcode 2/3

- Specifying J2OBJC Home path:
  - In order for Xcode to know about the J2OBJC and to compile we need to specify where the J2OBJC is.
  - In the *Build Settings* hit the + (near the search bar) and select *Add User-Defined Setting*.
  - Name the setting to J2OBJC_HOME and set the value to the J2objC folder



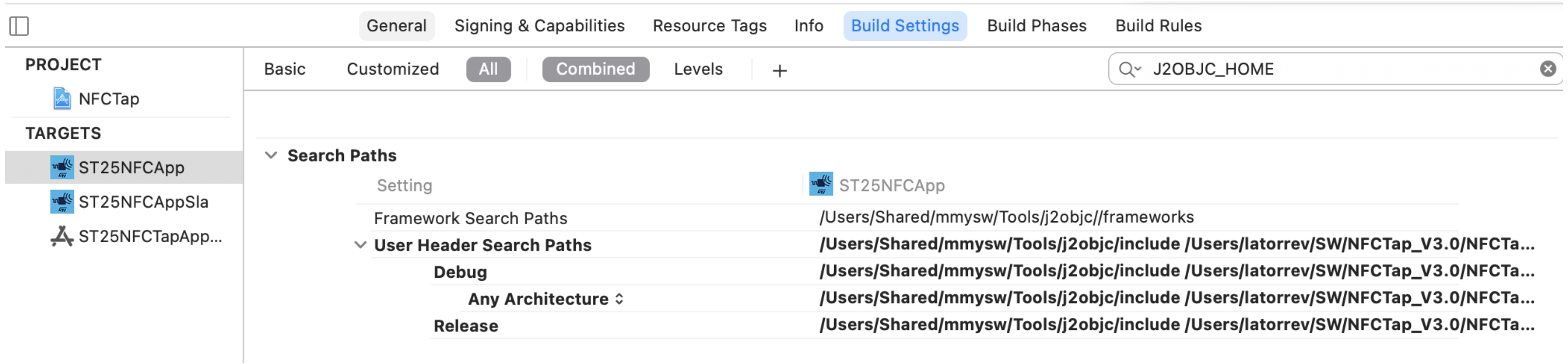
User-Defined

Setting	ST25NFCApp
J2OBJC_HOME	/Users/Shared/mmysw/Tools/j2objc/

Xcode

Setting up Xcode 3/3

- Updating the Search Path:
  - In the *Build Settings* under *Search Paths* append to the:
    - Framework Search Path: *${J2OBJC_HOME}/frameworks*
    - Library Search Path: *${J2OBJC_HOME}/lib*
    - User Header Search Paths: *${J2OBJC_HOME}/include*



Search Paths

Setting	ST25NFCApp
Framework Search Paths	/Users/Shared/mmysw/Tools/j2objc//frameworks
User Header Search Paths	**/Users/Shared/mmysw/Tools/j2objc/include /Users/latorrev/SW/NFCTap_V3.0/NFCTa...**
Debug	**/Users/Shared/mmysw/Tools/j2objc/include /Users/latorrev/SW/NFCTap_V3.0/NFCTa...**
Any Architecture ⇕	**/Users/Shared/mmysw/Tools/j2objc/include /Users/latorrev/SW/NFCTap_V3.0/NFCTa...**
Release	**/Users/Shared/mmysw/Tools/j2objc/include /Users/latorrev/SW/NFCTap_V3.0/NFCTa...**

