#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

- `NFCTag`

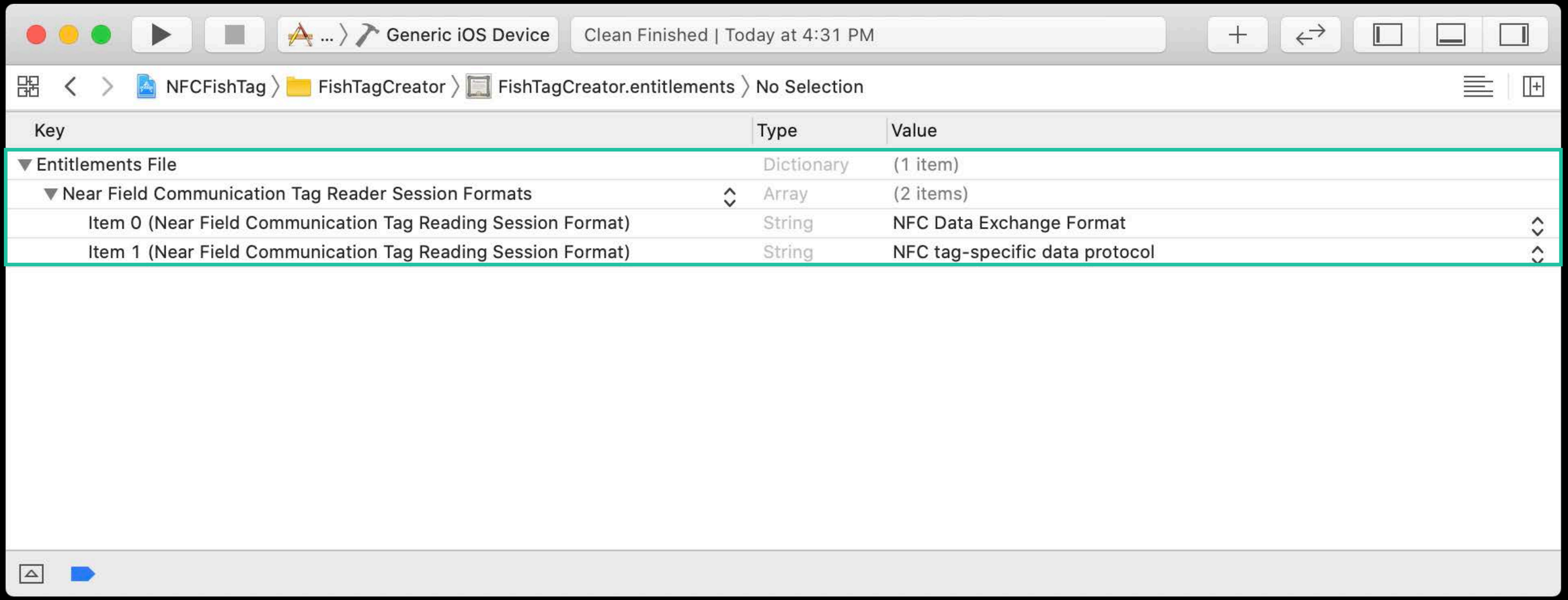
Protocols for different tag types

- `NFCNDEFTag`
- `NFCISO7816Tag`
- `NFCMiFareTag`
- `NFCISO15693Tag`
- `NFCFeliCaTag`

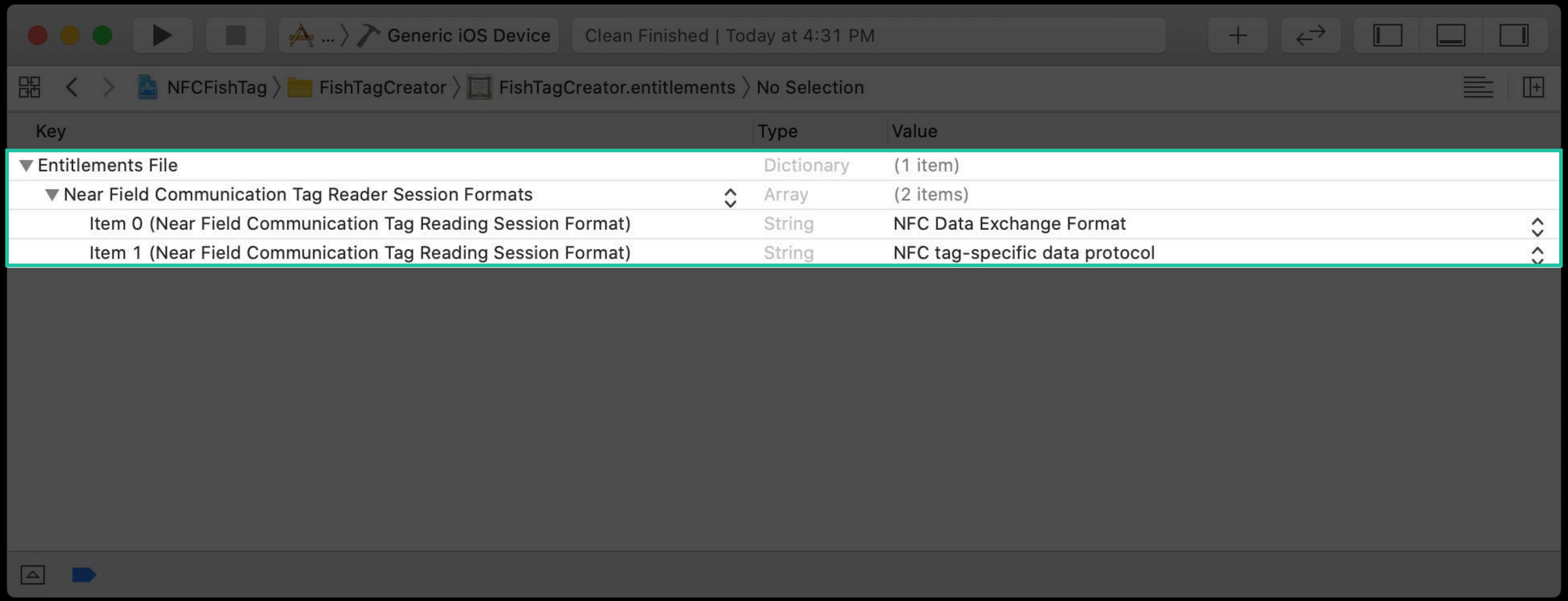
General API usage

Enable the NFC entitlement in Xcode

Xcode Configuration



Xcode Configuration



General API Usage

Enable the NFC entitlement in Xcode

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

General API Usage

Enable the NFC entitlement in Xcode

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

Receive detected tags in the discovery callback

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

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

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

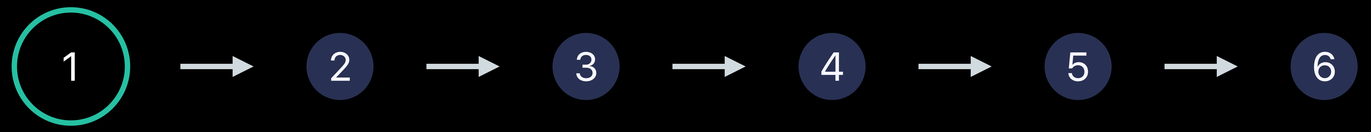
New `NFCNDEFReaderSessionDelegate` method to receive NDEF tag objects

```
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)
```

