citruspay / **citruspay-ios-sdk**

Enhanced iOS SDK for Superior Native Payments Experience

mukesh122 readme updte — Latest commit 7ca931e Feb 11, 2016

CitrusPay-Framework	v3.0.0 - Existing CitrusPay SDK Migrated to Framework - v3.0.0	Feb 11, 2016
CitrusPay	v3.0.0 - Existing CitrusPay SDK Migrated to Framework - v3.0.0	Feb 11, 2016
documentation	v2.1.4	Jan 13, 2016
.gitignore	Initial commit	Dec 21, 2015
CitrusPay.podspec	v3.0.0 - Existing CitrusPay SDK Migrated to Framework - v3.0.0	Feb 11, 2016
LICENSE	readme updated	Dec 31, 2015
README.md	readme updte	Feb 11, 2016
release_notes.md	v2.1.4	Jan 13, 2016

README.md

Getting Started {#mainpage}

CitrusPay

Introduction

The CitrusPay iOS SDK enables collection of payments via various payment methods.

It is meant for consumption by CitrusPay partners who are developing their own iOS apps aimed at merchants and/or consumers.

Features

CitrusPay iOS SDK broadly offers following features.

- Prepaid Payments.
- Direct credit/debit card (CC, DC) or netbanking payments (NB) .
- Saving Credit/Debit cards into user's account for easier future payments by abiding The Payment Card Industry Data Security Standard (PCI DSS).
- Loading Money into users Citrus prepaid account for Prepaid facility .

- Withdraw the money back into User's bank account from the Prepaid account .
- Creating Citrus account for the user .

ChangeLog

Migration from 3.0.x to 3.1.x

Usage

To run the example project, clone the repo, and run `pod install` from the Example directory first.

Requirements

- Xcode 6 or higher.

Citrus PG Prerequisites

- You need to enroll with Citrus as a merchant.
- You need to host Bill generator on your server
- You need to host Return Url Page on your server. (After the transaction is complete, Citrus posts a response to this URL.)
- Make sure that you have obtained following parameters from your Citrus admin panel
- Merchant Secret Key
- Merchant Access Key
- SignIn Key
- SignIn Secret
- SignUp Key
- SignUp Secret

Note: Please DO NOT PROCEED if the above mentioned requirements have not been met.

Installation

Using cocoapods (recommended)

- The recommended installation mechanism for CitrusPay is via CocoaPods. CocoaPods is an Objective-C library dependency manager that streamlines the process of installing, configuring, and updating third-party libraries. You can learn more about CocoaPods at the website: http://cocoapods.org/
- Add `pod "CitrusPay"` to your podfile
- Run `pod install`
- You should now be able to add `#import <CitrusPay/CitrusPay.h>` to any of your target's source files and begin using CitrusPay SDK!
- Done!

Using Submodule

- If you do not wish to use CocoaPods then the secondary recommendation is to use a submodule. This allows you to easily track updates using standard Git commands. The first step to installation is to add the submodule to your project:

```
$ cd /path/to/MyApplication
# If this is a new project, initialize git...
$ git init
$ git submodule add git://github.com/citruspay/citruspay-ios-sdk.git
$ git submodule update --init --recursive
$ open citruspay-ios-sdk
```

- Drag CitrusPay-Framework folder into your existing Xcode project
- Navigate to your project's settings, then select the target you wish to add CitrusPay-Framework to
- Navigate to `Build Phases` and expand the `Link Binary With Libraries` section
- Click the + and `CitrusPay.framework` appropriate to your target's platform
- Navigate to Build Phases and expand the `Copy Bundle Resources` section
- Click the + and `CitrusPay.bundle` appropriate to your target's platform
- Add depedency
    i. Download the JSONModel repository as a zip file or clone it
    ii. Copy the JSONModel sub-folder into your Xcode project
    iii. Link your app to SystemConfiguration.framework
- You should now be able to add `#import <CitrusPay/CitrusPay.h>` to any of your target's source files and begin using CitrusPay SDK!
- Done!

