

#WWDC19

Core NFC Enhancements

Gordon Scott, NFC Software
Lawrence Chung, NFC Software

API changes

API changes

NDEF tag writing

API changes

NDEF tag writing

Native tag reading

API changes

NDEF tag writing

Native tag reading

Demo

API Changes

Core NFC Review

Core NFC Review

Session based interface

Core NFC Review

Session based interface

Tag Reading UI

Core NFC Review

Session based interface

Tag Reading UI

Tag writing and native access are available in-app

Core NFC Review

Session based interface

Tag Reading UI

Tag writing and native access are available in-app

60 second maximum scan time

Reader Sessions



NEW

New `NFCTagReaderSession`

- Poll for ISO14443, ISO15693, and/or ISO18092 tags
- Discovery callback provides protocol specific tag objects
- Restart polling

Reader Sessions

NEW

New `NFCTagReaderSession`

- Poll for ISO14443, ISO15693, and/or ISO18092 tags
- Discovery callback provides protocol specific tag objects
- Restart polling

`NFCNDEFReaderSession`

- New discovery callback provides NDEF tag objects
- Read and write operations with NDEF tag objects

Tag Protocols

NEW

Basic attributes for all tag objects

- `NFCitag`

Protocols for different tag types

- `NFCNDEFitag`
- `NFCIS07816itag`
- `NFCMiFareitag`
- `NFCIS015693itag`
- `NFCFelicaitag`

General API usage

Enable the NFC entitlement in Xcode

Xcode Configuration



The screenshot shows the Xcode interface with the following details:

- Toolbar: Play, Stop, Run, and a button with a hammer icon. The target is set to "Generic iOS Device".
- Status Bar: "Clean Finished | Today at 4:31 PM".
- Navigation Bar: "NFCFishTag > FishTagCreator > FishTagCreator.entitlements > No Selection".
- Table with columns: Key, Type, Value.

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

Xcode Configuration



Generic iOS Device | Clean Finished | Today at 4:31 PM

NFCFishTag > FishTagCreator > FishTagCreator.entitlements > No Selection

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

General API Usage

Enable the NFC entitlement in Xcode

Use `NFCReaderSession` or `NFCNDEFReaderSession` to poll for tags

General API Usage

Enable the NFC entitlement in Xcode

Use `NFCReaderSession` or `NFCNDEFReaderSession` to poll for tags

Receive detected tags in the discovery callback

General API Usage

Enable the NFC entitlement in Xcode

Use `NFCReaderSession` or `NFCNDEFReaderSession` to poll for tags

Receive detected tags in the discovery callback

Connect to the desired tag

General API Usage

Enable the NFC entitlement in Xcode

Use `NFCReaderSession` or `NFCNDEFReaderSession` to poll for tags

Receive detected tags in the discovery callback

Connect to the desired tag

Perform operations using the tag object

General API Usage

Enable the NFC entitlement in Xcode

Use `NFCTagReaderSession` or `NFCNDEFReaderSession` to poll for tags

Receive detected tags in the discovery callback

Connect to the desired tag

Perform operations using the tag object

Invalidate the session when finished

NDEF Tag Writing

NFCNDEFReaderSession



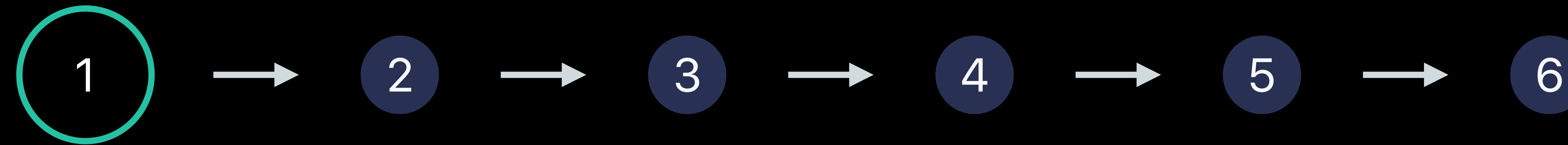
New `NFCNDEFReaderSessionDelegate` method to receive NDEF tag objects

```
optional func readerSession(_ session: NFCNDEFReaderSession, didDetect tags:
[NFCNDEFTag])
```

NDEF tag protocol

```
var isAvailable: Bool { get }
func queryNDEFStatus(completionHandler: @escaping (NFCNDEFStatus, Int, Error?) -> Void)
func readNDEF(completionHandler: @escaping (NFCNDEFMessage?, Error?) -> Void)
func writeNDEF(_ ndefMessage: NFCNDEFMessage, completionHandler: @escaping (Error?) -> Void)
func writeLock(completionHandler: @escaping (Error?) -> Void)
```

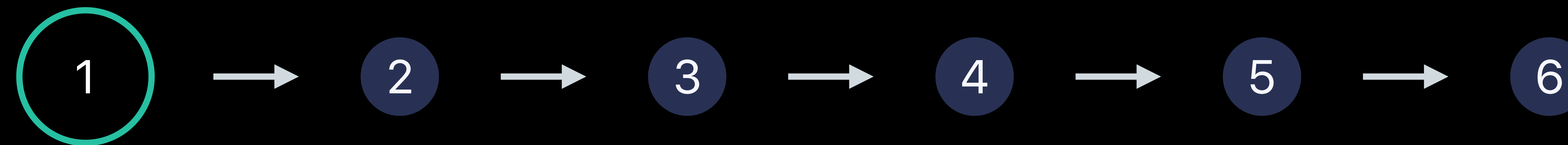

NFCNDEFReaderSession



Create session with a `NFCNDEFReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {
    session = NFCNDEFReaderSession(delegate: self, queue: nil, invalidateAfterFirstRead:
false)
    session?.alertMessage = "Hold your iPhone near the item to learn more about it."
    session?.begin()
}
```

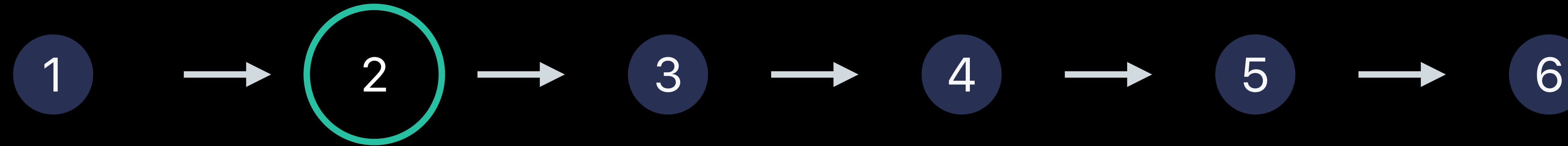
NFCNDEFReaderSession



Create session with a `NFCNDEFReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {  
    session = NFCNDEFReaderSession(delegate: self, queue: nil, invalidateAfterFirstRead:  
false)  
    session?.alertMessage = "Hold your iPhone near the item to learn more about it."  
    session?.begin()  
}
```

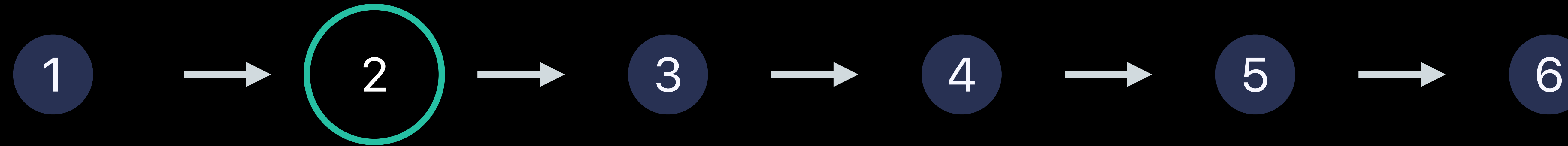
NFCNDEFReaderSession



Implement optional `readerSession(_:didDetect:)` method

```
func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {  
    // Connect to the found tag and perform NDEF message writing  
    let tag = tags.first!  
    session.connect(to: tag) { (error: Error?) in  
        ...  
    }  
}  
}
```

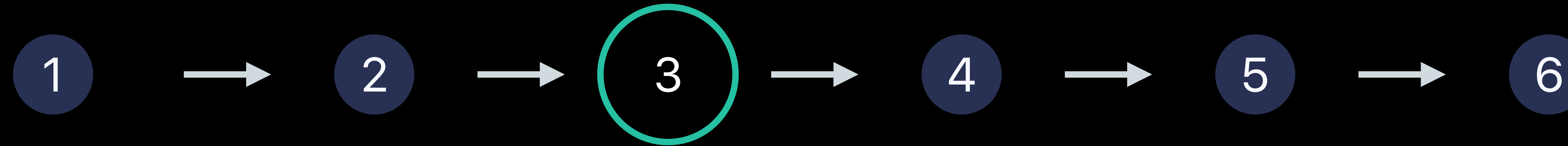
NFCNDEFReaderSession



Implement optional `readerSession(_:didDetect:)` method

```
func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {  
    // Connect to the found tag and perform NDEF message writing  
    let tag = tags.first!  
    session.connect(to: tag) { (error: Error?) in  
        ...  
    }  
}  
}
```

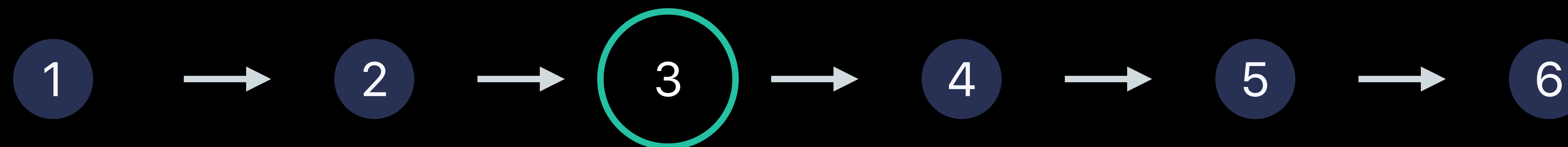
NFCNDEFReaderSession



Connect to the NDEF tag

```
func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {  
    // Connect to the found tag and perform NDEF message writing  
    let tag = tags.first!  
    session.connect(to: tag) { (error: Error?) in  
        ...  
    }  
}  
}
```

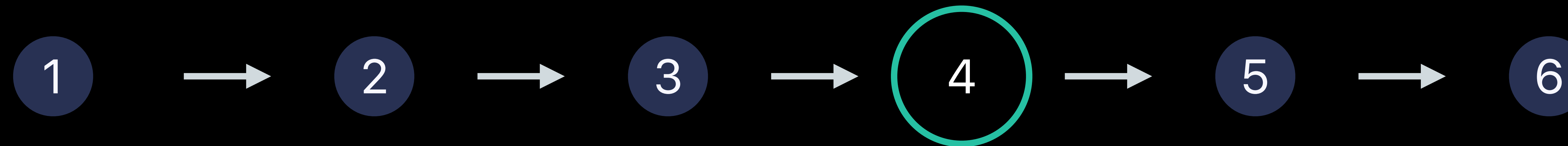

NFCNDEFReaderSession



Connect to the NDEF tag

```
func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {  
    // Connect to the found tag and perform NDEF message writing  
    let tag = tags.first!  
    session.connect(to: tag) { (error: Error?) in  
        ...  
    }  
}  
}
```

NFCNDEFReaderSession

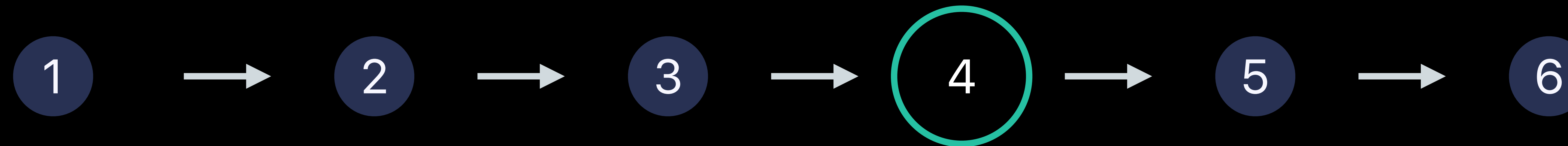


Query NDEF status

...

```
session.connect(to: tag) { (error: Error?) in
  tag.queryNDEFStatus() { (ndefStatus: NFCNDEFStatus, capacity: Int, error: Error?) in
    let myMessage = NFCNDEFMessage(data: Data())
    tag.writeNDEF(myMessage) { (error: Error?) in
      session.invalidate()
    }
  }
}
```