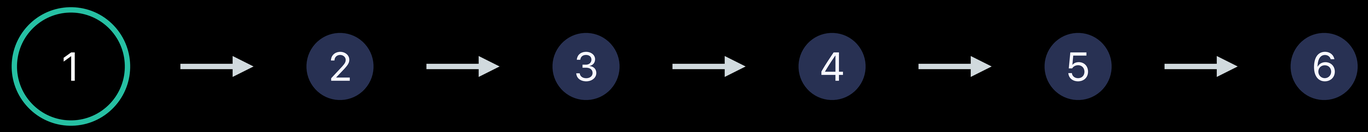
NFCNDEFReaderSession



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()
    }
```

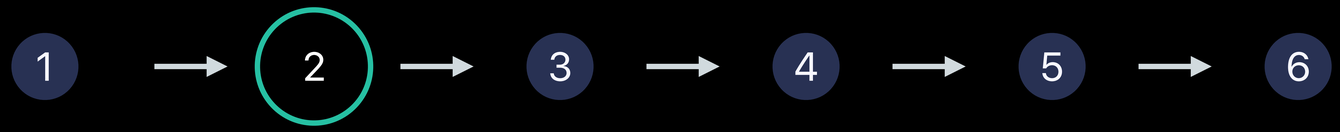
NFCNDEFReaderSession



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()
    }
```

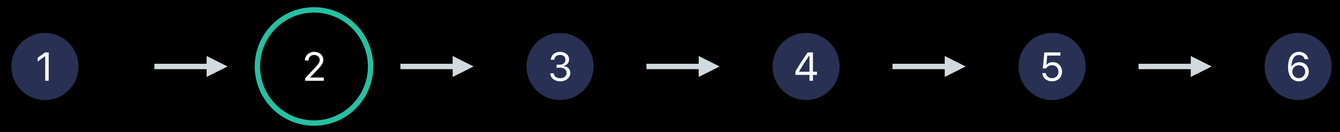
NFCNDEFReaderSession



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

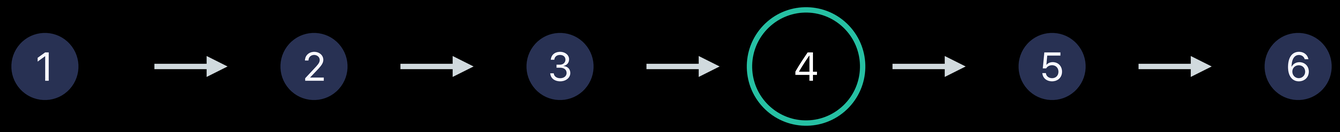
```
func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
 // Connect to the found tag and perform NDEF message writing
 let tag = tags.first!
 session.connect(to: tag) { (error: Error?) in
    …
  }
 }
}
```

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
    …
  }
 }
}
```

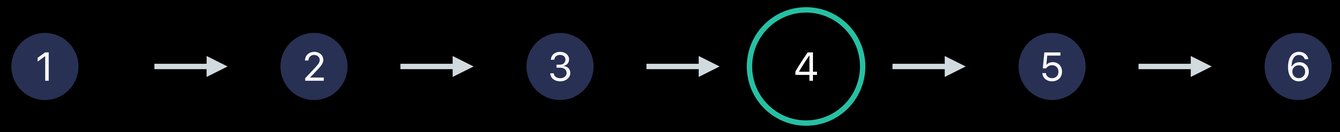
NFCNDEFReaderSession



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()
   }
  }
 }
```

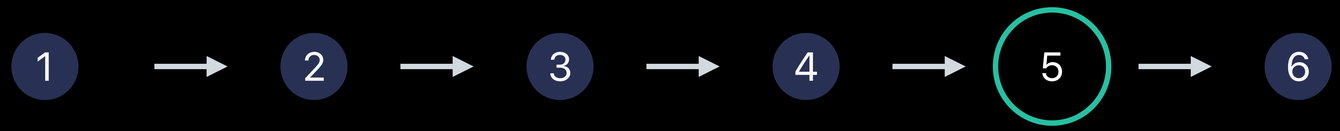
NFCNDEFReaderSession



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()
   }
  }
 }
```

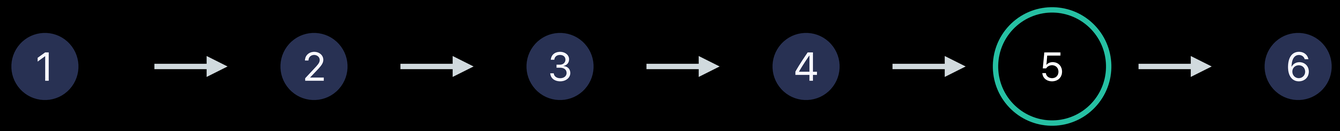
NFCNDEFReaderSession



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()
   }
  }
 }
```