Using Library Binaries (manual way)

- Download the latest zip file from our releases page or as a direct download
- Unzip the file
- Or clone it

```
$ git clone https://github.com/citruspay/citruspay-ios-sdk.git
$ open citruspay-ios-sdk
```

- Drag CitrusPay-Framework folder into your existing Xcode project
- In Xcode, go to your app's target settings. On the `Build Phases` tab, expand the `Link Binary With Libraries` section.
- Include the following framework:
    - `CitrusPay.framework`
- In Xcode, go to your app's target settings. On the `Build Phases` tab, expand the `Copy Bundle Resources` section.
- Include the following bundle:
    - `CitrusPay.bundle`
- Add depedency
    i. Download the JSONModel repository as a zip file or clone it
    ii. Copy the JSONModel sub-folder into your Xcode project
    iii. Link your app to SystemConfiguration.framework
- You should now be able to add `#import <CitrusPay/CitrusPay.h>` to any of your target's source files and begin using CitrusPay SDK!
- Done!

Documentation

HTML documentation is hosted on our CitrusPay Developer Guide.

Pdf documentation is available on the releases page or as a direct download.

SDK Organization

CitrusPay.h

`CitrusPay.h` is the starting point for consuming the SDK, and contains the primary class you will interact with. It exposes all the methods you can call to accept payments via the supported payment methods. Detailed reference documentation is available on the reference page for the `CitrusPay` class.

Data Models

All other classes in the SDK are data models that are used to exchange data between your app and the SDK. Detailed reference documentation is available on the reference page for each class.

Next Steps

Head over to the documentation to see all the API methods available. When you are ready, look at the samples below to learn how to interact with the SDK.

Samples

See the Example app for a working implementation of all API methods.

Note: make sure to open the project using `CitrusPay.xcworkspace` and not `CitrusPay.xcodeproj`.

Initializing the SDK

- Complete the installation steps (above).
- Include CitrusPay.h

```
#import <CitrusPay/CitrusPay.h>
```

How to configure KeyStore Object

As you must have noticed the SDK initialization requires you to pass the Keystore object please see below how to configure it.

```
CTSKeyStore *keyStore = [[CTSKeyStore alloc] init];
keyStore.signinId = @"test-signin";
keyStore.signinSecret = @"52f7e15efd4208cf5345dd554443fd99";
keyStore.signUpId = @"test-signup";
keyStore.signUpSecret = @"c78ec84e389814a05d3ae46546d16d2e";
keyStore.vanity = @"testing";
```

Setup working Enviroments

SDK operates in two different modes Sandbox and Production mode. for both the enviroments Citrus PG Prerequisites key sets are different. keys from one enviroment won't work on other. so please make sure you are using correct set of keys. During the developement you would always want to use the Sandbox mode. once you are done with your App development you can switch to production mode .

you need to use `[CitrusPaymentSDK initializeWithKeyStore: environment:]` to initialize the SDK

Sandbox:

```
[CitrusPaymentSDK initializeWithKeyStore:keyStore environment:CTSEnvSandbox];
```

Production:

```
[CitrusPaymentSDK initializeWithKeyStore:keyStore environment:CTSEnvProduction];
```

Only after you are done with initialization you can proceed with following guide

The SDK is logically divided into 3 modules/layers or interfacing classes

- CTSAuthLayer - handles all of the user creation related tasks .
- CTSProfileLayer - handles all of the user profile related tasks .
- CTSPaymentLayer - handles all of the payment related tasks .

To use any of the above layers your need to fetch their singlton instance from CitrusPaymentSDK's class methods,

```
// initialization in your .m file
CTSAuthLayer * authLayer = [CTSAuthLayer fetchSharedAuthLayer];
CTSProfileLayer * proifleLayer = [CTSProfileLayer fetchSharedProfileLayer];
CTSPaymentLayer * paymentLayer = [CTSPaymentLayer fetchSharedPaymentLayer];
```

Enable DEBUG Logs (By Default it's Disable Logs)

- Print Console logs

```
[CitrusPaymentSDK enableDEBUGLogs];
```

Get the Card's Schemes & Bank Logo Images

- Get the Card's Schemes Images

```
UIImage* image = [CTSUtility fetchSchemeImageBySchemeType:@"scheme"];
```

- Get the Bank Logo Images by Issure Code

```
UIImage* image = [CTSUtility fetchBankLogoImageByBankIssuerCode:@"code"];
```

- Get the Bank Logo Images by Bank Name

```
UIImage* image = [CTSUtility fetchBankLogoImageByBankName:@"bank"];
```

Following are the specific tasks related to each of the layer

Important Update for iOS 9

Doing direct payments

- CC, DC, NB Direct Payments
- Saved CC, DC Payments (A.K.A. Tokenized payments)

