ONLY FOR USE BY CUSTOMERS WHO CONNECTED TO WPAY PRIOR TO 2020

Wpay Digital Payments

Wpay Digital Payments Developer Hub. Everything you need to know to create a seamless commerce experience. Accept web and mobile payments, defend against fraud, and monitor your transactions in real time.

Version 1.0.7 - Generated on August 9th, 2024

Authentication Swagger

Wpay AP Swagger

Table of Contents

Getting Started

Getting Access

Get Setup as a Merchant

Contact us to find out more information on how to get set up as a Merchant with Wpay

API Keys

When you sign up for an account we will provide you with two sets of API keys one for the test environment (also known as the Sandbox), and one for the live system.

You authenticate with our APIs by providing your API Key in the request `X-Api-Key` header.

> 📘 **API Keys & IP Whitelist Restrictions**
>
> Your API key is used throughout our system to identify you as the merchant. Your egress IPs will be whitelisted to restrict sensitive services which should only be processed from your server-side applications. These will be configured against your merchant account during your merchant onboarding process.

IP Whitelisting

When you sign up with us, we will ask you to provide us the list of IP addresses that are authorised to make an API call. This ensures even if your API key is found or stolen, only your servers will be able to use it.

Authentication

Wpay's SDK APIs use Bearer Authentication to authenticate requests.

> 🚧 **Restricted API**
>
> This API is IP restricted to allow unauthenticated server-side calls.
> Your servers will need to be on an allow list to allow refunds.

You will need to provide your API key (`X-Api-Key`) as a part of the header along with a bearer token (`Authorization`) for all API requests (unless otherwise stated).

Once the backend has the token it can be passed to the front end for use with the client-side APIs calls as part of the header.

Token Generation

APIs that require authentication need a signed JSON Web Token (JWT). This authentication token (prefixed with bearer) needs to be provided in the `authorization` HTTP header.

Example of how to generate a bearer token for your authentication calls via our IDM API :

cURL JavaScript

```
curl --location \
--request POST 'https://{{environment}}.mobile-api.woolworths.com.au/wow/v1/idm/servers/token?access_toke
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourAPIKey}}' \
--data-raw '{
"shopperId": "70538197-f696-4f70-a025-93b03d48a03390",
"username": "jondoe@email.com",
"isGuest": false
}
'
```

Where:

- `yourAPIKey` contains your merchant API key which will be provided to you when joining Wpay
- `access_token_exp` is the time in seconds in which the access token will expire. This will default to 1 hour should no value be provided.
- `shopperId` is your unique customer identifier for your customer within your system. This is required to be unique for both registered and guest users. This should never be an email address or any other value that may change over time for the user.
- `isGuest` is an optional boolean flag that can be set based on your user type. If the user is a registered user in your system set this to `false` or if they are a guest user set this to `true`. This will default to false.
  - Guest users are your customers who have not registered their details with your store and therefore they cannot permanently save instruments to their customer wallet for later use.

Authentication Environments

The `{{environment}}` variable can be set to the below to access the required environment for the authentication API.

Environment	Variable
Test:	test
User Acceptance Testing	uat

Token Response

JSON

```
{
    "accessToken": "9ZhistH7jGgcao8QPH6ApAIsy2NW",
    "accessTokenExpiresIn": 3599,
    "refreshToken": "87eB6Pmp0x9I8x54NLtIYQo9nQ15FxoS",
    "refreshTokenExpiresIn": 2591999,
    "tokensIssuedAt": 1629782182152,
    "isGuestToken": false,
    "idmStatusOK": false
}
```

Where:

- `accessToken` is the bearer token value to be used in subsequent endpoint calls
- `accessTokenExpiresIn` is the time in seconds in which the access token will expire
- `tokensIssuedAt` is an epoch time in milliseconds `GMT: Tuesday, 24 August 2021 05:16:22.152`
  - reference Online epoch converter

Payment Methods Supported

Wpay currently supports multiple payment methods which are represented by the payment instrument type which you would like your customers to use when making payments.

These payment methods below can be used for a single payment or can be vaulted and saved to your customer's wallet for later use.

Supported Instrument Types

Cards

Wpay support credit and debit cards across the major providers namely; Mastercard, Visa, American Express, JCB and Diners Club allowing your customer to pay with and store their cards in their wallet for future use.

Gift Cards

Gift cards within the majority of gift card programs are supported with Wpay allowing your customer to pay with and store their gift cards in their wallet for future use.

PayPal

PayPal payments are supported through Wpay. Your customers can also store their PayPal account within their wallet allowing for easy one-click payments for future use using the Vault integration pattern. Alternatively, they can use the Checkout integration pattern for quick one time payments for both PayPal and PayPal Pay in 4.

> 📘 **PayPal Integration**
>
> To integrate your PayPal payments with Wpay, we require you to provide us with your PayPal account details so we can link them to your account.

Apple Pay

Apple Pay is a digital wallet and payment service provided by Apple and is the most secure way to pay in your website or app. Apple Pay allows your customers to make payments in iOS apps and on the web using Safari with the Apple Pay button and complete your purchase without the need to add to a cart or fill out a form.

Google Pay

Google Pay is a digital wallet and payment service provided by Google and is the convenient, efficient and secure way for customers to pay in mobile apps and on websites using their Google Account.

Cards

Card payments allow your customers to easily purchase goods or services from your business using credit and debit cards without the need for cash or cheques using one of the major providers such as; Mastercard, Visa, American Express, JCB and Diners Club. When a customer makes a purchase from your website the Acquirer will seek authorisation for the transaction from the customer's bank. When the transaction is approved the customer completes the purchase and a payment receipt is issued. The proceeds of the sale are then deposited to your account as part of the daily settlement process.

Cards High Level Payment Process

Card Acceptance Setup

Your Wpay account management representative will be able to support you through the steps of setting up credit and debit card acceptance as part of your integration process.

Tokenizing a Card

To tokenize a Card as an instrument using the Wpay Platform please follow Tokenizing a Card.

Making a Card Payment

To make a Card payment using the Wpay Platform please follow Making a Payment.

Card Acceptance Marks

The card acceptance marks of these schemes can be found in the following location:

- American Express
- Diners Club
- JCB
- Mastercard
- Visa

Gift Cards

Gift Cards have been a popular payment method for a long time seeing how they drive brand awareness and earn customer loyalty while encouraging repeat purchases.

As the Program Manager for all Woolworths brand gift cards, Wpay has the capability to support your business in the sale and redemption of Woolworths gift cards.

Wpay has a wide range of physical and digital gift card products that are offered on the Woolworths Gift Card Website

In addition, we work with a variety of Gifting Partners who act as resellers of Woolworths brand Gift cards. If you are interested in becoming a Gifting Partner, please follow us on Becoming a Gifting Partner

Beyond the sale of gift cards, Wpay also provides APIs that will support your platform in processing gift card payments. Even if your website does not sell gift cards, you can allow your customers to pay for their purchase using a gift card either entirely or partially.

Given your customer already has access to their Woolworths gift card, Wpay's APIs will allow them to

Add the Gift Card to their digital wallet

To tokenize a Gift Card as an instrument using the Wpay Platform please follow Tokenizing a Gift Card.

Make a Gift Card Payment

To make a Gift Card payment using the Wpay Platform please follow Making a Payment.

Check their Gift Card Balance

To check Gift Card balance using the Wpay Platform please follow Gift Card Balance Check.

Apple Pay

Apple Pay is a digital wallet and payment service provided by Apple Inc and it provides an easy and secure way to pay in iOS apps, watchOS apps, and websites on Safari. Apple Pay allows your customers with compatible devices to make payments and complete purchase without the need to add to a cart or fill out a form. Payments made using Apple Pay will benefit from full liability shift for supported card schemes including; Visa, MasterCard and AMEX.

