Methodology
Mobile API

Overview

There are two parts to the Mobile APIs, the end-to-end encryption method and the swipe device library.

End-to-End Encryption

The end-to-end encryption library allows credit card data to be encrypted on a mobile device before sending it to the Merchant's back-end server. During the sale process, the Merchant's server can send the encrypted card data to the Payment Gateway, where it is decrypted and treated like a normal credit card. This gives the merchant more control of mobile transactions without having to increase compliance costs.

The merchant's encryption key is an RSA public key that is unique to them. This means that the encrypted credit card data will only be able to be used to make a transaction in that merchant's payment gateway account. Only the Payment Gateway has access to the private key that corresponds to this public key.

Card data is encrypted using AES encryption with a new randomly generated key for every card. This key is then encrypted with the public key along with the card data. This packet (the encrypted card and AES key) is unreadable to anybody without the private key which is only known to the Payment Gateway.

Note: The public key cannot be used to decrypt an encrypted card. Once encrypted, the card is unusable except by the Gateway when it processes the payment for the merchant. For this reason, there is no need to keep the public key a secret.

Swipe Device Library

This library supports the encrypted card readers supported by the payment gateway. This includes parsing the data and notifying you when a card reader is connected, disconnected, ready to receive a swipe, etc.

Using the Library
Mobile API: Android

Creating a Project

The fastest way to get started is to check out the Client Encryption Example project that can be downloaded from the downloads section. Or if you prefer to create your own project, use these steps:

These directions are specific to the Eclipse IDE.

1. Download and extract the zip file from the integration section of the Payment Gateway.

2. In eclipse, select File -> New -> Project.
3. Import the SDK
    1. Right click on the libs directory in the new project, and select Import.
    2. Select File System underneath General, and click next.
    3. For the directory, click browse and inside of the extracted folder from before, select Payment Gateway Mobile SDK.
    4. Select all three Jar files and click finish.
4. Add SDK files to build path.
    1. Right click on the imported files and select Build Path -> Add to Build Path.
    2. Right click on the project and select Build Path -> Configure Build Path...
    3. In the Order and Export tab, select the SDK files.
    4. Click OK.

Network Usage Note

You may notice the library attempting to connect to IDTECH's website to download a file. Since the audio jack capabilities of different Android devices vary, the IDTECH Shuttle's library uses different communication settings for each supported device. IDTECH frequently updates a list of the supported devices and the communication settings for each which the library may attempt to download from IDTECH. Internet permission is required.

End-to-End Encryption Mobile API: Android

Acquiring a Public Key

a. After logging into the Payment Gateway, navigate to Settings -> Security Keys -> View Mobile SDK Key. You can click on the Java example button to get a version that can easily be copied and pasted into your project.

b. Use the Query API. In order to get the public key, you will need to use 'report_type=sdk_key'. The key will be returned in the <sdk_key> tag.

Encrypting a Card

The following is an example of the entire encryption process:

```
import com.SafeWebServices.PaymentGateway.PGEncrypt;
PGEncrypt pg = new PGEncrypt();
Pg. setKey(
    "***999|MIIEEjCCA3ugAwIBAgIBADANBgkqhkiG9w0BAQQFADCBvTELMAkGA1UEBh"
    "MCVVMxETAPBgNVBAgTCElsbGlub2lzMRMwEQYDVQQHEwpTY2hhdW1idXJnMRgwFg"
                    [Several lines omitted]
    "cNAQEEBQADgYEAKY8xYc91ESNeXZYTVxEsFA9twZDpRjSKShDCcbutgPlC0XcHUt"
    "a2MfFPsdgQoq0I8y1nEn1qJiOuEG1t9Uwux4GAvAPzsWSsKyKQkZhqxrxkJUB39K"
```

```
    "Pg57pPytfJnlQTgYiSrycCEVHdDvhk92X7K2cab3aVV1+j0rKlR/Sy6b4=***");
PGKeyedCard cardData = new PGKeyedCard(cardNumber, expiration, cvv);
Boolean includeCVV = true;
String encryptedCardData = pg.encrypt(cardData, includeCVV);
```

In this example, 'encryptedCardData' would now contain a string that can be passed to the Payment Gateway in place of credit card information. The parameter name to use when passing this value is 'encrypted_payment'.

For example, a simple DirectPost API string would look something like this:

(This example assumes your Merchant server is running a PHP script that has received the encrypted card data through a POST parameter called 'cardData'.)

```
//Business logic, validation, etc.  When ready to process the payment...
$cardData = $_POST['cardData'];
$postString = "username=demo&password=1234&type=sale&amount=1.00&encrypted_payme
//Post to Gateway
```

We suggest using POST instead of GET to reduce the possibility of the data being kept in a log file. For more information on how to communicate with the Payment Gateway, see the API documentation.