NFCNDEFReaderSession

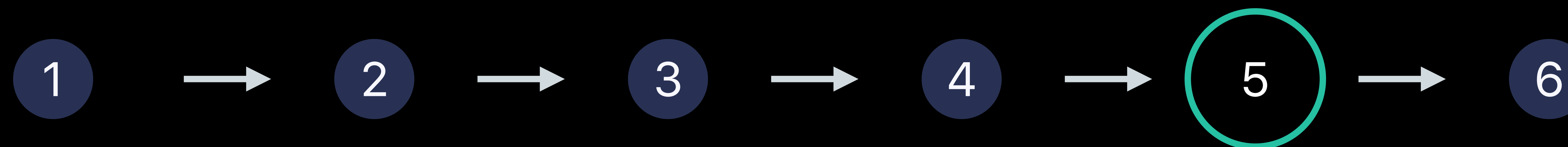


Query NDEF status

...

```
session.connect(to: tag) { (error: Error?) in
  tag.queryNDEFStatus() { (ndefStatus: NFCNDEFStatus, capacity: Int, error: Error?) in
    let myMessage = NFCNDEFMessage(data: Data())
    tag.writeNDEF(myMessage) { (error: Error?) in
      session.invalidate()
    }
  }
}
```

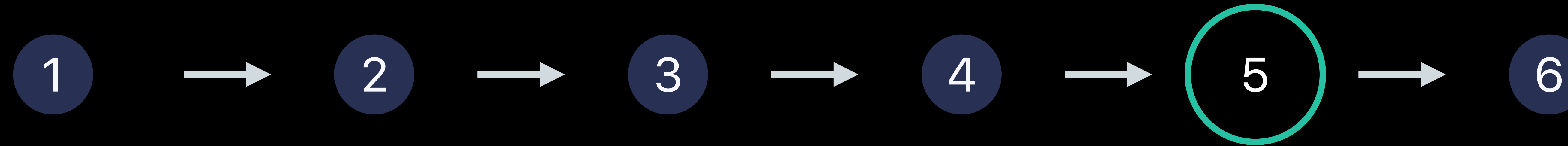

NFCNDEFReaderSession



Write NDEF message

```
...
session.connect(to: tag) { (error: Error?) in
  tag.queryNDEFStatus() { (ndefStatus: NFCNDEFStatus, capacity: Int, error: Error?) in
    let myMessage = NFCNDEFMessage(data: Data())
    tag.writeNDEF(myMessage) { (error: Error?) in
      session.invalidate()
    }
  }
}
```

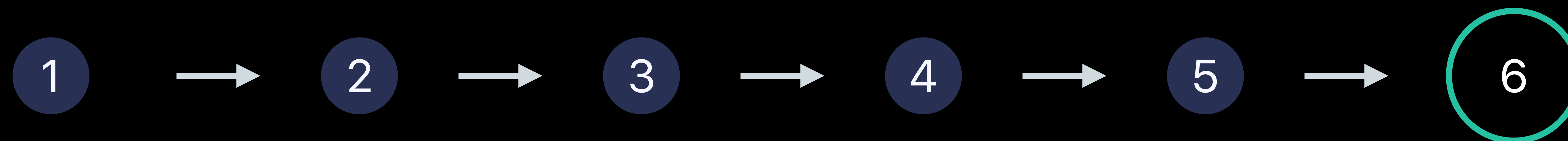
NFCNDEFReaderSession



Write NDEF message

```
...
session.connect(to: tag) { (error: Error?) in
  tag.queryNDEFStatus() { (ndefStatus: NFCNDEFStatus, capacity: Int, error: Error?) in
    let myMessage = NFCNDEFMessage(data: Data())
    tag.writeNDEF(myMessage) { (error: Error?) in
      session.invalidate()
    }
  }
}
```

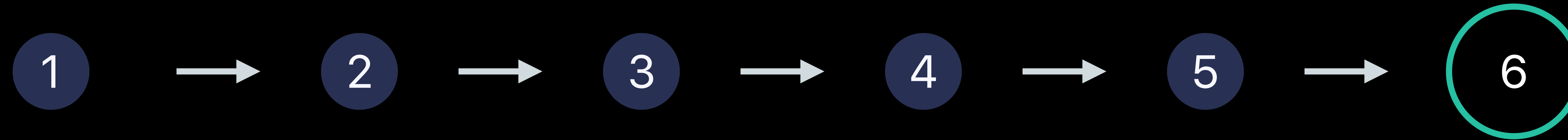
NFCNDEFReaderSession



Invalidate session on completion

```
...
session.connect(to: tag) { (error: Error?) in
  tag.queryNDEFStatus() { (ndefStatus: NFCNDEFStatus, capacity: Int, error: Error?) in
    let myMessage = NFCNDEFMessage(data: Data())
    tag.writeNDEF(myMessage) { (error: Error?) in
      session.invalidate()
    }
  }
}
```

NFCNDEFReaderSession



Invalidate session on completion

...

```
session.connect(to: tag) { (error: Error?) in
  tag.queryNDEFStatus() { (ndefStatus: NFCNDEFStatus, capacity: Int, error: Error?) in
    let myMessage = NFCNDEFMessage(data: Data())
    tag.writeNDEF(myMessage) { (error: Error?) in
      session.invalidate()
    }
  }
}
```

Native Tag Reading

ISO7816

ISO7816 Requirements

ISO7816 Requirements

Application Info.plist must contain list of application identifiers (AIDs)

ISO7816 Requirements

Application Info.plist must contain list of application identifiers (AIDs)

Detected tags callback is invoked if tag is ISO7816 compliant and contains an AID found in the Info.plist

ISO7816 Requirements

Application Info.plist must contain list of application identifiers (AIDs)

Detected tags callback is invoked if tag is ISO7816 compliant and contains an AID found in the Info.plist

Payment related application identifiers are not supported

ISO7816 AID Listing



Generic iOS Device | Clean Finished | Today at 4:31 PM

NFCFishTag > FishTagReader > Info.plist > No Selection

Key	Type	Value
Information Property List	Dictionary	(18 items)
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
ISO7816 application identifiers for NFC Tag Reader Session	Array	(1 item)
Item 0	String	D2760000850101
ISO18092 system codes for NFC Tag Reader Session	Array	(1 item)
Privacy - NFC Scan Usage Description	String	This app uses NFC to scan for fish tags
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	\$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES

ISO7816 AID Listing



Generic iOS Device | Clean Finished | Today at 4:31 PM

NFCFishTag > FishTagReader > Info.plist > No Selection

Key	Type	Value
Information Property List	Dictionary	(18 items)
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
ISO7816 application identifiers for NFC Tag Reader Session	Array	(1 item)
Item 0	String	D2760000850101
ISO18092 system codes for NFC Tag Reader Session	Array	(1 item)
Privacy - NFC Scan Usage Description	String	This app uses NFC to scan for fish tags
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	\$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES

NFCISO7816Tag

Properties

- `identifier` (UID)
- `historicalBytes`
- `applicationData`

Methods

```
func sendCommand(apdu: NFCISO7816APDU, completionHandler: @escaping (Data, UInt8, UInt8, Error?) -> Void)
```


NFCReaderSession

ISO7816



`NFCReaderSessionDelegate` protocol for session delegate object

```
func tagReaderSessionDidBecomeActive(_ session: NFCReaderSession)
func tagReaderSession(_ session: NFCReaderSession, didInvalidateWithError:
Error)
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCTag])
```

NFCReaderSession

ISO7816



Create session with a `NFCReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {  
    session = NFCReaderSession(pollingOption: .iso14443, delegate: self)  
    session?.alertMessage = "Hold your iPhone near the ISO7816 tag to begin  
transaction."  
    session?.begin()  
}
```

NFCReaderSession

ISO7816



Create session with a `NFCReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {  
    session = NFCReaderSession(pollingOption: .iso14443, delegate: self)  
    session?.alertMessage = "Hold your iPhone near the ISO7816 tag to begin  
transaction."  
    session?.begin()  
}
```


NFCReaderSession

ISO7816



Implements `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderSessionTag]) {
    if case let NFCReaderSession.Tag.iso7816(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
        }
    }
}
```

NFCReaderSession

ISO7816



Implements `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderSessionTag]) {  
    if case let NFCReaderSession.Tag.iso7816(tag) = tags.first {  
        session.connect(to: tag) { (error: Error?) in  
            ...  
        }  
    }  
}
```

NFCReaderSession

ISO7816



Connect to the ISO7816 tag

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession]) {  
    if case let NFCReaderSession.iso7816(tag) = tags.first {  
        session.connect(to: tag) { (error: Error?) in  
            ...  
        }  
    }  
}
```

NFCReaderSession

ISO7816



Connect to the ISO7816 tag

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession]) {  
    if case let NFCReaderSession.iso7816(tag) = tags.first {  
        session.connect(to: tag) { (error: Error?) in  
            ...  
        }  
    }  
}
```

NFCTagReaderSession

ISO7816



Send APDU and receive response

```
...
session.connect(to: tag) { (error: Error?) in
  let myAPDU = NFCISO7816APDU(instructionClass:0 instructionCode:0xB0 p1Parameter:0
  p2Parameter:0 data: Data() expectedResponseLength:16)
  tag.sendCommand(apdu: myAPDU) { (response: Data, sw1: UInt8, sw2: UInt8, error: Error?)
  in
    ...
  }
}
...

```


NFCTagReaderSession

ISO7816



Send APDU and receive response

```
...  
session.connect(to: tag) { (error: Error?) in  
  let myAPDU = NFCISO7816APDU(instructionClass:0 instructionCode:0xB0 p1Parameter:0  
    p2Parameter:0 data: Data() expectedResponseLength:16)  
  tag.sendCommand(apdu: myAPDU) { (response: Data, sw1: UInt8, sw2: UInt8, error: Error?)  
    in  
      ...  
    }  
  }  
}  
...
```

NFCReaderSession

ISO7816



Terminate session with error (optional)

```
...
tag.sendCommand(apdu: myAPDU) { (response: Data, sw1: UInt8, sw2: UInt8, error: Error?)
in
  guard error != nil && !(sw1 == 0x90 && sw2 == 0) else {
    session.invalidate(errorMessage: "Application failure")
    return
  }
}
```

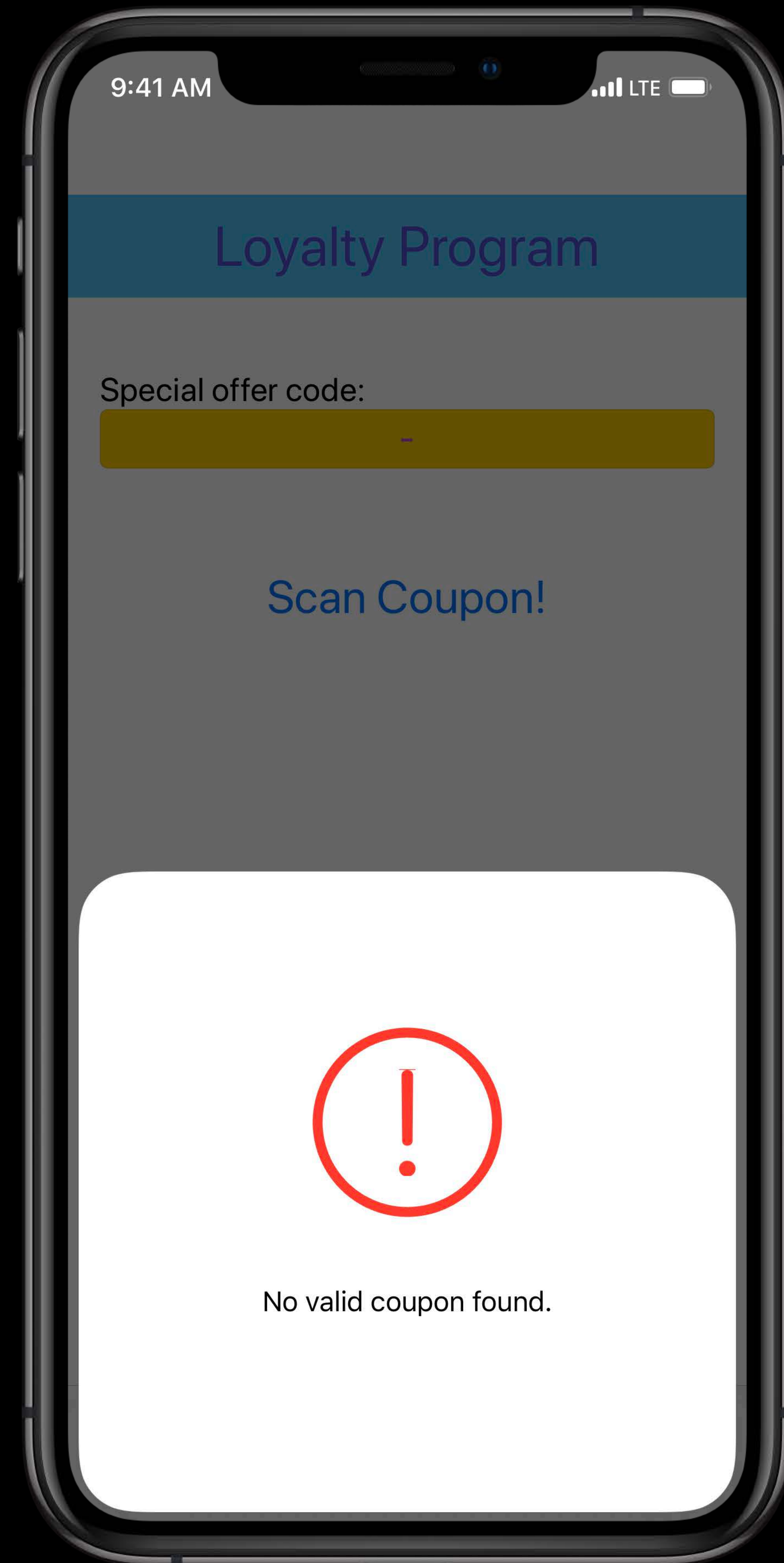

NFCReaderSession

ISO7816



Terminate session with error (optional)

```
...
tag.sendCommand(apdu: myAPDU) { (response: Data, sw1: UInt8, sw2: UInt8, error: Error?)
in
  guard error != nil && !(sw1 == 0x90 && sw2 == 0) else {
    session.invalidate(errorMessage: "Application failure")
  }
  return
}
```