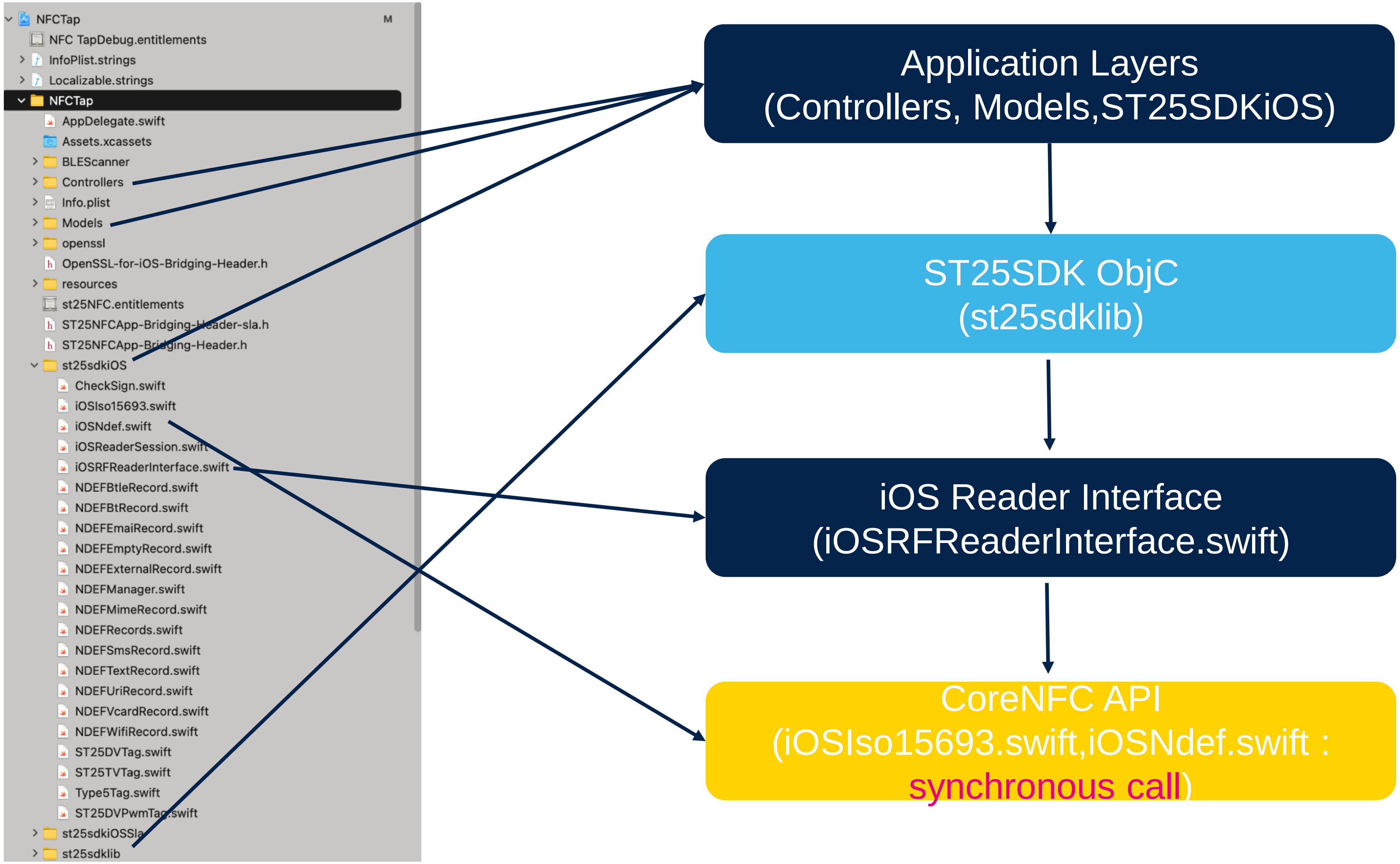
- At this point, you re ready to use ST25SDKiOS into your project !!!

life.augmented

Calling ST25SDKiOS code

- After all the setup we've reached a place where we can start to be productive. As you might have seen, J2ObjC will output Objective-C code (Duh!).
- Every ST25SDK Objc are prefixed with Java™ package name.
  - Ex : Java™ Class ***UriRecord*** in package ***com.st.st25sdk.ndef***
    => ObjC Class **ComStSt25sdkNdef*UriRecord***
- Swift :
  - If your project is a Swift project, we need to add a Bridging Header to the project. Next, open your bridging header and add the following.
    - *// Import all the Java classes below (ex : for UriRecord.h)*
    - *#import "UriRecord.h"*
  - As you can see, you need to import all the classes that have been translated from Java otherwise they will not be visible in Swift.

ST25SDK iOS App Architecture



```
NFCTap                                  M
  NFC TapDebug.entitlements
  InfoPlist.strings
  Localizable.strings
  NFCTap
    AppDelegate.swift
    Assets.xcassets
    BLEScanner
    Controllers
    Info.plist
    Models
    openssl
    OpenSSL-for-iOS-Bridging-Header.h
    resources
    st25NFC.entitlements
    ST25NFCApp-Bridging-Header-sla.h
    ST25NFCApp-Bridging-Header.h
    st25sdkiOS
      CheckSign.swift
      iOSIso15693.swift
      iOSNdef.swift
      iOSReaderSession.swift
      iOSRFReaderInterface.swift
      NDEFBtleRecord.swift
      NDEFBtRecord.swift
      NDEFEmaiRecord.swift
      NDEFEmptyRecord.swift
      NDEFExternalRecord.swift
      NDEFManager.swift
      NDEFMimeRecord.swift
      NDEFRecords.swift
      NDEFSmsRecord.swift
      NDEFTextRecord.swift
      NDEFUriRecord.swift
      NDEFVcardRecord.swift
      NDEFWifiRecord.swift
      ST25DVTag.swift
      ST25TVTag.swift
      Type5Tag.swift
      ST25DVPwmTag.swift
    st25sdkiOSSla
    st25sdklib
```

Application Layers
(Controllers, Models,ST25SDKiOS)

ST25SDK ObjC
(st25sdklib)

iOS Reader Interface
(iOSRFReaderInterface.swift)

CoreNFC API
(iOSIso15693.swift,iOSNdef.swift : synchronous call)