User Management

- Bind User (doesn't need password, enables user to save cards)
- See if anyone is logged in
- Link User (required for all Citrus Cash related operations)
- Signin the user for Citrus Cash access
- Reset User Password
- Sign Out

Card Management

- Save User Cards
- Get Saved Cards
- Delete Saved Cards

Using Citrus Cash a.k.a Prepaid Account

- Get User's Citrus Cash Balance
- Loading Money into Users Citrus Cash Account

- Paying via Citrus Cash account
- Save Cashout Bank Account
- Get Saved Cashout Bank Acoount
- Initiate Cashout Proccess into users Bank Account from Citrus Cash account
- Send Citrus Cash to another Citrus User

Dynamic Pricing Offer Coupons and Surcharge

- How to use dynamic pricing ?

Others

- Fetch Available Schemes and Banks for the Merchant

Common Integration Issues

- Could Not Connect to Internet
- postResponseiOS() error
- iOS 9 SSL Errors

Home

Mukesh Patil edited this page Dec 21, 2015 · 1 revision

Welcome to the citruspay-ios-sdk wiki!

1. Integrating CTSAuthLayer

Mukesh Patil edited this page Jan 4, 2016 · 2 revisions

There are two types of user privileges that a user can have:

- Save Card level - lets user save cards, get the saved cards, payment using Credit and Debit cards and delete cards.
- Citrus Cash level - includes all the privileges from Save Card level in addition to that User can now load and pay using their Citrus Cash account.

Bind User (for only Save Card level access)

bind the user, this grants cards saving, fetching, payment using CC and DC and delete cards privileges to the user, doesn't need password. call this method every time you want to initiate Citrus Saved card and related functionalities.

note : if you are going to integrate Citrus Cash feature then you don't need to do Bind User you can directly refer to Link User. with Link User you will get Save Card level access too

```
[authLayer requestBindUsername:@"test@gmail.com" mobile:@"9702222222" completionHandl
         if(error == nil){
            // Your code to handle success.
        }
        else {
            // Your code to handle error.
        }
    }];
```

Link User (for Citrus Cash level access, includes Save Card level)

Citrus Link user and Sign-in api simplifies the process of new user sign-up as well as enquiring the status of user's existing Citrus Cash account.

Acquiring the Citrus Cash access for the user is a two step process: First Linking the user and then presenting the sign-in screen with either OTP or password or both depending on the response of the Link User API

Citrus Link User

When User decides to Pay using Citrus Wallet, First thing you need to check is if user is already logged in with api `[authLayer isLoggedIn]` , if they are already logged in you can start the prepaid activities, if they are not then you need to call Citrus Link Api, Link User does all the checks on users account and responds with status that helps you to build sign-in screen. it also creates new account for new users and may fire otp to mobile or email .

```
[authLayer requestCitrusLink:@“test@mailinator.com” mobile:@“9700000000” completion:^(
        if (error) {
        [UIUtility toastMessageOnScreen:[error localizedDescription]];
        }
    else{
        [UIUtility toastMessageOnScreen:linkResponse.userMessage];

        switch (linkResponse.siginType) {
            case CitrusSiginTypeMOtpOrPassword:
                // Show Mobile otp and password sign in screen
                break;
            case CitrusSiginTypeMOtp:
                // Show Mobile otp sign in screen
                break;
            case CitrusSiginTypeEOtpOrPassword:
                // Show Email otp and password sign in screen
                break;
            case CitrusSiginTypeEOtp:
                // Show Email otp sign in screen
                break;
            default:
                break;
        }
     }
}];
```

Citrus Link User responds with CTSCitrusLinkRes object To decide whether to show Password or OTP or both in the next login screen can be decided using the enum field “linkResponse.siginType” in the link user response. Please see the code above .

