NFC Ink Screen SDK Integration Documentation

1. Introduction

This SDK allows communication with the NFC ink screen via NFC tags. The device scans NFC tags and transmits the data from the tags to the ink screen to trigger content updates. only the raw tag data is read and used to update the screen.

- **Supported Platform**: iOS 13.0 and above
- **Development Tools**: Xcode 14 or later
- **Hardware Requirements**: iPhone 12 and later, with NFC capabilities

2. Environment Configuration

2.1 Environment Requirements

- **Operating System**: iOS 13.0 and above
- **Development Tools**: Xcode 14 or later
- **Hardware Requirements**: iPhone 12 and above with NFC capabilities

2.2 SDK Installation

1. **Install via Framework**:
    - Download the provided SDK framework files.
    - Drag and drop the framework files into the `Frameworks` directory in your Xcode project.
    - In **Build Settings**, make sure the framework path is correctly configured.

2.3 Permissions Configuration

In the `Info.plist`, add the NFC permissions:

```
Plain Text
```

```
<key>Privacy - NFC Scan Usage Description</key>
  <string>This app requires NFC access to read tags and update the ink screen content.</string>
<key>com.apple.developer.nfc.readersession.felica.systemcodes</key>
  <array>
    <string>12FC</string>
  </array>
<key>com.apple.developer.nfc.readersession.iso7816.select-identifiers</key>
  <array>
    <string>D2760000850101</string>
  </array>
```

2.4 Configure Near Field Communication Tag Reader Session Formats (via Xcode Signing & Capabilities)

To configure the NFC tag reader session formats, follow these steps:

1. **Open your project in Xcode**.
2. **Navigate to the "Signing & Capabilities" tab** for your target.
3. **Click the "+ Capability" button** to add a new capability.

4. **Search for NFC** in the list of available capabilities.
5. **Select NFC** from the search results. This will automatically add NFC support to your project.
6. **Verify that an `.entitlements` file** has been created in your project. This file contains the necessary configurations, including the NFC tag reader session formats.
7. Inside the `.entitlements` file, the following configuration will be added::

```Plain Text
<key>Near Field Communication Tag Reader Session Formats</key>
<array>
    <string>Tag-Specific Data Protocol</string>
</array>
```

This ensures that NFC functionality is enabled and the device can interact with NFC tags using the **Tag-Specific Data Protocol (TAG)**.

8. In "**General**" -> "**Frameworks, Libraries, and Embedded Content**" when adding the InkScreenSDK.framework file, select "**Embed & Sign**"

3. Using the SDK

This section explains how to use the NFC Ink Screen SDK, from initialization to handling images, device information, and screen projections.

3.1 SDK Initialization

To begin using the SDK, initialize it via the singleton method:

```Objective-C
[NFCInkScreen shareInstance]
```

When initializing, you can set a block for handling device configuration mismatches. If the detected device does not match the configuration passed, the SDK will invoke this block, allowing you to read new device information or take other actions.

Additionally, the SDK provides a `bShowLog` variable to control whether logs should be printed for debugging purposes. When set to `YES`, you will see logs related to the data flow, which can help with integration and troubleshooting.

```
Plain Text
```

```
NFCInkScreen *inkScreen = [NFCInkScreen shareInstance];
inkScreen.bShowLog = YES;  // Enable log printing for debugging
inkScreen.nInterval = 20;  // waiting time  (0 - 60)s
inkScreen.handleDeviceCfgNotmatch = ^() {
    // Handle device configuration mismatch, e.g., update device info or take necessary actions
    NSLog(@"Device configuration mismatch");
};
// Set the compression method
inkScreen.compressBlock = ^(unsigned char * _Nonnull inData, size_t inLen, unsigned char * _Nonnull outData, size_t * _Nonnull outLen) {
    [FMCompressUtils FMCompressData:inData inLen:inLen outData:outData outLen:outLen];
};
```

3.2 Generate Image Data for NFC

To generate image data based on input images (such as content to be displayed on the ink screen), use the following method. It takes an input image and a preview image, then generates the corresponding image data for the screen.

```
Plain Text
```

```
- (NSData *)convertToNFCData:(UIImage *)inputImage
                previewImage:(UIImage * _Nonnull * _Nullable)previewImage
                   deviceCfg:(InkScreenSDKConfig *)deviceCfg;
```

This method allows you to convert an image into data that the ink screen can display. Additionally, it generates a preview image that can be used for displaying a preview before sending the final image.