ST25SDK iOS Layers

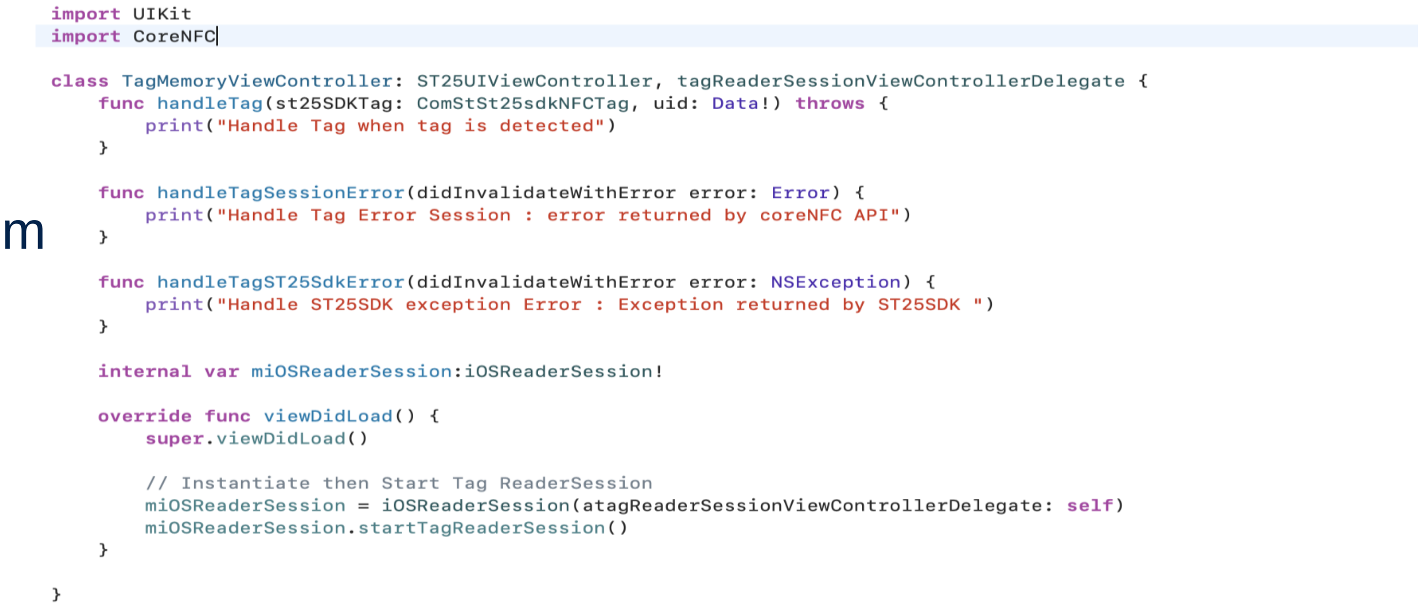
- iOS Application using ST25SDKiOS in composed of four main layers :
  - Application Layer :
    - Contains User Interface files (controller), Data Model files and Abstraction classes of ST25SDKiOS.
    - It uses the ST25SDK ObjC through the Commands, Tags or Helper classes.
  - ST25SDK ObjC Layer :
    - The ST25SDK transpiled into ObjC.
  - iOS ReaderInterface Layer (iOSRFReaderInterface.swift):
    - The reader interface is a contract between the ST25SDK Objc library and all reader classes. It ensures that all readers implement the same command set, making the library reader-independent.
    - In the case of iOS™ reader interface, the commands are transmitted to the coreNFC™ API.
    - It contains the *transceive()* method.
  - CoreNFC Api Layer (iOSIso15693.swift and iOSNdef.swift):
    - Native Interface commands to communicate with the NFC controller present on the smartphone. As ST25SDK Objc uses Synchronous commands, we have developed the *iOSIso15693.swift* and *iOSNdef.swift* files wrapping **Asynchronous** CoreNFC functions into **Synchronous** functions.

life.augmented

Code Example1: ReadSingleBlock 1/2

- This example shows how to use ST25SDKiOS in an iOS app to read block 0 of ST25 Type5 Tag (ST25DV or ST25TV).

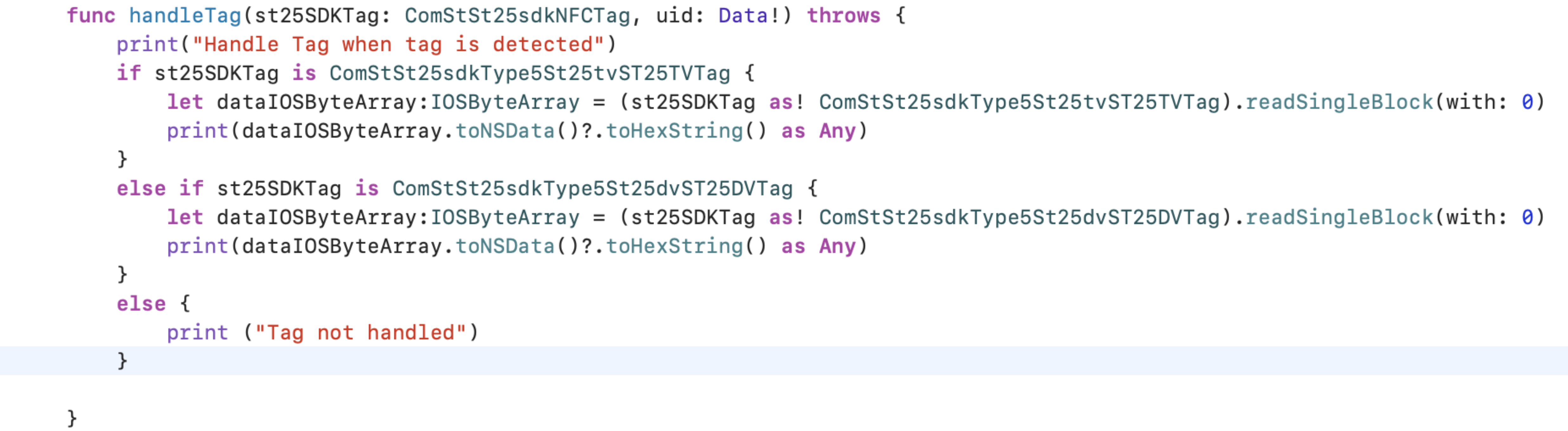
- Instantiate then Start a Tag reader session (ref to *iOSReaderSession.swift*) :
  - Create an *iOSReaderSession* object.
  - The *iOSReaderSession* requires a delegate object that conforms to the *tagReaderSessionViewControllerDelegate* protocol.
  - Start *iOSReaderSession*.
  - Adopting this protocol allows the delegate to receive notifications from the reader session when it:
    - Detects a Type5 Tag.
    - Encountering an error.



```
import UIKit
import CoreNFC

class TagMemoryViewController: ST25UIViewController, tagReaderSessionViewControllerDelegate {
    func handleTag(st25SDKTag: ComStSt25sdkNFCTag, uid: Data!) throws {
        print("Handle Tag when tag is detected")
    }

    func handleTagSessionError(didInvalidateWithError error: Error) {
        print("Handle Tag Error Session : error returned by coreNFC API")
    }

    func handleTagST25SdkError(didInvalidateWithError error: NSException) {
        print("Handle ST25SDK exception Error : Exception returned by ST25SDK ")
    }

    internal var miOSReaderSession:iOSReaderSession!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Instantiate then Start Tag ReaderSession
        miOSReaderSession = iOSReaderSession(atagReaderSessionViewControllerDelegate: self)
        miOSReaderSession.startTagReaderSession()
    }

}
```

