Heartland

A Global Payments Company

Heartland iOS SDK Integrators Guide

Heartland

A **Global Payments** Company

Notice

Table of Contents

Notice……………………………………….2
History of revision………………………...4
SDK Configuration…..…...……………...5
Bluetooth Pairing………………………….5
Transaction Listener……………………...7
Transaction Processing…………………..8
OTA Updating………………………………11
Surcharge……..……………………………14
Troubleshooting……………………………16

Version	Author	Date	Revisions
1.0	Shane Logsdon	May 18, 2021	Initial documentation release
1.1	Shane Logsdon	May 21, 2021	Add timeout reversal example
1.2	Brandon Stojak	Apr 11, 2023	Add custom ClientTxnID
1.3	Francis Legaspi	Jun 22, 2024	Add OTA Update
1.4	David Frahm	Nov 18, 2024	Add Surcharge
1.5	David Frahm	Feb 6, 2025	Add Surcharge Custom Rate

SDK Configuration

While the SDK manages the communication with the device and the gateway, neither it nor the device maintains a copy of the merchant account credentials used, so these will need to be set by your application at runtime. Below are code snippets for SDK imports to be used as well as configuring the SDK with credentials:

Note - Please use your own Heartland test account information in place of the x's below in the sample (username, password, licenseID, siteID, and deviceID):

```
// Used Imports example code

#import <Heartland_iOS_SDK/Heartland-iOS-SDK-umbrella.h>
#import <Heartland_iOS_SDK/Heartland_iOS_SDK-Swift.h>

// Configuration example code

HpsConnectionConfig *config = [[HpsConnectionConfig alloc] init];
config.versionNumber = @"3409";
config.developerID = @"002914";
config.username = @"xxxxxxxxx";
config.password = @"xxxxxxxxx";
config.siteID = @"xxxxxx";
config.deviceID = @"xxxxxx";
config.licenseID = @"xxxxxx";

self.device = [[HpsC2xDevice alloc] initWithConfig:config];
```

Bluetooth Pairing

Before creating a bluetooth connection with the device, your application requires some additional code in the form of a device listener / delegate object. Here's a brief description of each of the available methods and how they can assist in your application development:

- `onBluetoothDeviceList` - Once the bluetooth scanning process completes, this method will be called by the SDK, signaling that no additional devices are currently available. The user should select one of the available devices or check their settings before initiating another bluetooth scan.
- `onConnected` - This method will be called by the SDK once a successful bluetooth connection has been established with the selected device.

- `onDisconnected` - This method will be called by the SDK once a bluetooth connection has been closed with the selected device.
- `onError` - This method will be called by the SDK if any errors occur during the bluetooth connection process.
- `onTerminalInfoReceived` - This method may be called by the SDK if any information about the device is available.

Your application will need to initiate the bluetooth scanning process by calling the `initialize` method on the `device` object. Any progress along the way will be indicated by the SDK calling the appropriate method on your application's device listener / delegate object.

Once the user selects their correct bluetooth device, your application will need to update the SDK by calling the `connectDevice` method on the `device` object, and this method will require an object obtained by the `onBluetoothDeviceList` method.

```
// Device Bluetooth Connection example code

@interface MyDeviceDelegate : NSObject<HpsC2xDeviceDelegate>
@end

@implementation MyDeviceDelegate

- (void)onBluetoothDeviceList:(NSMutableArray *)peripherals {
    // receive list of available bluetooth devices
    // present to user for selection/confirmation

    // connect to the selected device
    [self.device connectDevice:[peripherals objectAtIndex:i]];
}
- (void)onConnected {
    // device is connected
    // allow processing
}
- (void)onDisconnected {
    // device is not connected
    // do not allow processing
}
- (void)onError:(NSString*)deviceError {
    // handle errors
}
```

```
    - (void)onTerminalInfoReceived:(HpsTerminalInfo *)terminalInfo {
        // handle received device information
    }

    @end


    device.deviceDelegate = [[MyDeviceDelegate alloc] init];
    // initiate a bluetooth scan
    [device initialize];
```

Transaction Listener

Before starting a transaction request, your application requires some additional code in the form of a transaction listener / delegate object. Here's a brief description of each of the available methods and how they can assist in your application development:

- `onStatusUpdate` - As the EMV process occurs, this method will be called by the SDK when the device may need to prompt the user with information about the current status. Any status update received by this method should be displayed to the user.
- `onConfirmApplication` - As the EMV process occurs, this method will be called by the SDK when the card holder needs to select an EMV application.
- `onConfirmAmount` - As the EMV process occurs, this method will be called by the SDK when the card holder needs to confirm the final authorization amount.
- `onTransactionCompleted` - Once the authorization request has been submitted to the gateway, this method will be called by the SDK, passing the authorization response to your application.
- `onTransactionCancelled` - This method will be called by the SDK if the transaction is cancelled by the device at any point.
- `onError` - This method will be called by the SDK if any errors occur during the transaction process.

```
    // Transaction Response Handling example code

    @interface MyTransactionDelegate :
NSObject<HpsC2xTransactionDelegate>
    @end

    @implementation MyTransactionDelegate
    - (void)onStatusUpdate:(HpsTransactionStatus)transactionStatus {
        // inform user of status updates
    }
    - (void)onConfirmApplication:(NSArray<AID *> *)applications {
        // prompt user to select desired application
        [self.device confirmApplication:[applications
objectAtIndex:0]];
    }
    - (void)onConfirmAmount:(NSDecimal)amount {
        // prompt user to confirm final amount
        [self.device confirmAmount:amount];
    }
    - (void)onTransactionComplete:(HpsTerminalResponse *)response {
        // handle response
    }
    - (void)onTransactionCancelled {
        // handle cancellation
    }
    - (void)onError:(NSError *)emvError {
        // handler errors
    }

    @end

    device.transactionDelegate = [[MyTransactionDelegate alloc]
init];
```

Transaction Processing

Note: Required device kernel and firmware version: ANHE_Kernel 4.3g_v8, and its firmware version is 1.00.03.45

Your application will need to initiate the transaction request process by creating the correct request builder, preparing any required data / information, and executing the builder. Any progress along the way will be indicated by the SDK calling the appropriate method on your application's transaction listener / delegate object.

```
    // Initiate Transaction - CreditAuth - Manual Entry example code

    HpsCreditCard *card = [[HpsCreditCard alloc] init];
    card.cardNumber = @"424242******4242";
    card.expMonth = 12;
    card.expYear = 2025;
    card.cvv = @"123";

    HpsC2xCreditAuthBuilder *builder = [[HpsC2xCreditAuthBuilder
alloc] initWithDevice:device];
    builder.amount = [[NSDecimalNumber alloc] initWithDouble:10.00];
    builder.creditCard = card;
    [builder execute];

    // Initiate Transaction - CreditAuth - Dip/Swipe example code

    HpsC2xCreditAuthBuilder *builder = [[HpsC2xCreditAuthBuilder
alloc] initWithDevice:device];
    builder.amount = [[NSDecimalNumber alloc] initWithDouble:10.00];
    [builder execute];

    // Initiate Transaction - CreditAddToBatch - example code

    HpsC2xCreditCaptureBuilder *builder =
[[HpsC2xCreditCaptureBuilder alloc] initWithDevice:device];
    builder.amount = [[NSDecimalNumber alloc]initWithDouble:10.00];
    builder.transactionId = transactionId;
    [builder execute];

    // Initiate Transaction - CreditSale - Manual Entry example code

    HpsCreditCard *card = [[HpsCreditCard alloc] init];
    card.cardNumber = @"424242******4242";
    card.expMonth = 12;
    card.expYear = 2025;
    card.cvv = @"123";

    HpsC2xCreditSaleBuilder *builder = [[HpsC2xCreditSaleBuilder
alloc] initWithDevice:device];
    builder.amount = [[NSDecimalNumber alloc] initWithDouble:10.00];
    builder.creditCard = card;
    [builder execute];
    // Initiate Transaction - CreditSale - Dip/Swipe example code

    HpsC2xCreditSaleBuilder *builder = [[HpsC2xCreditSaleBuilder
alloc] initWithDevice:device];
    builder.amount = [[NSDecimalNumber alloc] initWithDouble:10.00];
    [builder execute];

    // Initiate Transaction - Tip Adjust - example code

    HpsC2xCreditAdjustBuilder *builder = [[HpsC2xCreditAdjustBuilder
alloc] initWithDevice:device];
    builder.amount = [[NSDecimalNumber alloc]initWithDouble: 12.00];
    builder.gratuity = [[NSDecimalNumber alloc]initWithDouble: 2.00];
    builder.transactionId = transactionId;
    [builder execute];

    // Initiate Transaction - CreditReturn by Transaction ID -
example code

    HpsC2xCreditReturnBuilder *builder = [[HpsC2xCreditReturnBuilder
alloc] initWithDevice:device];
    builder.amount = [[NSDecimalNumber alloc]initWithDouble:1.00];
    builder.transactionId = transactionId;
    [builder execute];

    // Initiate Transaction - CreditReversal - Timeout Reversal -
example code

    HpsC2xCreditReversalBuilder *b = [[HpsC2xCreditReversalBuilder
alloc] initWithDevice:self.device];
    b.clientTransactionId = originalResponse.clientTransactionIdUUID;
    b.reason = ReversalReasonCodeTIMEOUT;
    b.amount = builder.amount;
   [builder execute];

    // Initiate Transaction - CreditVoid - example code

    HpsC2xCreditVoidBuilder *builder = [[HpsC2xCreditVoidBuilder
alloc] initWithDevice:device];
    builder.transactionId = transactionId;
    [builder execute];
```