9:41 AM

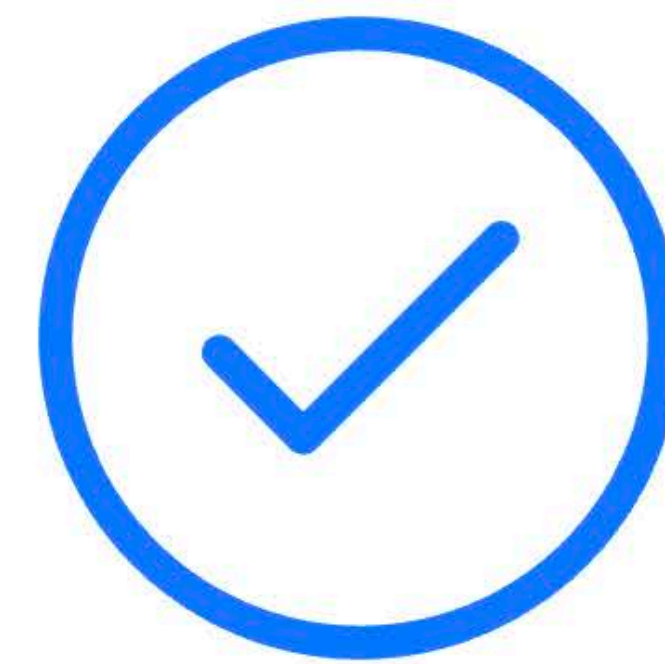
LTE

Loyalty Program

Special offer code:

FISH50OFF

Scan Coupon!



Valid coupon found.

Native Tag Reading

MIFARE[®]

NFCMiFareTag

Properties

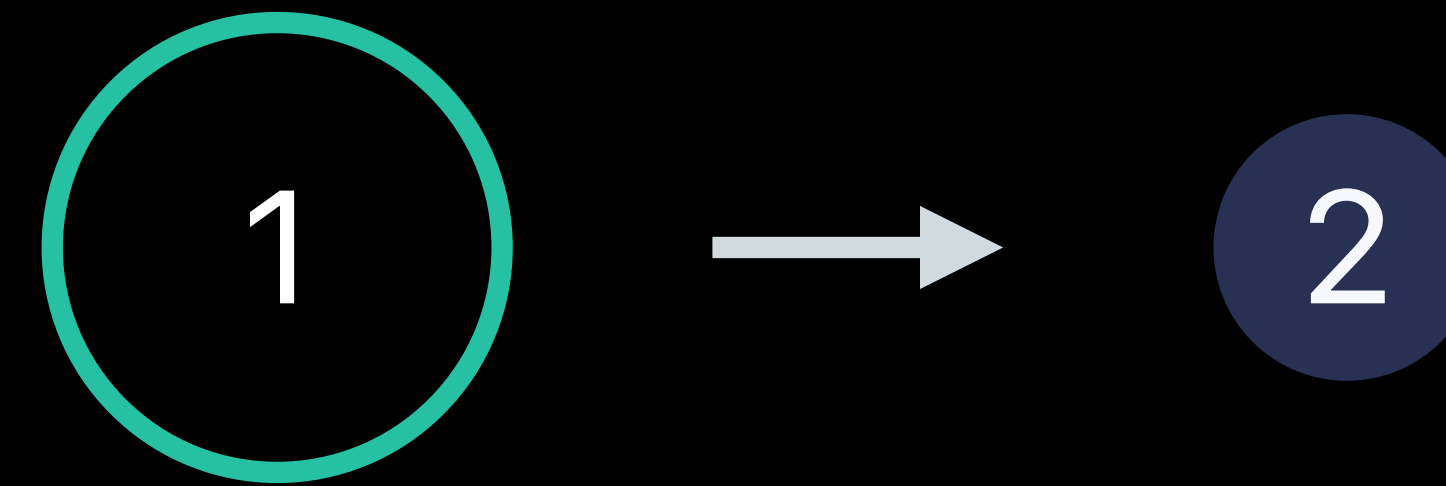
- `identifier` (UID)
- `historicalBytes`
- `mifareFamily` – Ultralight, Plus, DESFire

Methods

```
func sendMiFareCommand(commandPacket command: Data, completionHandler: @escaping (Data, Error?) -> Void)
func sendMiFareISO7816Command(_ apdu: NFCISO7816APDU, completionHandler: @escaping (Data, UInt8, UInt8, Error?) -> Void)
```


NFCReaderSession

MIFARE®

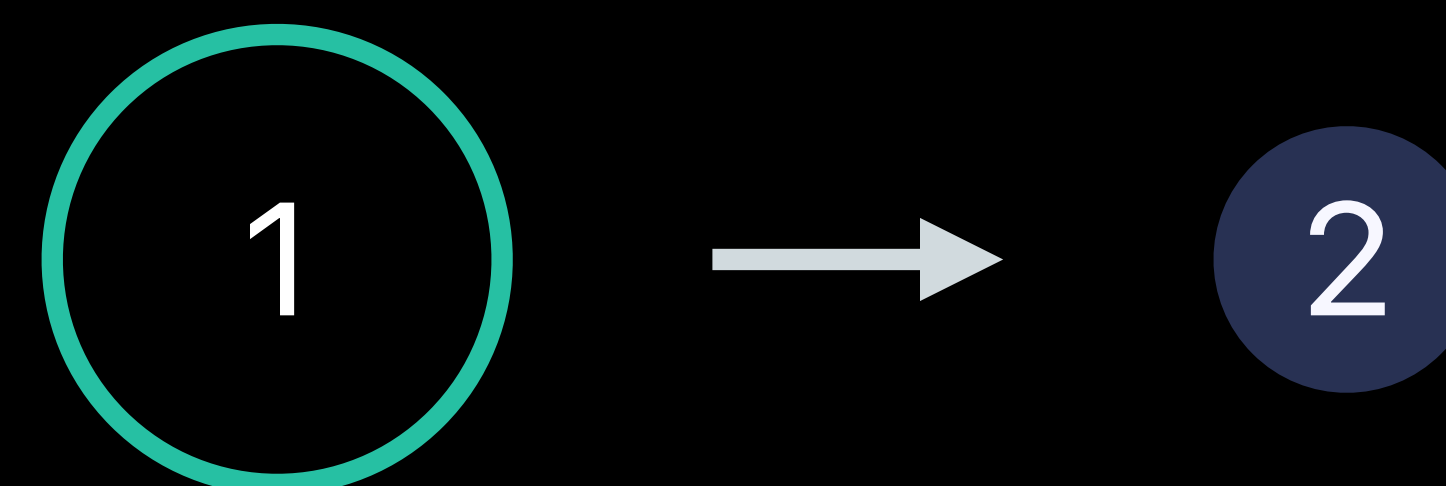


Create session with an `NFCReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {
    session = NFCReaderSession(pollingOption: .iso14443, delegate: self)
    session?.alertMessage = "Hold your iPhone near the MIFARE tag to begin transaction."
    session?.begin()
}
```

NFCReaderSession

MIFARE®

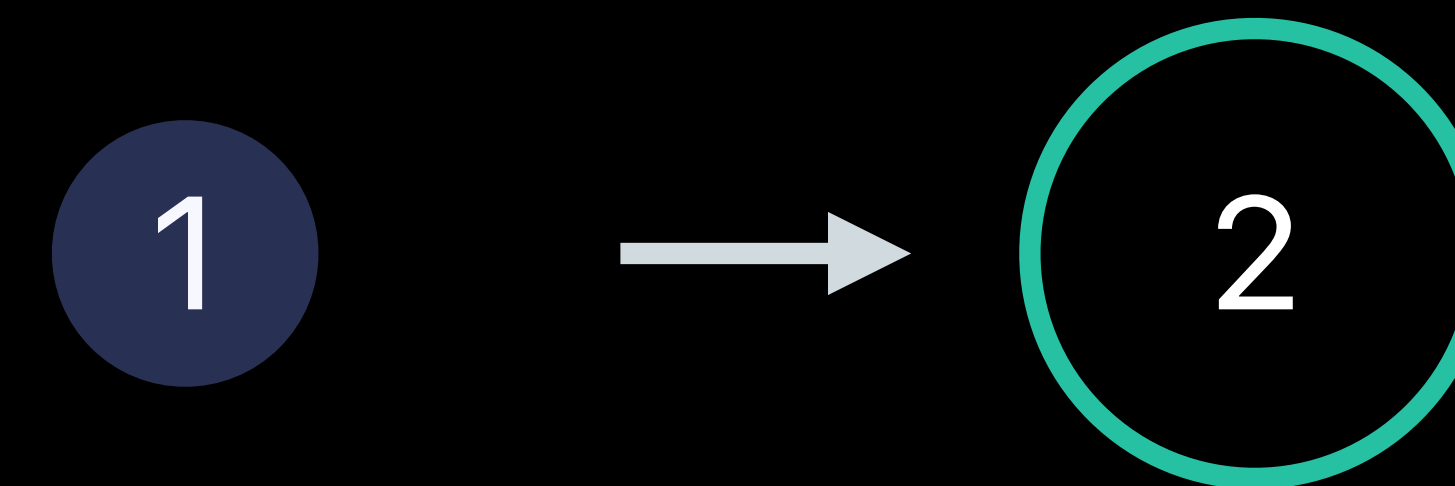


Create session with an `NFCReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {  
    session = NFCReaderSession(pollingOption: .iso14443, delegate: self)  
    session?.alertMessage = "Hold your iPhone near the MIFARE tag to begin transaction."  
    session?.begin()  
}
```


NFCReaderSession

MIFARE®

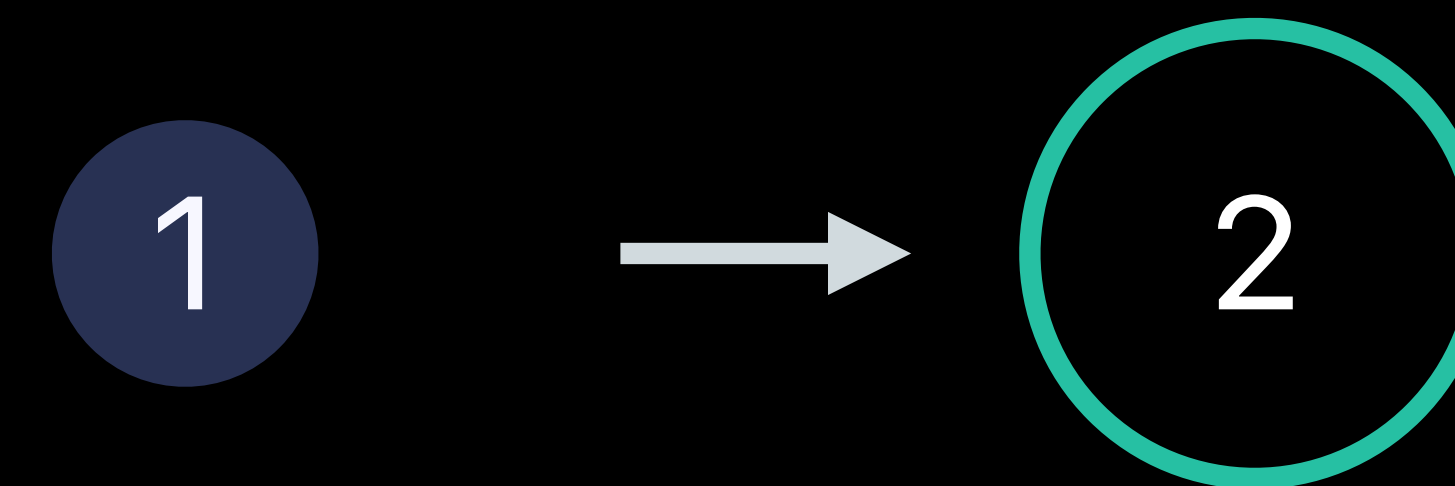


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderTag]) {
    if case let NFCReaderSession.NFCReaderTag.mifare(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.sendMifareCommand(commandPacket: command) { (response: Data, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