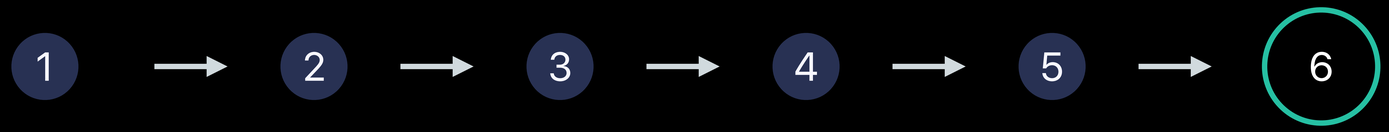
NFCNDEFReaderSession



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()
   }
  }
 }
```

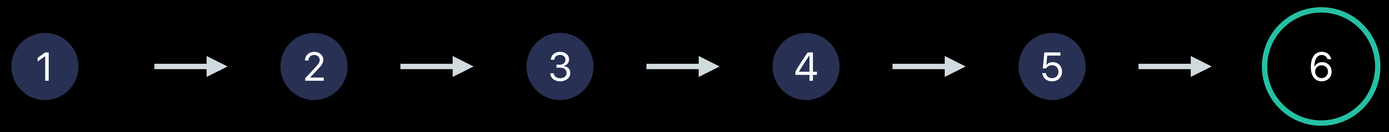
NFCNDEFReaderSession



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

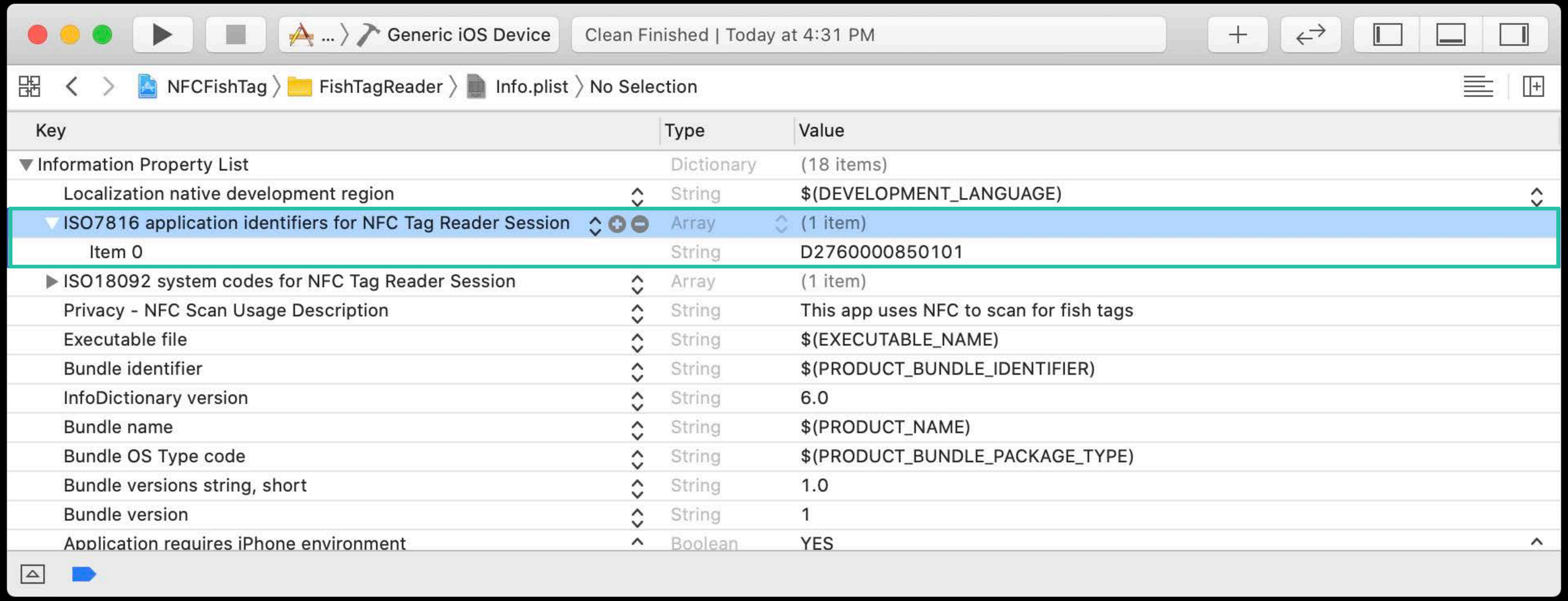
Application Info.plist must contain list of application identifiers (AIDs)

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

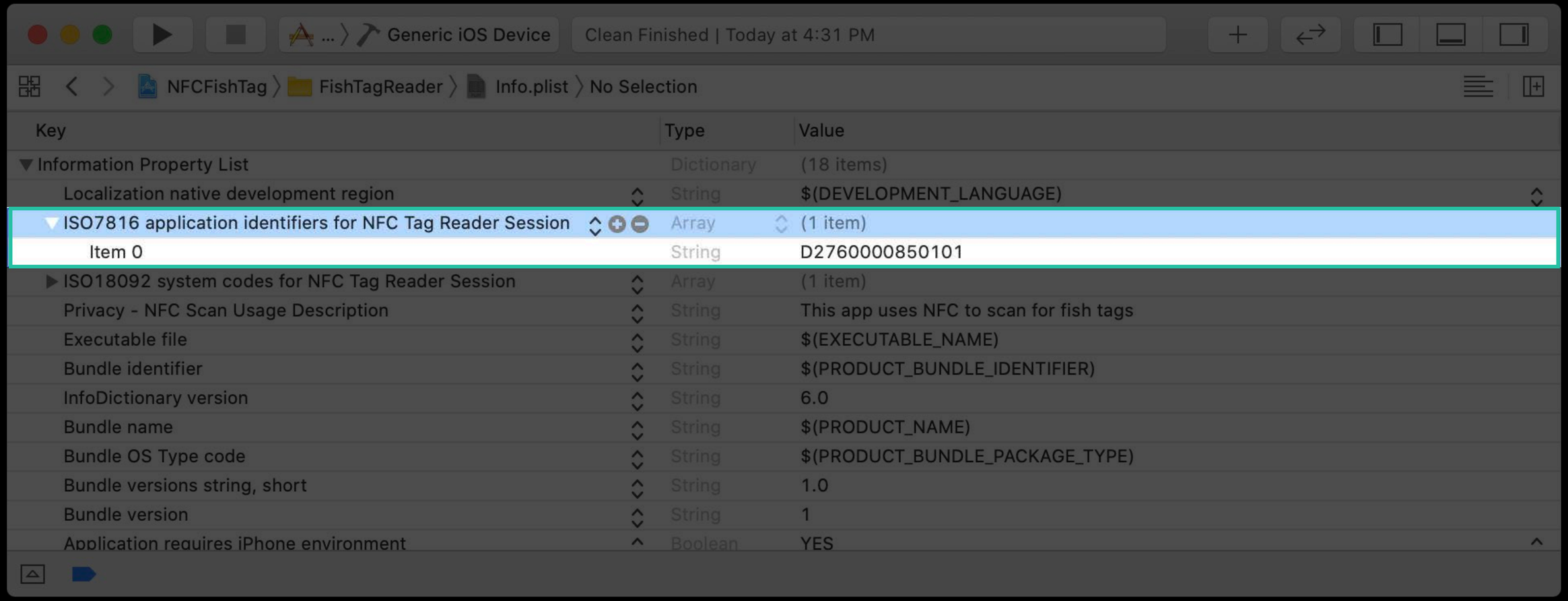
ISO7816 Requirements

- 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



ISO7816 AID Listing



NFCISO7816Tag

Properties

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

Methods

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

NFCTagReaderSession

ISO7816

`NFCTagReaderSessionDelegate` protocol for session delegate object