Code Example1: ReadSingleBlock 2/2

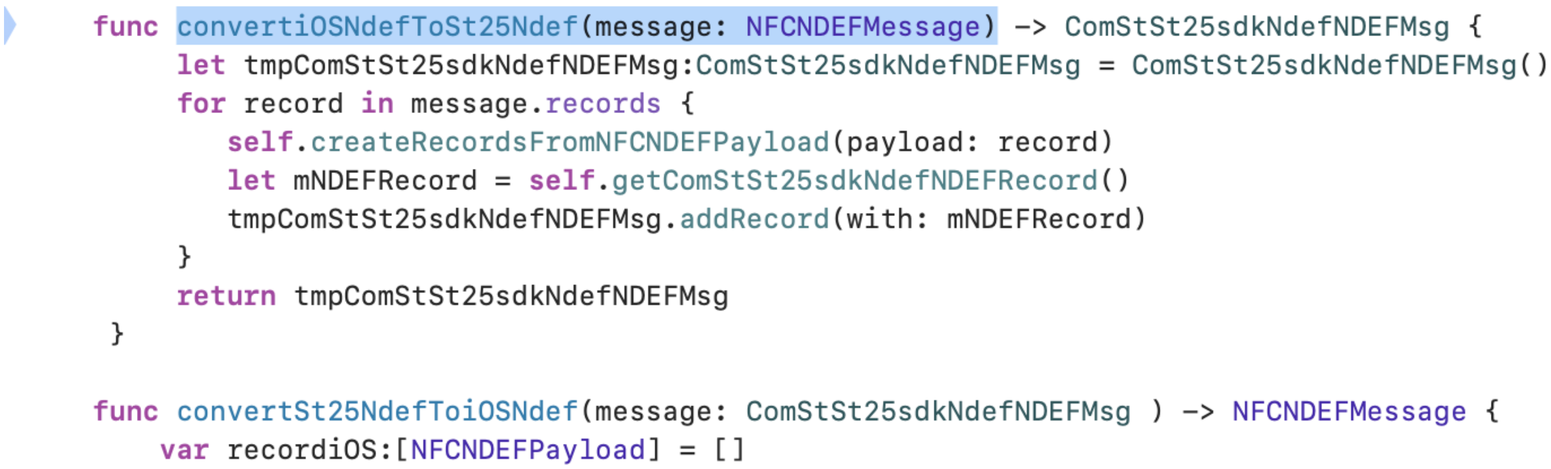
- Whenever iOSReaderSession detects Type5 tag, it calls the delegate method ‘*HandleTag*’ with the instance of *ComStSt25sdkNFCTag* and its *UID*.
- Call the ‘readSingleBlock’ ST25SDK function depending on ST25 Tag type.



```
func handleTag(st25SDKTag: ComStSt25sdkNFCTag, uid: Data!) throws {
    print("Handle Tag when tag is detected")
    if st25SDKTag is ComStSt25sdkType5St25tvST25TVTag {
        let dataIOSByteArray:IOSByteArray = (st25SDKTag as! ComStSt25sdkType5St25tvST25TVTag).readSingleBlock(with: 0)
        print(dataIOSByteArray.toNSData()?.toHexString() as Any)
    }
    else if st25SDKTag is ComStSt25sdkType5St25dvST25DVTag {
        let dataIOSByteArray:IOSByteArray = (st25SDKTag as! ComStSt25sdkType5St25dvST25DVTag).readSingleBlock(with: 0)
        print(dataIOSByteArray.toNSData()?.toHexString() as Any)
    }
    else {
        print ("Tag not handled")
    }

}
```

Code Example2: Read NDEF URI 1/3

- This example shows how to use ST25SDKiOS in an iOS app to read an NDEF URI from ST25 NFC Tag.
- !!! Tag MUST at least contains an empty Ndef for enabling the read/write of NDEF (coreNFC Limitation)!!!
- As coreNFC API comes with its own NDEF structure, we have developed the ‘*NDEFManager.swift*’ file that converts the coreNFC NDEFs into ST25SDKiOS NDEFs and vice-versa.