Custom Client Transaction ID

To use your own Client Transaction ID, set the value that you want by using the setReferenceNumber method of the builder, as shown below. If you do not set your own reference number, a transaction ID will be generated automatically.

```
//Set the reference number
builder.clientTransactionId(yourClientTxnId);
```

Once the transaction is complete, you will see the same value returned inside the TerminalResponse object parameter of onTransactionComplete.

OTA Updating

The SDK enables over-the-air (OTA) updates for the C2X firmware and configuration. To check and install updates, you must establish a Bluetooth connection with the device. First, your application should request the available versions updates of either the firmware or configuration.

```
import Foundation
import Heartland_iOS_SDK
import UIKit

class ViewController: UIViewController {
    var device: HpsC2xDevice? {
        didSet {
            device?.otaFirmwareUpdateDelegate = self
        }
    }

    @IBAction func checkForUpdatesButtonAction(_: Any) {
       device.requestTerminalVersionData()
    }
}
```

ViewController class must conform to the **GMSDeviceFirmwareUpdateDelegate** to handle responses from the device.

The code triggers the **onTerminalVersionDetails** callback, which retrieves and displays the current firmware version installed on the device.

```
var currentFirmwareVerion: String = "" {
    didSet {
        device?.getAllVersionsForC2X()
    }
}

extension ViewController: GMSDeviceFirmwareUpdateDelegate {
     func onTerminalVersionDetails(info: [AnyHashable: Any]?) {
        if let info = info, let firmwareVersion = info["firmwareVersion"],
           let stringFirmwareVersion = firmwareVersion as? String
        {
            print("Info: \(stringFirmwareVersion)")
            self.currentFirmwareVerion = stringFirmwareVersion
        }
    }
}
```

When the code triggers the **onTerminalVersionDetails** callback, it retrieves and displays the latest firmware version.

```
var lastFirmwareVersion: String = "" {
    didSet {
        self.compareVersions()
     }
}

extension ViewController: GMSDeviceFirmwareUpdateDelegate {
  func listOfVersionsFor(results: [Any]?) {
      if let results = results,
      let lastVersion = results.reversed().first as? [String: AnyObject],
      let firmwareVersion = lastVersion.first?.value {
          lastFirmwareVersion = firmwareVersion as? String ?? ""
          print(firmwareVersion)
      }
  }
}
```

We can then now then compare the **currentFirmwareVerion** and **lastestFirmwareVersion**

```
func compareVersions() {
  let versionCompare = currentFirmwareVerion
                          .compare(lastFirmwareVersion, options: .numeric)
  if versionCompare == .orderedAscending {
       // There is a new update
  } else {
       // No new update
   }
}
```

Assuming there is a new update, we can request update to C2X, supplying the latest firmware version

```
@IBAction func updateFirmwareButtonAction(_: Any) {
   device.setVersionDataFor(versionString: lastFirmwareVersion)
   device.requestUpdateVersionForC2X()
}
```

We can also handle the callbacks of the device in these functions

```
extension ViewController: GMSDeviceFirmwareUpdateDelegate {
   func terminalOTAResult(resultType: TerminalOTAResult,
                          info _: [String: AnyObject]?,
                          error _: Error?) {}

   func otaUpdateProgress(percentage: Float) {}

   func onReturnSetTargetVersion(message _: String) {}
}
```

Surcharge

Sample App: See the sample Xcode project in the SDK GitHub repo for a working example.

With surcharge enabled, a 3% charge will be added to the total for sale/auth transactions that use a surcharge-eligible credit card. No charge is added if the eligibility is unable to be detected.