```
func tagReaderSessionDidBecomeActive(_ session: NFCTagReaderSession)

func tagReaderSession(_ session: NFCTagReaderSession, didInvalidateWithError: Error)

func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag])
```

NFCTagReaderSession

ISO7816

Create session with a `NFCTagReaderSessionDelegate` object

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

NFCTagReaderSession

ISO7816

Create session with a `NFCTagReaderSessionDelegate` object

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

NFCTagReaderSession

ISO7816

Implements `tagReaderSession(_:didDetect:)` method

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

NFCTagReaderSession

ISO7816

Implements `tagReaderSession(_:didDetect:)` method

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

NFCTagReaderSession

ISO7816

Connect to the ISO7816 tag

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

NFCTagReaderSession

ISO7816

Connect to the ISO7816 tag

```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.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
   …
  }
 }
 …
```

NFCTagReaderSession

ISO7816

Terminate session with error (optional)

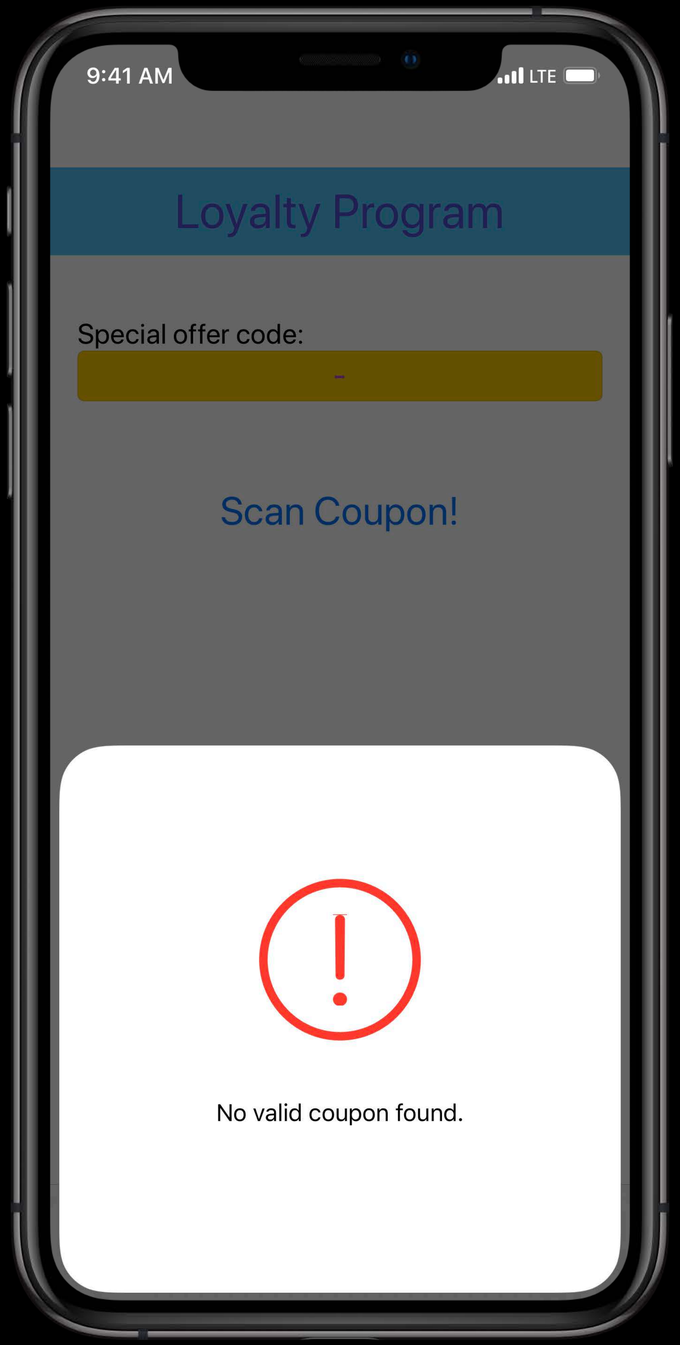
```
…
tag.sendCommand(apdu: myAPDU) { (response: Data, sw1: UInt8, sw2: UInt8, error: Error?)
in
 guard error != nil && !(sw1 == 0x90 && sw2 == 0) else {
  session.invalidate(errorMessage: "Application failure")
  return
 }
```

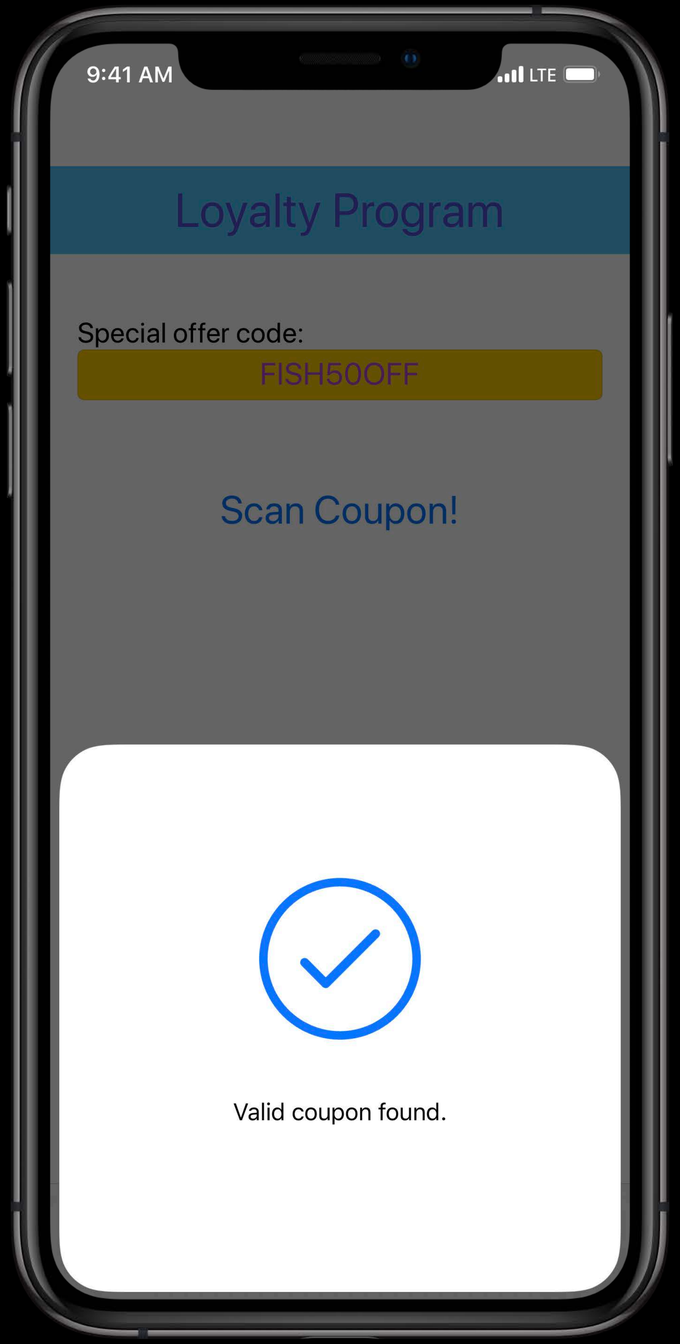
NFCTagReaderSession

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:
Scan Coupon!
No valid coupon found.


9:41 AM
LTE
Loyalty Program
Special offer code:
FISH50OFF
Scan Coupon!
Valid coupon found.

Native Tag Reading

MIFARE®

NFCMiFareTag

Properties

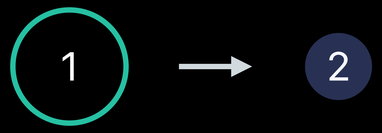
- `identifier` (UID)
- `historicalBytes`
- `mifareFamily` — Ultralight , Plus, DESFire

Methods

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

NFCTagReaderSession

MIFARE®

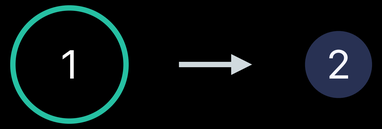


Create session with an `NFCTagReaderSessionDelegate` object

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

NFCTagReaderSession

MIFARE®

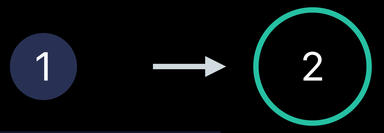


Create session with an `NFCTagReaderSessionDelegate` object

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

NFCTagReaderSession

MIFARE®

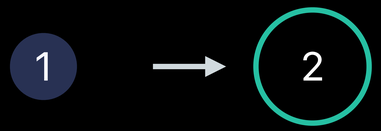


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.miFare(tag) = tags.first {
  session.connect(to: tag) { (error: Error?) in
   …
   tag.sendMiFareCommand(comandPacket: command) { (response: Data, error: Error?) in
    …
   }
  }
 }
}
```