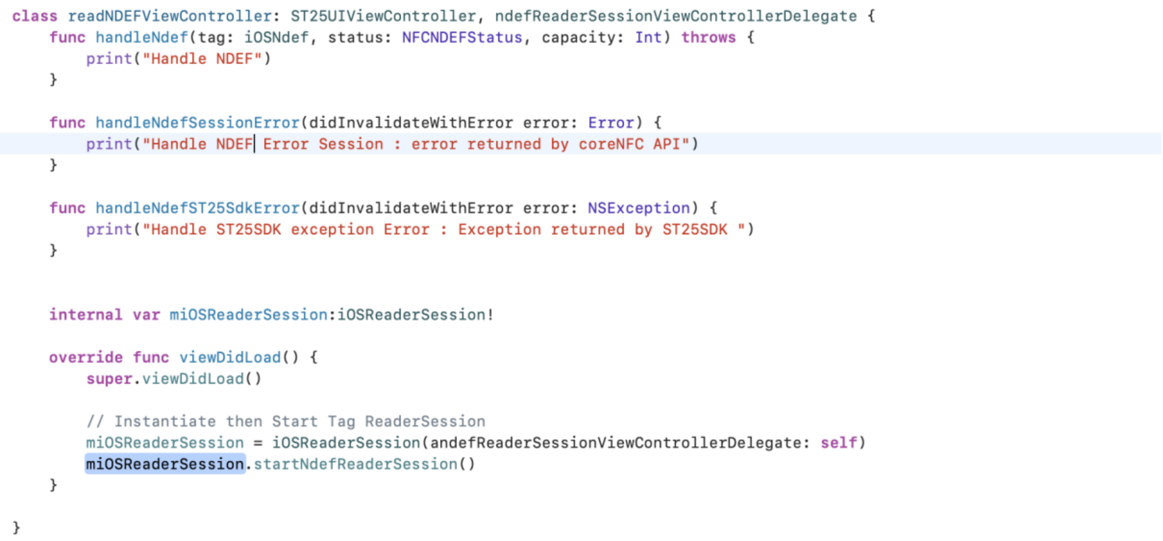


```
func convertiOSNdefToSt25Ndef(message: NFCNDEFMessage) -> ComStSt25sdkNdefNDEFMsg {
     let tmpComStSt25sdkNdefNDEFMsg:ComStSt25sdkNdefNDEFMsg = ComStSt25sdkNdefNDEFMsg()
     for record in message.records {
        self.createRecordsFromNFCNDEFPayload(payload: record)
        let mNDEFRecord = self.getComStSt25sdkNdefNDEFRecord()
        tmpComStSt25sdkNdefNDEFMsg.addRecord(with: mNDEFRecord)
     }
     return tmpComStSt25sdkNdefNDEFMsg
 }

func convertSt25NdefToiOSNdef(message: ComStSt25sdkNdefNDEFMsg ) -> NFCNDEFMessage {
    var recordiOS:[NFCNDEFPayload] = []
```

Code Example2: Read NDEF URI 2/3

- Instantiate then Start an NDEF reader session (ref to *iOSReaderSession.swift*) :
  - Create an *iOSReaderSession* object.
  - The *iOSReaderSession* requires a delegate object that conforms to the *ndefReaderSessionViewControllerDelegate* protocol.
  - Start *iOSReaderSession*.