When your customer selects Apple Pay, they are presented with a pre-populated Payment Sheet where they can view the order, choose a card and confirm their shipping and contact details. The final step is payment authorisation through Face ID or Touch ID authentication to confirm the purchase. [1]

> 📘 **Supported Apple Pay Experiences**
>
> As a merchant you can choose to support any of the patterns below. Depending on which of the below patterns your business wishes to support, the setup steps will vary.
>
> - Apple Pay in iOS apps
> - Apple Pay on the Web via Safari
> - Apple Pay on both iOS apps and on the Web via Safari.

Apple Pay Setup

To start accepting Apple Pay payments via Web and Mobile Apps please follow the instructions below:

- **Apple Pay on the Web:** You don't need to create your own Apple Pay certificate for web integration because you will use Wpay Apple Pay Certificate. Your Wpay account management representative will be able to support you

through the integration process.

- **Apple Pay in App:** You will need to enable Apple Pay with your own certificates and share your certificates with Wpay to configure against your merchant profile. These certificates will be generated from your Apple Developer Account which can be created by following the instructions here Before You Enrol - Apple Developer Program. Please share this with your account management representative who can support you through the integration process.

Starting a Session with Apple for Apple Pay

Apple Pay Web: Wpay `paymentsession` API must first be called from the client side so that Wpay can correctly validate the merchant domain and return an opaque Apple Pay merchant session object. The Apple Pay session object can be used to encrypt payment data. Upon successful call of `Paymentsession` API, you may then present a Payment Sheet to the user to review the purchase and authorise the payment.

cURL JavaScript Swift

```
// Dev to check
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/applepay/paymentsession' \
--header 'X-Api-Key: {{yourAPIKey}}' \
--header 'Content-Type: application/json' \
--header 'authorization: Bearer {{yourBearerToken}}' \
--header 'Origin: https://{{yourDomain}}' \
--data-raw '{
  "validationUrl": "https://apple-pay-gateway-cert.apple.com/paymentservices/paymentSession",
  "displayName": "{{yourStoreName}}"
}'
```

Where:

- `Origin` header will be automatically passed on by customer's browser
- `validationUrl` is validation URLs provided by Apple when you set up your server (see Requesting An Apple Pay Payment Session for more detail)

Apple Pay Merchant Session Object

A sample of merchant session object can be seen below. You may pass this whole object to the completion method, completeMerchantValidation to enable the user to authorize a transaction.

JSON

```
{
    "epochTimestamp": 1658987403707,
    "expiresAt": 1658991003707,
    "merchantSessionIdentifier": "SSHDC1221C404******",
    "nonce": "eed0adec",
    "merchantIdentifier": "4FCE55AEEED3DACF3FE85****************",
    "domainName": "localhost:8080",
    "displayName": "My Store",
    "signature": "308006092a864886**********************************",
    "operationalAnalyticsIdentifier": "My Store:4FCE55AEEED3DACF3************",
    "retries": 0,
    "pspId": "B17C76FBD980CE281**********************"
}
```

Apple Pay in App: you can validate your merchant domain by calling Apple validation URL directly (see Requesting An Apple Pay Payment Session for more detail). It will return an opaque Apple Pay session object that you can use to encrypt the payment data. Since Wpay already stores your Apple Pay Certificates against your profile in our secure environment, we will be able to decrypt the payment data on our end and [provide you with a payment token](doc:apple-pay#tokenizing-apple-pay) that you can use to make payments or store in your wallet.

Tokenizing Apple Pay

To tokenize an Apple Pay instrument using the Wpay platform please follow Tokenizing Apple Pay.

Making an Apple Pay Payment

To make an Apple Pay payment using the Wpay platform please follow Making a Payment.

> 🚧 **Testing Apple Pay Integration**
>
> Testing for Apple Pay on web requires a sandbox account. The steps to create a sandbox accounts and add dummy cards is documented here at Apple Pay - Sandbox Testing - Apple Developer. Once your test integration is complete you can test your setup using the Apple Pay on the Web Demo .

References

1. Apple Pay - Marketing Guidelines
2. Apple Pay Programming Guide
3. Preparing Merchant Domains for Verification
4. Register Merchant
5. Apple Pay Interactive Demo
6. Registering with Apple Pay and Applying to Use the API
7. Requesting An Apple Pay Payment Session

Google Pay

Google Pay™ is a payment service provided by Google and is a convenient, efficient and secure way for customers to pay in mobile apps and on websites using their Google Account. Google Pay gives you access to hundreds of millions of cards saved to Google Accounts worldwide.

When your customer chooses Google Pay, a Payment Sheet will be displayed, which lists the goods or services, payment methods saved to their Google Account, plus other optional fields such as shipping address and additional information. The customer can then quickly review the purchase, select a payment method, add an optional shipping address and confirm the payment. See the videos in Start using Google Pay today for more detail.

Supported Google Pay Experiences

1. Google Pay payments via Android Mobile App.
2. Google Pay payments on iOS devices via Chrome Browser
3. Google Pay on PC via supported browsers.

Google Pay Setup

To start accepting Google Pay payments, you will need to be registered with Wpay as one of our merchants. Once you are registered, we will provide you with two values - `gateway` and `gatewayMerchantId` for you to request a payment token from Google Pay. Set `wpayaus` as your `gateway` and use `merchantId` found in your Merchant Profile for `gatewayMerchantId`. You may then pass the payment token information to Wpay for us to process the payment. Please get in touch with your Account Management representative to get your merchant profile setup.

Tokenizing Google Pay

To tokenize a Google Pay instrument using the Wpay Platform please follow Tokenizing Google Pay.

Making a Google Pay Payment

To make a Google Pay payment using the Wpay Platform please follow Making a Payment.

Supported Google Pay Configuration

Wpay supports both PAN_ONLY and CRYPTOGRAM_3DS authentication methods:

- PAN_ONLY - the card is stored on file within your customer's Google account and not bound to an Android device.
- CRYPTOGRAM_3DS - the payment credentials is bound to an Android device.

Supported payment cards can be found here.

If you require a billing address to be submitted for address verification purposes, set `billingAddressRequired` to `true` and `billingAddressParameters` according to your need. For more information, please see Billing Address Parameter.

Enabling 3D Secure (3DS) for Google Pay

If you would like to enable 3DS for PAN_ONLY credentials returned via Google Pay API, please refer to the following documentation for more information:

- 3D Secure (3DS)
- 3DS Payment Integration
- 3DS Card Capture Integration

> 🚧 **Testing Google Pay Integration**
>
> Google provides a test environments for merchants to test Google Pay integration. The test environment does not return live chargeable tokens in the `PaymentData` response, but it allows developers to test elements of the purchase such as; confirmation pages, receipts, billing, shipping, and so on. Google also provide a set of sample cards and sample payment tokens that can be used during testing. For more information, please see "Integration checklist".

References

1. Google Pay Overview
2. Google Pay Android developer documentation
3. Google Pay Android integration checklist
4. Google Pay Android brand guidelines
5. Google Pay Web developer documentation
6. [Google Pay Web brand guidelines] (https://developers.google.com/pay/api/web/guides/brand-guidelines)
7. [Google Pay Web Integration checklist] (https://developers.google.com/pay/api/web/guides/test-and-deploy/integration-checklist)
8. Google Pay UX best practices - Android
9. Google Pay UX best practices - Web

PayPal

PayPal

PayPal allows your customers with one click, to get directly from the product page or the shopping cart to the PayPal payment page without any detours. PayPal remembers their financial and shipping details, so customers don't have to re-enter them on your site.