NFCTagReaderSession

MIFARE®

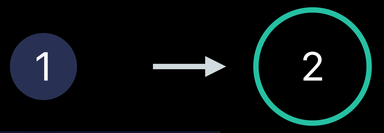


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.miFare(tag) = tags.first {
  session.connect(to: tag) { (error: Error?) in
   …
   tag.sendMiFareCommand(comandPacket: command) { (response: Data, error: Error?) in
    …
   }
  }
 }
}
```

NFCTagReaderSession

MIFARE®

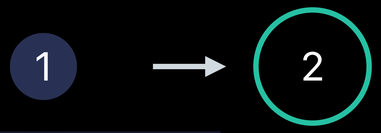


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.miFare(tag) = tags.first {
  session.connect(to: tag) { (error: Error?) in
   …
   tag.sendMiFareCommand(comandPacket: command) { (response: Data, error: Error?) in
    …
   }
  }
 }
}
```

NFCTagReaderSession

MIFARE®

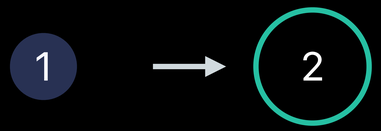


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.miFare(tag) = tags.first {
  session.connect(to: tag) { (error: Error?) in
   …
   tag.sendMiFareCommand(comandPacket: command) { (response: Data, error: Error?) in
    …
   }
  }
 }
}
```

NFCTagReaderSession

MIFARE®

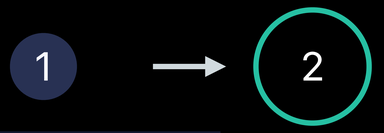


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.miFare(tag) = tags.first {
  session.connect(to: tag) { (error: Error?) in
   …
   tag.sendMiFareCommand(comandPacket: command) { (response: Data, error: Error?) in
    …
   }
  }
 }
}
```

NFCTagReaderSession

MIFARE®



Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.miFare(tag) = tags.first {
  session.connect(to: tag) { (error: Error?) in
   …
   tag.sendMiFareCommand(comandPacket: 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

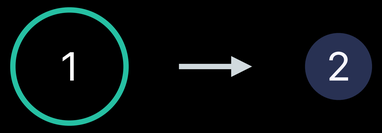
Write DSID

Lock DSID

Get system info

NFCTagReaderSession

ISO15693

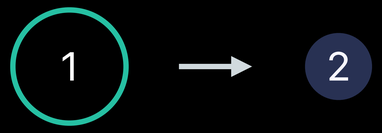


Create session with a `NFCTagReaderSessionDelegate` object

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

NFCTagReaderSession

ISO15693

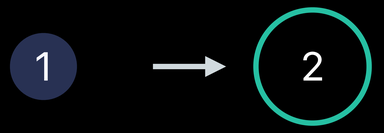


Create session with a `NFCTagReaderSessionDelegate` object

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

NFCTagReaderSession

ISO15693

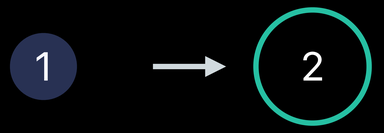


Implement `tagReaderSession(_:didDetect:)` method

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

NFCTagReaderSession

ISO15693

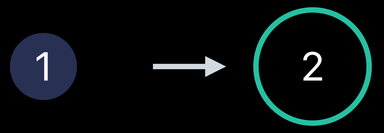


Implement `tagReaderSession(_:didDetect:)` method

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

NFCTagReaderSession

ISO15693

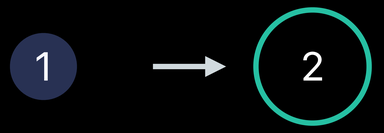


Implement `tagReaderSession(_:didDetect:)` method

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

NFCTagReaderSession

ISO15693

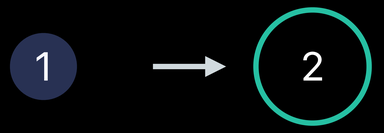


Implement `tagReaderSession(_:didDetect:)` method

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

NFCTagReaderSession

ISO15693

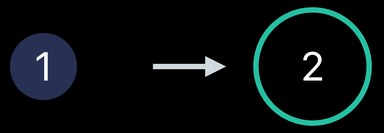


Implement `tagReaderSession(_:didDetect:)` method

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

NFCTagReaderSession

ISO15693



Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.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

