TO THE CORE OF

CORE NFC

@TOLKIANA

👋

HELLO SWIFTFEST!

I'M NELIDA

@TOLKIANA

DETROIT LABS

WHY **CORE NFC?**

AGENDA

- NFC CHARACTERISTICS
- NFC MODES
- NFC TAGS
- NFC DATA EXCHANGE FORMAT - NDEF
- CORE NFC TAG READING
- CORE NFC TAG WRITING

WHAT MAKES **NFC** DIFFERENT?

NEAR FIELD COMMUNICATION

- REQUIRES LITTLE POWER CONSUMPTION

NEAR FIELD COMMUNICATION

- REQUIRES LITTLE POWER CONSUMPTION
- **SHORT-RANGE WIRELESS COMMUNICATION**

NEAR FIELD COMMUNICATION

- REQUIRES LITTLE POWER CONSUMPTION
- SHORT-RANGE WIRELESS COMMUNICATION
- **IT CAN BE SETUP FOR ONE WAY OR TWO WAY COMMUNICATION**

TYPES OF **NFC DEVICES**

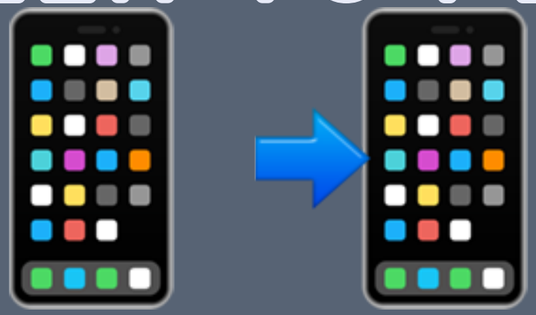
PASSIVE  <---> ACTIVE 

NFC MODES

1. TAG READER/WRITER

@TOLKIANA

2. PEER TO PEER



3. CARD EMULATION

PHOTO BY KARIN HISELIUS ON UNSPLASH

PHOTO BY JAMIE STREET ON UNSPLASH

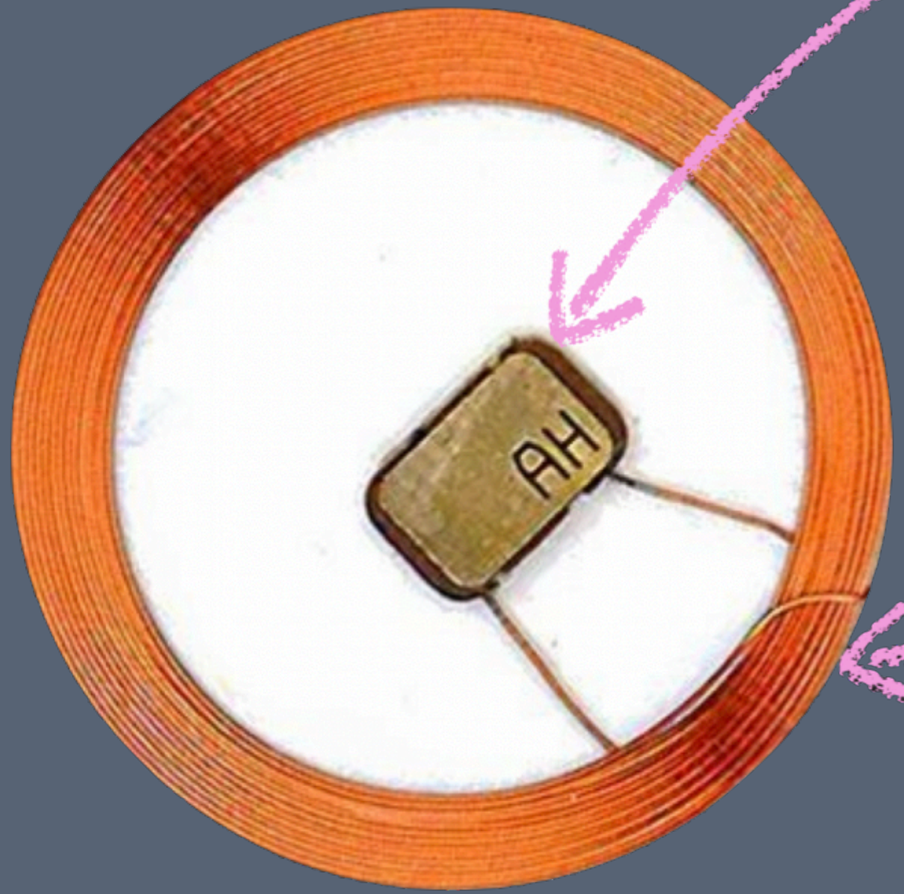
WHAT DOES THIS MEAN FOR THE **LATEST RELEASE?**

MODE	2017	2018	2019
TAG READER/ WRITER	ONLY NDEF TAGS READING	BACKGROUND READING	NATIVE TAG READING & WRITTING
PEER TO PEER	🚫	🚫	🚫
CARD EMULATION	🚫	🚫	🚫

NFC TAGS STRUCTURE

CHIP

ANTENNA



@TOLKIANA

NFC CHIP

NFC CHIP

- UNIQUE IDENTIFIER

NFC CHIP

- UNIQUE IDENTIFIER
- READ/WRITE USER MEMORY

NFC CHIP

- UNIQUE IDENTIFIER
- READ/WRITE USER MEMORY

- PASSWORD PROTECTION

NFC TAG TYPES

IT'S VERY EASY!