Swipe Devices
Mobile API: Android

Permissions

You will need to grant the application multiple permissions in order to use a swipe device. This can be done by modifying the manifest file by adding:

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
```

In the class that intends to handle swipe events, add a PGSwipeController property called swipeController, and then in your init function, initialize the object with this line:

```
//This example is for the iPS Encrypted Mobile Card ReaderswipeController = new I
```

If you want to change the default settings, you can change them now. Here are some examples:

```
swipeController.getDevice().setSwipeTimeout(30);
swipeController.getDevice().setAlwaysAcceptSwipe(false);
swipeController.getDevice().setActivateReaderOnConnect(false);
```

Your class will have to implement the PGSwipeListener protocol. If you are only interested in knowing when a card is swiped, you can safely leave every other event handler empty, as shown here (or add your own code to, for example, display an image indicating that the swipe reader is ready for a swipe). In this example, when the swipe is received, the card data is saved in a property (swipedCard) for eventual transmission to the Gateway

(not shown), and two TextView variables (cardNumberField and expirationField) are set to show the masked card number and expiration date. If a bad swipe occurs, onSwipedCard is still called, but "card" will be null.

```
@Override
public void onDeviceConnected(final PGSwipeDevice device)
{
}
@Override
Public void onDeviceDisconnected(final PGSwipeDevice device)
{
}
@Override
public void onDeviceActivationFinished(final PGSwipeDevice device)
{
}
@Override
public void onDeviceDeactivated(final PGSwipeDevice device)
{
}
@Override
public void onDeviceReadyForSwipe(final PGSwipeDevice device)
{
}
@Override
public void onDeviceUnreadyForSwipe(final PGSwipeDevice device,
    PGSwipeDevice.ReasonUnreadyForSwipe reason)
{
}
@Override
public void onSwipedCard(final PGSwipedCard card, final PGSwipeDevice device)
{
    if (card != null) {
      this.runOnUiThread(new Runnable() {
      public void run() {
          TextView cardNumberField = (TextView)findViewById(R.id.cardNumber);
        cardNumberField.setText((CharSequence)card.getMaskedCardNumber());
        }
      }
    } else {
        //A null card means that there was a swipe but it was unsuccessful.    }
}
```

Classes Overview
Mobile API: Android

PGEncrypt

The PGEncrypt class contains all necessary to encrypt data to be sent to the payment gateway. Merchants wanting to send transaction data to their servers before processing the transaction will want to use this method in order to prevent their server from touching sensitive data.

- void setKey(String key)

  This method takes in the public key and sets it to be used with the encrypt method.

- String encrypt(String plaintext)

  This method accepts a string to be encrypted. Although any string can be passed, the Payment Gateway will only pull fields related to credit cards from the encrypted text.

- String encrypt(PGCard card, boolean includeCVV)

  This is the preferred way of getting the encrypted card data. It will format and encrypt the card data for you to pass on to the gateway.

PGSwipeDevice

This class represents the functionality that is common to the swipe reader devices. A PGSwipeDevice object is passed along with every even generated by the devices in order to identify the device type and access device-specific features by casting it to the specific swipe device.

- enum ReasonUnreadyForSwipe {DISCONNECTED, TIMED_OUT, CANCELED, REFRESHING, SWIPE_DONE }

  Used to explain why the device can no longer accept a swipe.

- enum SwipeDevice { UNIMAG, IPS }

  Used to identify the type of device being used.

- boolean getIsConneced()

  Returns true if the swipe device is connected.

- boolean getIsActivated()

  Returns true if the swipe device is activated.

- boolean getIsReadyForSwipe()

  Returns true if the swipe device is ready.

- SwipeDevice getDeviceType()

  Returns the current device type.

- void setListener(SwipeListener value)

  Sets the event listener.

- boolean setSwipeTimeout(int seconds)

Sets the timeout interval for the swipe device.

- void setAlwaysAcceptSwipe(boolean alwaysAcceptSwipe)

  True by default, if this is set to false, a swipe must be requested once the device is ready.

- void setActivateReaderOnConnect(boolean activateReaderOnConnect)

  True by default, if this is to false, the device must be activated before it can be used.

- boolean requestSwipe()

  Notifies the reader to start waiting for a swipe. The device must be active before this can be called.

- void cancelSwipeRequest()

  Cancels a swipe request.

- void stopSwipeController()

  Cancels the current swipe request, unregisters the swipe device, and frees resources. Will not receive any information from the device until it is resumed.

- void restartSwipeController()

  Registers the swipe device. Should only be called after calling stopSwipeController()

- String getDefaultMsg()

  Returns the default message for the current device state.

PGSwipeIPS extends PGSwipeDevice

This class handles communications with the iPS Encrypted Mobile Card Reader.

- void InitializeReader(Context ctx)

  This class is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create a PGSwipeIPS instance to interact with the IPS device.

PGSwipeUniMag extends PGSwipeDevice

This class handles communication with the IDTECH Unimag device.

- void InitializeReader(Context ctw)

  This class is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create an instance of PGSwipeUniMag to interact with the Shuttle device.

- void updateCompatableDeviceList()

  The UNIMAG device uses an xml compatibility list that consists of specific device settings that are unique to every device. This function should be called to handle new devices.

PGCard

This is a simple base class for the different types of cards that can be used. There is no reason to ever explicitly declare this.

- void setCVV(String CVV)

  Sets the CVV for the credit card data.

- String getCVV()

  Returns the CVV for the card.

- String getDirectPostString(boolean includeCVV)

  Returns a query string consisting of the card data that can be passed to the Payment Gateway through the Direct Post API.

PGKeyedCard extends PGCard

This class should be used when accepting credit card information from a keyboard.

- PGKeyedCard(String ccnumber, String expiration, String cvv)

  The standard constructor for this class. It should be used most of the time.

- PGKeyedCard(String ccnumber, String expiration, String cvv, String startDate, String issueNum)

  This constructor accepts two more values that would be used for Maestro cards.

- void setCardNumber(String value)

  Sets the card number to be used for the current card.