Surcharge is supported for specific transaction types: credit sale, credit authorization, and manual card entry.

To enable surcharge, you need to configure the transaction builder.

```
let builder = HpsC2xCreditAuthBuilder(device: device)
builder.isSurchargeEnabled = true
...Operations for credit sale, credit auth, manual card entry
builder.execute()
```

Optional: Custom Surcharge Rate
When enabled, the surcharge rate defaults to 3%. This can be overridden, such as in the case where local limits require a surcharge rate of less than 3%. The overriding value must be 2% or greater, and less than 3%.

```
builder.surchargeFee = NSDecimalNumber(string: "2.75")
```

With surcharge enabled, an approved payment transaction response includes surcharge information.

Surcharge Fee (ie, surcharge %)

```
let surchargeFee = (Decimal(string: response.surchargeFee ?? "0") ?? 0) * 100
```

Surcharge Amount

```
let surchargeAmount = NSDecimalNumber(string: response.surchargeAmount ?? "0")
```

Surcharge Eligibility: Y/N/U (Yes, No, Unknown)

```
if case response.surchargeRequested = SurchargeEligibility.Y { }
if case response.surchargeRequested = SurchargeEligibility.N { }
if case response.surchargeRequested = SurchargeEligibility.U { }
```

Note: Eligibility of Unknown should be rare; it exists to ensure that the core payment transaction is not rejected in any scenario where eligibility cannot be fully verified.

Surcharge requires that the cardholder is given the opportunity to approve/decline this charge. This is supported via a cardholder interaction type of `surchargeRequested`. This interaction request will occur when surcharge is enabled and the card was determined to be a credit card.

```
func onTransactionWaitingForSurchargeConfirmation(
    result: HpsTransactionStatus, response: HpsTerminalResponse
) {
    if result == .surchargeRequested, let builder = self.builder,
       let surchargeFee = response.surchargeFee {
        let alertController = UIAlertController(title: "Surcharge Confirmation Required",
                message: "There will be a \(surchargeFee) surcharge added to your purchase",
                preferredStyle: .alert)
        // Create the actions
        let okAction = UIAlertAction(
            title: "Accept",
            style: UIAlertAction.Style.default
        ) {
            UIAlertAction in
            NSLog("OK Pressed")
            self.device?.confirmSurcharge(true)
        }
        let cancelAction = UIAlertAction(
            title: "Decline",
            style: UIAlertAction.Style.cancel
        ) {
            UIAlertAction in
            NSLog("Cancel Pressed")
            self.device?.confirmSurcharge(false)
            self.showProgress(false)
        }
        // Add the actions
        alertController.addAction(okAction)
        alertController.addAction(cancelAction)
        // Present the controller
        self.present(alertController, animated: true, completion: nil)
    }
}
```

Troubleshooting

Device cannot be paired

- Please press the power on button to restart your device.
- Please check to see if you can find the device's "Serial Number" (Shown on the back of the device) in the "Scanned Device List" of your smartphone or tablet.

Device loses the connection with your smartphone or tablet when the device has gone into auto-off mode

- Please press the power on button to turn on the device again. The device will automatically
- connect with your smartphone or tablet again.
- The device may be at lower battery level, please use the USB cable to recharge it, then retry.
- Please ensure the device or smartphone/tablet is within the reception range.

Device does not work with your phone or tablet

- Please ensure the Bluetooth® function of your smartphone or tablet is turned on.
- Please check the version of your operating system is supported for this device's operation.

Device cannot read your card successfully

- Please press the power on button to turn on the device again.
- The device will automatically connect with your smartphone or tablet again.
- The device may be at lower battery level, please use the USB cable to recharge it, then retry.
- Please ensure the device or smartphone/tablet is within the reception range.
- Inserting card
  - Please check if the device has power when operating and ensure devices are connected.
  - Please follow the application instructions to insert or tap the card.
  - Please ensure that there is no obstacle in the card slots.
  - Please check if the chip of the card is facing the right direction when inserting the card.
  - Please ensure that your phone/ tablet is a supported model for this device's operation.
  - Please insert the card with a more constant speed.
- Tap Card/Mobile Wallet
  - Please check if your card supports NFC payment.

- Please ensure if your card is placed within 4 cm range on top of the NFC marking.
- Please take out your NFC payment card from your wallet or purse for payment to avoid any interference.

Device has no response

- Please use a paperclip to press the reset button at the bottom for reboot.