Sample Code

Building an NFC Tag-Reader App

Read NFC tags with NDEF messages in your app.

Download

iOS 13.0+ | iPadOS 13.0+ | Xcode 11.0+

Overview

This sample code project shows how to use Core NFC in an app to read Near Field Communication (NFC) tags of types 1 through 5 that contains NFC Data Exchange Format (NDEF) data. To use this sample, download the project and build it using Xcode. Run the sample app on your iPhone. Tap the Scan button to start scanning for tags, then hold the phone near an NFC tag.

To read a tag, the sample app creates an NFC NDEF reader session and provides a delegate. The running reader session polls for NFC tags and calls the delegate when it finds tags that contain NDEF messages, passing the messages to the delegate. The delegate then stores the messages so the user can view them later.

Configure the App to Detect NFC Tags

Begin building your tag reader by configuring your app to detect NFC tags. Turn on Near Field Communication Tag Reading under the Capabilities tab for the project's target (see Add a capability to a target). This step:

- Adds the NFC tag-reading feature to the App ID.
- Adds the `Near Field Communication Tag Reader Session Formats Entitlement` to the entitlements file.

Next, add the `NFCReaderUsageDescription` key as a string item to the `Info.plist` file. For the value, enter a string that describes the reason the app needs access to the device's NFC reader. If the app attempts to read a tag without providing this key and string, the app exits.

Start a Reader Session

Create an NFCNDEFReaderSession object by calling the init(delegate:queue:invalidateAfterFirstRead:) initializer method and passing in:

- The reader session delegate object.
- The dispatch queue to use when calling methods on the delegate.
- The invalidateAfterFirstRead flag to determine whether the reader session reads only a single tag or multiple tags.

After creating the reader session, give instructions to the user by setting the alertMessage property. For example, you might tell users, “Hold your iPhone near the item to learn more about it.” The system displays this message to the user while the phone is scanning for NFC tags. Finally, call begin() to start the reader session. This enables radio-frequency polling on the phone, and the phone begins scanning for tags.

The sample app starts a reader session when the user taps the Scan button. The app configures the reader session to invalidate the session after reading the first tag. To read additional tags, the user taps the Scan button again.

```
@IBAction func beginScanning(_ sender: Any) {
    guard NFCNDEFReaderSession.readingAvailable else {
        let alertController = UIAlertController(
            title: "Scanning Not Supported",
            message: "This device doesn't support tag scanning.",
            preferredStyle: .alert
        )
        alertController.addAction(UIAlertAction(title: "OK", style: .default, handle
        self.present(alertController, animated: true, completion: nil)
        return
    }

    session = NFCNDEFReaderSession(delegate: self, queue: nil, invalidateAfterFirstF
    session?.alertMessage = "Hold your iPhone near the item to learn more about it.'
    session?.begin()
}
```

Adopt the Reader Session Delegate Protocol

The reader session requires a delegate object that conforms to the NFCNDEFReaderSession Delegate protocol. Adopting this protocol allows the delegate to receive notifications from the reader session when it:

- Reads an NDEF message
- Becomes invalid due to ending the session or encountering an error

```
class MessagesTableViewController: UITableViewController, NFCNDEFReaderSessionDelega
```

Read an NDEF Message

Each time the reader session retrieves a new NDEF message, the session sends the message to the delegate by calling the `readerSession(_:didDetectNDEFs:)` method. This is the app's opportunity to do something useful with the data. For instance, the sample app stores the message so the user can view it later.

```
func readerSession(_ session: NFCNDEFReaderSession, didDetectNDEFs messages: [NFCNDE
    DispatchQueue.main.async {
        // Process detected NFCNDEFMessage objects.
        self.detectedMessages.append(contentsOf: messages)
        self.tableView.reloadData()
    }
}
```

Handle an Invalid Reader Session

When a reader session ends, it calls the delegate method `readerSession(_:didInvalidateWithError:)` and passes in an error object that gives the reason for ending the session. Possible reasons include:

- The phone successfully read an NFC tag with a reader session configured to invalidate the session after reading the first tag. The error code is `NFCReaderError.Code.readerSessionInvalidationErrorFirstNDEFTagRead`.
- The user canceled the session, or the app called `invalidate()` to end the session. The error code is `NFCReaderError.Code.readerSessionInvalidationErrorUserCanceled`.
- An error occurred during the reader session. See `NFCReaderError.Code` for the complete list of error codes.