Citrus Link Sign-in

To Signin with Password or OTP use following API

Password Sign-in

```
[authLayer requestCitrusLinkSignInWithPassoword:@“CitrusPassword” passwordType:Passwor
        LogTrace(@"error %@",error);
        if (error) {
            // Handle Error
        }
        else{
            // Handle Success
        }
    }];
```

OTP Sign-in

```
[authLayer requestCitrusLinkSignInWithPassoword:@“3453” passwordType:PasswordTypeOtp c
        LogTrace(@"error %@",error);
        if (error) {
            // Handle Error
        }
        else{
            // Handle Success
        }
    }];
```

See if anyone is logged in

To check if user is already logged into the SDK.

```
if([authLayer isLoggedIn]){
    [UIUtility toastMessageOnScreen:@"user is signed in"];
}
else{
    [UIUtility toastMessageOnScreen:@"no one is logged in"];
}
```

Reset User Password

It happens to all of us, we tend to forget our passwords. so if user forgets their password and wants to reset it following is the way to do it

```
[authLayer requestResetPassword:@"test@gmail.com" completionHandler:^(NSError *error)
    if(error == nil){
            // Your code to handle success.
        }
        else {
            // Your code to handle error.
        }
  }];
```

Sign out

To do a local sign out from the SDK you have to call following method

```
[authLayer signOut];
```

2. Integrating CTSProfileLayer

Mukesh Patil edited this page Dec 21, 2015 · 1 revision

CTSProfile layer deals with tasks related to profile of the user.

Save User Cards

Once you have successfully Linked the user you can now save cards. There are two types of cards thats can be saved Credit and Debit.

```
 //How to initialise required card?
 CTSPaymentDetailUpdate *paymentInfo = [[CTSPaymentDetailUpdate alloc] init];

 CTSElectronicCardUpdate *creditCard = [[CTSElectronicCardUpdate alloc] initCreditCard
 creditCard.cvv = @"123";
 creditCard.number = @"5105105105105100";
 creditCard.expiryDate = @"03/2018"; //mm/yyyy format
 creditCard.scheme = [CTSUtility fetchCardSchemeForCardNumber:creditCard.number]; //fe
 creditCard.ownerName = @"Tester"; //no special characters allowed here
 [paymentInfo addCard:creditCard];



// Save the card
[proifleLayer updatePaymentInformation:paymentInfo withCompletionHandler:^(NSError *er
    if(error == nil){
        // Your code to handle success.
    }
    else {
        // Your code to handle error.
    }
}];
```

Get Saved Cards

Use following code to read the saved cards, a Saved card also will have a token which looks like this 5115669e6129247a1e7a3599ea58e947, this can be used for payment using this card. you won't need to collect the card information except CVV from user again.

```
 [proifleLayer requestPaymentInformationWithCompletionHandler:^(CTSProfilePaymentRes *
    if (error == nil) {
        // Your code to handle success.
        if([paymentInfo.paymentOptions count]){
            //process the save cards here
        }
        else{
             // no saved cards
        }
    } else {
        // Your code to handle error.
       }
}];
```

Pages 7

Home

1. Integrating CTSAuthLayer

2. Integrating CTSProfileLayer

3. Integrating CTSPaymentLayer

4. Common Errors

5 Migration From V 3.0.x to V 3.1.x

ChangeLog

Delete Saved Cards

Saved card can be deleted by sending its token to following api.

```
[proifleLayer requestDeleteCardWithToken:card.token withCompletionHandler:^(NSError *
     if(error == nil){
       // Your card is successfully deleted.
     }
     else {
       // Your code to handle error.
     }
 }];
```

Get User's Prepaid Balance

After user is linked then you can fetch their prepaid account balance if user has a prepaid account enabled.

```
[proifleLayer requetGetBalance:^(CTSAmount *amount, NSError *error) {

    if (error) {
       //your code to handle the error
    }
    else{
         // your code to handle success.
        LogTrace(@" value %@ ",amount.value);
        LogTrace(@" currency %@ ",amount.currency);
    }
}];
```

Save Cash-out Bank Account

User can request withdraw their prepaid account balance into a bank account and such an account can be stored in users profile in the following manner

```
CTSCashoutBankAccount *bankAccount = [[CTSCashoutBankAccount alloc] init];
bankAccount.owner = @"Tester";
bankAccount.branch = @"HSBC0000123";
bankAccount.number = @"123456789987654";

[proifleLayer requestUpdateCashoutBankAccount:bankAccount withCompletionHandler:^(NSEr
    if (error) {
        [UIUtility toastMessageOnScreen:[error localizedDescription]];
    }
    else{
        [UIUtility toastMessageOnScreen:@"Succesfully stored bank account"];
    }
}];
```

Get Saved Cashout Bank Acoount

```
[proifleLayer requestCashoutBankAccountCompletionHandler:^(CTSCashoutBankAccountResp *
    if(error){
        // handle error
    }
    else {
        //handle sucess
    }
}];
```

3. Integrating CTSPaymentLayer

Mukesh Patil edited this page Feb 5, 2016 · 2 revisions

Payment layer lets you do the payments .