- void setExpirationDate(String value)

  Sets the expiration date to be used for the current card.

- void setCardStartDate(String value)

  Sets the start date for the current card.

- void setCardIssueNumber(String value)

Sets the issue number for the current card.

- String getCardNumber()

  Returns the current card number.

- String getExpirationDate()

  Returns the current expiration date.

- String getCardStartDate()

  Returns the current start date.

- String getCardIssueNumber()

  Returns the current issue number.

PGSwipedCard extends PGCard

This class should only be used along with an unencrypted swipe device.

- PGSwipedCard(String track1, String track2, String track3, String cvv)

  The constructor that sets the card data accordingly.

- void setTrack1(String value)

  Sets track1 for the current card.

- void setTrack2(String value)

  Sets track2 for the current card.

- void setTrack3(String value)

  Sets track3 for the current card.

- void setMaskedCardNumber(String value)

  Sets the masked card number for the current card.

- void setCardholderName(String value)

  Sets the name on the current card.

- void setExpirationDate(String value)

  Sets the expiration date for the current card.

- String getTrack1()

Returns the track1 data.

- String getTrack2()

  Returns the track2 data.

- String getTrack3()

  Returns the track3 data.

- String getMaskedCardNumber()

  Returns the masked card number. This should be used when trying to display card information to the user.

- String getCardholderName()

  Returns the name on the card.

- String getExpirationDate()

  Returns the expiration date.

PGEncryptedSwipedCard extends PGSwipedCard

This class should be used for all encrypted swipe devices.

- PGEncryptedSwipedCard(String track1, String track2, String track3, String ksn, String cvv)

  The constructor accepts all class variables.

- void setKsn(String value)

  Sets the KSN that is used to decrypt the card information at the gateway.

- String getKsn()

  Returns the KSN.

PGSwipeController

The PGSwipeController class is used to maintain the swipe device.

- PGSwipeController(Object source, PGSwipeDevice.SwipeDevice deviceType)

  This constructor sets the type of device to be used and initializes it.

- PGSwipeDevice getDevice()

Returns the device that is currently initialized. Only one should be initialized at a time.

- PGSwipeUniMag getUnimag()

  Can be used instead of getDevice, will produce the same result as long as a UNIMAG device is being used.

- PGSwipeIPS getIPS()

  Can be used instead of getDevice, will produce the same result as long as an IPS device is being used'.

PGSwipeController.SwipeListener

This interface must be implemented in order to receive events from the card readers

- void onSwipedCard(PGSwipedCard card, PGSwipeDevice swipeDevice)

  Method called when a card is swiped. It accepts the card data and the device used.

- void onDeviceReadyForSwipe(PGSwipeDevice swipeDevice)

  Called when the device is ready to read a swipe.

- void onDeviceUnreadyForSwipe(PGSwipeDevice swipeDevice, PGSwipeDevice.ReasonUnreadyForSwipe reason)

  Is called when the device can no longer read a card. It is passed the device and the reason it can no longer accept a swipe.

- void onDeviceConnected(PGSwipeDevice swipeDevice)

  This method is called when the swipe device is connected.

- void onDeviceDisconnected(PGSwipeDevice swipeDevice)

  This method is called when the swipe device is unplugged from the android device.

- void onDeviceActivationFinished(PGSwipeDevice swipeDevice)

  This method is called when a swipe can be requested.

- void onDeviceDeactivated(PGSwipeDevice swipeDevice)

  This method is called when the device is stopped. Once this is called, the device has to be restarted to function again.

Using the Library
Mobile API: iOS

Creating a Project

The fastest way to get started is to check out the PaymentGatewayEncryptionExample and PaymentGatewaySwipeExample projects that can be downloaded from the Payment Gateway's Integration section. If you prefer to create your own project instead, use these steps (current as of Xcode 5.0):

1. Download the Mobile API .zip file from the Integration Portal by using the "Downloads" link under the Mobile API section. This file contains both the iOS and Android libraries.
2. Create a new Xcode Project.
3. Copy the files in the .zip file into your project folder, and add them to your Xcode project. The files you will need are PGMobileSDK.a and the entire folder PGMobileSDK containing the headers. These are found in the .zip file under Apple iOS -> Payment Gateway SDK.
4. Under the project's Build Phases settings, add these libraries to the Link Binary With Libraries section:
    - AudioToolbox.framework
    - AVFoundation.framework
    - ExternalAccessory.framework
    - MediaPlayer.framework
    - Security.framework
    - libstdc++.6.0.9.dylib
5. (Optional - see note below) - In your Info.plist, add a row for "Supported external accessory protocols", and add "com.gatewayprocessingservices.iprocess" as Item 0. This enables connection to the iDynamo swipe reader.

Note: You may wish to skip step 5 if you do not need to support the iDynamo. Apple requires manufacturers of accessories that use the dock connector to add your app to their product plan before approving your app for the app store. You will need to contact MagTek in order to have your app added to their product plan. Contact MagTek for more details.

Viewing documentation in Xcode

Adding the doc set to Xcode allows the most up-to-date, relevant documentation to appear in the IDE as you type. To enable access to the SDK documentation from inside Xcode:

1. Under the Xcode menu, click Preferences
2. Navigate to the Downloads page
3. On the Documentation tab, click Add.
4. On the "Enter a doc set feed URL" window that pops up, enter: https://secure.safewebservices.com/merchants/resources/integration/docset/iOSSDK.atom
5. Click Add
6. Click the newly-added install button

Important Info About the App Store

The Apple App Store's current policy is to require mobile apps to purchase digital goods (e.g. downloadable content, etc.) through the App Store. For that reason, this SDK is intended only for use in apps selling real-world goods and services. Please direct questions about Apple's App Store policies to Apple. Their policies are subject to change at their discretion.

End-to-End Encryption Mobile API: iOS

Acquiring a Public Key

a. After logging into the Payment Gateway, navigate to Settings->Security Keys->View Mobile SDK Key. You can click on the Objective-C example link to get a version that can easily be copied and pasted into your project.

b. Use the Query API. In order to get the public key, you will need to use 'report_type=sdk_key'. The key will be returned in the <sdk_key> tag.

Encrypting a Card