PayPal Pay in 4

PayPal Pay Later offers for Australian customers include PayPal Pay in 4 , which is a short-term instalment product that provides eligible PayPal customers the option to split purchases into 4 interest-free payments for transactions between $30 AUD - $1,500 AUD. The first payment is due at the time of the transaction with subsequent payments due every two weeks. [1]

The Wpay implementation of PayPal Pay in 4 uses the PayPal Checkout method which utilizes the Braintree SDK. Checkout with PayPal is a one-time payment checkout experience that gives you more control over the entire checkout process. It offers a streamlined checkout flow that keeps customers local to your website during the payment authorization process. Unlike the PayPal Vault method, Checkout with PayPal does not provide the ability to store a customer's PayPal account in the Vault. [5]

🚧 Minimum Braintree SDK version

Braintree SDK JavaScript v3 3.69.0 or higher is required on the client side.

Supported Integration Patterns

Below are the current PayPal integration methods further detail about each integration method can be found on the Braintree Website.

Payment Integration Method	Wpay Support	Payment Types	Save for Future Use
Vault Flow	Yes	PayPal	Yes
Checkout Flow	Yes	PayPal PayPal Pay in 4	No
Checkout Flow with Vault Flow	Not currently	PayPal PayPal Pay in 4	Yes

! Number of Integration Methods

Please note you can only use one integration method at a time and they can't be used simultaneously. Therefore, its important to consider which integration method best supports your current and future use cases for PayPal.

PayPal Account Setup

As a Merchant you will first need to setup your PayPal Developer Account which can be used for both Sandbox testing and Production. Follow the PayPal instructions to ensure you sign up with the correct account type as either Personal or Business.

📘 PayPal Account Sign Up

Sign Up: Create a PayPal Account - PayPal Australia

Note: If you setup the PayPal Developer Account as a Business type there are a few things to consider:

- Who will be the Primary Account User this individual is essentially the owner of the account.
- Roles and Permissions you may want to have for different groups of users in your business who require access to the account to perform their role.

Braintree Account Setup

As a Merchant you will also need to setup your Braintree Developer Account which can be used for both Sandbox testing and Production. In order, to link your PayPal and Braintree Accounts together you should follow these instructions from Braintree. As part of the setup process you will need to enter the following details:

- Sandbox Account Email

- Client ID
- Secret

This information can be found in the My Apps & Credentials menu item inside your PayPal Developer Account.

PayPal Client Token

Obtaining the current PayPal client token may be retrieved from your Merchant Profile. The PayPal client token is located under the `payPal.clientToken` path.

> 🚧 **Client Token Validity**
>
> The client token is only valid for 24 hours before it expires and new one is required to be generated.

PayPal Buttons Integration Tool

Once the PayPal and Braintree Sandboxes have been linked together and PayPal enabled as a Payment Processing Option in the Braintree control panel it is possible to test the integration using the PayPal Buttons Integration Tool. This will allow you to test your integration is setup correctly by observing if the PayPal Buttons render in the tool in the sandbox environment. A prerequisite to the PayPal buttons appearing is a Client Token must be generated and inserted into the code as shown in the extract below.

HTML

```
<!-- Load the Braintree components -->
    <script src="https://js.braintreegateway.com/web/3.85.2/js/client.min.js"></script>
    <script src="https://js.braintreegateway.com/web/3.85.2/js/paypal-checkout.min.js"></script>
    <script>
      // Client Token
      var ClientToken =
        "eyJ2ZXJzaW9uIjoyLCJhdXRob3JpemF0aW9uRmluZ2VycHJpbnQiOiJleUowZVhBaU9pSktWMVFpTENKaGJHY2lPaUpGVXpJ
```

PayPal Buttons

The PayPal Checkout method allows you to render either the single PayPal or also a Pay in 4 button depending on your preference as documented here by PayPal. For more detailed information regarding styling of PayPal buttons please refer to this guide on the PayPal Developer Portal.

The Pay in 4 button provides direct access to Pay Later offers in PayPal Checkout. Specify the `fundingSource: paypal.FUNDING.PAYLATER` option when rendering the Pay Later button, and additionally pass `enable-funding: paylater` as a query param in the PayPal SDK. Passing `dataAttributes.amount` when calling `loadPayPalSDK` is required for the Pay in 4 button to render. [2] It's also possible to render multiple standalone payment buttons for supported payment method. [3]

JavaScript HTML

```
paypalCheckoutInstance.loadPayPalSDK({
  components: 'buttons,messages'
  currency: 'AUD',
  'enable-funding': 'paylater',
  dataAttributes: {
    amount: '10.00'
  }
  // Other config options here
}).then(function () {
  var button = paypal.Buttons({
    fundingSource: paypal.FUNDING.PAYLATER,

    createOrder: function () {
      return paypalCheckoutInstance.createPayment({
        flow: 'checkout', // Required
        amount: '10.00', // Required
        currency: 'AUD' // Required
      });
    },
    onApprove: function (data, actions) {
      return paypalCheckoutInstance.tokenizePayment(data).then(function (payload) {
        // Submit `payload.nonce` to your server
      });
    },
  });
  if (!button.isEligible()) {
    // Skip rendering if not eligible
    return;
  }

  button.render('#pay-later-button');
});
```

Pay in 4 Messaging

PayPal provides a dynamic messaging to let customers know they can buy now and pay later. With dynamic messaging, PayPal will show them the right order for what they're buying. You must show the Pay Later button if you present Pay Later messaging.

Qualifying amount message example [1]

Non-qualifying amount or no amount message example [1]

To present this messaging, you must include the PayPal messaging component in JavaScript while loading PayPal SDK, and also include a `div` on the HTML page where you want to render a message. [2]

JavaScript HTML

```
paypalCheckoutInstance.loadPayPalSDK({
  components: 'buttons,messages'
  // Other config options here
}, function () {
  // set up PayPal buttons (see next section)
});
```

For more detailed information about the presentment of PayPal Pay in 4 messaging please refer to the documentation on the PayPal Developer Portal here.

Tokenizing PayPal

To tokenize a PayPal instrument using the Wpay Platform please follow Tokenizing a Payment Instrument.

Making a PayPal Payment

To make a PayPal payment using the Wpay Platform please follow Making a Payment.

Using PayPal Seller Protection

To use PayPal Seller Protection when making a payment please follow PayPal Seller Protection.

References

1. Pay Later Offers (AU)
2. Pay Later Offers
3. Standalone Payment Buttons
4. Braintree PayPal Guide
5. Checkout Method

Wallets and Instruments

Wallets

The wallet allows you to securely store and manage customers' payment instruments in our vault. This will allow you and your customer to reuse these stored instruments in future checkouts without the need to recapture all the payment details again. Think of this as a real wallet but for the online world.

Wallet Creation

We handle the complexity of setting up a new wallet each time a new customer saves a payment instrument during checkout. We will re-use a customer's existing wallet if they have already saved one or more payment instruments as part of their previous checkout.

Retrieving Instruments in the Wallet

Payment Instruments stored in the customer's wallet for your website can be retrieved using the customer's unique identifier.

Instruments

An instrument is a unique reference to a payment method that has been tokenized and stored in our vault, either as a single-use or multi-use instrument. When tokenization is successful we securely store the payment information as a payment instrument in our vault and each instrument is uniquely identified by an instrument ID and payment token.

These instruments can be single-use or can be saved to a customer's wallet and used multiple times.

Single-use Instrument Tokens

An instrument token that is single-use will not be stored in the customer's wallet and cannot be used to make multiple payments. An example of this would be the instruments generated for a guest user or where a customer has selected not to save their payment details for future checkouts.

Multi-use Instrument Tokens

An instrument token that is multi-use is where the customer has selected to save their payment details so that they can use them at a later stage without having to recapture all their payment details. We will store the payment token against the wallet as an instrument which you will be able to retrieve along with some header level details of the payment method when the customer comes back for a future checkout.

Instrument Types

An instrument includes all payment methods, e.g. debit card, credit card, gift card and PayPal.

Integration Options

With Wpay you can integrate with us via our easy to use SDK's and REST API's.

SDK's

Our SDK's allow for easy integration, hosting and customisation of PCI compliant frames for credit card capture.