- TYPE 1
- TYPE 2
- TYPE 3
- TYPE 4
- TYPE 5

HMM NOT REALLY

- TYPE 1?
- TYPE 2?
- TYPE 3?
- TYPE 4?
- TYPE 5?

PHOTO BY TADEUSZ LAKOTA ON UNSPLASH

UNDERSTANDING **NFC TAG TYPES**

TYPE	FUNCTIONALITY	OTHER CHARACTERISTICS	CORE NFC PROTOCOLS	UNDERLYING TECHNOLOGY
1	**READ ONLY APPLICATIONS, STORE SMALL AMOUNTS OF DATA SUCH A WEBSITE URL**	SLOW AND CHEAP	**NFCNDEFTAG**	ISO14443A
2	**EVENT TICKETS, BADGING SYSTEMS**	THE MOST POPULAR, FASTER THAN TYPE 1	**NFCNDEFTAG, NFCMIFARETAG**	ISO14443A
3	**TRANSIT TICKETS, PAYMENTS, ELECTRONIC IDS, HEALTH CARE DEVICES, HOME ELECTRONICS**	USED PRIMARILY IN JAPAN, IT'S MORE EXPENSIVE	**NFCNDEFTAG, NFCFELICATAG**	ISO18092
4	**PASSPORTS, PAYMENTS**	MOST IMPORTANT FEATURE IT'S SECURITY, MODERATE TO HIGH PRICE TAG	**NFCNDEFTAG, NFCISO7816TAG, NFCMIFARETAG**	ISO14443A, ISO14443B
5	**RETAIL, INDUSTRIAL APPLICATIONS, HEALTH CARE**	ALSO CALLED VICINITY TAGS, BACK-END DATABASE SUPPORT	**NFCNDEFTAG, NFCISO15693TAG**	ISO15693