```
#import "PGEncrypt.h"
#import "PGCards.h"
PGEncrypt encryption = [[PGEncrypt alloc] init];
[encryption setKey:
    @"***999|MIIEEjCCA3ugAwIBAgIBADANBgkqhkiG9w0BAQQFADCBvTELMAkGA1UEBh"
    "MCVVMxETAPBgNVBAgTCElsbGlub2lzMRMwEQYDVQQHEwpTY2hhdW1idXJnMRgwFg"
                    [Several lines omitted]
    "cNAQEEBQADgYEAKY8xYc91ESNeXZYTVxEsFA9twZDpRjSKShDCcbutgPlC0XcHUt"
    "a2MfFPsdgQoq0I8y1nEn1qJiOuEG1t9Uwux4GAvAPzsWSsKyKQkZhqxrxkJUB39K"
    "Pg57pPytfJnlQTgYiSrycCEVHdDvhk92X7K2cab3aVV1+j0rKlR/Sy6b4=***"];
PGCard *cardData = [[PGKeyedCard alloc] initWithCardNumber:cardNumberField.text
                                            expirationDate:expirationField.text
                                                       cvv:cvvField.text];
NSString *encryptedCardData = [encryption encrypt:cardData includeCVV:NO];
```

encryptedCardData will contain a string that can be passed to the Payment Gateway in place of credit card information. The parameter name to use when passing this value through DirectPost is "encrypted_payment". For example, a simple DirectPost API string would look something like this:

(This example assumes your Merchant server is running a PHP script that has received the encrypted card data through a POST parameter called 'cardData'.)

```
//Business logic, validation, etc.  When ready to process the payment...
$cardData = $_POST['cardData'];
$postString = "username=demo&password=1234&type=sale&amount=1.00&encrypted_payme
```

```
//Post to Gateway
```

For more information on how to communicate with the Payment Gateway, see the API documentation.

Swipe Devices
Mobile API: iOS

Creating the Controller

In the class that intends to handle swipe events, create a PGSwipeController object in your init method. Initialize the object with this line to support Shuttle readers:

```
swipeController = [[PGSwipeController alloc] initWithDelegate:self audioReader:Au
```

or for the IPS Encrypted Card Reader:

```
swipeController = [[PGSwipeController alloc] initWithDelegate:self audioReader:Au
```

Only a single model of audio jack-connected reader can be enabled at a time. The audioReader parameter allows you to choose which type, UniMag (Shuttle) or IPS Encrypted Card Reader, you want to allow. See the PGSwipeController's initWithDelegate:audioReader: documentation for more details.

Your class will have to implement the PGSwipeDelegate protocol. If you are only interested in knowing when a card is swiped, you can safely leave every other event handler empty, as shown here (or add your own code to, for example, display an image indicating that the swipe reader is ready for a swipe). In this example, when the swipe is received, the card data is saved in a property (swipedCard) for eventual transmission to the Gateway (not shown), and two UITextField properties (cardNumberField and expirationField) are set to show the masked card number and expiration date.

If a bad swipe occurs, didSwipeCard:device: may still be called, but "card" will be nil. An error message is displayed in this example. Note: Not all card reader models give feedback when a bad swipe is received.