MIFARE®

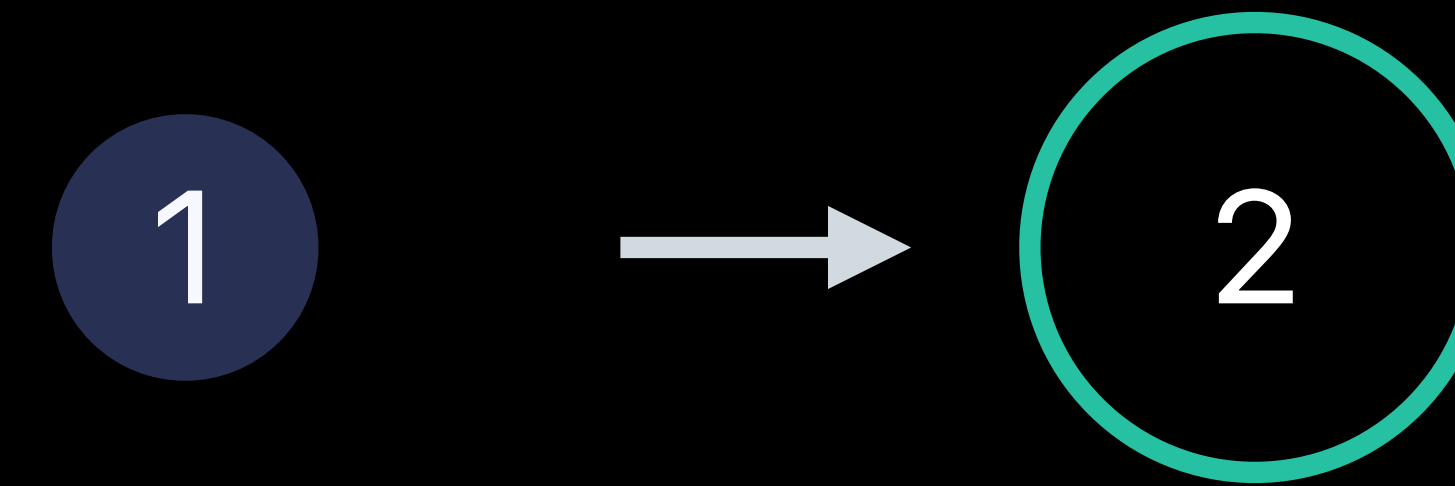


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderTag]) {
    if case let NFCReaderTag.mifare(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.sendMifareCommand(commandPacket: command) { (response: Data, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

MIFARE®

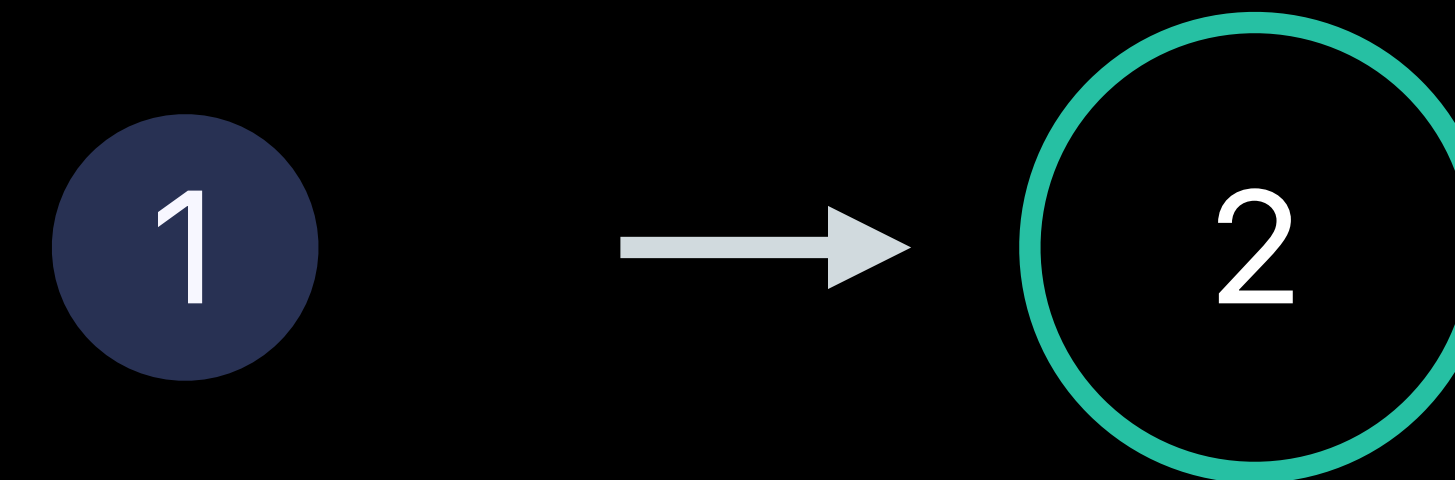


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderTag]) {
    if case let NFCReaderSession.NFCReaderTag.mifare(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.sendMifareCommand(commandPacket: command) { (response: Data, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

MIFARE®

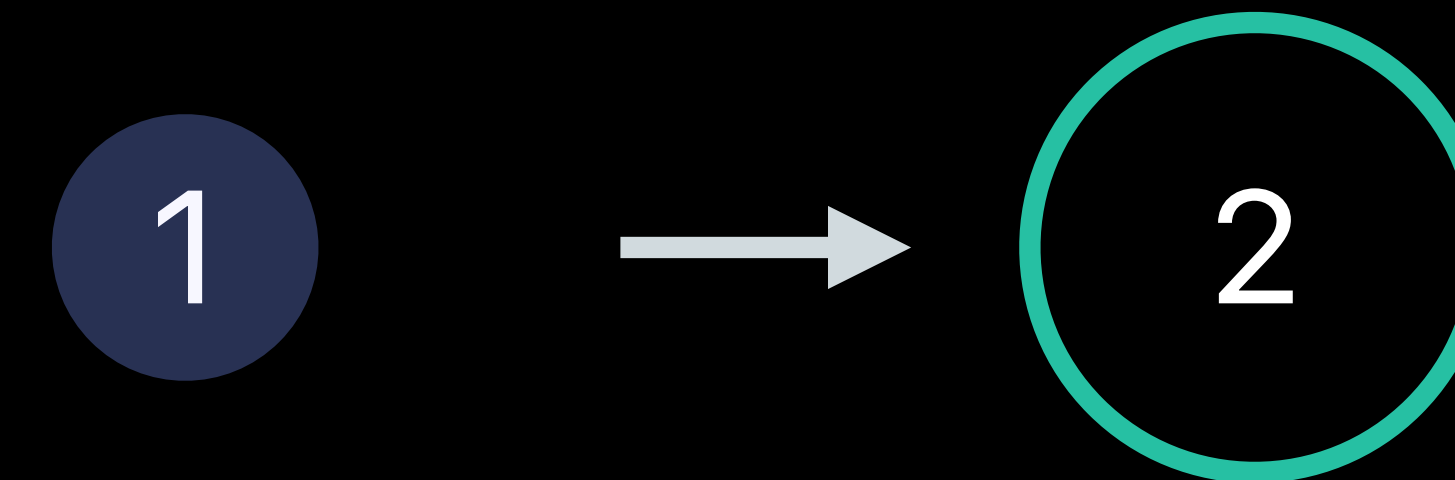


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderTag]) {
    if case let NFCReaderSession.NFCReaderTag.mifare(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.sendMifareCommand(commandPacket: command) { (response: Data, error: Error?) in
                ...
            }
        }
    }
}
```


NFCReaderSession

MIFARE®

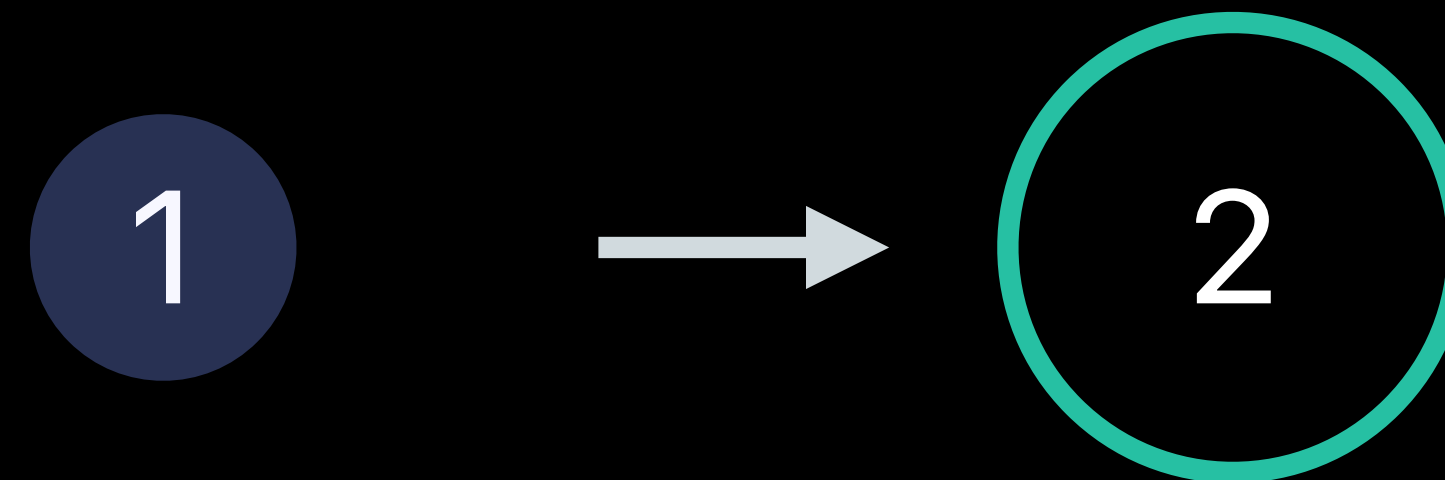


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderTag]) {
    if case let NFCReaderSession.NFCReaderTag.mifare(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.sendMifareCommand(commandPacket: command) { (response: Data, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

MIFARE®



Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderTag]) {
    if case let NFCReaderSession.NFCReaderTag.mifare(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.sendMifareCommand(commandPacket: command) { (response: Data, error: Error?) in
                ...
            }
        }
    }
}
```