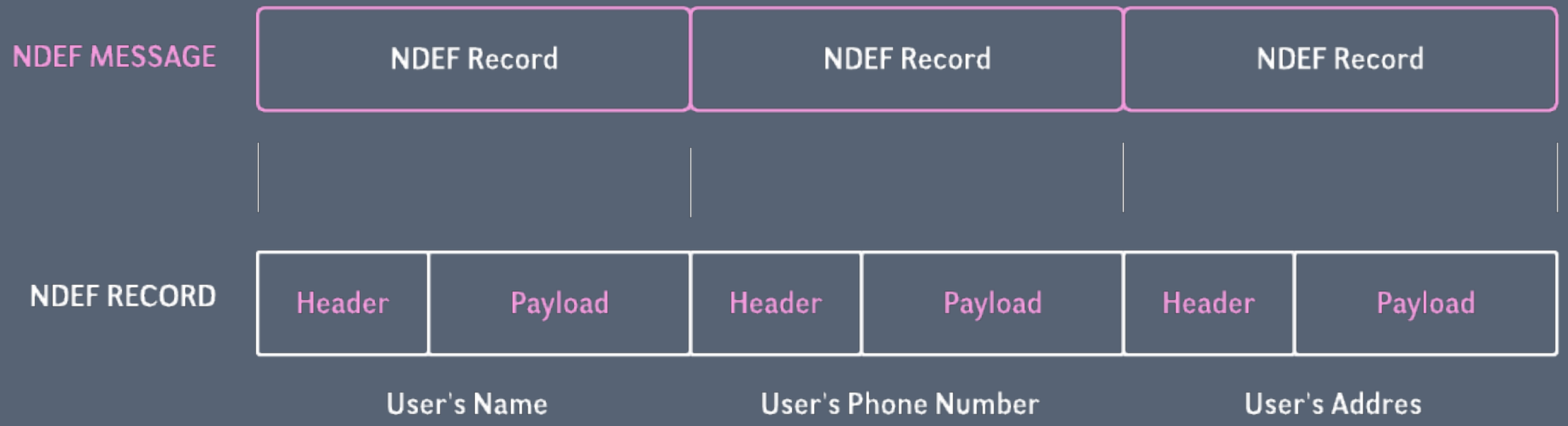
@TOLKIANA

WHAT IS **NDEF?**

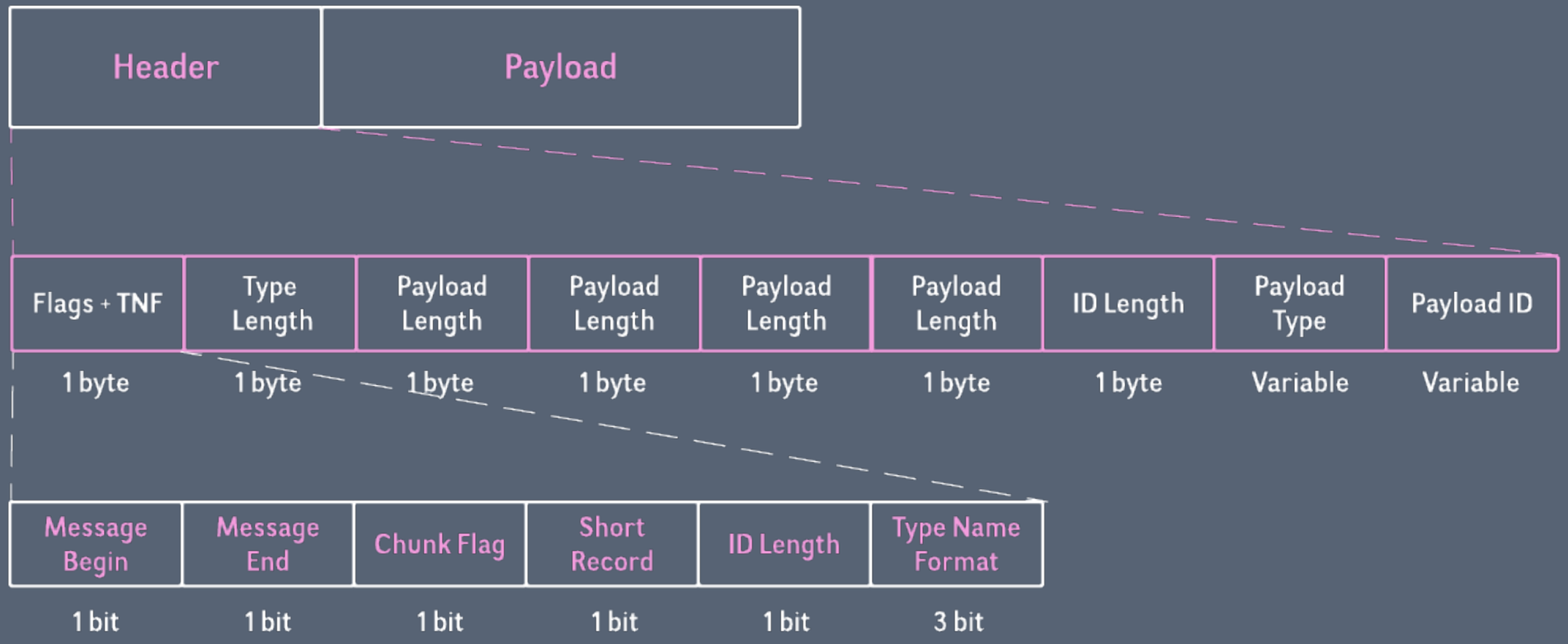
NFC DATA EXCHANGE FORMAT

@TOLKIANA

NFC DATA EXCHANGE FORMAT



NDEF RECORD

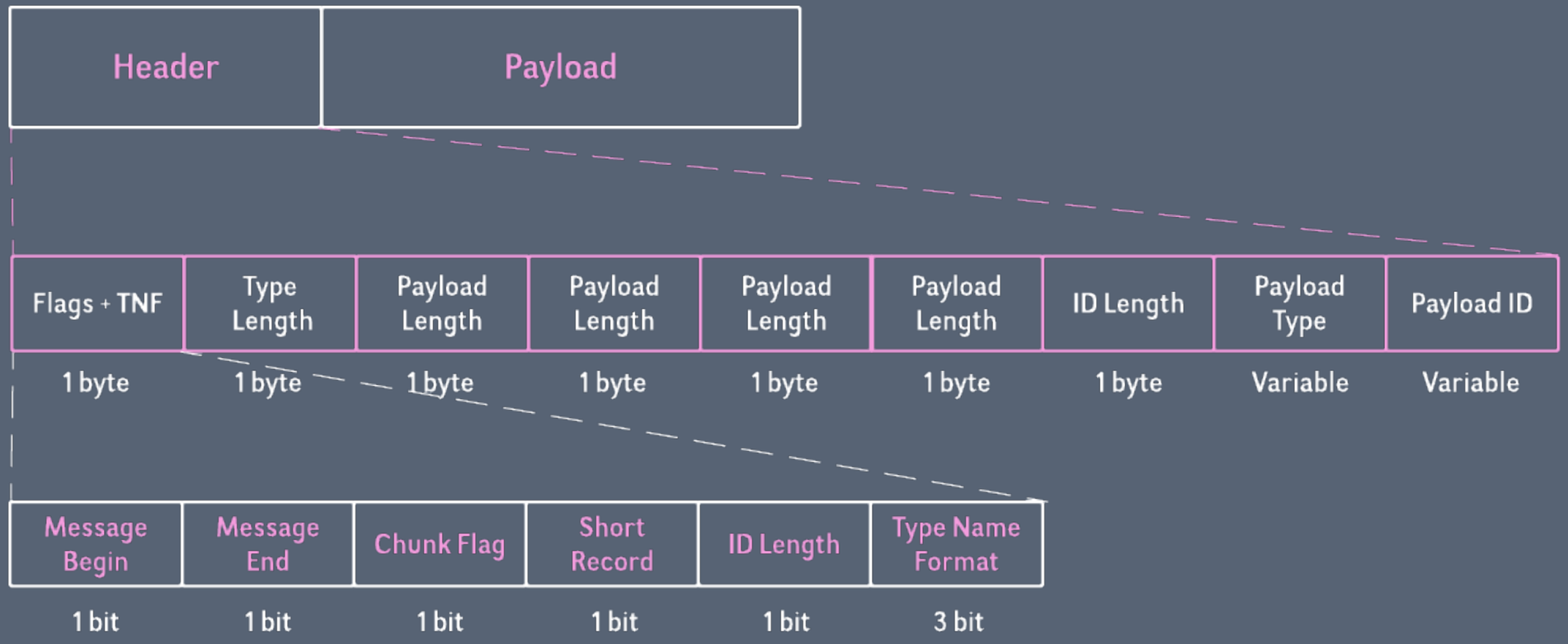


@TOLKIANA

TYPE NAME FORMAT

0 EMPTY
1 WELL-KNOWN
2 MIME MEDIA-TYPE
3 ABSOLUTE URI
4 EXTERNAL
5 UNKNOWN
6 UNCHANGED
7 RESERVED

NDEF RECORD



@TOLKIANA

CORE NFC

TAG WRITING

@TOLKIANA

- URI: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh"
- SONG NAME: "Cheek to Cheek"
- ARTIST: "Ella Fitzgerald"

IDENTIFY YOUR TAG

TYPE	FUNCTIONALITY	OTHER CHARACTERISTICS	CORE NFC PROTOCOLS	UNDERLYING TECHNOLOGY
1	READ ONLY APPLICATIONS, STORE SMALL AMOUNTS OF DATA SUCH A WEBSITE URL	SLOW AND CHEAP	NFCNDEFTAG&	ISO14443A
2	**EVENT TICKETS, BADGING SYSTEMS**	**THE MOST POPULAR, FASTER THAN TYPE 1**	**NFCNDEFTAG, NFCMIFARETAG**	**ISO14443A**