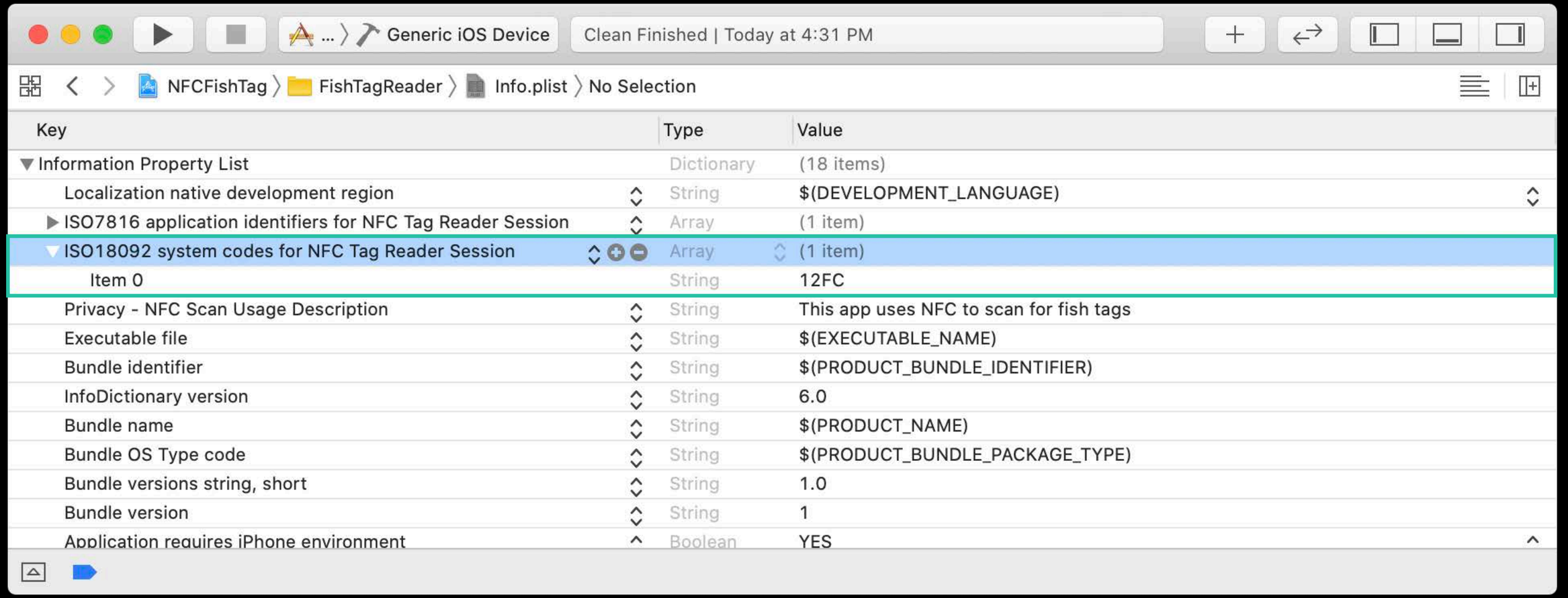
Application Info.plist must contain list of system codes

FeliCa® Requirements

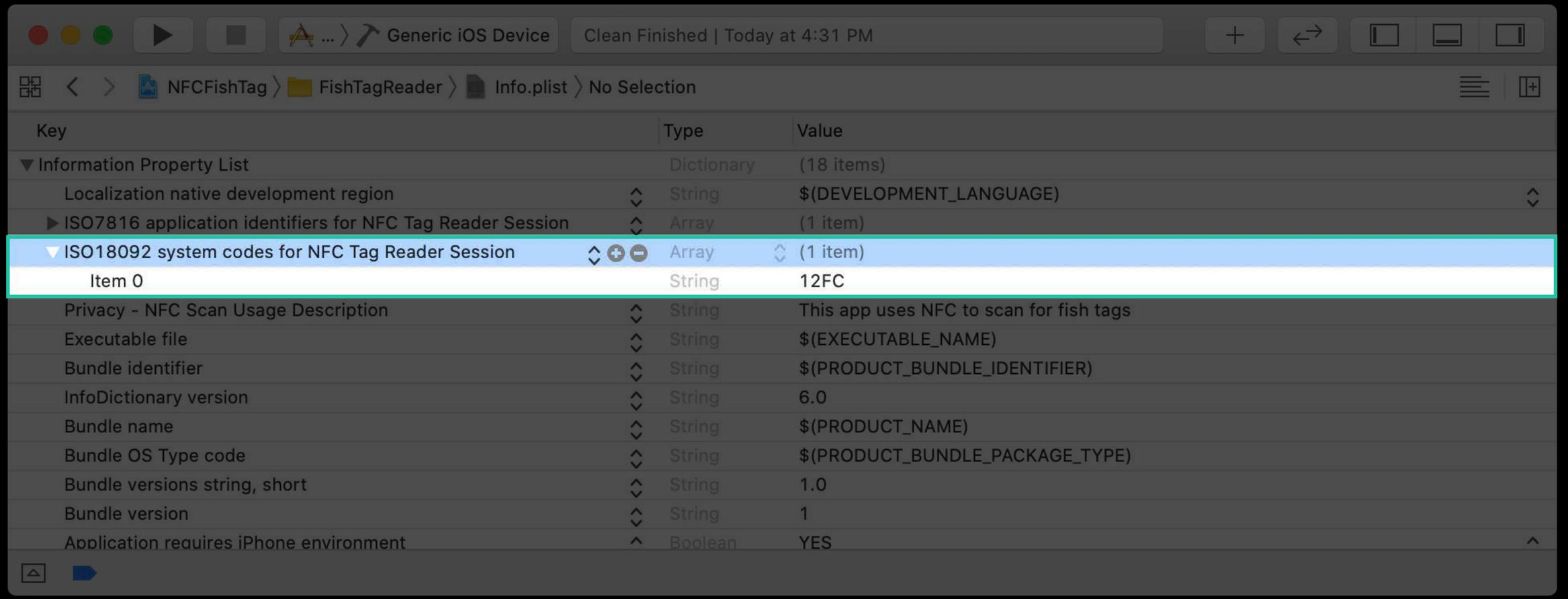
Application Info.plist must contain list of system codes

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

FeliCa® System Code Listing



FeliCa® System Code Listing



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

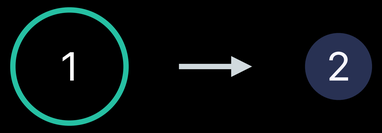
Write without encryption

Request specification version

Send FeliCa command packet

NFCTagReaderSession

FeliCa®

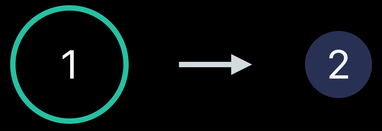


Create session with a `NFCTagReaderSessionDelegate` object

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

NFCTagReaderSession

FeliCa®

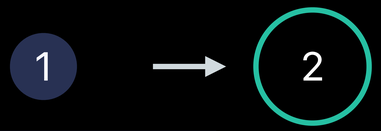


Create session with a `NFCTagReaderSessionDelegate` object

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

NFCTagReaderSession

FeliCa®

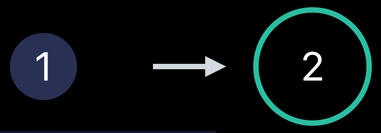


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, 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
    …
   }
  }
 }
}
```

NFCTagReaderSession

FeliCa®

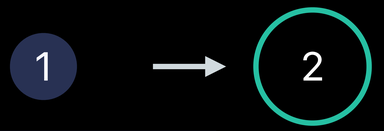


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, 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
    …
   }
  }
 }
}
```