API's

Our API's allow you to interact with our payments and wallet management services and allow full control over your integration.

For the full specification see our API reference

Payments

Overview

Once you've successfully set up your integration and have received your `API key`, you're able to start receiving and tokenizing your customers' payment instruments. After this tokenization process, you're ready to create a payment. To do this, you can use either our API's or our easy to integrate SDK's.

How it works

Tokenization

Tokenization or tokenizing a payment instrument is the process of taking sensitive payment information and converting it into a payment token. This payment token can then be used for making payments and setting up payment agreements through our payment services. Payment tokens are used for both single-use payments such as; guest payments or the token can be vaulted and stored in the customers wallet and used multiple times for your registered customers.

Payments

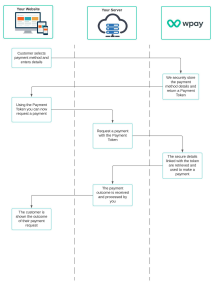
- Once you have tokenized a payment instrument you can then use the payment token to send through a payment request.
- We use the payment token to retrieve the payment information from our secure vault and use this information to process a payment with the relative issuer/provider.
- Once a transaction is complete you will receive back a response with the outcome of the payment request and a `Payment Transaction ID` and `Payment Transaction Reference`. These can be used for subsequent Refunds, Voids or Completions.

Tokenizing a Payment Instrument

This guide covers the high level process for tokenizing a payment instrument and provides detailed guides on supported payment instruments which can be tokenized using the Wpay Platform.

How it works

1. The customer then enters their payment details directly on your checkout page.
2. Your customer's sensitive information is passed from your website and is processed by us. We return a payment token (`instrumentId`) which is a unique representation of the card without any sensitive card information.
3. The `instrumentId` can be used in our payments services to make a payment or even set up a recurring payment.



Tokenization Guides

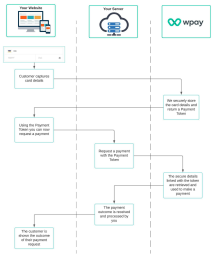
Follow the guides below for detailed instructions on how to perform tokenization for different payments instruments using the Wpay Platform.

1. Tokenizing a Card
2. Tokenizing a Gift Card
3. Tokenizing PayPal
4. Tokenizing Apple Pay
5. Tokenizing Google Pay

Tokenizing a Card

To tokenize a card you are required to use our Frames SDK. This is due to the PCI compliance requirements around capturing and handling sensitive card payment information. We currently do not support the tokenization of card information via our standard API's, however, if you are an organization with strong **PCI Compliance** please Contact us

High Level Flow



How it works

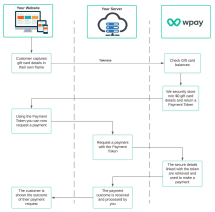
1. Embed the Wpay Frames on the checkout page of your website.
2. The customer then enters their payment details directly on your checkout page.
3. Your customer's sensitive information is passed from the frames and is processed by us. We return a payment token (`instrumentId`) which is a unique representation of the card without any sensitive card information.
4. The `instrumentId` can be used in our payments services to make a payment or even set up a recurring payment.

Due to PCI compliance requirements, we require you to use our Frames in order to capture and tokenize credit card information. To see more on tokenizing a credit card via our Frames please see our Frames SDK

Tokenizing a Gift Card

To tokenize a gift card for a gift card program supported by your site, you can make use of our tokenize gift card API's. Unlike our scheme card tokenization service which necessitates the use of our PCI compliant Frames SDK , for gift cards you can host your own frames to capture the `gift card number` and `pin`.

High-level flow



How it works

1. Create and host frames on your site allowing your customers to capture a `gift card number` and `pin` for a supported gift card program
2. Your customer's gift card information is processed by us and we return a `payment token` which is a unique representation of the gift card without any sensitive information.
3. The `payment token` can be used in our payments services to make a payment

Tokenizing a Gift Card

This method should be used to tokenize gift cards. The same API can be used for registered and guest customers.

cURL | JavaScript | Swift | Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/v1/apm/tokenize' \
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw '{
    "data": {
        "paymentInstrumentType": "GIFTCARD",
        "payload": {
            "cardNumber": "628000**********",
            "pinCode": "****",
            "primary": false,
            "save": true
        }
    },
    "meta": {}
}'
```

Where:

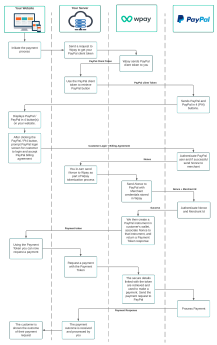
- `cardNumber` and `pinCode` are the values for the gift card your customer wishes to tokenize
- `primary` can be set to `true` or `false` and this indicates whether it is a primary or secondary instrument when saving the card to the customer's wallet

- `save` can be set to `true` or `false` and this indicates whether to save the card in the customer's wallet after tokenizing (multi-use token) or to only have a token be valid for a single transaction (single-use token)

Tokenizing PayPal

Currently Wpay supports both the Checkout and Vault flows for PayPal. Please refer to the Wpay PayPal documentation to get further detailed information about each of the supported PayPal integration patterns.

High Level Flow



Tokenizing PayPal for Checkout Flow

The Checkout with PayPal integration pattern will allow you to tokenize a PayPal or Pay in 4 `nonce` into a single-use payment instrument. This means payment instruments generated using the PayPal Checkout integration will never be saved to the customer's wallet.

How it works

1. Integrate your site with PayPal to facilitate the checkout journey and allow a customer to log in to their PayPal account and approve the payment.
2. Once the payment has been approved PayPal will provide you with a `nonce` which can then be provided to Wpay to use for payments.
3. The PayPal `nonce` is processed by us and we return a `payment token` which is a unique representation of the PayPal account without any sensitive information.
4. The `payment token` can be used in our payment services to make a payment.

cURL | JavaScript

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/v1/apm/tokenize' \
--header 'Authorization: Bearer 2Q0BRJbdbJAljXsX6q35fuyN6w9X' \
--header 'Content-Type: application/json' \
--header 'X-Api-Key: 9JMPM102iV1PtnO6HwZoorYNpdfqAWap' \
--data-raw '{
    "data": {
        "paymentInstrumentType": "PAYPAL",
        "payload": {
            "nonce": "35ecab49-4d75-0687-72ba-2794b490e071"
        }
    },
    "meta": {}
}'
```

Where:

- `nonce` is the value PayPal returns upon successfully validating the payment request.

Tokenizing PayPal for Vault Flow

To tokenize your customer's PayPal account and use it to make a payment or store it in the customer's wallet, you will need to first provide us with your PayPal account information so that we can link this to your Wpay account.

To tokenize a customer's PayPal account you can make use of either our SDKs or APIs.

How it works

1. Integrate your site with PayPal to facilitate the checkout journey and allow a customer to log in to their PayPal account and approve the payment.
2. Once the payment has been approved PayPal will provide you with a `nonce` which can then be provided to Wpay to use for payments or to even store within the customer's wallet to allow for easy one-click payments in the future.
3. The PayPal `nonce` is processed by us and we return a `payment token` which is a unique representation of the PayPal account without any sensitive information.
4. The `payment token` can be used in our payments services to make a payment.