3	TRANSIT TICKETS, PAYMENTS, ELECTRONIC IDS, HEALTH CARE DEVICES, HOME ELECTRONICS	USED PRIMARILY IN JAPAN, IT'S MORE EXPENSIVE	NFCNDEFTAG, NFCFELICATAG	ISO18092
4	PASSPORTS, PAYMENTS	MOST IMPORTANT FEATURE IT'S SECURITY, MODERATE TO HIGH PRICE TAG	NFCNDEFTAG, NFCISO7816TAG, NFCMIFARETAG	ISO14443A, ISO14443B
5	RETAIL, INDUSTRIAL APPLICATIONS, HEALTH CARE	ALSO CALLED VICINITY TAGS, BACK-END DATABASE SUPPORT	NFCNDEFTAG, NFCISO15693TAG	ISO15693

@TOLKIANA

WHAT KIND OF SESSION DO YOU NEED?

```
NFCNDEFReaderSession // To use tag with NDEF Format
NFCTagReaderSession  // To use tag native features
```

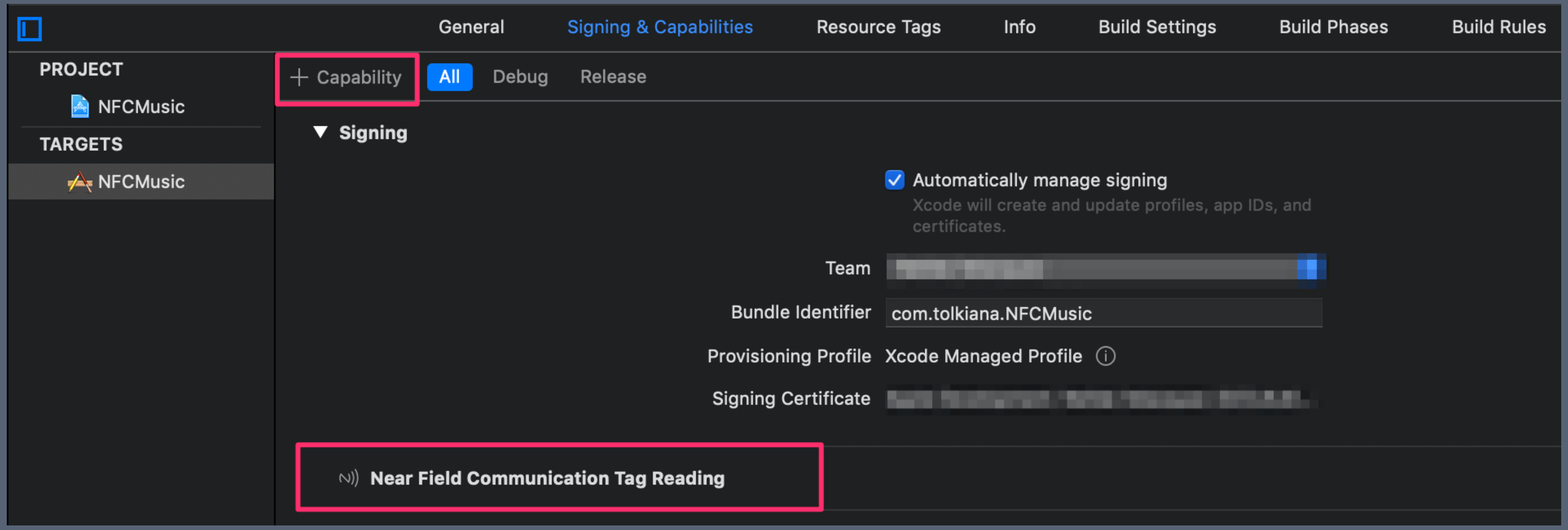
WHAT KIND OF SESSION DO YOU NEED?

```
NFCNDEFReaderSession -> NFCNDEFTag // To use tag with NDEF Format
NFCTagReaderSession -> NFCMiFareTag  // To use tag native features
```