PG Payments using CC, DC and NB

You can pay using Credit, Debit, Netbanking see the following code:

```
//CC, DC payments
CTSElectronicCardUpdate *creditCard = [[CTSElectronicCardUpdate alloc] initCreditCard]
creditCard.number = @"4028530052708001";
creditCard.expiryDate = @"03/2020"; //only mm/yyyy format
creditCard.scheme = [CTSUtility fetchCardSchemeForCardNumber:creditCard.number]; //fet
creditCard.ownerName = @"Tester"; // no special characters here
creditCard.cvv = @"123";

CTSPaymentDetailUpdate *paymentInfo = [[CTSPaymentDetailUpdate alloc] init];
[paymentInfo addCard:creditCard];

[CTSUtility requestBillAmount:@"10" billURL:BillUrl callback: ^(CTSBill *bill , NSErro
        if(error){
            [UIUtility toastMessageOnScreen:error.localizedDescription];
        }
        else {
            [paymentLayer requestChargePayment:paymentInfo withContact:contactInfo wit
                if(error){
                    [UIUtility toastMessageOnScreen:error.localizedDescription];
                }
                else {
                    [UIUtility toastMessageOnScreen:[NSString stringWithFormat:@"Payme
                }
            }];
       }
}];

//netbanking payments
CTSPaymentDetailUpdate *paymentInfo = [[CTSPaymentDetailUpdate alloc] init];

// Update bank details for net banking payment.
CTSNetBankingUpdate* netBank = [[CTSNetBankingUpdate alloc] init];
netBank.code = @"CID001"; //you can obtain this from pgSetting service
netBank.name = @"Tester";
[paymentInfo addNetBanking:netBank];

[CTSUtility requestBillAmount:@"10" billURL:BillUrl callback: ^(CTSBill *bill , NSErro

        if(error){
            [UIUtility toastMessageOnScreen:error.localizedDescription];
        }
        else {
            [paymentLayer requestChargePayment:paymentInfo withContact:contactInfo wit
                if(error){
                    [UIUtility toastMessageOnScreen:error.localizedDescription];
                }
                else {
                    [UIUtility toastMessageOnScreen:[NSString stringWithFormat:@"Payme
                }
            }];
        }
}];
```

Saved CC, DC Payments (A.K.A. Tokenized payments)

Obtain users saved cards and use this method to pay using saved card, Citrus server returns a token in each of the saved cards. You only need send this token and CVV(cvv is not saved along with the other card information) for doing the payment, rest of the information will be fetched by the Citrus server from its database using this token.

```
CTSPaymentDetailUpdate *tokenizedCardInfo = [[CTSPaymentDetailUpdate alloc] init];

// Update card for tokenized payment.
CTSElectronicCardUpdate *tokenizedCard = [[CTSElectronicCardUpdate alloc] initCreditC
tokenizedCard.cvv= @"123";
tokenizedCard.token= @"5115669e6129247a1e7a3599ea58e947";
[tokenizedCardInfo addCard:tokenizedCard];

[CTSUtility requestBillAmount:@"10" billURL:BillUrl callback: ^(CTSBill *bill , NSErr
       if(error){
           [UIUtility toastMessageOnScreen:error.localizedDescription];
       }
       else {
            [paymentLayer requestChargePayment:tokenizedCardInfo withContact:contact
             if(error){
                  [UIUtility toastMessageOnScreen:error.localizedDescription];
              }
              else {
                  [UIUtility toastMessageOnScreen:[NSString stringWithFormat:@"Payme
              }
           }];
       }
}];
```

Loading Money into Users Citrus Prepaid Account

Money can be loaded into users citrus account using following method, this can be done using direct payments as well as saved cards

```
CTSPaymentDetailUpdate *creditCardInfo = [[CTSPaymentDetailUpdate alloc] init];

CTSElectronicCardUpdate *creditCard = [[CTSElectronicCardUpdate alloc] initCreditCard]
creditCard.number = @"4028530052708001";
creditCard.expiryDate = @"03/2020"; //only mm/yyyy format
creditCard.scheme = [CTSUtility fetchCardSchemeForCardNumber:creditCard.number]; //fet
creditCard.ownerName = @"Tester"; // no special characters here
creditCard.cvv = @"123";

[creditCardInfo addCard:creditCard];

[paymentLayer requestLoadMoneyInCitrusPay:creditCardInfo withContact:contactInfo withA
   if(error){
              //handle error
            }
            else{
                //handle success
            }
}];
```