Tokenizing PayPal for a Registered Customer

This method should be used when dealing with a registered customer on your website and where the user has been authenticated as a registered customer.

cURL JavaScript Swift Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/paypal/tokenize' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw '
{
     "primary": true,
     "nonce": "8ca99905-2419-09ab-742d-2794b490e071"
}
'
```

Where:

- `nonce` is the value PayPal returns upon successfully validating the payment request
- `primary` can be set to `true` or `false` and this indicates whether it is a primary or secondary instrument when saving the card to the customer's wallet

Tokenizing PayPal for a Guest Customer

This method should be used when dealing with a guest customer on your website and where the user has been authenticated as a guest customer.

cURL | JavaScript | Swift | Kotlin

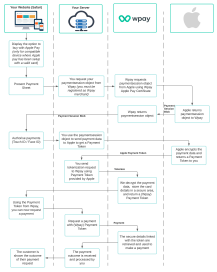
```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/guest/paypal/tokenize' \
--header 'Accept: application/json' \
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw '
{
     "nonce": "8ca99905-2419-09ab-742d-2794b490e071"
}
'
```

Tokenizing Apple Pay

To tokenize an Apple Pay instrument and use it to make a payment or store it in the customer's Wpay wallet, we will first need to set up your merchant profile. To tokenize a customer's Apple Pay instrument you can make use of our APIs.

> 📘 **Apple Pay on the Web (Safari)**
>
> For Apple Pay on the web, you will use Wpay Apple Pay certificate that we can configure against your merchant's profile in our system.

High-level flow - Web



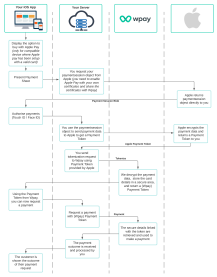
How it works

1. Integrate your websites with Apple Pay to facilitate the checkout journey and allow a customer to select an instrument stored in their Apple Pay account.
2. Get a `paymentsession` object from Wpay to create and encrypt payment data (since you will be using Wpay Apple Pay certificate).
3. Once the user authorises the payment using Touch ID / Face ID, send the encrypted payment data to Apple servers, where it is re-encrypted using Wpay Payment Processing certificate and receive back a `PaymentToken` from Apple.
4. This `PaymentToken` can then be provided to Wpay to be decrypted in our secure environment for tokenization. Please see the Payment Token Format for more information.
5. The Apple Pay data is processed by us during tokenization and we return a `Wpay PaymentToken` which is a unique representation of the Apple Pay instrument without any sensitive information.

6. The `Wpay PaymentToken` can be used in our payments services to make a payment.

> ### Apple Pay on Mobile Apps
>
> For Apple Pay on Mobile Apps you will need to first provide us with your Apple Pay account information so that we can link this to your Wpay account.

High level flow - iOS App



How it works

1. Integrate your iOS app with Apple Pay to facilitate the checkout journey and allow a customer to select an instrument stored in their Apple Pay account.
2. Get a `paymentsession` object from Apple using your own Apple Pay certificate to create and encrypt payment data.
3. Once the user authorises the payment using Touch ID / Face ID, send the encrypted payment data. to Apple servers, where it is re-encrypted using your Payment Processing certificate and receive back a `PaymentToken` from Apple.
4. This `PaymentToken` can then be provided to Wpay to be decrypted in our secure environment for tokenization. Please see the Payment Token Format for more information.
5. The Apple Pay data is processed by us during tokenization and we return a `Wpay PaymentToken` which is a unique representation of the Apple Pay instrument without any sensitive information.
6. The `Wpay PaymentToken` can be used in our payments services to make a payment.

Tokenizing Apple Pay

This method should be used to tokenize Apple Pay payment token data. The same API can be used for registered and guest customers.

cURL JavaScript Swift

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/v1/apm/tokenize' \
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw '{
    "data": {
        "paymentInstrumentType": "APPLEPAY",
        "payload": {
            "data": "a15owFtc**********************",
            "ephemeralPublicKey": "MFkwEwY**********************************",
            "publicKeyHash": "AwqjNE*********************************",
            "transactionId": "87a124b**********************",
            "signature": "MIAGCSq*********************************",
            "version": "EC_V1",
            "instrumentType": "AMEX",
            "primary": true,
            "comment": "AMEX-0001",
            "applicationData":"3B4893D8B*****************************************"
        }
    },
    "meta": {}
}'
```

Where:

- `paymentInstrumentType` should be set to APPLEPAY for Apple Pay tokenization.
- `data` within `payload` can be retrieved from the decrypted `PaymentToken` from Apple. This contains encrypted payment data.
- `ephemeralPublicKey` can be retrieved from the decrypted `PaymentToken` from Apple. This is an Ephemeral public key bytes.
- `publicKeyHash` can be retrieved from the decrypted `PaymentToken` from Apple. This is a hash of the encoded public key of your merchant's certificate.
- `transactionId` can be retrieved from the decrypted `PaymentToken` from Apple. This is a transaction identifier that is generated on the device.
- `signature` can be retrieved from the decrypted `PaymentToken` from Apple. The signature includes the signing certificate, its intermediate CA certificate, and information about the signing algorithm.
- `version` can be retrieved from the decrypted `PaymentToken` from Apple. The token uses `EC_V1` for ECC-encrypted data, and `RSA_V1` for RSA-encrypted data.
- `instrumentType` is the payment network of the card selected.
- `primary` can be set to `true` or `false` and this indicates whether it is a primary or secondary instrument when saving the card to the customer's wallet.
- `comment` is the display name of the card selected, generally, this is the payment network following by the last 4 digits of the selected card.
- `applicationData` can be retrieved from `PaymentToken` from Apple. This field contains application-specific data or state.

Sample Tokenization Response

JSON


```
{
  "data": {
    "paymentInstrumentId": "2484***",
    "allowed": true,
    "status": "VERIFIED",
    "paymentInstrumentType": "APPLE_PAY",
    "paymentToken": "2f3ddc79-****-****-****-************",
    "stepUpToken": "e80c40fe-****-****-****-************"
  },
  "meta": {}
}
```

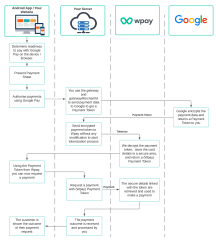

Where:

- `paymentInstrumentId` is the new payment instrument id to be used for payments.
- `allowed` is a flag to indicate if the merchant profile in the container is allowed to use this payment instrument.
- `status` indicates the status of the payment instrument in the container.
- `paymentInstrumentType` is the type of instrument for which token has been generated. For Apple Pay, this value will be set to APPLE_PAY.
- `paymentToken` is the Apple pay payment token. Payment token is a unique identifier for the payment instrument.
- `stepUpToken` is the step-up token to be used for payments.

Tokenizing Google Pay

To tokenize a Google Pay™ instrument and use it to make a payment or store it in the customer's Wpay wallet, we will first need to set up your merchant profile. To tokenize a customer's Google Pay instrument you can make use of our APIs.

High-level flow



How it works

1. Integrate your app or websites with Google Pay to facilitate the checkout journey and allow a customer to select an instrument stored in their Google Pay account and approve the payment.
2. Once the payment has been approved, Google Pay will provide you with a payment token data payload which can then be sent to Wpay for payments.
3. The Google Pay token data is decrypted and processed by us. We then return a `Wpay PaymentToken` which is a unique representation of the Google Pay instrument without any sensitive information.
4. The `Wpay PaymentToken` can be used in our payments services to make a payment.