NFCTagReaderSession

FeliCa®

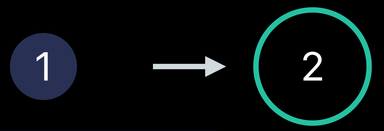


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, 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
    …
   }
  }
 }
}
```

NFCTagReaderSession

FeliCa®

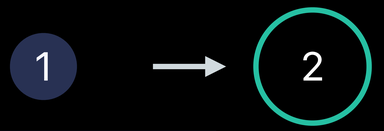


Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, 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
    …
   }
  }
 }
}
```

NFCTagReaderSession

FeliCa®



Implement `tagReaderSession(_:didDetect:)` method

```
func tagReaderSession(_ session: NFCTagReaderSession, 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
    …
   }
  }
 }
}
```

NFCTagReaderSession

FeliCa®

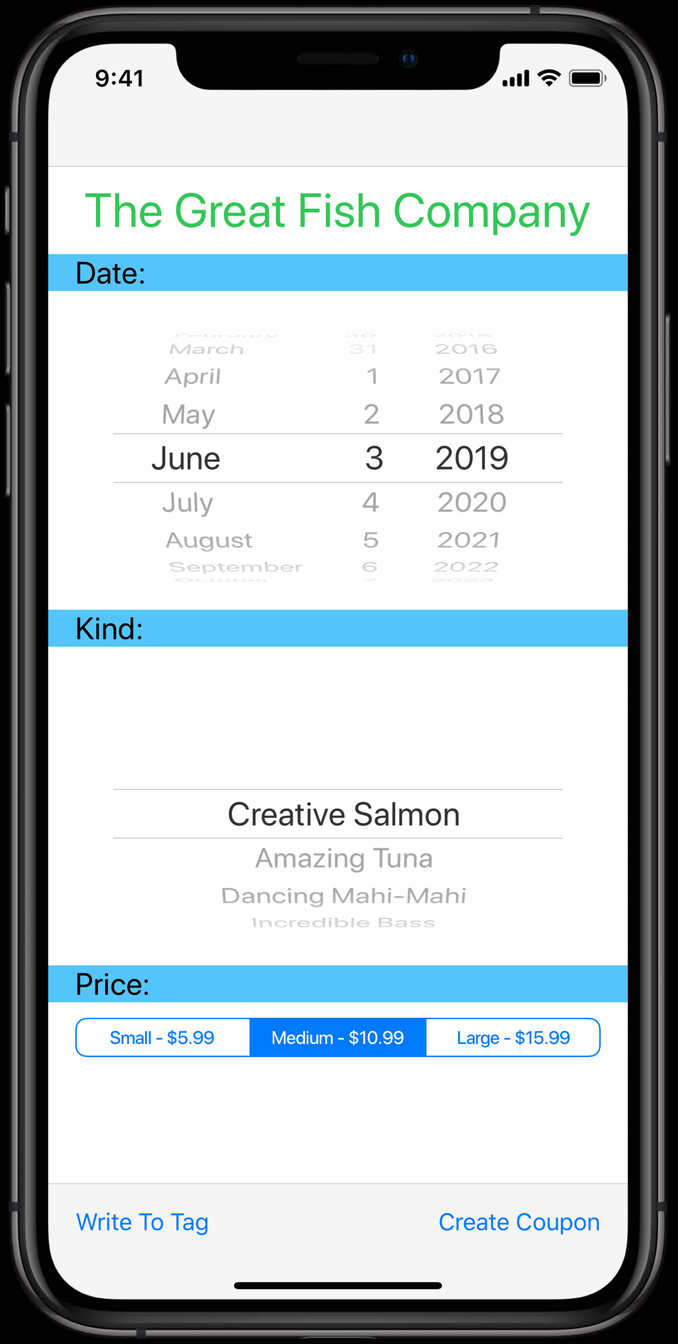
Implement `tagReaderSession(_:didDetect:)` method

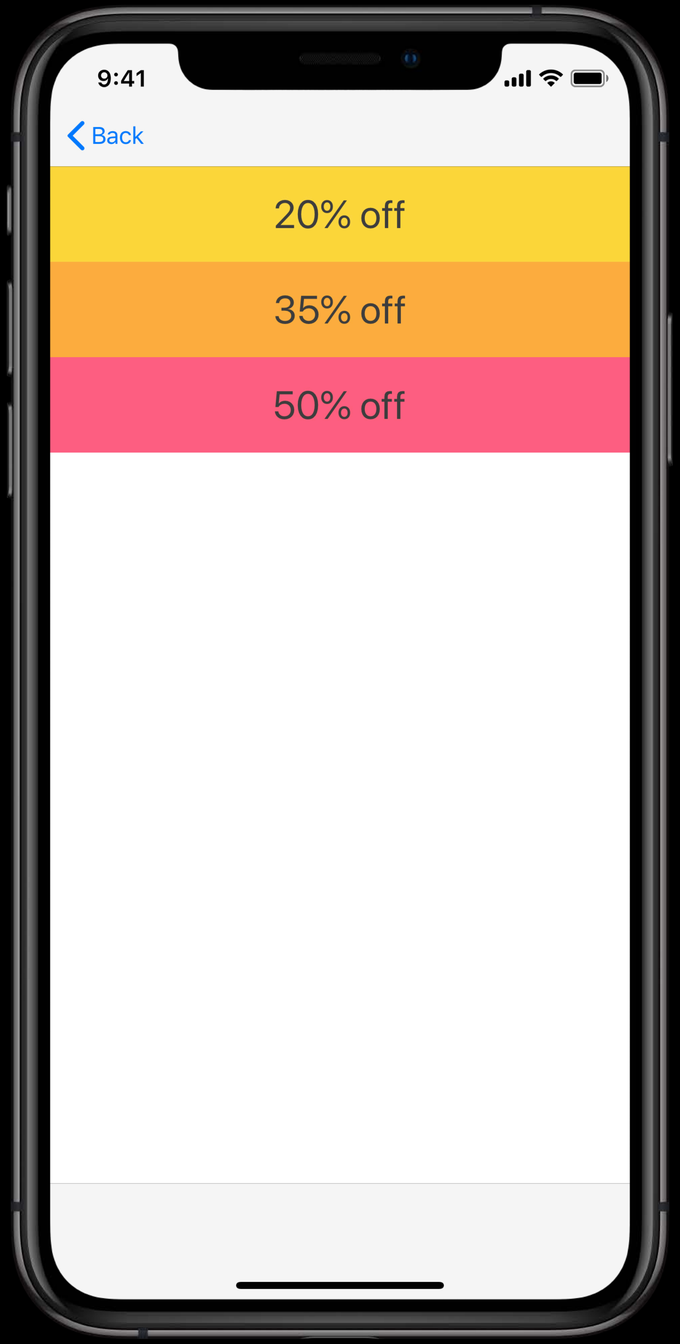
```
func tagReaderSession(_ session: NFCTagReaderSession, didDetect tags: [NFCTag]) {
 if case let NFCTag.feliCa(tag) = tags.first {
  session.connect(to: tag) { (error: Error?) in
   …
   tag.requestResponse() { (mode: Int, error: Error?) in
    …
   }
  }
 }
}
```