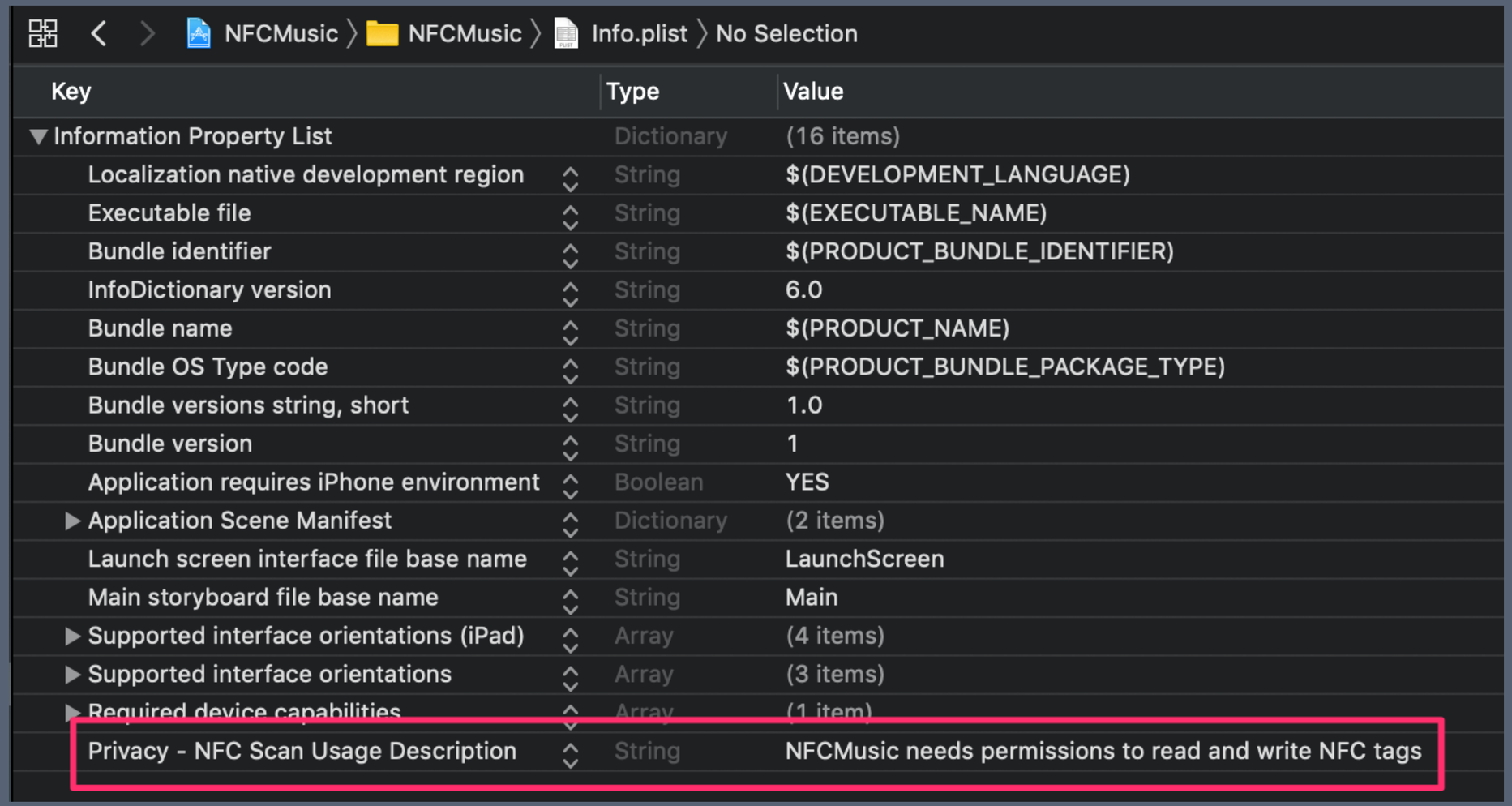
DON'T FORGET YOUR ENTITLEMENTS

Key	Type	Value
▼ Entitlements File	Dictionary	(1 item)
▼ Near Field Communication Tag Re...	Array	(2 items)
Item 0 (Near Field Communication...	String	NFC Data Exchange Format
Item 1 (Near Field Communication...	String	NFC tag-specific data protocol