```
-(void)deviceConnected:(PGSwipeDevice *)sender
{
}
-(void)deviceDisconnected:(PGSwipeDevice *)sender
{
}
-(void)deviceActivationFinished:(PGSwipeDevice *)sender result:(SwipeActivationR
{
}
-(void)deviceDeactivated:(PGSwipeDevice *)sender
{
}
-(void)deviceBecameReadyForSwipe:(PGSwipeDevice *)sender
{
}
-(void)deviceBecameUnreadyForSwipe:(PGSwipeDevice *)sender reason:(SwipeReasonUn
{
}
```

```
-(void)didSwipeCard:(PGSwipedCard *)card device:(PGSwipeDevice *)sender
{
    if (card != nil) {
        swipedCard = [card retain];
        cardNumberField.text = card.maskedCardNumber;
        expirationField.text = card.expirationDate;
    } else {
        //A nil card means that there was a swipe but it was unsuccessful.
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Swipe Error"
                                                        message:@"The reader was
                                                       delegate:nil
                                              cancelButtonTitle:@"OK"
                                              otherButtonTitles:nil];
        [alert show];
        [alert release];
    }
}
```

Supported Devices

IPS Encrypted Card Reader

The IPS is an audio jack-connected card reader. Unlike the IDTECH Shuttle, the IPS is powered by an internal battery. The IPS has a fast startup time and does not produce a constant tone through the audio jack.

Because the IPS connects through the audio port and there is no way to immediately detect the device type, you will receive a deviceConnected: event even if the user has only plugged in headphones. Since there is no activation with the IPS, a deviceActivated: and deviceBecameReadyForSwipe: will also be sent immediately. In order to be sure that the device is an IPS reader, the PGSwipeIps provides a beginTestCommunication: method you can use to attempt to communicate with the device. If it returns success, the device is an IPS reader. This is not done by default to eliminate a delay before the device becomes active.

IDTECH Shuttle

The Shuttle (referred to in code as a UniMag device) is an audio jack-connected card reader. It is powered by a tone from the iPod / iPad / mobile phone. Before the Shuttle can receive swipes, it must be powered up.

Because the Shuttle connects through the audio port and there is no way to detect the device type until the device is activated, you will receive deviceConnected events whenever any device is attached to that port. For example, if the user attaches headphones, you will receive a connection event from the Mobile SDK.

The Mobile SDK can be configured to automatically attempt to power on the swipe reader immediately (this is the default), or you can disable the automatic activation and only activate the device when desired (e.g. on a payment screen, or when the user clicks a button).

Important: When powering on the device, the audio volume must be at maximum (done automatically by default). The tone generated through the audio port to activate the device can be very painful to a listener if they have connected speakers or headphones. For this reason, swipeController.uniMagReader.messageOptions.activateReaderWithoutPromptingUser is set to NO by default, causing the SDK to prompt the user for confirmation before activating the reader.

The Shuttle saves battery by only allowing swipes when a swipe has been requested, and a timeout occurs if a swipe is not received quickly enough (20 seconds by default). For simplicity, the SDK defaults to automatically requesting a swipe on activation and continuously renewing the swipe request. If you have issues with battery life, you can set swipeController.uniMagReader.alwaysAcceptSwipe to NO and manually call [swipeController.uniMagReader requestSwipe] when ready for a swipe.

iDynamo

The iDynamo connects to the mobile device via Apple's dock connector and is only compatible with iOS devices that use the older 30-pin (non-Lightning) dock connector.

When physically attached, the iDynamo is almost immediately ready to receive swipe events. When connected, the Swipe Delegate should expect a deviceConnected: message, immediately followed by a deviceActivationFinished: message, then a deviceBecameReadyForSwipe: message.

When the device is physically detached, the delegate receives the events in reverse order, i.e. deviceBecameUnreadyForSwipe:, deviceDeactivated:, deviceDisconnected:.

App Store: To support the iDynamo on an app distributed through the App Store, Apple may require you to contact MagTek for information before they will process your submission. To disable iDynamo support, do not add it to "Supported external accessory protocols" in your info.plist. You will still receive connect and disconnect events, but activation will fail, so be sure to check if the sending device is the iDynamo object and ignore it if so.

Known Issue with the iDynamo: There is an issue with device disconnection with the iDynamo and iOS's ExternalAccessory framework. Upon disconnection, the stream communicating with the device is closed, during which you may receive the warning: *[NSCondition dealloc]: condition (<NSCondition: 0x1d54ce90> '(null)') deallocated while still in use.* After reconnecting, a later disconnect may randomly cause the app to crash with an attempt to send a message to the deallocated instance. This does not occur frequently, and is more likely to occur when rapidly opening and closing the application (which sends a disconnect followed by a reconnect when the app re-opens). This issue is with Apple's accessory-handling framework. Apple is aware of the issue and may fix it in a future iOS release.

Classes Overview
Mobile API: iOS

PGSwipeController

The PGSwipeController contains a set of swipe reader classes that control individual swipe readers. This is the main Mobile Swipe SDK class required for using swipe devices, intended to be instantiated near the app's startup. The delegate you set on the PGSwipeController is the object that will receive all of the SDK's swipe events.