Tokenizing Google Pay

This method should be used to tokenize Google Pay payment token data. The same API can be used for registered and guest customers.

cURL JavaScript Swift Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/v1/apm/tokenize' \
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw '{
    "data": {
        "paymentInstrumentType": "GOOGLEPAY",
        "payload": {
            "instrumentType": "AMEX",
            "comment": "AMEX-5232",
            "tokenData": "{\"protocolVersion\":\"ECv2\",\"signature\":\"MEQ**********==\",\"intermediateS
        }
    },
    "meta": {}
}'
```

Where:

- `paymentInstrumentType` should be set to `GOOGLEPAY` for Google Pay tokenization
- `instrumentType` is the payment network of the card selected.
- `comment` is the display name of the card selected, generally, this is the payment network followed by the last 4 digits of the selected card.
- `tokenData` is the JSON Escaped token data returned by Google Pay upon successfully authenticating the payment.

Making a Payment

Once a payment instrument has been tokenized the payment token (`instrumentId`) can be used to make a payment through our APIs. The `instrumentId` representing the tokenized instrument can be used to make payments in the same way regardless of the payment method the token represents.
The payments API will also be used to handle payments for your registered as well as guest customers.

Transaction Types

When making a payment you have the option to apply two different transaction types depending on your business's requirements and use cases.

Pre-auth

The Pre-auth flow allows you to make a purchase and reserve the funds on the customer's card and complete the transaction at a later stage to take the funds or should the need arise voids the full transaction amount. Once a transaction has been completed it can be refunded for either part of or the full transaction amount.

> 📘 **Preauth Flow**
>
> Pre-authorisation is only applicable to debit and credit cards either in the Wpay wallet and **some** gift card programs.

Purchase

The Purchase flow allows you to process a transaction where the funds on the customer's card are taken immediately. Once a purchase has been successfully processed you can use refund to refund either part of or the full transaction amount.

Make a Payment

To submit a payment request using an `Instrument Id` generated during the tokenization process, please call the Payment API below.

cURL | JavaScript | Swift | Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/payments
--header 'X-Api-Key: {{yourAPIKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--header 'Content-Type: application/json' \
--data-raw '{
  "data": {
    "transactionType": {
      "creditCard": "PREAUTH",
      "giftCard": "PURCHASE",
      "payPal": "PURCHASE",
      "googlePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      },
      "applePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      }
    },
    "clientReference": "UNIQUE_CLIENT_REFERENCE",
    "orderNumber": "UNIQUE_ORDER_NO",
    "shippingAddress": {
                        "firstName": "John",
                        "lastName": "Doe",
                        "company": "Wpay",
                        "extendedAddress": "4th Floor",
                        "streetAddress": "407 Elizabeth Street",
                        "suburb": "Surry Hills",
                        "stateOrTerritory": "NSW",
                        "postalCode": "2010",
                        "countryCode": "AU"
                },
    "payments": [
      {
        "paymentInstrumentId": "213553",
        "amount": 10.5
      },
       {
        "paymentInstrumentId": "215319",
        "amount": 6.5
      }
    ]
  },
  "meta": {
    "fraud": {
        "provider": "cybersource",
        "version": "CyberSourceTransaction_1.101",
        "format": "XML",
        "responseFormat": "XML",
        "message": "<?xml version=\"1.0\" encoding=\"Windows-1252\"?>\r\n<RequestMessage xmlns:xsd=\"http
    },
    "challengeResponses": [
      {
        "instrumentId": "213553",
        "type": "STEP_UP",
```

Where:

- `transactionType` is the method by which you want to process the transaction, either as a pre-auth or as a straight purchase.
- `clientReference` is your application specific reference number. This number should uniquely identify the transaction in your system
- `orderNumber` is your order number of the transaction as generated during the order creation.
- `shippingAddress` is the customer's shipping address.
- `paymentInstrumentId` is the tokenized payment instrument with which you want to make the payment. This is either from the frames SDK response or the list payment instruments response.
- `fraud` is the CyberSource fraud payload for the transaction. Please see Fraud Detection for more information around the fraud payload.
- `challengeResponses` contains the payment challenge data such as the step-up token for CVV capture or PayPal Seller Protection `deviceData`. See PayPal Seller Protection for more information.

Split Payments

Wpay also supports split payments across multiple instruments in our Payments API. When you choose to enable this option in your store, the customer may be presented a checkout page with all available payment instruments and this allows them to pay using some or all instruments in a single payment request.

Supported schemes:

- Payment split across multiple debit/credit cards.
- Payment split across multiple gift cards.
- Payment split across multiple debit/credit cards + multiple gift cards.

> 📘 **A note on Split Payments**
>
> If part of the split payment is rejected (e.g. not enough funds in one of the gift cards), Wpay will return `partialSuccess` as `true` in the payment response. Wpay will attempt to roll back the already completed transactions and the status of this rollback can be seen in the `rollback` field. Should this be `FAILED` you will need to work out how to handle this as part of your process internally. Some possible options would be to submit a Refund or Void payment requests or manually handling the partial rejection through manual process.

Challenge Responses During Payments

A challenge-response may be required during payments based on various factors such as the payment instrument type being used or whether 3DS has been invoked.

Step Up Tokens

A step up token is a token linked to the CVV which is captured as part of the card tokenization process. CVV's are not stored within the vault due to PCI compliance legislation and as such a temporary token is given when a CVV is captured. See the Step Up Token Process for more information on generating a step up token.

A step up token is usually required:

- On un-verified cards i.e. on cards where a transaction has not yet been completed (either a $0.01 pre-auth or a full transaction).
- Where your merchant settings indicate that a CVV is always required.
- When setting up a payment agreement (recurring payment).

A step up token is **not** required:

- For a short period after the card has been captured and tokenized as the step-up token from the card capture will still be valid.
- When your merchant settings indicate that a CVV is not required on verified cards saved in your customer's wallets.
- When charging a payment agreement as part of a recurring payment.

> 📘 **3DS2 Integration**
>
> If you are interested in 3DS2, please review the 3D Secure (3DS).

Transaction Outcomes

After a payment has been processed you will receive a detailed response showing the outcome of the payment request.

JSON


```
{
    "data": {
        "transactionId": "75a37436-6f59-475c-b915-b370ab5bf513",
        "paymentRequestId": "12e4623d-af7e-4bfd-9cab-070ccd3bbd52",
        "type": "PAYMENT",
        "status": "APPROVED",
        "grossAmount": 17,
        "executionTime": "2021-09-27T23:13:00.857Z",
        "merchantId": "10001",
        "merchantReferenceId": "20170505065",
        "clientReference": "T5ESYRPWJKPHk997",
        "instruments": [
            {
                "paymentInstrumentId": "215319",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "PREAUTH",
                        "executionTime": "2021-09-27T23:13:01.379Z",
                        "paymentTransactionRef": "1000000007817560",
                        "status": "APPROVED",
                        "amount": 6.5
                    }
                ]
            },
            {
                "paymentInstrumentId": "215318",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "PREAUTH",
                        "executionTime": "2021-09-27T23:13:01.379Z",
                        "paymentTransactionRef": "1000000007817559",
                        "status": "APPROVED",
                        "amount": 10.5
                    }
                ]
            }
        ],
        "subTransactions": [
            {
                "transactionReceipt": "1000000007817558",
                "partialSuccess": false,
                "fraudResponse": {
                    "clientId": "6327843829656897803007",
                    "reasonCode": "481",
                    "decision": "REJECT"
                },
                "paymentResponses": [
                    {
                        "paymentInstrumentId": "215319",
                        "paymentToken": "de770338-5fde-4e15-a232-5efd1246ef62",
                        "paymentTransactionRef": "1000000007817560",
                        "threeDS": {
                            "sli": null,
```


Transaction Outcomes

The status of your payment request is provided in the response and can be seen at both the overall transaction as well as sub-transaction level. The outcome of a payment can either be **APPROVED** or **REJECTED**.

Transaction Reference

The transaction ID `transactionId` is returned as part of a payment response and is a unique reference to the full payment request. A payment can be made up of multiple sub-transactions each with a `paymentTransactionRef` which is the unique reference of the sub-transaction.

The `transactionId` and `paymentTransactionRef` are important as these will be a required piece of information when either completing a pre-authorised transaction, voiding the transaction or processing a refund.

Fraud Outcomes

Should you use the fraud services of Wpay the fraud outcome (`fraudResponse`) in the payment response will give you the outcome of the fraud decision manager check. This can be **Accept**, **Reject** or **Review**.

- **Accept**: No fraud detected. Advice is that payment can proceed.
- **Reject**: Fraud likely. Advice is that payment should be voided or refunded.
- **Review**: Fraud potential. Payment should be manually reviewed to determine fraud decision.