3.3 Send Image Data to the Ink Screen

Once the image data is generated, you can send it to the ink screen for projection. Pass the image data, the device's UUID, and success/failure callbacks.

```
- (void)sendImageData:(NSData *)imageData
                uuID:(NSString *)uuID
             success:(SuccessBlock)success
             failure:(FailedBlock)failure;
```

If the UUID of the detected device does not match the provided UUID, the SDK will call the `handleDeviceCfgNotmatch` block. This allows you to handle mismatches and take necessary actions, such as updating the device information or retrying the operation.

3.4 Get Device Information

To retrieve device information (such as the device version and screen dimensions), use the following method:

```
- (void)getDeviceInfo:(SuccessBlock)success
              failure:(FailedBlock)failure;
```

This method will retrieve the device's information and pass it back through the success or failure block. If you need to close the session immediately after retrieving device info, you can use the following method to display a prompt and decide whether to end the session:

3.5 Display Alert and Handle Session End

To display alert messages and handle session termination, use the following method:

```
- (void)showAlertMsg:(NSString *)message
    invalidaSession:(BOOL)isInvalidaSession;
```

This method will display a prompt and allow you to choose whether to end the session. After retrieving the device information, if you need to continue the screen projection, you can use the previously obtained device information to generate image data and continue projecting the screen. If you need to end the session, you can pass `YES` to end the session.

4. Examples

4.1 Initialize the SDK and Set Device Configuration Mismatch Callback

After initializing the SDK, you can set the `handleDeviceCfgNotmatch` callback to handle cases where the device configuration does not match the passed configuration. If the current ink screen device does not match the configuration, the SDK will trigger this callback.

```
NFCInkScreen *inkScreen = [NFCInkScreen shareInstance];
inkScreen.bShowLog = YES;  // Enable logging for debugging
inkScreen.nInterval = 20;  // Since frequent use of the NFC module may cause overheating and lead to screen refresh failures, you can set a refresh interval, with a default of 20 seconds.

inkScreen.handleDeviceCfgNotmatch = ^() {
    // Handle device configuration mismatch

    // Update device information or take other necessary actions here
    [inkScreen getDeviceInfo:^(id  _Nonnull obj) {
        InkScreenSDKConfig *sdkConfig = (InkScreenSDKConfig *)obj;
        self.deviceCfg = sdkConfig;

        //A. If you want to end the session.
        [inkScreen showAlertMsg:@"You can read device information here and continue screen mirroring" invalidaSession: true];

        //B. If you want to continue projecting the screen.
        NSData *imageData = [inkScreen convertToNFCData:inputImage previewImage:&previewImage deviceCfg:sdkConfig];
        // Send the image data to the ink screen
        [inkScreen sendImageData:imageData uuID:deviceUUID success:^(id  _Nonnull obj) {
             NSLog(@"Image sent successfully!");
        } failure:^(id  _Nonnull error) {
            NSLog(@"Image send failed with error: %@", error.localizedDescription);
        }];
    } failure:^(id  _Nonnull error) {

    }];
};

// Set the compression method
inkScreen.compressBlock = ^(unsigned char * _Nonnull inData, size_t inLen, unsigned char * _Nonnull outData, size_t * _Nonnull outLen) {
    [FMCompressUtils FMCompressData:inData inLen:inLen outData:outData outLen:outLen];
};
```

4.2 Retrieve Device Information

After retrieving the device information using the `getDeviceInfo` method, you can choose to either continue projecting the screen or end the session. **After retrieving the device information, it is recommended to save this information for future use. The next time the device is detected, you can directly use the saved information. If a device mismatch is detected, you can re-read the device information.**

Plain Text

```
[inkScreen getDeviceInfo:^(id  _Nonnull obj) {
    // Successfully retrieved device information, continue projection or take other actions
    NSLog(@"Device Info: %@", obj);

    // After retrieving the device information, it is recommended to save this information for future use. The next time the device is detected, you can directly use the saved information. If a device mismatch is detected, you can re-read the device information.

    // Show an alert and choose whether to continue the projection
    [inkScreen showAlertMsg:@"Device info retrieved, do you want to continue projection?" invalidaSession:YES];

} failure:^(NSError *error) {
    // Failed to retrieve device information
    NSLog(@"Failed to retrieve device info, error: %@", error.localizedDescription);
}];
```