Through this class, you can access the controller classes for individual swipe device types (PGSwipeIps *ipsReader, PGSwipeIDynamo *iDynamoReader, and PGSwipeUniMag *uniMagReader).

You should be sure to call initWithDelegate rather than the parameterless init because during initialization a check is made to see if any devices are already connected and sends a deviceConnected event if they are. If the parameterless init is used, the initial connection message will be missed.

- -(id)initWithDelegate:(id<PGSwipeDelegate>)delegate audioReader:(AudioJackReaderType)readerType

  Initializes the PGSwipeController and the individual swipe reader classes. Init checks if any devices are connected and sends a deviceConnected event if they are, so initWithDelegate: should always be used rather than init to ensure that a connection event is received if the device is already connected.

  The audioReader: parameter selects which type of audio jack-connected card reader to enable. Only one type of audio jack-connected reader can be used at a time to prevent more than one device library from attempting to access the audio system at the same time. AudioJackReaderUnimag enables the Shuttle library, and AudioJackReaderIps enables the IPS Encrypted Card Reader library. You may also select AudioJackReaderNone to disable both libraries, or AudioJackReaderAutodetectOnConnect to allow the SDK to attempt to determine the type on connection. Autodetection has several drawbacks. See PGSwipeController beginAutodetectAudioJackCardReader for more information.

- -(void)setAudioJackReaderType:(AudioJackReaderType)audioJackReaderType messageOptions:(PGSwipeUniMagMessageOptions *)messageOptions

  Sets the enabled audioJackReaderType. This can be used to enable support for either the IPS Encrypted Card Reader or the UniMag (Shuttle) reader. Since the underlying libraries may not always unload cleanly, you should avoid calling this repeatedly to change the supported device type. Doing so could cause the reader to malfunction or be damaged. Setting this to AudioJackReaderAutodetectOnConnect will disable any currently selected audioJackReaderType and autodetect upon device connection.

  messageOptions will be used only when audioJackReaderType is AudioJackReaderUnimag to replace the default message options. For any other AudioJackReaderType, or to use the default message options for AudioJackReaderUnimag, this should be nil.

- -(void)beginAutodetectAudioJackCardReader;

  Asynchronously attempts to detect the card reader type currently attached to the audio jack.

  A communication test is first attempted for an IPS reader. If that fails, an attempt is made to power up a UniMag (Shuttle) card reader. If either test succeeds, the audioJackReaderType is set to the correct value, and the device will be made ready for use. The result of the autodetect is reported to the delegate through deviceAutodetectComplete:.

  Note: Both the IPS communication test and the UniMag power up produce very loud tones through the audio jack. If speakers or headphones are attached, the tones would be unpleasant to the user. It is recommended that the user be warned and allowed to remove headphones before calling this function. This library suppresses user notifications from the UniMag reader during autodetect.

Because the device is powered up in order to test it, you will not receive connection / activation / ready for swipe events during detection. When your delegate receives its deviceAutodetectComplete message, check the isConnected, isActive, and isReadyForSwipe properties for its current state and to complete any initialization.<

In order to detect the devices, all of the underlying card reader libraries must be loaded. Under some circumstances, these libraries may not unload cleanly, resulting in unreliable use of the card reader. Autodetect is also a very slow process. For these reasons, you should not rely on autodetection for each use of the app.

Because communication through the audio jack is not always perfect, autodetect does not always return a correct result. The most common failure type is returning CardReaderAutodetectResultFail even though a supported device is connected.

If it is known in advance which card reader type will be used, that type should be specified when initializing the PGSwipeController. If multiple devices must be supported, it is strongly recommended that the result of the autodetect be saved (e.g. in NSUserDefaults) and re-used on app startup.

PGSwipeDevice

The PGSwipeDevice class represents the functionality that is common to the swipe reader devices. PGSwipeIDynamo and PGSwipeUniMag both use PGSwipeDevice as a base class.

A PGSwipeDevice object is passed along with every event generated by the swipe devices to allow you to identify the device type and access device-specific features by casting to the specific swipe type.

- bool isConnected

  True when the reader is physically attached to the device.

- bool isActivated

  True when the reader is powered up / initialized.

- bool isReadyForSwipe

  True when the reader is able to accept card swipes from the user.

- id<PGSwipeDelegate>delegate

  Sets the delegate that will receive the device's events. You should not set the delegate directly. Setting the delegate on the PGSwipeController sets the delegate for each of its members.

PGSwipeDelegate

The PGSwipeDelegate protocol must be implemented by the class that intends to receive swipe reader events. The following event handlers will need to be implemented.

- -(void)didSwipeCard:(PGSwipedCard *)card device:(PGSwipeDevice *)sender

  This event is sent whenever the user swipes a card. Normally "card" will be either a PGEncryptedSwipedCard or PGMagnesafeSwipedCard (depending on the swipe reader) with track data, masked card number, expiration date, and cardholder name. If the swiped card cannot be read, "card" will be nil.