Complete a Pre-authorised Payment

Completing a transaction allows you to take either the full or partial funds from a previously pre-authorised payment.

How it works

Utilising our completions feature you are able to take either the full amount reserved on the customer's payment instrument or a partial amount **less than the total amount reserved** utilising the original payment transaction ID and reference number.

Completions can only be processed on payments made using the pre-authorisation flow if the transaction has not been voided.

> 🚧 **Restricted API**
>
> This API is IP restricted to allow unauthenticated server-side calls. Your servers will need to be on an allow list to allow completions.

Completing a Payment

Using the `transactionId` and optionally the `paymentTransactionRef` from a pre-authorised payment you are able to complete for an amount up to the amount pre-authorised. You can simply call this method with the `transactionId` and all sub-transactions will be completed for their full amounts or you can specify the sub-transactions `paymentTransactionRef` and the amount you wish to complete the transaction for.

cURL JavaScript

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/merchant/transact
--header 'content-type: application/json' \
--header 'content-type: application/json' \
--header 'X-Api-Key: {{yourAPIKey}}' \
--data-raw '{
    "data": {
        "orderNumber": "UNIQUE_ORDER_NO",
        "clientReference": "UNIQUE_CLIENT_REFERENCE",
        "completions":[{
            "paymentTransactionRef": "1000000007828829",
            "amount": 10.5
        }]
    },
    "meta": {}
}'
```

Where:

- `transactionId` is the unique reference to pre-authorised payment outcome received from us when making a payment.
- `clientReference` is your application-specific reference number. This number should uniquely identify the transaction in your system
- `orderNumber` is your order number of the transaction as generated during the order creation.
- `paymentTransactionRef` is the specific sub transaction reference within the parent `transactionId`.
- `amount` is the value for which you want to complete the sub transaction.

Completion Outcome

Following the completion, an outcome of the transaction is returned with the status of the completion as well as the transaction references.

JSON


```
{
    "data": {
        "transactionId": "23fe9174-fdf3-4ba0-88d0-f0678c510cab",
        "merchantReferenceId": "b717215d-4afc-4760-ab08-0fac46150309",
        "walletId": "08d92644-121e-47cb-a711-7e7389cb576d",
        "paymentRequestId": "9074e03f-05e1-4b55-a7eb-f1a0336acccc",
        "grossAmount": 50.5,
        "clientReference": "ref",
        "executionTime": "2021-10-06T06:00:21.035Z",
        "type": "COMPLETION",
        "status": "APPROVED",
        "instruments": [
            {
                "paymentInstrumentId": "215931",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "COMPLETION",
                        "executionTime": "2021-10-06T06:00:21.035Z",
                        "paymentTransactionRef": "1000000007828829",
                        "completionTransactionRef": "1000000007828838",
                        "status": "APPROVED",
                        "amount": 10.5
                    }
                ]
            }
        ],
        "subTransactions": [
            {
                "transactionReceipt": "1000000007828838",
                "partialSuccess": false,
                "completionResponses": [
                    {
                        "paymentTransactionRef": "1000000007828829",
                        "completionTransactionRef": "1000000007828838",
                        "amount": 10.5,
                        "externalServiceCode": "00",
                        "externalServiceMessage": "APPROVED"
                    }
                ]
            }
        ]
    },
    "meta": {}
}
```


Void a Pre-authorised Payment

Voiding a transaction allows you to cancel the reserved amount which has been held on the customer's card as part of a purchase made using the pre-authorisation flow.

How it works

Utilising our voids feature you are able to void the full transaction amount prior to taking the reserved funds from the customer's card utilising the original payment transaction ID and reference number.

Voids can only be processed on payments made using the pre-authorisation flow if the transaction has not yet been completed.

> 🚧 **Restricted API**
>
> This API is IP restricted to allow unauthenticated server-side calls. Your servers will need to be on an allow list to allow voids.

Voiding a Payment

Using the `transactionId` and optionally the `paymentTransactionRef` from a pre-authorised payment you are able to void a pre-authorised transaction which will void the full amount reserved. You can simply call this method with the `transactionId` and all subtransactions will be voided for their full amounts or you can specify the sub-transactions `paymentTransactionRef` you wish to void should you only wish to void part of the transaction.

cURL JavaScript

```
// probably should use :transactionId in the below URL to indicate it as a parameter
// same for Javascript examples
// the URL used in Swift/Kotlin examples seems incorrect.

curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/merchant/transacti
--header 'content-type: application/json' \
--header 'X-Api-Key: {{yourAPIKey}}' \
--data-raw '{
    "data": {
        "orderNumber": "UNIQUE_ORDER_NO",
        "clientReference": "UNIQUE_CLIENT_REFERENCE",
        "voids": [
            {
                "paymentTransactionRef": "1000000007818208"
            },
            {
                "paymentTransactionRef": "1000000007818209"
            }
        ]
    },
    "meta": {}
}'
```

Void Outcome

Following the void, an outcome of the transaction is returned with the status of the void as well as the transaction references.

JSON


```
{
    "data": {
        "transactionId": "d7af9019-056d-4920-b6c1-adff66eab0eb",
        "merchantReferenceId": "f69ba188-4d09-43d2-af19-8f0eba93cde0",
        "walletId": "87e47ba3-d2d0-437d-bc7d-d93581ae4e8f",
        "paymentRequestId": "a3be273e-4a62-4956-b1f0-4a2f2770f76e",
        "grossAmount": 0,
        "clientReference": "T8VZS5KQH0N278D",
        "executionTime": "2021-09-28T06:31:44.565Z",
        "type": "VOID",
        "status": "APPROVED",
        "instruments": [
            {
                "paymentInstrumentId": "215353",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "VOID",
                        "executionTime": "2021-09-28T06:31:44.565Z",
                        "paymentTransactionRef": "1000000007818209",
                        "voidTransactionRef": "1000000007818214",
                        "status": "APPROVED",
                        "amount": 0
                    }
                ]
            },
            {
                "paymentInstrumentId": "215352",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "VOID",
                        "executionTime": "2021-09-28T06:31:44.565Z",
                        "paymentTransactionRef": "1000000007818208",
                        "voidTransactionRef": "1000000007818213",
                        "status": "APPROVED",
                        "amount": 0
                    }
                ]
            }
        ],
        "subTransactions": []
    },
    "meta": {}
}
```


Refund a Payment

Refunds allow you to refund a previous transaction to the original instrument from where the purchase was made.

How it works

Utilising our refunds feature you are able to refund an amount up to the value of the full transaction amount utilising the original payment transaction ID and reference number.

Multiple refunds are able to be processed on the transaction up to the original transaction amount.

Refunds can only be processed on payments made using the purchase flow or after a transaction using the pre-authorisation flow has been completed.

> 🚧 **Restricted API**
>
> This API is IP restricted to allow unauthenticated server-side calls. Your servers will need to be on our allow list to allow refunds.

Refund a Payment

Using the `transactionId` and optionally the `paymentTransactionRef` from a payment you are able to refund a transaction up to the amount of the original transaction. You can simply call this method with the `transactionId` and all subtransactions will be refunded for their full amounts or you can specify the sub-transactions `paymentTransactionRef` and the amount you wish to refund the transaction for.

cURL JavaScript

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/merchant/transacti
--header 'content-type: application/json' \
--header 'X-Api-Key: {{yourAPIKey}}' \
--data-raw '{
        "data" : {
                "reason": "Customer returned item",
                "clientReference": "UNIQUE_CLIENT_REFERENCE",
        "subTransactions": [
            {
            "subTransactionRef": "1000000007818223",
            "amount": 20.5
            },
            {
            "subTransactionRef": "1000000007818224",
            "amount": 10.5
            }
        ]
        },
        "meta": {}
}'
```

Refund Outcome

Following the refund, an outcome of the transaction is returned with the status of the refund as well as the transaction references.