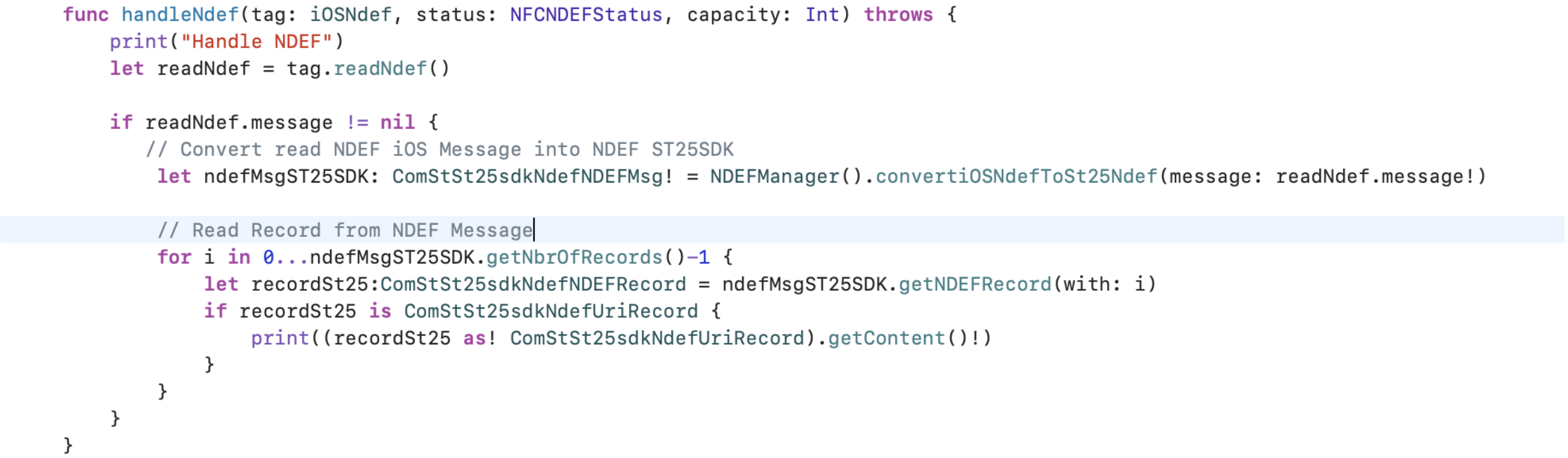
  - Adopting this protocol allows the delegate to receive notifications from the reader session when it:
    - Detects a NDEF message.
    - Encountering an error.



```swift
class readNDEFViewController: ST25UIViewController, ndefReaderSessionViewControllerDelegate {
    func handleNdef(tag: iOSNdef, status: NFCNDEFStatus, capacity: Int) throws {
        print("Handle NDEF")
    }

    func handleNdefSessionError(didInvalidateWithError error: Error) {
        print("Handle NDEF Error Session : error returned by coreNFC API")
    }

    func handleNdefST25SdkError(didInvalidateWithError error: NSException) {
        print("Handle ST25SDK exception Error : Exception returned by ST25SDK ")
    }


    internal var miOSReaderSession:iOSReaderSession!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Instantiate then Start Tag ReaderSession
        miOSReaderSession = iOSReaderSession(andefReaderSessionViewControllerDelegate: self)
        miOSReaderSession.startNdefReaderSession()
    }

}
```

Code Example2: Read NDEF URI 3/3

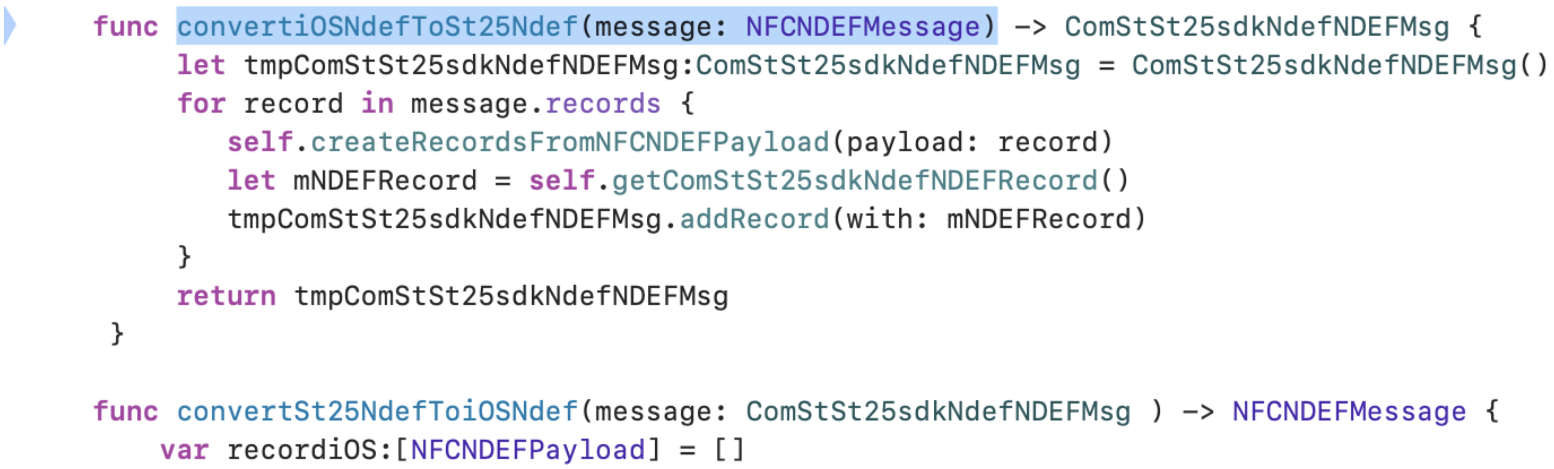
- Whenever iOSReaderSession reads NDEF message, it calls the delegate method ‘*HandleNdef*’ with an instance of *iOSNdef (ref to iOSNdef.swift)*.
- Call ‘NDEFManager’ to convert the CoreNFC NDEF into ST25SDK NDEF.
- Use the ST25SDK objects : ‘*ComStSt25sdkNDEFMsg*’, ‘*ComStSt25sdkNdefRecord*’ then ‘*ComStSt25sdkNdefUriRecord*’ to display URI.



```
func handleNdef(tag: iOSNdef, status: NFCNDEFStatus, capacity: Int) throws {
    print("Handle NDEF")
    let readNdef = tag.readNdef()

    if readNdef.message != nil {
       // Convert read NDEF iOS Message into NDEF ST25SDK
        let ndefMsgST25SDK: ComStSt25sdkNdefNDEFMsg! = NDEFManager().convertiOSNdefToSt25Ndef(message: readNdef.message!)

        // Read Record from NDEF Message
        for i in 0...ndefMsgST25SDK.getNbrOfRecords()-1 {
            let recordSt25:ComStSt25sdkNdefNDEFRecord = ndefMsgST25SDK.getNDEFRecord(with: i)
            if recordSt25 is ComStSt25sdkNdefUriRecord {
                print((recordSt25 as! ComStSt25sdkNdefUriRecord).getContent()!)
            }
        }
    }
}
```