In the sample app, the delegate displays an alert when the reader session ends for any reason other than reading the first tag during a single-tag reader session, or the user canceling the session. Also, you cannot reuse an invalidated reader session, so the sample app sets `self.session` to `nil`.

```
func readerSession(_ session: NFCNDEFReaderSession, didInvalidateWithError error: Er
    // Check the invalidation reason from the returned error.
    if let readerError = error as? NFCReaderError {
        // Show an alert when the invalidation reason is not because of a
        // successful read during a single-tag read session, or because the
        // user canceled a multiple-tag read session from the UI or
        // programmatically using the invalidate method call.
        if (readerError.code != .readerSessionInvalidationErrorFirstNDEFTagRead)
            && (readerError.code != .readerSessionInvalidationErrorUserCanceled) {
            let alertController = UIAlertController(
                title: "Session Invalidated",
                message: error.localizedDescription,
                preferredStyle: .alert
            )
            alertController.addAction(UIAlertAction(title: "OK", style: .default, ha
            DispatchQueue.main.async {
                self.present(alertController, animated: true, completion: nil)
            }
        }
    }

    // To read new tags, a new session instance is required.
    self.session = nil
}
```

Write an NDEF Message

To write to a tag, the sample app starts a new reader session. This session must be active to write an NDEF message to the tag, so this time, `invalidateAfterFirstRead` is set to `false`, preventing the session from becoming invalid after reading the tag.

```
@IBAction func beginWrite(_ sender: Any) {
    session = NFCNDEFReaderSession(delegate: self, queue: nil, invalidateAfterFirstR
    session?.alertMessage = "Hold your iPhone near an NDEF tag to write the message.
    session?.begin()
}
```

When the reader session detects a tag, it calls the `readerSession(_:didDetectNDEFs:)` delegate method. However, because the session doesn't become invalid after reading the first tag, it's possible for the session to detect more than one tag. The sample app writes to one tag only, so it checks that the session detected only one tag. If the session detected more than one, the app asks the user to remove the tags, and then restarts polling to scan for a new tag.

After the app confirms that it has only one tag, it connects to the tag and verifies that it's writable. The app then writes the NDEF message it read earlier to the tag.

```
func readerSession(_ session: NFCNDEFReaderSession, didDetect tags: [NFCNDEFTag]) {
    if tags.count > 1 {
        // Restart polling in 500 milliseconds.
        let retryInterval = DispatchTimeInterval.milliseconds(500)
        session.alertMessage = "More than 1 tag is detected. Please remove all tags
        DispatchQueue.global().asyncAfter(deadline: .now() + retryInterval, execute:
            session.restartPolling()
        })
        return
    }

    // Connect to the found tag and write an NDEF message to it.
    let tag = tags.first!
    session.connect(to: tag, completionHandler: { (error: Error?) in
        if nil != error {
            session.alertMessage = "Unable to connect to tag."
            session.invalidate()
            return
        }

        tag.queryNDEFStatus(completionHandler: { (ndefStatus: NFCNDEFStatus, capacit
            guard error == nil else {
                session.alertMessage = "Unable to query the NDEF status of tag."
                session.invalidate()
                return
            }

            switch ndefStatus {
            case .notSupported:
                session.alertMessage = "Tag is not NDEF compliant."
                session.invalidate()
            case .readOnly:
                session.alertMessage = "Tag is read only."
                session.invalidate()
            case .readWrite:
                tag.writeNDEF(self.message, completionHandler: { (error: Error?) in
                    if nil != error {
                        session.alertMessage = "Write NDEF message fail: \(error!)"
                    } else {
                        session.alertMessage = "Write NDEF message successful."
                    }
                    session.invalidate()
                })
            @unknown default:
                session.alertMessage = "Unknown NDEF tag status."
                session.invalidate()
            }
        })
    })
}
```

Support Background Tag Reading

To learn how to set up your app to process tags that iOS reads in the background, see Adding Support for Background Tag Reading.

See Also

Essentials

Adding Support for Background Tag Reading

Allow users to scan NFC tags without an app using background tag reading.

`NFCReaderUsageDescription`

A message that tells people why the app is requesting access to the device's NFC hardware.