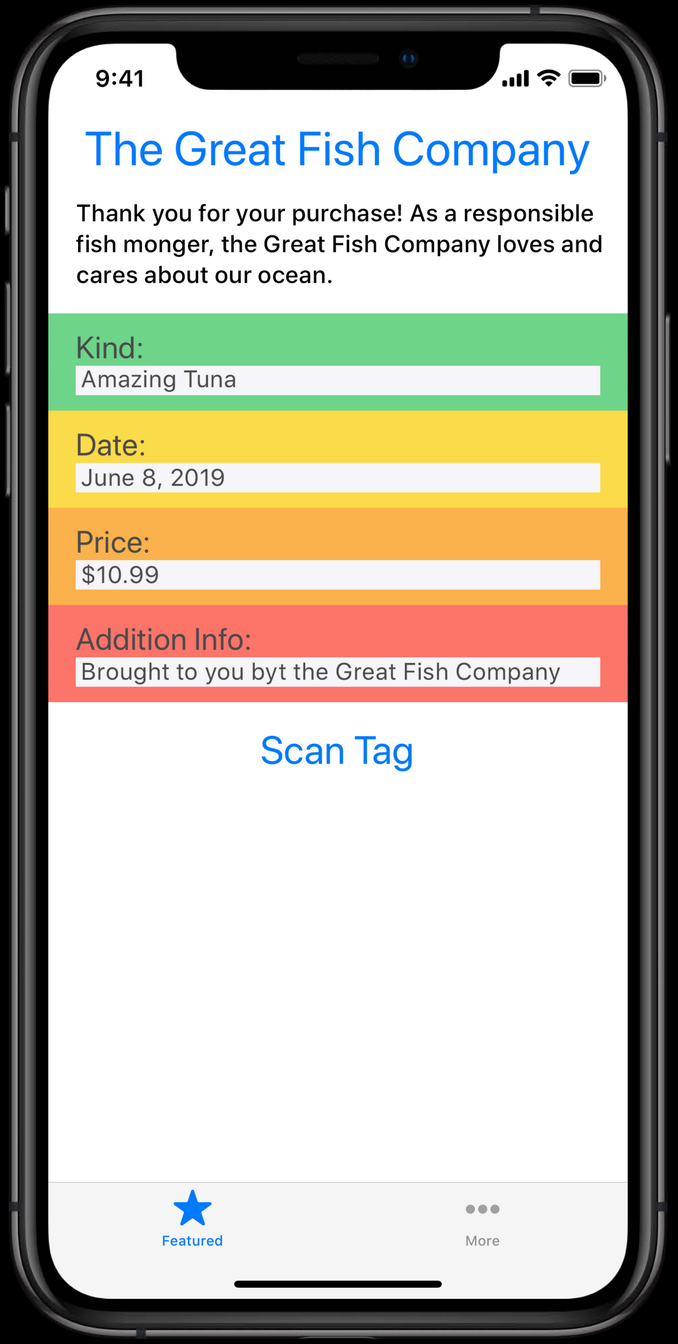
Demo

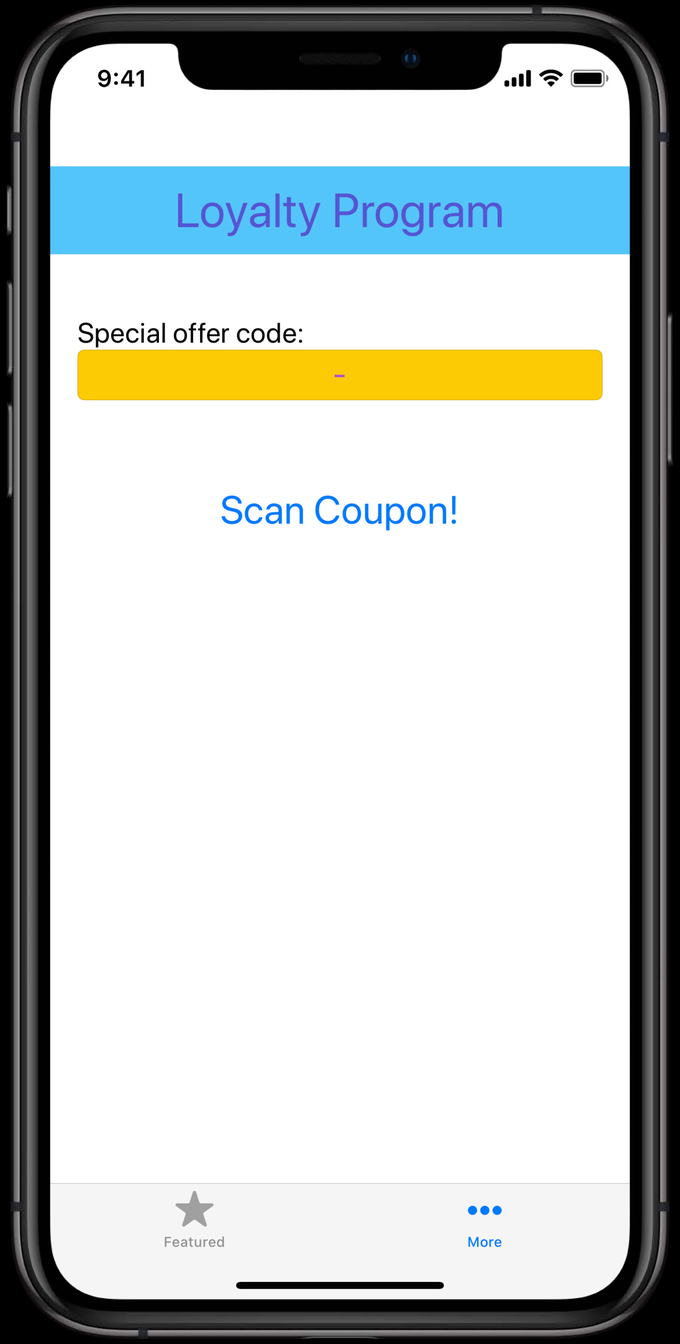
Story of a fish monger

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


WWDC19