Paying via Prepaid account/Citrus Cash

Once there is some money in user's prepaid account it can be used to do payments. for that see the following method

```
[CTSUtility requestBillAmount:@"10" billURL:BillUrl callback: ^(CTSBill *bill , NSErro
        if(error){
            [UIUtility toastMessageOnScreen:error.localizedDescription];
        }
        else {
           [paymentLayer requestChargeCitrusWalletWithContact:contactInfo address:addr
               if(error){
                  //handle error
               }
               else{
                 //handle success
               }
           }];
        }
}];
```

Initiate Cashout Process into users Account from Citrus prepaid account

User can request to put money from Citrus cash account back into their bank account

```
CTSCashoutBankAccount *bankAccount = [[CTSCashoutBankAccount alloc] init];
bankAccount.owner = @"Tester";
bankAccount.branch = @"HSBC0000123";
bankAccount.number = @"123456789987654";

[paymentLayer requestCashoutToBank:bankAccount amount:@"5" completionHandler:^(CTSCash
    if(error){
               //handle error
           }
           else{
             //handle success
           }
}];
```

Fetch Available Schemes and Banks for the Merchant

Not all banks and card types are enabled for every merchant, they can be added, removed by contacting your sales contact at Citrus. To fetch what all is available for you use following method

```
[paymentLayer requestMerchantPgSettings:VanityUrl withCompletionHandler:^(CTSPgSetting
    if(error){
      //handle error
    }
    else {
        LogTrace(@" pgSettings %@ ", pgSettings);
        for (NSString* val in pgSettings.creditCard) {
            LogTrace(@"CC %@ ", val);
        }
        for (NSString* val in pgSettings.debitCard) {
            LogTrace(@"DC %@ ", val);
        }
        for (NSDictionary* arr in pgSettings.netBanking) {
            LogTrace(@"bankName %@ ", [arr valueForKey:@"bankName"]);
            LogTrace(@"issuerCode %@ ", [arr valueForKey:@"issuerCode"]);
        }
    }
}];
```

Fetch the PG Health

You can fetch the Health of the PG using following method

```
[paymentLayer requestGetPGHealthWithCompletionHandler:^(CTSPGHealthRes* pgHealthRes, N
    if(error){
              //handle error
           }
           else{
             //handle success
           }
 }];
```

Send money to another Citrus User

Money can be sent from one user's Citrus Wallet to another users wallet

```
[paymentLayer requestTransferMoneyTo:@"9800000000" amount:@"20" message:@"Here is Some
   if(error){
              //handle error
           }
           else{
             //handle success
           }
}];
```

Dynamic Pricing, offer coupons, surcharge.

Using dynamic pricing you can apply surcharge or discount or even issue a coupon to the user.

Dynamic pricing happens in 2 stages first applying the rule and second is processing the payment

There are three ways in which rule can be applied

SearchAndApply

Search and apply will search for the possible rule depending upon the input data. The rule can be applied on the payment mode or card bin range or user details etc. The system will check for matching rule using the input data and apply the rule and return the altered amount if proper rule was found. Once the request is successful the user can make transaction for the altered amount. The input parameters will be transaction amount, payment details, userDetails , valid bill url .

```
//Payment details
CTSElectronicCardUpdate *creditCard = [[CTSElectronicCardUpdate alloc] initCreditCard]
creditCard.number = @"4028530052708001";
creditCard.expiryDate = @"03/2020"; //only mm/yyyy format
creditCard.scheme = [CTSUtility fetchCardSchemeForCardNumber:creditCard.number]; //fet
creditCard.ownerName = @"Tester"; // no special characters here
creditCard.cvv = @"123";

CTSRuleInfo *ruleInfo = [[CTSRuleInfo alloc] init];
ruleInfo.originalAmount = @"100";
ruleInfo.operationType = DPRequestTypeSearchAndApply;

CTSPaymentDetailUpdate *paymentInfo = [[CTSPaymentDetailUpdate alloc] init];
[paymentInfo addCard:instrument];

CTSUser *user = [[CTSUser alloc] init];
user.email = @"test@gmail.com";
user.mobile = @"9700000000";

[paymentLayer requestPerformDynamicPricingRule:ruleInfo paymentInfo:paymentInfo billUr
        if(error){
          //handle error
    }
    else {
         //handle success
        [UIUtility toastMessageOnScreen:  [NSString stringWithFormat:@"Request Status:
    }
}];
```