- -(void)deviceBecameReadyForSwipe:(PGSwipeDevice *)sender

  This event is sent when isReadyForSwipe becomes true. Between this event and the receipt of deviceBecameUnreadyForSwipe, any swipe should produce a didSwipeCard event.

- -(void)deviceBecameUnreadyForSwipe:(PGSwipeDevice *)sender reason:(SwipeReasonUnreadyForSwipe)reason

  This event is sent when isReadyForSwipe becomes false. There are many reasons the reader could become unready to receive swipe events, e.g. the swipe request times out, the device is disconnected, etc. Check the value of "reason" to determine the cause.

  On Shuttle readers, if you have set the device to alwaysAcceptSwipe, the reason may be set to SwipeReasonUnreadyForSwipeRefreshing. In that case, there is no need to request a new swipe. The "unready" state is momentary while the device automatically renews after a timeout, swipe, or other event.

- -(void)deviceConnected:(PGSwipeDevice *)sender;

  Occurs when the reader is physically connected to the device. With audio-port connected devices like the Shuttle, this event can be sent when the user attaches headphones. See the PGSwipeUniMag documentation for more information.

- -(void)deviceDisconnected:(PGSwipeDevice *)sender;

  Occurs when the reader is physically disconnected from the device. With audio-port connected devices like the Shuttle, this event can be sent when the user detaches headphones. See the PGSwipeUniMag documentation for more information.

- -(void)deviceActivationFinished:(PGSwipeDevice *)sender result:(SwipeActivationResult)result;

  Occurs when the device has finished an attempt to power up/initialize. This may occur at the same time as deviceConnected or later, depending on the device and settings. See the individual device documentation for specifics.

  Receiving this event does not mean the initialization succeeded. Be sure to check the value of "result" to verify that it is SwipeActivationResultSuccess.

- -(void)deviceDeactivated:(PGSwipeDevice *)sender;

  Occurs when the device has powered down. This may occur when the device is disconnected or, for certain swipe readers, when you make a call to power down the device.

- -(void)deviceAutodetectStarted;

Occurs when an attempt to detect the type of an audio jack-connected card reader has started. This can be triggered by a manual call to PGSwipeController beginAutodetectAudioJackCardReader or automatically when the PGSwipeController is in AudioReaderAutodetectOnConnect mode and an object is attached to the audio jack by the user.

- -(void)deviceAutodetectComplete:(CardReaderAutodetectResult)result;

  Occurs when an attempt to detect the type of an audio jack-connected card reader has finished. The result may be CardReaderAutodetectResultUniMag, CardReaderAutodetectResultIps, or CardReaderAutodetectResultFail if the type could not be determined.

  When this message is received, the PGSwipeController's audioJackReaderType will have been set to the appropriate value and the card reader will be activated. Since the device is powered up while autodetecting, no events for connection, activation, or readyForSwipe will be received. Check the isConnected, isActivated, and isReadyForSwipe properties to determine the device's state.

PGSwipeIDynamo

This class is the interface to the iDynamo reader. It is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create a PGSwipeIDynamo instance to interact with the iDynamo device.

The iDynamo has no configurable options. When the device is attached, it is active and ready for swipe. The only property for the PGSwipeIDynamo class is a delegate to receive events, which should not be set directly. When the delegate is set for the PGSwipeController, the same delegate is passed to the PGSwipeIDynamo instance it contains.

PGSwipeUniMag

This class is the interface to the IDTECH Shuttle reader. It is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create a PGSwipeUniMag instance to interact with the Shuttle device.

There are several flags and methods available for the Shuttle. For an app that does not need much specific control of the swipe device and is mostly interested in the swipe event, the defaults can be kept and the device will power up and become ready for swipe when attached.

- id delegate;

  Gets or sets the delegate that will receive events. You should not set this directly. When the PGSwipeController's delegate is set, it will pass it through to this delegate.

- PGSwipeUniMagMessageOptions *messageOptions;

Contains a set of options for interactions with the user, e.g. whether to prompt before powering up the Shuttle and the text of error messages. See the PGSwipeUniMagMessageOptions section for specific settings.

- BOOL activateReaderOnAttach;

  If this is true, the SDK will attempt to power-up the reader when attachment is detected. There are 3 things to be aware of:

  1. If the user attaches headphones to the mobile device, it will be treated as a swipe reader and an attempt to power it up will be made.
  2. Before the attempt to activate the reader, if messageOptions.activateReaderWithoutPromptingUser is set to false (it is false by default), the user will receive a prompt asking to confirm activation. If they decline, no activation will be attempted.
  3. If you call powerDown to deactivate the device, leaving activateReaderOnAttach set to true will cause the device to immediately power back up.
- BOOL automaticallySetVolumeToMaxOnActivate;

  If this is set to true, the device's volume will be set to maximum immediately before any attempt to power on the reader. Since the reader requires full volume to activate, this defaults to true and should normally remain true.

- BOOL alwaysAcceptSwipe;