Code Example3: Write NDEF URI 1/3

- This example shows how to use ST25SDKiOS in an iOS app to write an NDEF URI to ST25 NFC Tag.
- !!! Tag MUST at least contains an empty Ndef for enabling the read/write of NDEF (coreNFC Limitation)!!!
- As coreNFC API comes with its own NDEF structure, we have developed the '*NDEFManager.swift*' file that converts the coreNFC NDEFs into ST25SDKiOS NDEFs and vice-versa.



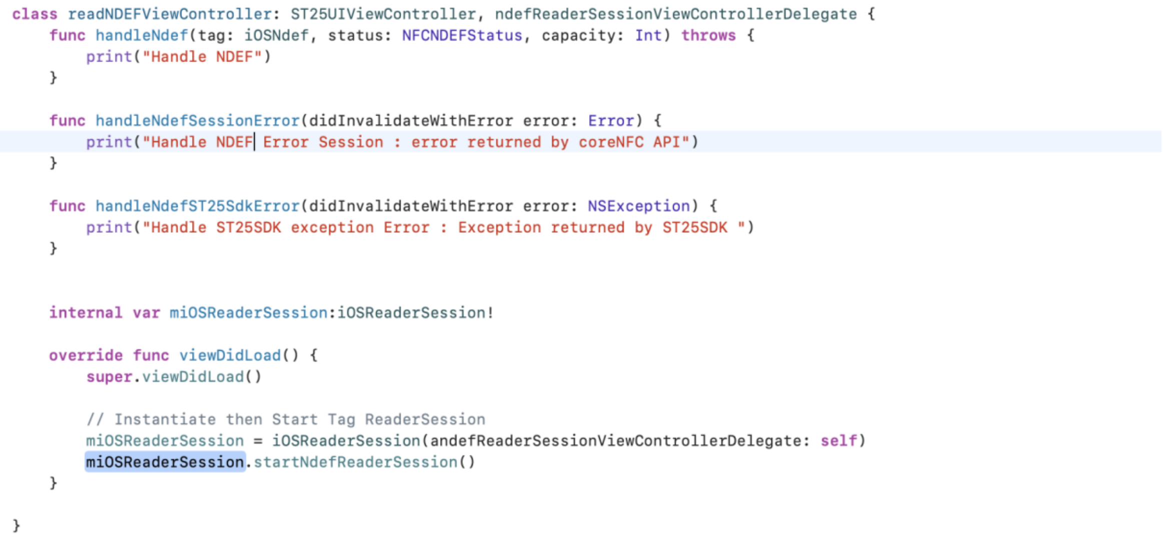
```
func convertiOSNdefToSt25Ndef(message: NFCNDEFMessage) -> ComStSt25sdkNdefNDEFMsg {
    let tmpComStSt25sdkNdefNDEFMsg:ComStSt25sdkNdefNDEFMsg = ComStSt25sdkNdefNDEFMsg()
    for record in message.records {
       self.createRecordsFromNFCNDEFPayload(payload: record)
       let mNDEFRecord = self.getComStSt25sdkNdefNDEFRecord()
       tmpComStSt25sdkNdefNDEFMsg.addRecord(with: mNDEFRecord)
    }
    return tmpComStSt25sdkNdefNDEFMsg
 }

func convertSt25NdefToiOSNdef(message: ComStSt25sdkNdefNDEFMsg ) -> NFCNDEFMessage {
   var recordiOS:[NFCNDEFPayload] = []
```

life.augmented

Code Example3: Write NDEF URI 2/3

- Instantiate then Start an NDEF reader session (ref to *iOSReaderSession.swift*) :
  - Create an *iOSReaderSession* object.
  - The *iOSReaderSession* requires a delegate object that conforms to the *ndefReaderSessionViewControllerDelegate* protocol.
  - Start *iOSReaderSession*.