CalculatePricing

Calculate pricing will search for the given rule name and apply the rule, if rule with given name is found, it will return the altered amount. Once the request is successful the user can make transaction for altered amount. The input parameters will be transaction amount, payment details, user details and rule name, valid bill url

```
//payment details
CTSElectronicCardUpdate *creditCard = [[CTSElectronicCardUpdate alloc] initCreditCard]
creditCard.number = @"4028530052708001";
creditCard.expiryDate = @"03/2020"; //only mm/yyyy format
creditCard.scheme = [CTSUtility fetchCardSchemeForCardNumber:creditCard.number]; //fet
creditCard.ownerName = @"Tester"; // no special characters here
creditCard.cvv = @"123";

CTSRuleInfo *ruleInfo = [[CTSRuleInfo alloc] init];
ruleInfo.ruleName = @"MonsoonSale";
ruleInfo.originalAmount = @"100";
ruleInfo.operationType = DPRequestTypeCalculate;

CTSPaymentDetailUpdate *paymentInfo = [[CTSPaymentDetailUpdate alloc] init];
[paymentInfo addCard:instrument];

CTSUser *user = [[CTSUser alloc] init];
user.email = @"test@gmail.com";
user.mobile = @"9700000000";

[paymentLayer requestPerformDynamicPricingRule:ruleInfo paymentInfo:paymentInfo billUr
    if(error){
          //handle error
    }
    else {
         //handle success
        [UIUtility toastMessageOnScreen:  [NSString stringWithFormat:@"Request Status:
    }
}];
```

ValidateRule

Validate rule will be used to validate the whether the rule is properly applied or not. The input values will be transaction amount, payment details, user details, rule name and altered amount. System will search for the given rule and apply the rule and check whether the given altered amount matches with the generated altered amount. Once the rule is validated you can proceed for the transaction with altered amount.

```
CTSElectronicCardUpdate *creditCard = [[CTSElectronicCardUpdate alloc] initCreditCard]
creditCard.number = @"4028530052708001";
creditCard.expiryDate = @"03/2020"; //only mm/yyyy format
creditCard.scheme = [CTSUtility fetchCardSchemeForCardNumber:creditCard.number]; //fet
creditCard.ownerName = @"Tester"; // no special characters here
creditCard.cvv = @"123";

CTSRuleInfo *ruleInfo = [[CTSRuleInfo alloc] init];
ruleInfo.ruleName = @"50PrecentOff";
ruleInfo.alteredAmount = @"50";
ruleInfo.originalAmount = @"100";
ruleInfo.operationType = DPRequestTypeValidate;

CTSPaymentDetailUpdate *paymentInfo = [[CTSPaymentDetailUpdate alloc] init];
[paymentInfo addCard:instrument];

CTSUser *user = [[CTSUser alloc] init];
user.email = @"test@gmail.com";
user.mobile = @"9700000000";

[paymentLayer requestPerformDynamicPricingRule:ruleInfo paymentInfo:paymentInfo billUr
    if(error){
          //handle error
    }
    else {
         //handle success
        [UIUtility toastMessageOnScreen:  [NSString stringWithFormat:@"Request Status:
    }
}];
```

Processing the payments for dynamic pricing

only after one of the above services is applied user can proceed with the actual payments.to do that use the following method.

```
contactInfo = [[CTSContactUpdate alloc] init];
contactInfo.firstName = @"first";
contactInfo.lastName = @"last";
contactInfo.email = @"Ytest@gmail.com";
contactInfo.mobile = @"9700000000";

addressInfo = [[CTSUserAddress alloc] init];
addressInfo.city = @"Mumbai";
addressInfo.country = @"India";
addressInfo.state = @"Maharashtra";
addressInfo.street1 = @"Golden Road";
addressInfo.street2 = @"Pink City";
addressInfo.zip = @"401209";

[paymentLayer requestChargeDynamicPricingContact:contactInfo withAddress:addressInfo c
    if(error){
        //handle error
    }
    else {
       //handle success
    }
}];
```