ADD THE NFC CAPABILITY



AND THE USAGE DESCRIPTION

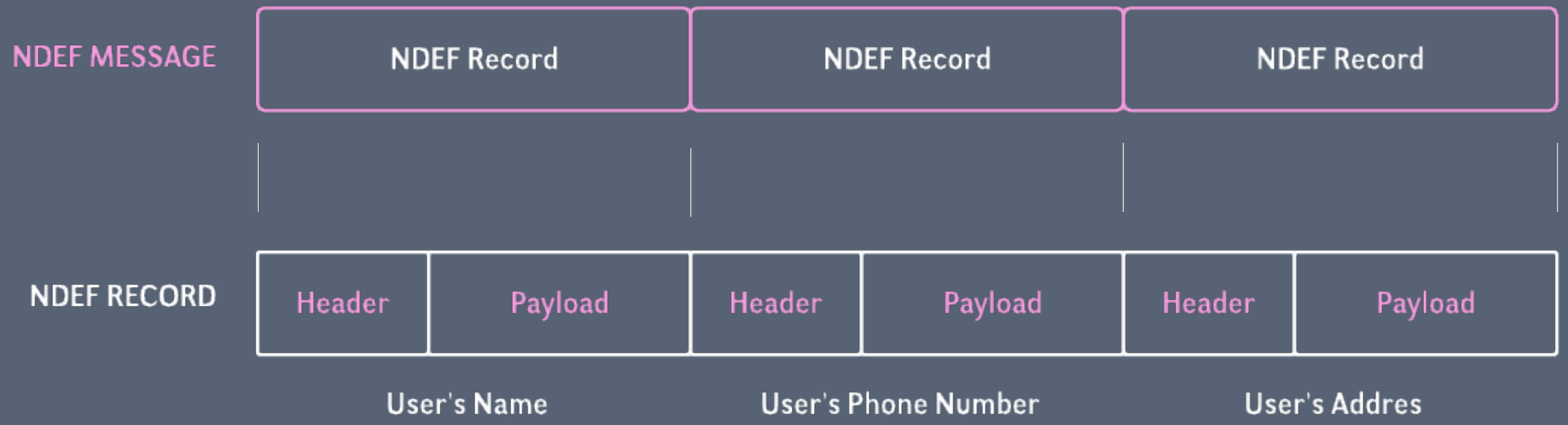


NFCNDEFMessage

```
init(records: [NFCNDEFPayload])
```

NFCNDEFPayload

```
init(format: NFCTypeNameFormat,
     type: Data,
     identifier: Data,
     payload: Data)
```


NDEF MESSAGE
NDEF Record
NDEF Record
NDEF Record
NDEF RECORD
Header
Payload
Header
Payload
Header
Payload
User's Name
User's Phone Number
User's Addres

NFCTypeNameFormat

```
typedef NS_ENUM(uint8_t, NFCTypeNameFormat) {
    NFCTypeNameFormatEmpty                  = 0x00,
    NFCTypeNameFormatNFCWellKnown           = 0x01,
    NFCTypeNameFormatMedia                  = 0x02,
    NFCTypeNameFormatAbsoluteURI            = 0x03,
    NFCTypeNameFormatNFCExternal            = 0x04,
    NFCTypeNameFormatUnknown                = 0x05,
    NFCTypeNameFormatUnchanged              = 0x06
};
```

NFCNDEFPayload

```
class func wellKnownTypeURIPayload(url: URL) -> Self?
class func wellKnownTypeTextPayload(string text: String,
                                    locale: Locale) -> Self?
```

NFCNDEFPayload

```
class func wellKnownTypeURIPayload(url: URL) -> Self?
class func wellKnownTypeTextPayload(string text: String,
                                    locale: Locale) -> Self?

init(format: NFCTypeNameFormat,
     type: Data,
     identifier: Data,
     payload: Data)
```

BEGIN THE SESSION 

```
func beginReadingSession() {
    let session = NFCNDEFReaderSession(delegate: self,
                                       queue: DispatchQueue.main,
                                       invalidateAfterFirstRead: true)
    session.begin()
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.write(to: tag)
        }
    }
}
```

WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

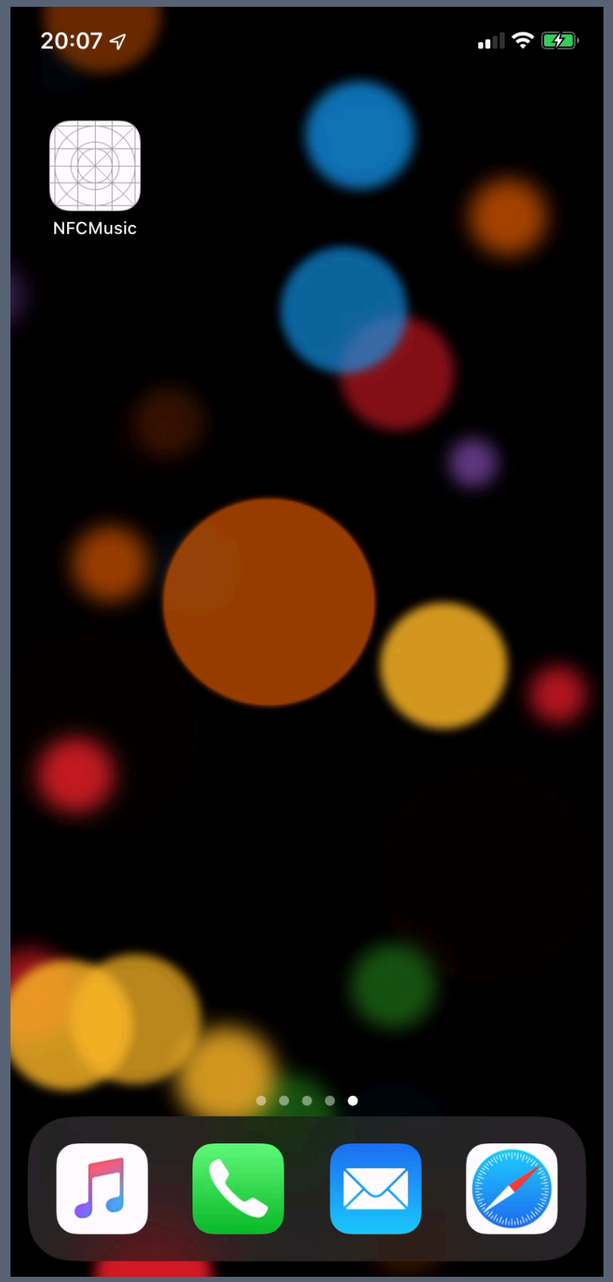
WRITE YOUR TAG! ✍️

```
@available(iOS 13.0, *)
private func write(to tag: NFCNDEFTag) {
    tag.queryNDEFStatus() { (status: NFCNDEFStatus, _, error: Error?) in
        guard status == .readWrite else {
            self.writeSession?.invalidate(errorMessage: "Tag not valid.")
            return
        }

        guard let uriPayload = NFCNDEFPayload.wellKnownTypeURIPayload(string: "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh") else {
            self.writeSession?.invalidate(errorMessage: "Invalid Song URI.")
            return
        }
        let myMessage = NFCNDEFMessage(records: [uriPayload])
        tag.writeNDEF(myMessage) { (error: Error?) in
            if (error != nil) {
                self.writeSession?.invalidate(errorMessage: error?.localizedDescription ?? "There was an error")
            }
            self.writeSession?.alertMessage = "Song successfully saved"
            self.writeSession?.invalidate()
        }
    }
}
```

CORE NFC

TAG READING



@TOLKIANA

BEGIN THE SESSION 

```
func beginReadingSession() {
    let session = NFCNDEFReaderSession(delegate: self,
                                       queue: DispatchQueue.main,
                                       invalidateAfterFirstRead: true)
    session.begin()
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return;
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.read(tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return;
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.read(tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return;
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.read(tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return;
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.read(tag)
        }
    }
}
```

DETECT THE TAG 🔎

```
class NFCMusicWriter: NSObject, NFCNDEFReaderSessionDelegate {

    @available(iOS 13.0, *)
    func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
        guard let tag = tags.first else {
            session.invalidate(errorMessage: "Couldn't read tag")
            return;
        }

        session.connect(to: tag) { (error: Error?) in
            if error != nil {
                session.invalidate(errorMessage: "Connection error. Please try again.")
                return
            }
            self.read(tag)
        }
    }
}
```

READ THE TAG 📖

```
private func read(_ tag: NFCNDEFTag) {
    tag.readNDEF { (message: NFCNDEFMessage?, error: Error?) in
        guard error == nil else {
            self.session?.invalidate(errorMessage: error!.localizedDescription)
            return
        }
        guard let message = message else {
            self.session?.invalidate(errorMessage: "There's no message")
            return
        }
        self.decoder.decode(message)
        self.session?.alertMessage = "Message successfully read"
        self.session?.invalidate()
    }
}
```

READ THE TAG 📖

```
private func read(_ tag: NFCNDEFTag) {
    tag.readNDEF { (message: NFCNDEFMessage?, error: Error?) in
        guard error == nil else {
            self.session?.invalidate(errorMessage: error!.localizedDescription)
            return
        }
        guard let message = message else {
            self.session?.invalidate(errorMessage: "There's no message")
            return
        }
        self.decoder.decode(message)
        self.session?.alertMessage = "Message successfully read"
        self.session?.invalidate()
    }
}
```