Native Tag Reading

ISO15693

NFCISO15693Tag

Properties

- `identifier` (UID)
- `icManufacturerCode`
- `icSerialNumber`

ISO15693 Convenience Methods

Read single block

Read multiple blocks

Extended read

Write single block

Write multiple blocks

Extended write

Lock block

Select

Reset

Send custom command

Get block security

Stay quiet

Write AFI

Lock AFI

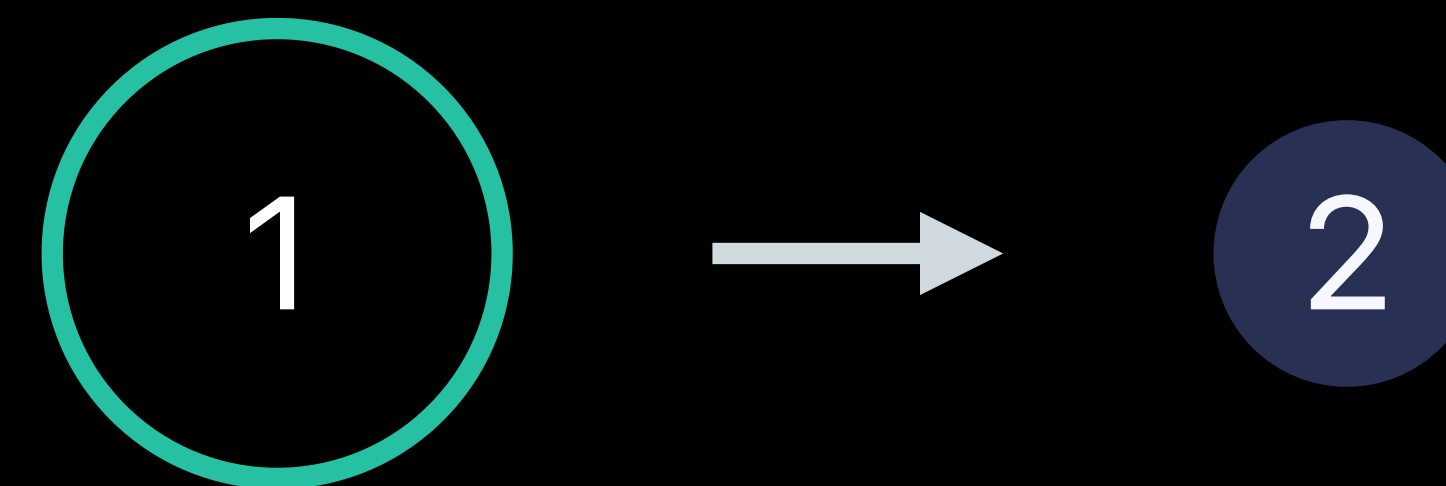
Write DSID

Lock DSID

Get system info

NFCReaderSession

ISO15693

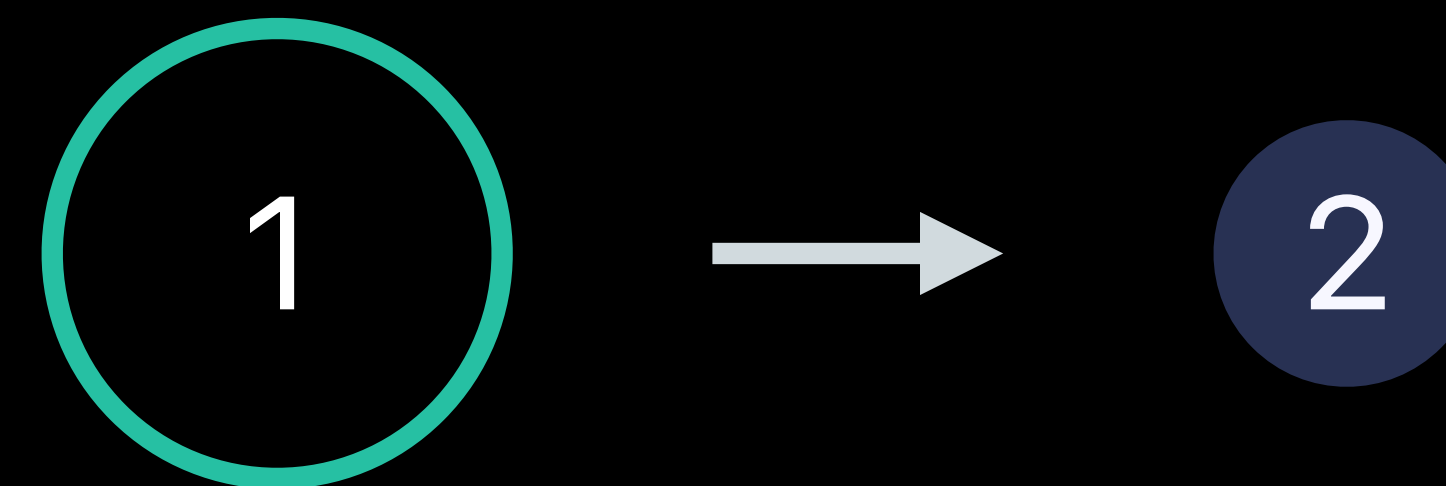


Create session with a `NFCReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {  
    session = NFCReaderSession(pollingOption: .iso15693, delegate: self)  
    session?.alertMessage = "Hold your iPhone near the ISO15693 tag to begin  
transaction."  
    session?.begin()  
}
```

NFCReaderSession

ISO15693

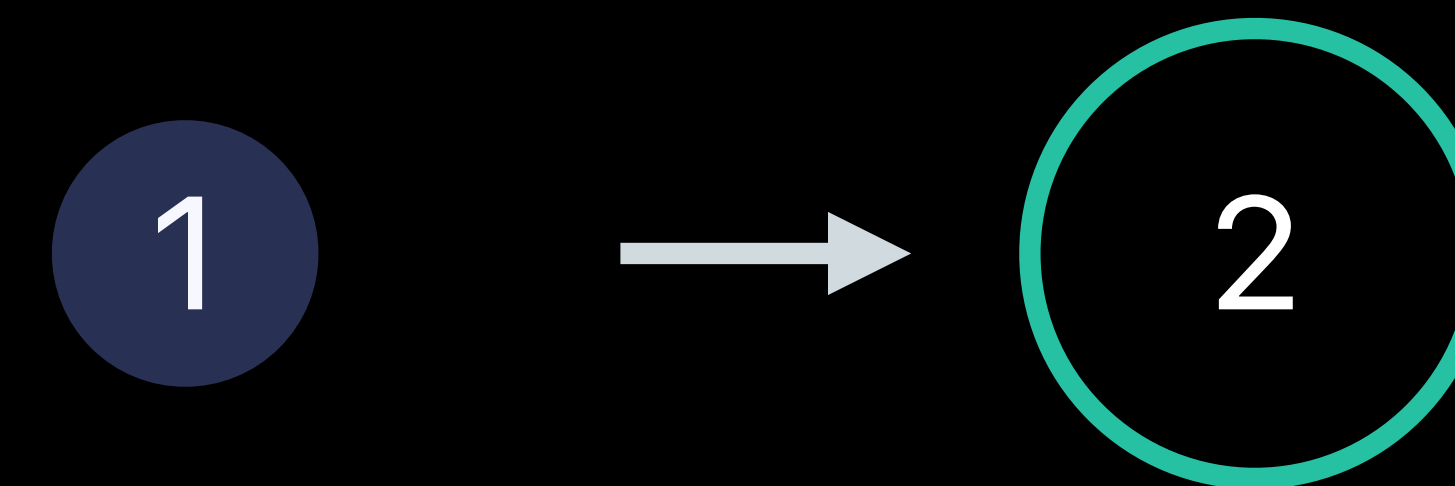


Create session with a `NFCReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {  
    session = NFCReaderSession(pollingOption: .iso15693, delegate: self)  
    session?.alertMessage = "Hold your iPhone near the ISO15693 tag to begin  
transaction."  
    session?.begin()  
}
```

NFCReaderSession

ISO15693

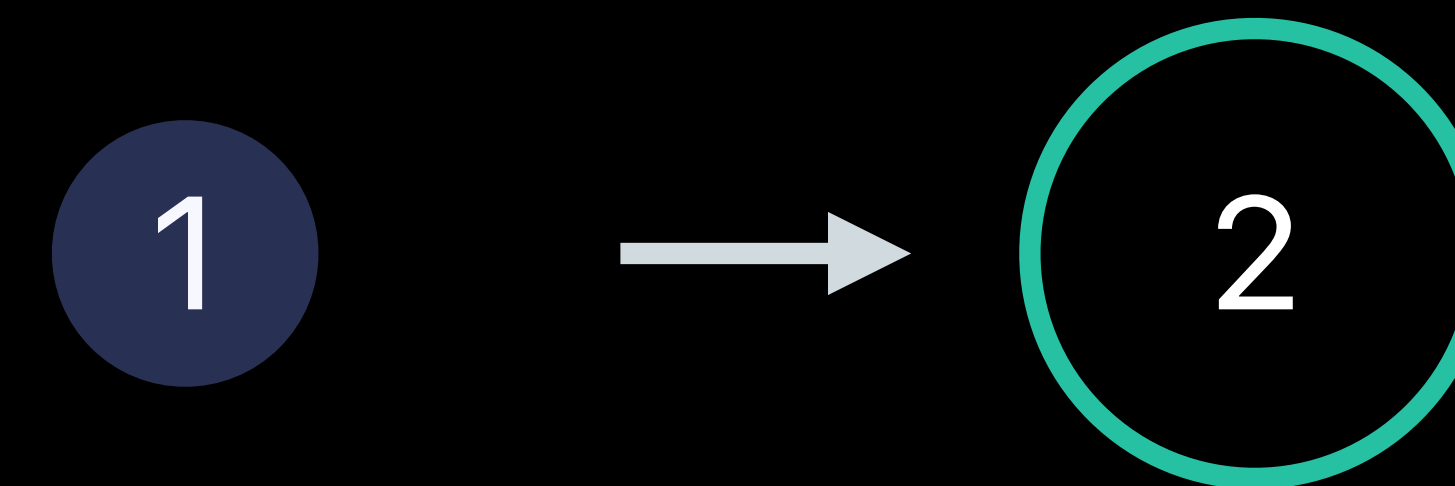


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderSessionTag]) {
    if case let NFCReaderSessionTag.iso15693(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.readSingleBlock(requestFlags: [.highDataRate, .address], blockNumber: 0) { (response:
                Data, error: Error?) in
                ...
            }
        }
    }
}
```


NFCReaderSession

ISO15693

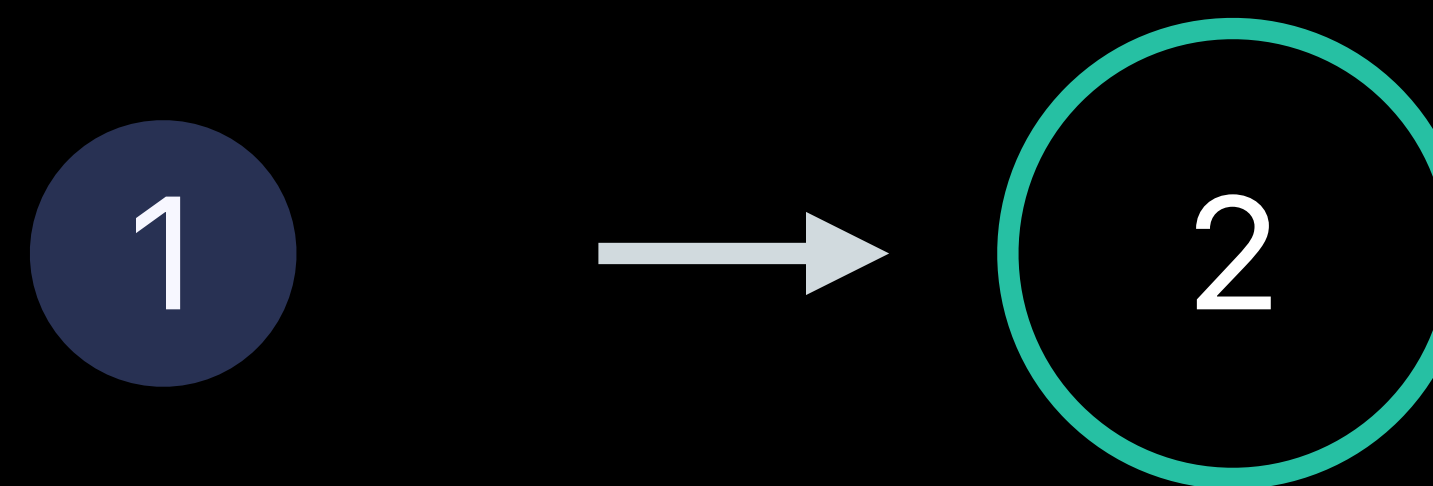


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderSessionTag]) {
    if case let NFCReaderSession.Tag.iso15693(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.readSingleBlock(requestFlags: [.highDataRate, .address], blockNumber: 0) { (response:
                Data, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

ISO15693

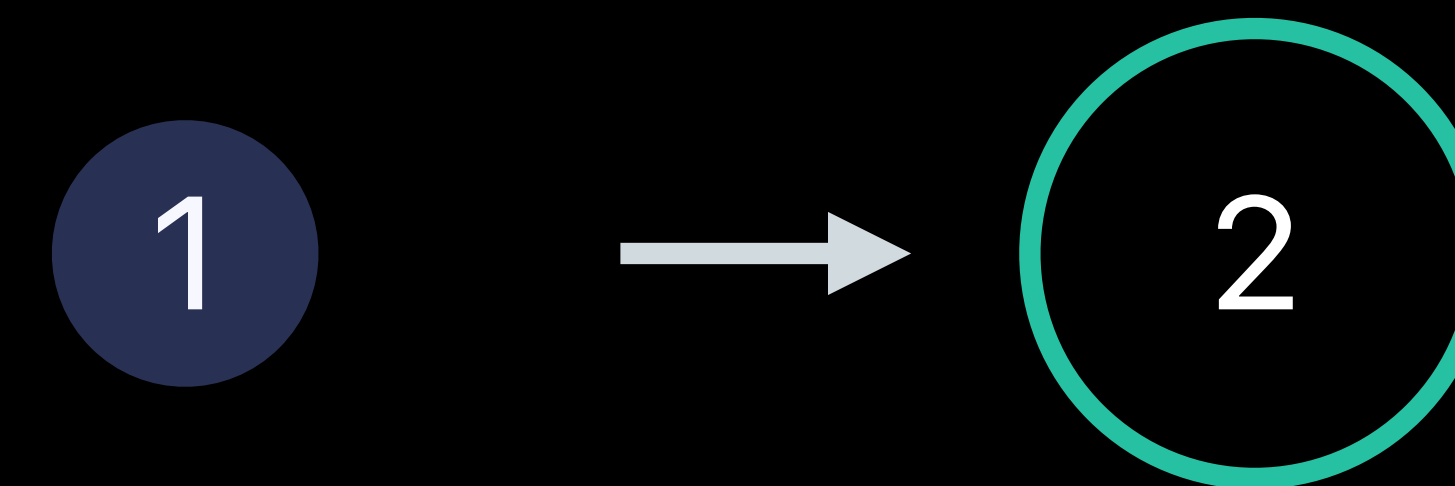


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderTag]) {
    if case let NFCReaderTag.iso15693(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.readSingleBlock(requestFlags: [.highDataRate, .address], blockNumber: 0) { (response:
                Data, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

ISO15693

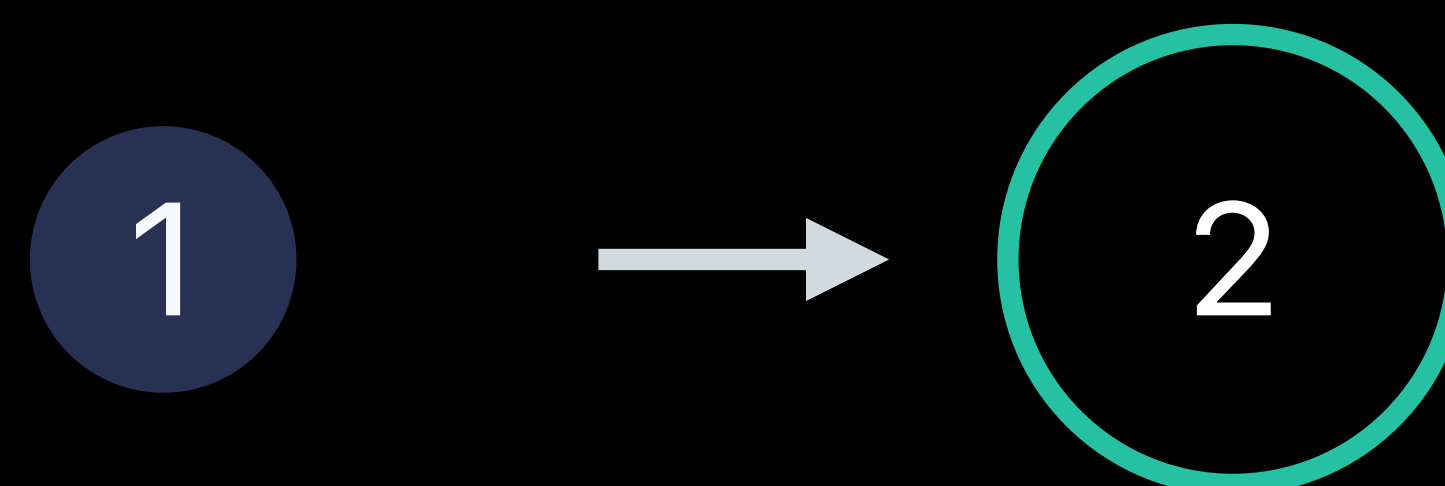


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderTag]) {  
    if case let NFCReaderTag.iso15693(tag) = tags.first {  
        session.connect(to: tag) { (error: Error?) in  
            ...  
            tag.readSingleBlock(requestFlags: [.highDataRate, .address], blockNumber: 0) { (response:  
                Data, error: Error?) in  
                ...  
            }  
        }  
    }  
}
```