4. Common Errors

Mukesh Patil edited this page Jan 13, 2016 · 2 revisions

Could Not Connect to Internet

Many times during testing you may face this error, please verify following things to resolve this

- Internet connection is working fine
- Keys in MerchantConstants.h are correct according to BASE_URL mode (sandbox/production). keys are not interchangeable between the modes
- Reset your simulator
- Delete app from device and reinstall
- Also see iOS 9 Security level issue

postResponseiOS() error

you may face this error during the transaction, what this means is your return url is not hosted properly Please see the return url hosting section below to resolve this

Return Url iOS

after the transaction is over Citrus server posts(HTTP POST) various parameters to you return url web page related to this transaction. your return URL web page now needs do a signature verification of these parameters. After the signature verification on your return URL SDKs expect this data to be returned in the app to know that the transaction is over,the way this happens in android and in iOS is a a bit different due to inherent nature of OSs and APIs provided by them, for iOS return url should implement a JS function named exactly "postResponseiOS()" and should return JSON string only.

if this function returns nil, non json, or has different name or doesn't exist at all then you will face "postResponseiOS()" error

iOS 9 SSL Errors & Fix

Apple introduced App Transport Security (ATS) to actively enforce best practices of secure communication between an app and its backend. It is by default for iOS 9 apps built using Xcode 7 and forces network traffic to be sent over HTTPS. This means Requests that are not HTTPS will not be allowed to pass through.even HTTPS should use TLS 1.2 and PFS cipher suites else they block the connection from OS level. you may see errors like

```
NSURLSession/NSURLConnection HTTP load failed (kCFStreamErrorDomainSSL, -9802)

Error Domain=NSURLErrorDomain Code=-1004 "Could not connect to the server." UserInfo=0

App Transport Security has blocked a cleartext HTTP (http://) resource load since it i
```

Though All the Citrus servers use HTTPS that matches with Apple's laid standards for ATS, Some banks are still using older versions of HTTPS, which means you may face SSL error during transactions (mostly for net-banking)

To solve this error do the following

Add following element in your apps info.plist by opening it in any text editor like Sublime or TextWrangler You can add it under the main `</dict>` object.This disables the ATS for your APP

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
</dict>
```

more info ...

5 Migration From V 3.0.x to V 3.1.x

Mukesh Patil edited this page Jan 13, 2016 · 2 revisions

This guide will help you to migrate to V 3.1.0

Along with performance improvements, bug fixes, new features we have also changed a few basics things in integration to make it more flexible.

Integration related changes

SDK Initialization

SDK now needs to be initialized using `[CitrusPaymentSDK initializeWithKeyStore: environment:]` before you can use any of its features, please see it here

MerchantConstants.h

Previous versions of the SDK used to rely on this constant file for all the merchant related configurations, like subscription keys, secret keys, base urls. now this is obsolete. V 3.1.X no longer relies on this. you will need to directly configure these keys at the time of SDK initialization.please see it here

Sandbox OR Production

In previous versions of the SDK you would need to configure the BASE_URL constant in the MerchantConstants.h to change the SDK's mode of operation, in V3.1.X this needs to be done at the time of SDK initialization. please see it here

Layers initialization

If you have used earlier version of the SDK you already know that SDK functions using three main layers/classes. CTSAuthLayer, CTSProfileLayer, CTSPaymentLayer; for initialization of these layers instead of doing alloc init directly, you will need to get their instances using the respective class methods of `CitrusPaymentSDK`

```
CTSPaymentLayer *payment = [CitrusPaymentSDK fetchSharedPaymentLayer];
CTSAuthLayer *auth = [CitrusPaymentSDK fetchSharedAuthLayer];
CTSProfileLayer *profile = [CitrusPaymentSDK fetchSharedProfileLayer];
```

Features

- Update : Delete Card API
- Update : Wallet Pay API
- Update : Check if anyone is logged in
- New : Dynamic Pricing, Coupons, Surcharge APIs

ChangeLog

Mukesh Patil edited this page Feb 5, 2016 · 3 revisions

[3.1.4]

[enhancement] seamless migration for user on new authentication flow

[bugfix] prepaid account activation on new link flow

[3.1.3]

[enhancement] improved signup and sign in flow, now OTP login is supported, new link