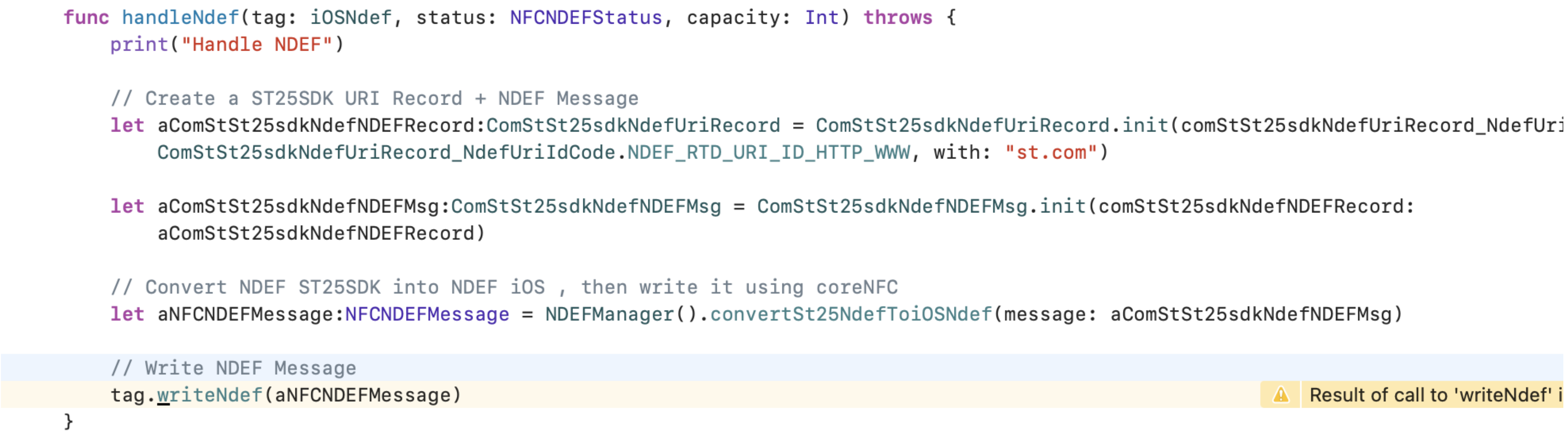
  - Adopting this protocol allows the delegate to receive notifications from the reader session when it:
    - Detects a NDEF message.
    - Encountering an error.



```
class readNDEFViewController: ST25UIViewController, ndefReaderSessionViewControllerDelegate {
    func handleNdef(tag: iOSNdef, status: NFCNDEFStatus, capacity: Int) throws {
        print("Handle NDEF")
    }

    func handleNdefSessionError(didInvalidateWithError error: Error) {
        print("Handle NDEF Error Session : error returned by coreNFC API")
    }

    func handleNdefST25SdkError(didInvalidateWithError error: NSException) {
        print("Handle ST25SDK exception Error : Exception returned by ST25SDK ")
    }


    internal var miOSReaderSession:iOSReaderSession!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Instantiate then Start Tag ReaderSession
        miOSReaderSession = iOSReaderSession(andefReaderSessionViewControllerDelegate: self)
        miOSReaderSession.startNdefReaderSession()
    }

}
```

Code Example3: Write NDEF URI 3/3

- Whenever iOSReaderSession detects at least an emtpy NDEF message, it calls the delegate method ‘*HandleNdef*’ with an instance of *iOSNdef (ref to iOSNdef.swift)*.
- Create a ST25SDK Uri Record, then add it into a ST25SDK Ndef Message.
- Call ‘NDEFManager’ to convert the ST25SDK NDEF into CoreNFC NDEF.
- Then write coreNFC NDEF message.



```
func handleNdef(tag: iOSNdef, status: NFCNDEFStatus, capacity: Int) throws {
    print("Handle NDEF")

    // Create a ST25SDK URI Record + NDEF Message
    let aComStSt25sdkNdefNDEFRecord:ComStSt25sdkNdefUriRecord = ComStSt25sdkNdefUriRecord.init(comStSt25sdkNdefUriRecord_NdefUri
        ComStSt25sdkNdefUriRecord_NdefUriIdCode.NDEF_RTD_URI_ID_HTTP_WWW, with: "st.com")

    let aComStSt25sdkNdefNDEFMsg:ComStSt25sdkNdefNDEFMsg = ComStSt25sdkNdefNDEFMsg.init(comStSt25sdkNdefNDEFRecord:
        aComStSt25sdkNdefNDEFRecord)

    // Convert NDEF ST25SDK into NDEF iOS , then write it using coreNFC
    let aNFCNDEFMessage:NFCNDEFMessage = NDEFManager().convertSt25NdefToiOSNdef(message: aComStSt25sdkNdefNDEFMsg)

    // Write NDEF Message
    tag.writeNdef(aNFCNDEFMessage)
}
```

Conclusion

- ST25SDKiOS offers the same level of features as ST25SDK Java™ aimed at accelerating the development process of iOS applications based on ST RF tags.
- Same logic shared between iOS and Android = Same behavior and less divergent
- Shared code = Faster implementation, code once use twice ;)
- iOS NFC Tap comes with ‘wrappers’ and utilities to simplify usage of ST25SDKiOS within coreNFC Api.
  - *iOSIso15693.swift* & *iOSNdef.swift* : coreNFC APIs synchronous functions.
  - *iOSReaderSession.swift* : wrapper for coreNFC Tag/Ndef reader session + Handles ST25SDK Exception errors.
  - *iOSRFReaderInterface.swift* : allows the ST25SDKiOS to abstract the interactions with the iPhone NFC reader.
  - *NDEFManager.swift* : NDEF helper file to convert coreNFC NDEFs into/from ST25SDK NDEFs

life.augmented

References

- J2OBJC :
  - https://developers.google.com/j2objc
- CoreNFC Api :
  - https://developer.apple.com/documentation/corenfc
- ST25SDKiOS + iOS NFC Tap source code :
  - https://www.st.com/content/st_com/en/products/embedded-software/st25-nfc-rfid-software/stsw-st25ios001.html