NFCReaderSession

ISO15693

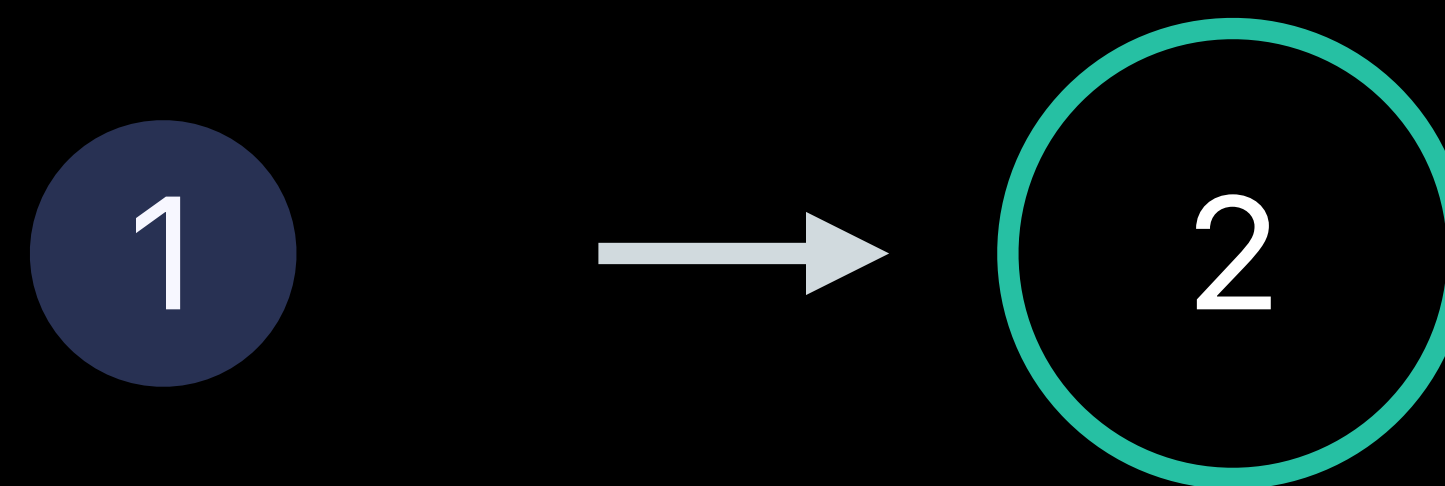


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderSessionTag]) {
    if case let NFCReaderSessionTag.iso15693(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.readSingleBlock(requestFlags: [.highDataRate, .address], blockNumber: 0) { (response:
                Data, error: Error?) in
                ...
            }
        }
    }
}
```


NFCReaderSession

ISO15693



Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderSessionTag]) {
    if case let NFCReaderSession.Tag.iso15693(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.readSingleBlock(requestFlags: [.highDataRate, .address], blockNumber: 0) { (response:
                Data, error: Error?) in
                ...
            }
        }
    }
}
```


Native Tag Reading

FeliCa[®]

FeliCa[®] Requirements

FeliCa[®] Requirements

Application Info.plist must contain list of system codes

FeliCa[®] Requirements

Application Info.plist must contain list of system codes

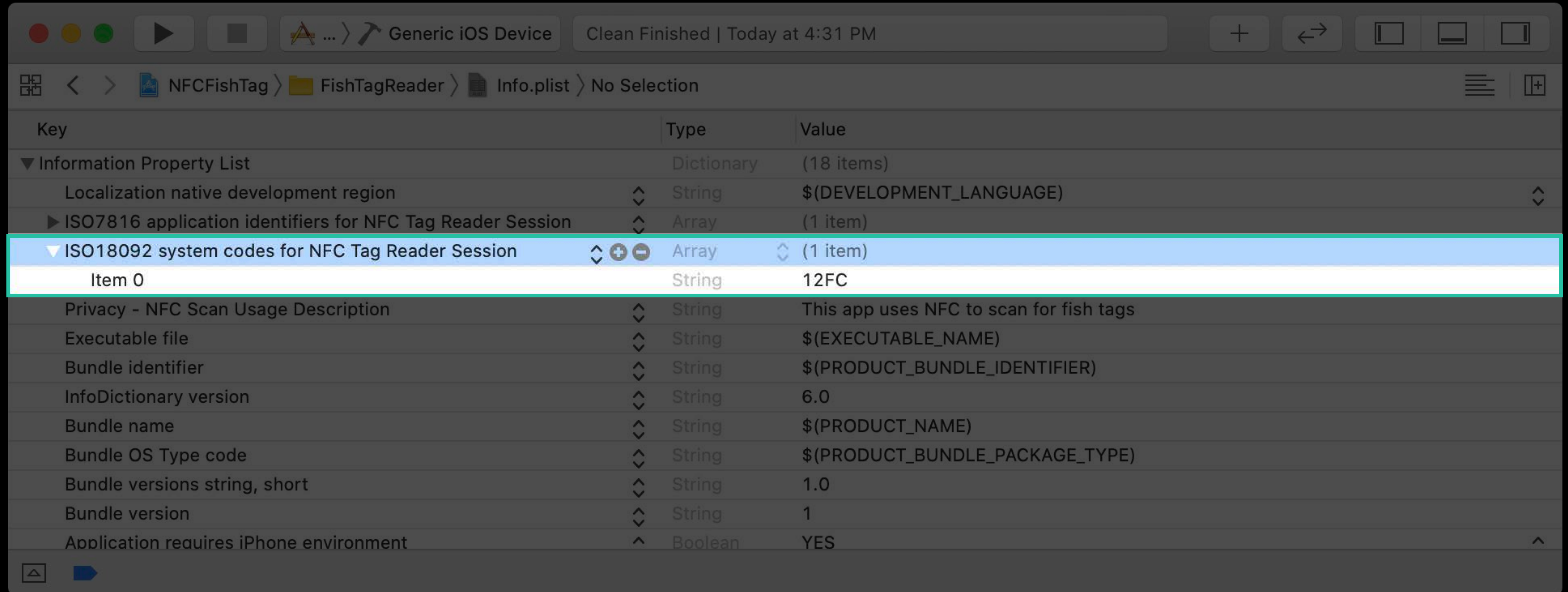
Detected tags callback is invoked if tag contains a system code found in the Info.plist

FeliCa[®] System Code Listing

The screenshot shows the Xcode interface with the Info.plist file open. The breadcrumb path is NFCFishTag > FishTagReader > Info.plist > No Selection. The table below lists the keys and their values in the Info.plist file.

Key	Type	Value
Information Property List	Dictionary	(18 items)
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
ISO7816 application identifiers for NFC Tag Reader Session	Array	(1 item)
ISO18092 system codes for NFC Tag Reader Session	Array	(1 item)
Item 0	String	12FC
Privacy - NFC Scan Usage Description	String	This app uses NFC to scan for fish tags
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	\$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES

FeliCa[®] System Code Listing



The screenshot shows the Xcode interface with the following details:

- Toolbar: Generic iOS Device, Clean Finished | Today at 4:31 PM
- Navigation: NFCFishTag > FishTagReader > Info.plist > No Selection
- Table with columns: Key, Type, Value

Key	Type	Value
Information Property List	Dictionary	(18 items)
Localization native development region	String	\$(DEVELOPMENT_LANGUAGE)
ISO7816 application identifiers for NFC Tag Reader Session	Array	(1 item)
ISO18092 system codes for NFC Tag Reader Session	Array	(1 item)
Item 0	String	12FC
Privacy - NFC Scan Usage Description	String	This app uses NFC to scan for fish tags
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	\$(PRODUCT_BUNDLE_PACKAGE_TYPE)
Bundle versions string, short	String	1.0
Bundle version	String	1
Application requires iPhone environment	Boolean	YES

NFCFeliCaTag

Properties

- `currentSystemCode`
- `currentIDM`

Methods

```
func sendFeliCaCommand(commandPacket: Data, completionHandler: @escaping (Data, Error?) -> Void)
```