4.3 Generate Image Data and Send to Ink Screen

The following code demonstrates how to use the `convertToNFCData` method to generate image data, then send it to the ink screen using the `sendImageData` method.

```
// Input image to be processed
UIImage *inputImage = [UIImage imageNamed:@"example_image.png"];

// Due to the difference between the screen pixel ratio and the actual screen width-to-height ratio, this operation simulates a scenario where a user selects an image and then edits it. When the user selects an image to be cast, editing may be required. In this case, a canvas with screenWidth and screenHeight can be generated to hold the image. Once the editing is complete, the image data can be processed and cast to the screen.
  if (self.deviceCfg.ratio != 0x00) {
      inputImage = [self renderImageInAspectRatio:inputImage targetSize:CGSizeMake(self.deviceCfg.screenWidth, self.deviceCfg.screenHeight)];
  }

// Generate image data and preview image
UIImage *previewImage = nil;
NSData *imageData = [inkScreen convertToNFCData:inputImage previewImage:&previewImage deviceCfg:deviceCfg];

// Device UUID (assumed to be obtained through getDeviceInfo)
NSString *deviceUUID = @"123e4567-e89b-12d3-a456-426614174000";

// Send the image data to the ink screen
[inkScreen sendImageData:imageData uuID:deviceUUID success:^(id  _Nonnull obj) {
    NSLog(@"Image sent successfully!");
} failure:^(id  _Nonnull error) {
    NSLog(@"Image send failed with error: %@", error.localizedDescription);
    if ([error isKindOfClass:[NSDictionary class]]) {
        if ([error[@"code"] integerValue] == 1203) {
            NSLog(@"Please wait %@ seconds", error[@"value"]);
        }
    }
}];
```

During prolonged NFC communication, the device may overheat, leading to performance degradation and potential screen refresh failures. To ensure a better user experience, you can set a waiting period after each screen casting session. Waiting for approximately 20 seconds can improve the success rate of screen refresh and better protect the user's phone.

4.4 nternationalization of Prompts

The SDK also provides the following methods for internationalization handling. These methods allow you to set messages that will be displayed when switching between languages:

setNearThePhoneMessage("Place NFC close to the back of the phone");
setLowOSVersionMessage("The current iOS version of the phone is too low and does not support NFC");

setModelNotSupportMessage("This model does not support NFC");

Notes

1. **Compatibility**
   Ensure that the device you're testing with supports NFC. This SDK requires iPhone 12 or later, with iOS 13 or later, and Xcode 14 or later for development.
2. **Device Information Management**
   When retrieving device information, it's recommended to store the device configuration data for future use. This will allow you to reuse the configuration information without having to retrieve it again, unless a mismatch occurs.
3. **Handling Device Mismatches**
   When a device mismatch occurs (e.g., when the UUID does not match), the `handleDeviceCfgNotmatch` block is triggered. Be sure to implement this block to handle the mismatch situation, such as re-reading device information or taking corrective action.
4. **Logging for Debugging**
   The SDK includes a `bShowLog` variable to control whether logs are displayed. It is recommended to enable logging during development and debugging to help identify and resolve issues. Be sure to disable it in production to avoid unnecessary logging.
5. **Memory Management**
   When using large image data (such as the images you will generate using `convertToNFCData`), ensure that memory is managed properly. Release unused objects promptly to avoid memory leaks.
6. **Multilingual Support**
   The SDK provides methods for setting language-specific messages. Make sure to update all relevant messages when switching languages to ensure that your app provides an optimal user experience.
7. **Error Handling**
   Always implement the success and failure callbacks properly. If an operation fails, ensure

that meaningful error messages are displayed to users to guide them through troubleshooting.

8. **Testing on Real Devices**
   Since NFC functionality requires real hardware, make sure to test on actual devices. Testing on simulators may not give accurate results for NFC-based operations.
9. **Session Management**
   Keep track of session states carefully. If a session is invalid or needs to be ended, use the `showAlertMsg` method to prompt the user and ask if they want to end the session or continue.
10. **Device Authentication**
    The SDK requires channel code verification. If the channel code does not match, screen casting will not be allowed. Please ensure the correctness of the channel code to avoid issues with screen casting.