  The Shuttle does not accept swipes from the user unless a swipe has been requested. If alwaysAcceptSwipe is true, the SDK will immediately request a swipe and renew the request any time the old swipe request times out or ends. You will still receive periodic didBecomeUnreadyForSwipe: messages, but the reason will be SwipeReasonUnreadyForSwipeRefreshing to indicate that you should be receiving a didBecomeReadyForSwipe: message immediately after without any interaction.

  The mobile device's battery may deplete faster if the swipe reader is always awaiting a swipe. If battery life is a concern, consider setting this to false and using requestSwipe when a swipe is expected, or only setting alwaysAcceptSwipe to true when a swipe is expected.

  If alwaysAcceptSwipe is true, you should not use requestSwipe or cancelSwipeRequest. By default, alwaysAcceptSwipe is true.

- -(void)powerUpDevice:(BOOL)powerUp;

  Powers up the reader if powerUp is true or cancels a power up if powerUp is false. If activateReaderOnAttach is true, this is called automatically after connection to power up the device. If you wish to only power up the device after user interaction, you should wait until a deviceConnected: event is received, then call powerUpDevice:YES when they choose to power up. This should only be used if activateReaderOnAttach is false.

- -(void)powerDown;

  Powers down the reader. This may extend mobile device battery life. A deviceConnected event will be received after shut-down, which will trigger a power-up if activateReaderOnAttach is true, so be sure to

set activateReaderOnAttach to false before powering down. It is not necessary to power down the reader before disconnecting it from the device.

- -(void)requestSwipe;

  Starts listening for swipe events. You will never need to call this if you set alwaysAcceptSwipe to true (it is true by default). After receipt of a didBecomeUnreadyForSwipe message, you may request a new swipe (unless the reason is SwipeUnreadyForSwipeReasonRefreshing). The request will timeout after 20 seconds, or the amount of time you set in setSwipeTimeoutDuration:.

- -(void)cancelSwipeRequest;

  Cancels a swipe request to stop listening for swipe events. You should not use this if you did not manually start the swipe request with a call to requestSwipe.

- -(void)setSwipeTimeoutDuration:(int)seconds;

  Sets the time between requestSwipe and when swipes will no longer be accepted. Default and maximum are 20 seconds. The minimum is 3 seconds. This still applies even if alwaysAcceptSwipe is true, but the swipe request will be automatically renewed in that case.

PGSwipeUniMagMessageOptions

This class contains a set of user-interaction options for the Shuttle device.

- BOOL activateReaderWithoutPromptingUser;

  If this is set to true (false by default), the reader is activated automatically immediately after you receive a deviceConnected: event. If this is false, you will need to call powerUpDevice: during or after the deviceConnected event to power the device or cancel powering up.

- BOOL showInitializingReaderMessage;

  If true, an "Initializing Card Reader…" alert is shown while the reader powers up and is dismissed once power-up completes.

- NSString *cardReaderActivationPrompt;

  Gets or sets the prompt that will be displayed to confirm that the user would like to power up the reader. If you change this prompt, you should also change cardReaderActivationAtMaxVolumePrompt. Note: an activation prompt is only shown before activation if messageOptions.activateReaderWithoutPromptingUser is false. This message is meant to include a warning to indicate that the volume will be set to max. If automaticallySetVolumeToMaxOnActivate is false, cardReaderActivationAtMaxVolumePrompt is shown instead of this.

- NSString *cardReaderActivationAtMaxVolumePrompt;

  Gets or sets the prompt that will be displayed to confirm that the user would like to power up the reader. If you change this prompt, you should also change cardReaderActivationPrompt. Note: an activation

prompt is only shown before activation if messageOptions.activateReaderWithoutPromptingUser is false. If automaticallySetVolumeToMaxOnActivate is true and the volume is not at maximum, cardReaderActivationPrompt is shown instead of this.

- NSString *cardReaderTimeoutMessage;

  Gets or sets the alert that is shown if the reader times out while attempting to power up. If you prefer to handle this differently, set this message to nil to prevent it from being shown, then handle the activationDidComplete:result: message with a result of SwipeActivationResultTimeout.

- NSString *volumeTooLowMessage;

PGSwipeIps

This class is the interface to the IPS Encrypted Card Reader. It is not intended to be instantiated directly. Instantiate a PGSwipeController instead. The PGSwipeController will create a PGSwipeIps instance to interact with the IPS device.

- id delegate;

  Gets or sets the delegate that will receive events. You should not set this directly. When the PGSwipeController's delegate is set, it will pass it through to this delegate.

- -(void)shutdown;

  Closes the card reader's connections and disables event handling to allow it to be deallocated. You should not call this directly. It is called by the PGSwipeController when necessary.

- -(void)beginTestCommunication(id)communicationTestDelegate;

  Asynchronously sends a communication test message to the card reader device and waits for a response. This can be used to detect whether the connected device is an IPS Encrypted Card Reader. A message is sent via the audio jack and if no response is received from the device within 5 seconds, the attached object is assumed not to be an IPS reader. Note that calling this will produce a short, loud tone through the audio jack if headphones are attached.

  The result is returned to the communcationTestDelegate by calling ipsCommunicationTestCompleteWithResult:(BOOL)succes with success set to YES or NO, depending on if a response was sent by the device.