NFCFeliCaTag Convenience Methods

Polling command

Request response

Request service

Request service V2

Request system code

Reset mode

Read without encryption

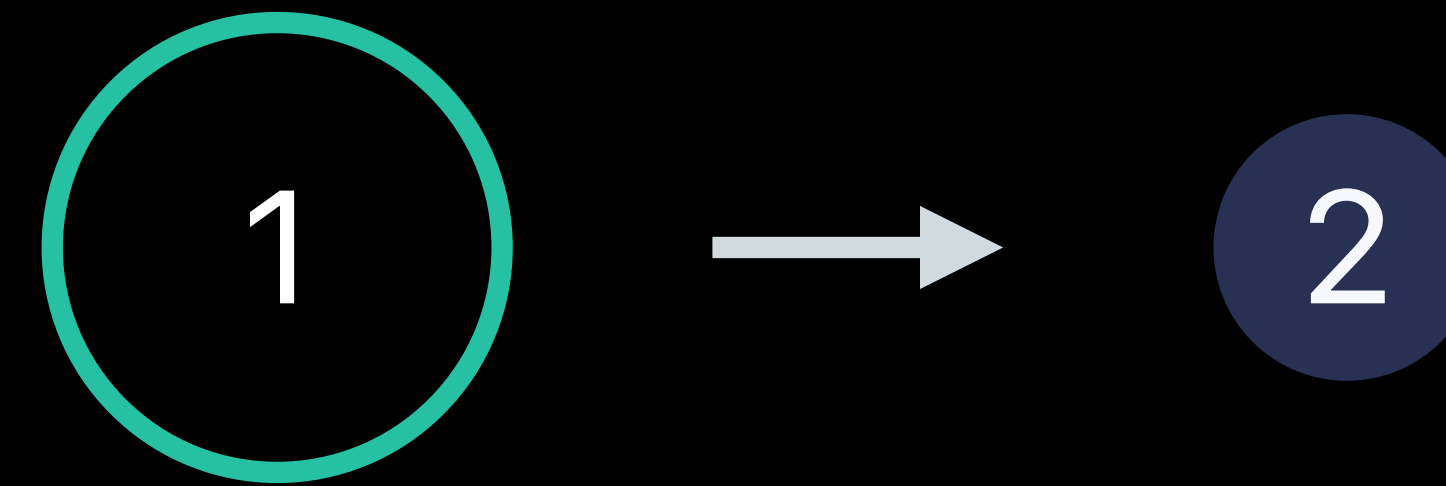
Write without encryption

Request specification version

Send FeliCa command packet

NFCReaderSession

FeliCa[®]

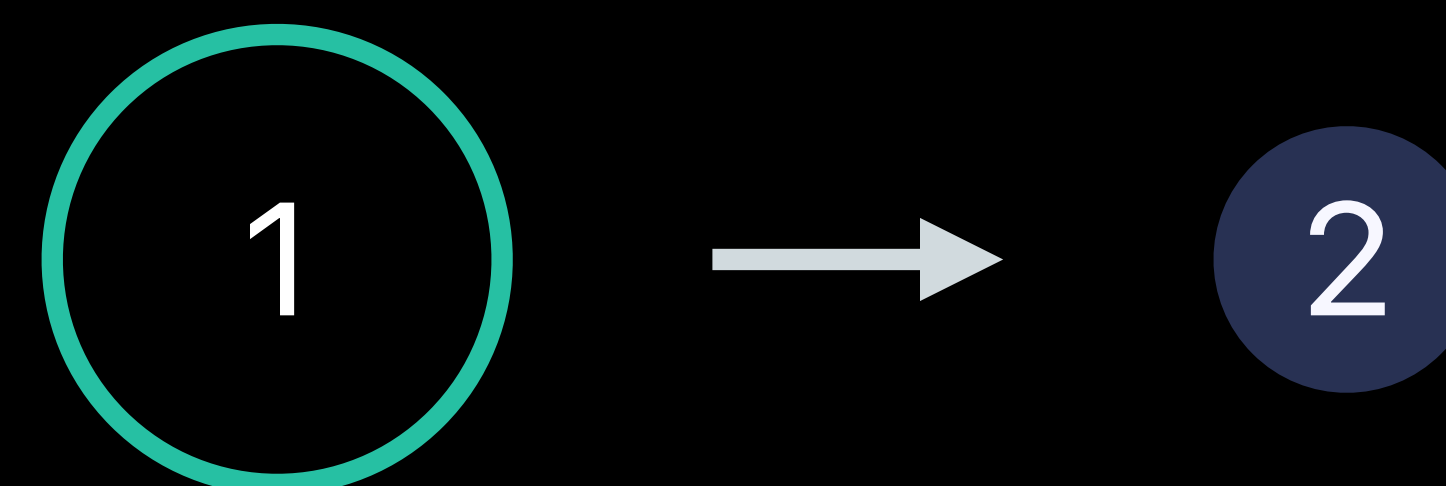


Create session with a `NFCReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {  
    session = NFCReaderSession(pollingOption: .iso18092, delegate: self)  
    session?.alertMessage = "Hold your iPhone near the tag to begin transaction."  
    session?.begin()  
}
```

NFCReaderSession

FeliCa®

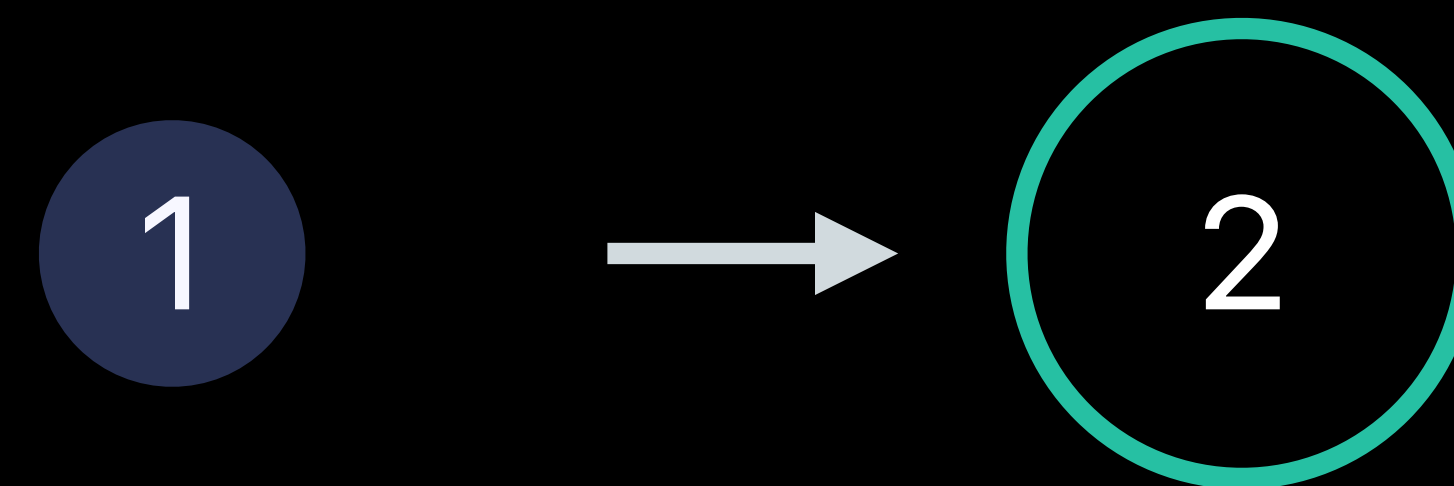


Create session with a `NFCReaderSessionDelegate` object

```
@IBAction func beginScanning(_ sender: Any) {  
    session = NFCReaderSession(pollingOption: .iso18092, delegate: self)  
    session?.alertMessage = "Hold your iPhone near the tag to begin transaction."  
    session?.begin()  
}
```


NFCReaderSession

FeliCa®

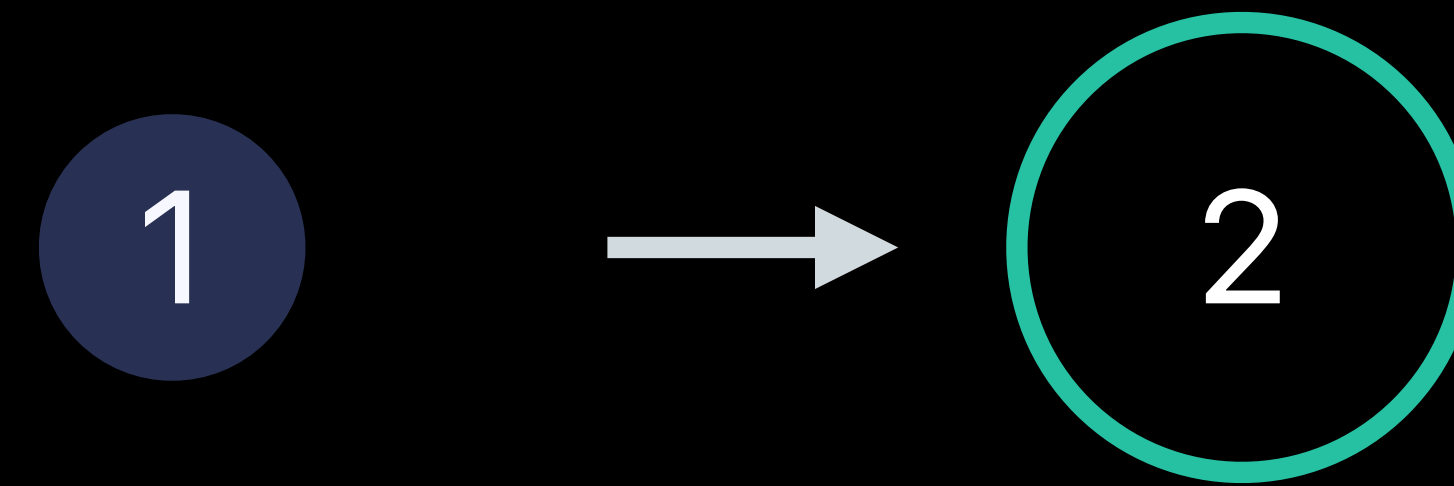


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCTag]) {  
    if case let NFCTag.feliCa(tag) = tags.first {  
        session.connect(to: tag) { (error: Error?) in  
  
            ...  
            tag.requestResponse() { (mode: Int, error: Error?) in  
  
                ...  
            }  
        }  
    }  
}
```

NFCReaderSession

FeliCa[®]

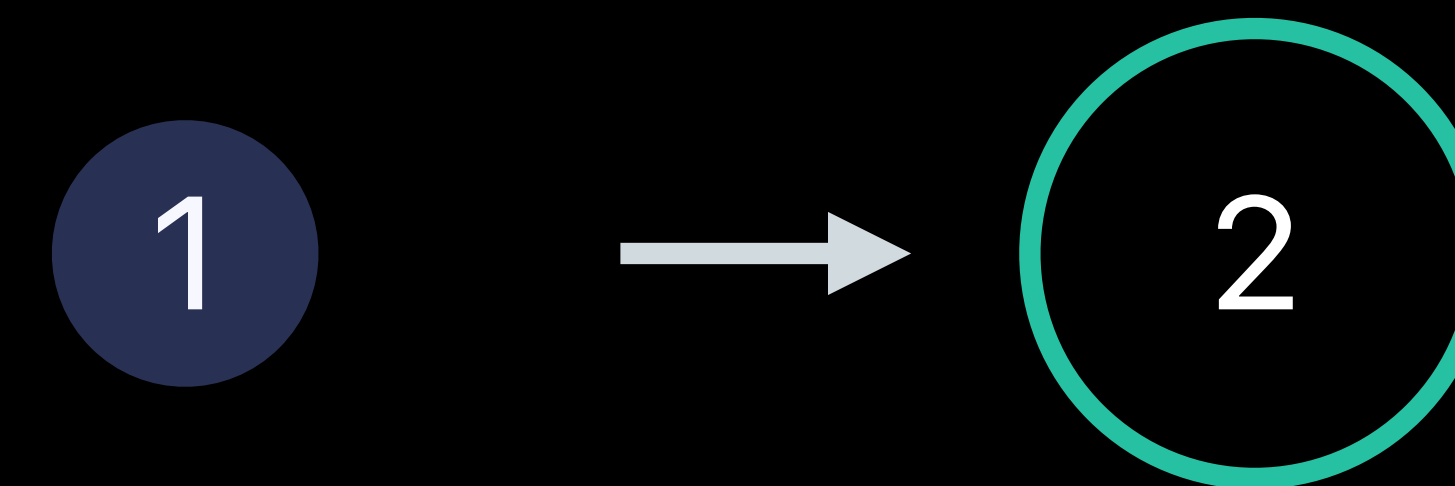


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderTag]) {
    if case let NFCReaderSession.NFCReaderTag.feliCa(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.requestResponse() { (mode: Int, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

FeliCa[®]