READ THE TAG 📖

```
private func read(_ tag: NFCNDEFTag) {
    tag.readNDEF { (message: NFCNDEFMessage?, error: Error?) in
        guard error == nil else {
            self.session?.invalidate(errorMessage: error!.localizedDescription)
            return
        }
        guard let message = message else {
            self.session?.invalidate(errorMessage: "There's no message")
            return
        }
        self.decoder.decode(message)
        self.session?.alertMessage = "Message successfully read"
        self.session?.invalidate()
    }
}
```

READ THE TAG 📖

```
private func read(_ tag: NFCNDEFTag) {
    tag.readNDEF { (message: NFCNDEFMessage?, error: Error?) in
        guard error == nil else {
            self.session?.invalidate(errorMessage: error!.localizedDescription)
            return
        }
        guard let message = message else {
            self.session?.invalidate(errorMessage: "There's no message")
            return
        }
        self.decoder.decode(message)
        self.session?.alertMessage = "Message successfully read"
        self.session?.invalidate()
    }
}
```

READ THE TAG 📖

```
private func read(_ tag: NFCNDEFTag) {
    tag.readNDEF { (message: NFCNDEFMessage?, error: Error?) in
        guard error == nil else {
            self.session?.invalidate(errorMessage: error!.localizedDescription)
            return
        }
        guard let message = message else {
            self.session?.invalidate(errorMessage: "There's no message")
            return
        }
        self.decoder.decode(message)
        self.session?.alertMessage = "Message successfully read"
        self.session?.invalidate()
    }
}
```

READ THE TAG 📖

```
private func read(_ tag: NFCNDEFTag) {
    tag.readNDEF { (message: NFCNDEFMessage?, error: Error?) in
        guard error == nil else {
            self.session?.invalidate(errorMessage: error!.localizedDescription)
            return
        }
        guard let message = message else {
            self.session?.invalidate(errorMessage: "There's no message")
            return
        }
        self.decoder.decode(message)
        self.session?.alertMessage = "Message successfully read"
        self.session?.invalidate()
    }
}
```

READ THE TAG 📖

```
private func read(_ tag: NFCNDEFTag) {
    tag.readNDEF { (message: NFCNDEFMessage?, error: Error?) in
        guard error == nil else {
            self.session?.invalidate(errorMessage: error!.localizedDescription)
            return
        }
        guard let message = message else {
            self.session?.invalidate(errorMessage: "There's no message")
            return
        }
        self.decoder.decode(message)
        self.session?.alertMessage = "Message successfully read"
        self.session?.invalidate()
    }
}
```

READ THE TAG 📖

```
private func read(_ tag: NFCNDEFTag) {
    tag.readNDEF { (message: NFCNDEFMessage?, error: Error?) in
        guard error == nil else {
            self.session?.invalidate(errorMessage: error!.localizedDescription)
            return
        }
        guard let message = message else {
            self.session?.invalidate(errorMessage: "There's no message")
            return
        }
        self.decoder.decode(message)
        self.session?.alertMessage = "Message successfully read"
        self.session?.invalidate()
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

DECODE YOUR MESSAGE! 👀

```
class NFCMessageDecoder {

    func decode(_ message: NFCNDEFMessage) -> [String] {
        return message.records.compactMap { record in
            return self.decode(record)
        }
    }

    private func decode(_ record: NFCNDEFPayload) -> String? {
        let type = PayloadType(rawValue: record.type.decode())
        switch type {
        case .url: // Raw value: "U"
            return record.wellKnownTypeURIPayload()?.absoluteString
        case .text: // Raw value: "T"
            return record.wellKnownTypeTextPayload().0
        default:
            return nil
        }
    }
}
```

READY TO LISTEN 🎧

```
// This is the decoded NFCNDEFMessage

let song = [
  "spotify:track:33jt3kYWjQzqn3xyYQ5ZEh", // this was a payload of type URI
  "Cheek to Cheek", // this was a payload of type Text
  "Ella Fitzgerald" // this was a payload of type Text
]
```

TAKE AWAYS

TOOLS 

TOOLS 🛠️

- YOU NEED AN IPHONE 7 OR NEWER

TOOLS 🛠

- YOU NEED AN IPHONE 7 OR NEWER
- YOU NEED TO INSTALL IOS 13 BETA

TOOLS 🛠

- YOU NEED AN IPHONE 7 OR NEWER
- YOU NEED TO INSTALL IOS 13 BETA
- NFC TAGS

TAGS

TAGS 🏷

- NDEF IS GREAT TO STORE SIMPLE DATA

TAGS 🏷️

- NDEF IS GREAT TO STORE SIMPLE DATA
- SOPHISTICATED FEATURES REQUIRE LEARN SPECIFIC STANDARDS

FUTURE 

FUTURE 🔮

- CORE NFC IS STILL LIMITED

FUTURE 🔮

- CORE NFC IS STILL LIMITED
- BUT I'M STILL EXCITED AND HOPING FOR THE FUTURE

WHAT ELSE DO YOU THINK CAN WE DO WITH **CORE NFC?**

@TOLKIANA

@TOLKIANA

HTTPS://GITHUB.COM/TOLKIANA/NFCMUSIC

PHOTO BY HOWARD RIMINTON ON UNSPLASH