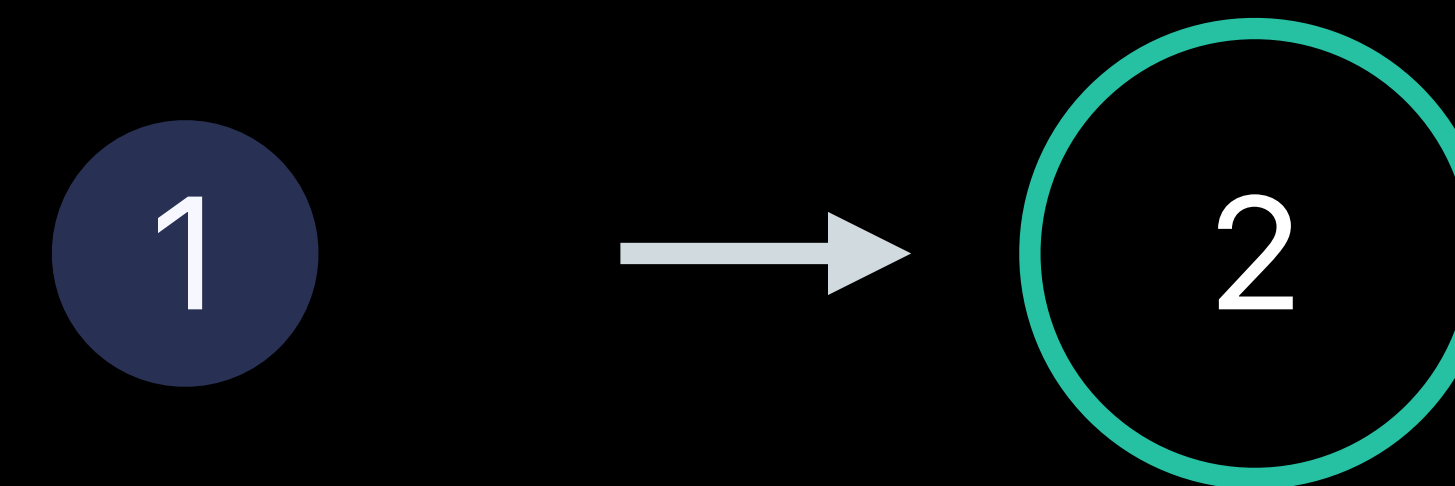


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderSessionTag]) {
    if case let NFCReaderSessionTag.feliCa(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.requestResponse() { (mode: Int, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

FeliCa[®]

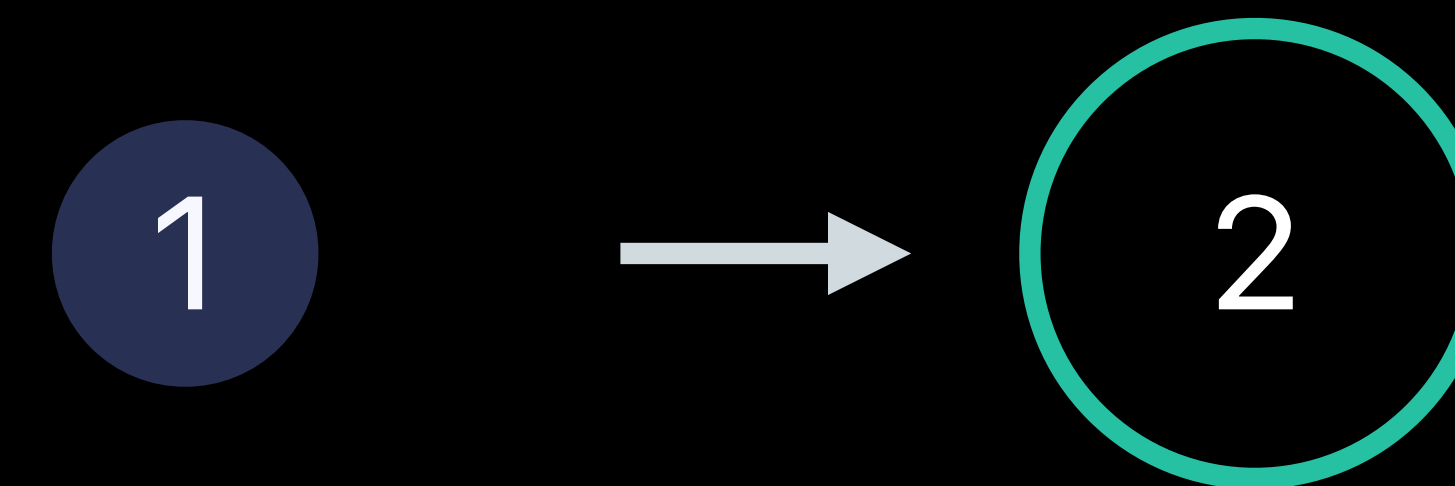


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCTag]) {
    if case let NFCTag.feliCa(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.requestResponse() { (mode: Int, error: Error?) in
                ...
            }
        }
    }
}
```

NFCReaderSession

FeliCa[®]

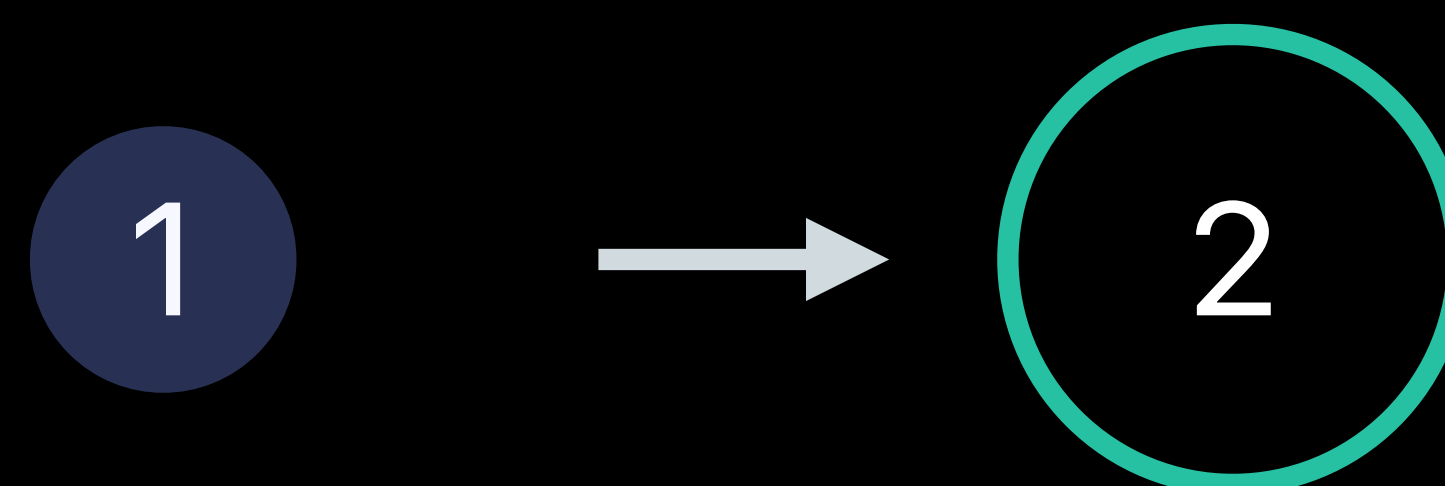


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderSession.NFCReaderSessionTag]) {
    if case let NFCReaderSessionTag.feliCa(tag) = tags.first {
        session.connect(to: tag) { (error: Error?) in
            ...
            tag.requestResponse() { (mode: Int, error: Error?) in
                ...
            }
        }
    }
}
```


NFCReaderSession

FeliCa[®]



Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCReaderSession, didDetect tags: [NFCReaderTag]) {  
    if case let NFCReaderTag.feliCa(tag) = tags.first {  
        session.connect(to: tag) { (error: Error?) in  
            ...  
            tag.requestResponse() { (mode: Int, error: Error?) in  
                ...  
            }  
        }  
    }  
}
```

Demo

Story of a fish monger

Lawrence Chung, NFC Software

9:41



The Great Fish Company

Date:

March	31	2016
April	1	2017
May	2	2018
June	3	2019
July	4	2020
August	5	2021
September	6	2022

Kind:

Creative Salmon

Amazing Tuna

Dancing Mahi-Mahi

Incredible Bass

Price:

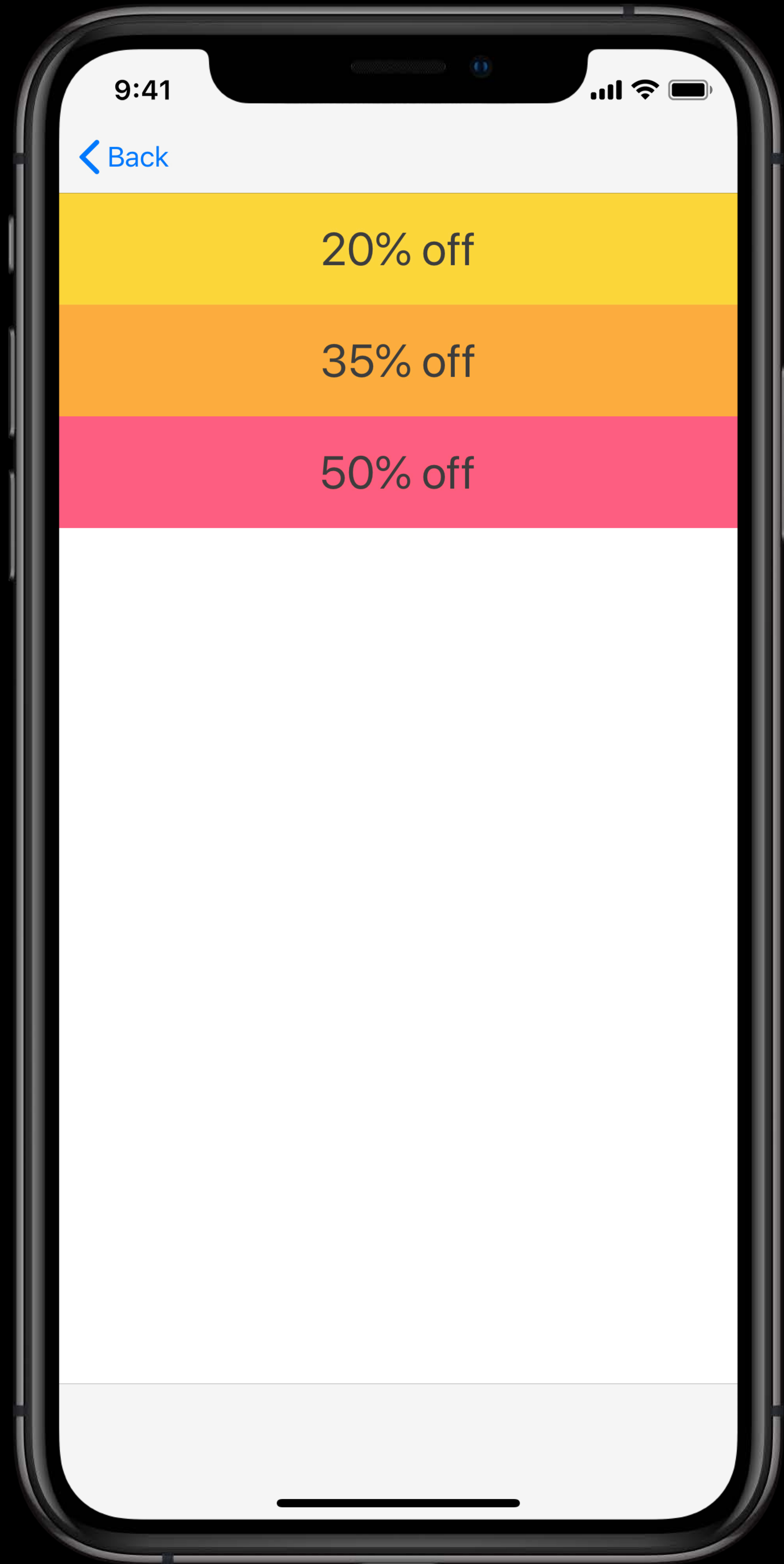
Small - \$5.99

Medium - \$10.99

Large - \$15.99

Write To Tag

Create Coupon



9:41



[← Back](#)

20% off

35% off

50% off

9:41



The Great Fish Company

Thank you for your purchase! As a responsible fish monger, the Great Fish Company loves and cares about our ocean.

Kind:

Amazing Tuna

Date:

June 8, 2019

Price:

\$10.99

Addition Info:

Brought to you byt the Great Fish Company

Scan Tag



Featured



More

9:41



Loyalty Program

Special offer code:

Scan Coupon!



Featured



More

Summary

Use `NFCNDEFReaderSession` to get NDEF tags

Use `NFCTagReaderSession` to get native tags

Summary

Use `NFCNDEFReaderSession` to get NDEF tags

Use `NFCTagReaderSession` to get native tags

Receive tag objects in the callback

Summary

Use `NFCNDEFReaderSession` to get NDEF tags

Use `NFCTagReaderSession` to get native tags

Receive tag objects in the callback

Perform read and write operations using tag objects

Summary

Use `NFCNDEFReaderSession` to get NDEF tags

Use `NFCTagReaderSession` to get native tags

Receive tag objects in the callback

Perform read and write operations using tag objects

Indicate success or failure in your invalidation call

More Information

developer.apple.com/wwdc19/715