JSON


```
{
    "data": {
        "transactionId": "c1de8321-64dc-48a8-ae00-5f8ab341cf05",
        "merchantReferenceId": "c691fdb2-c072-426a-b2ec-324c53366747",
        "walletId": "7aa8634e-3c4e-41f7-9f15-0963515de94a",
        "paymentRequestId": "1ed7bd1a-1068-4f7a-9655-8874aaa6685c",
        "refundReason": "Customer returned item",
        "grossAmount": 31,
        "clientReference": "UNIQUE_CLIENT_REFERENCE",
        "executionTime": "2021-09-28T06:45:54.138Z",
        "type": "REFUND",
        "status": "APPROVED",
        "instruments": [
            {
                "paymentInstrumentId": "215355",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "REFUND",
                        "executionTime": "2021-09-28T06:45:54.138Z",
                        "paymentTransactionRef": "1000000007818223",
                        "refundTransactionRef": "1000000007818227",
                        "status": "APPROVED",
                        "amount": 20.5
                    }
                ]
            },
            {
                "paymentInstrumentId": "215354",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "REFUND",
                        "executionTime": "2021-09-28T06:45:54.138Z",
                        "paymentTransactionRef": "1000000007818224",
                        "refundTransactionRef": "1000000007818226",
                        "status": "APPROVED",
                        "amount": 10.5
                    }
                ]
            }
        ],
        "subTransactions": [
            {
                "transactionReceipt": "1000000007818225",
                "refunds": [
                    {
                        "paymentTransactionRef": "1000000007818223",
                        "refundTransactionRef": "1000000007818227",
                        "amount": 20.5,
                        "externalServiceCode": "00",
                        "externalServiceMessage": "APPROVED"
                    },
                    {
                        "paymentTransactionRef": "1000000007818224",
```

Customer Wallet Management

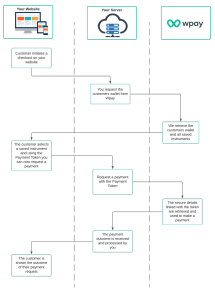
Overview

When a card is tokenized you are able to select to store this against customers wallet in the Wpay secure cardholder environment. The payment token provided back during tokenization is a unique payment token for this saved instrument for your merchant account.

Once a card has been stored against your customer's wallet you are able to retrieve the wallet and view the payment instruments stored. These can then be represented back to your customer allowing them to select from a list of saved payment methods and instruments when making a payment on your merchant site.

For more information on making a payment using a payment token see: Making a Payment

How it works



Retrieve a Customers Wallet

Where a registered customer has saved a payment instrument to their wallet during payment you are able to retrieve these saved payment instruments during subsequent checkouts allowing for easy checkout using saved payment details.

The wallet stores all tokenized payment instruments for your customer along with any configured payment agreements set up for recurring payments. This allows for simple checkout using an already saved payment method.

List Instruments

A list of saved payment instruments with you can be retrieved using the list instrument feature.

cURL | JavaScript | Swift | Kotlin

```
curl --location --request GET 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/instrument
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
```

Your customers saved instruments and payment agreements will be returned and allow you to show their saved instruments for selection during checkout.

JSON


```
{
    "data": {
        "creditCards": [
            {
                "paymentInstrumentId": "213###",
                "paymentToken": "7578a5bc-0aaa-####-####-############",
                "status": "VERIFIED",
                "createdOn": "2021-09-07T15:45:39.311+10:00",
                "lastUpdated": "2021-09-20T16:50:03.090+10:00",
                "lastUsed": "2021-09-20T16:50:02.592+10:00",
                "primary": false,
                "allowed": true,
                "scheme": "MASTERCARD",
                "cardSuffix": "0407",
                "cardName": "CHAPMAN",
                "expiryMonth": "01",
                "expiryYear": "23",
                "cvvValidated": false,
                "expired": false,
                "requiresCVV": true,
                "updateURL": "https://iframe.environment.payments.woolworths.com.au/container-ws/getCaptu
                "stepUp": {
                    "type": "CAPTURE_CVV",
                    "mandatory": true,
                    "url": "https://iframe.environment.payments.woolworths.com.au/container-ws/getCapture
                    "sessionId": "4f71251c-788a-####-####-############"
                }
            },
            {
                "paymentInstrumentId": "214###",
                "paymentToken": "c8b3cb08-f2b2-####-####-############",
                "status": "UNVERIFIED_PERSISTENT",
                "createdOn": "2021-09-22T16:24:32.622+10:00",
                "lastUpdated": "2021-09-22T16:24:32.622+10:00",
                "primary": false,
                "allowed": true,
                "expiryYear": "23",
                "scheme": "VISA",
                "expiryMonth": "02",
                "cardName": "My Card",
                "cardSuffix": "0608",
                "cvvValidated": false,
                "expired": false,
                "requiresCVV": true,
                "updateURL": "https://iframe.environment.payments.woolworths.com.au/container-ws/getCaptu
                "stepUp": {
                    "type": "CAPTURE_CVV",
                    "mandatory": true,
                    "url": "https://iframe.environment.payments.woolworths.com.au/container-ws/getCapture
                    "sessionId": "4f71251c-788a-####-####-############"
                }
            }
        ],
        "giftCards": [],
        "payPal": [
```


Where:

- `paymentInstrumentId` is the payment token of the associated instrument saved in the customer's wallet
- `paymentToken` is the payment token unique GUID of the associated instrument saved in the customer's wallet
- `status` is either **VERIFIED** or **UNVERIFIED_PERSISTENT**. Verified indicates that a successful verification or purchase has occurred using the instrument. Unverified indicates that the card has not yet been verified or used in a purchase.
- `lastUpdated` is the date the instruments information was last updated.
- `lastUsed` is the date the instrument was last used to make a payment.
- `allowed` indicates whether the instrument is an allowed payment method based on your merchant config with Wpay.
- `scheme` indicates the issuer scheme of the tokenized card.
- `cardSuffix` provides the last 4 digits of the tokenized credit card for display purposes.
- `cardName` is the name given to the card at the point of tokenization.
- `expiryMonth` indicates the month to which the card will be valid. This is indicated as a 2 digit MM field.
- `expiryYear` indicates the year in which the card expires. This is indicated as a 2 digit YY field.
- `cvvValidated`
- `expired` indicates if the card has expired based on the cards expiry month and year as compared to the current date.
- `requiresCVV` indicates if the CVV is required when making a payment utilizing the saved card. This is based on your merchant preferences with Wpay. Where this is **true** a step up token will need to be provided during payment. See Step Up Process
- `stepUp: mandatory` will align with the requiresCVV indicator.

List Instruments including Gift Card Balance

You can also include an optional `include=GC_BALANCE` parameter when calling Get Payment Instruments List to retrieve all saved payment instruments and get the gift card balance at the same time. Refer to Retrieve Gift Card Balance using List Instruments.

Retrieve a Gift Card

In some circumstances, you may want to retrieve your registered customers gift card from their wallet. Provided you already know the `paymentInstrumentId` from List Instrument API, you may call the API below to retrieve the gift card information securely.

cURL | JavaScript | Swift | Kotlin

```
curl --location --request GET 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/instrument
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}'
```

Where

- `uriEncodedPublicKey` query parameter is the RSA / ECC public key
- `algo` query parameter is either set to `rsa` for RSA encryption or `ec` for ECC encryption.
- `paymentInstrumentId` is the tokenized payment instrument of the gift card. This can be derived the list payment instruments response.

📘 Gift Card Retrieval Encryption

To retrieve gift card number and pin securely, you must generate either an RSA (Rivest-Shamir-Adleman) or ECC (Elliptic Curve Cryptography) public and private keys on your server and embed the public key into the request query parameter. The key pair must remain valid either for the duration of the customer session or for one-time use in a single request / response cycle. Provided the gift card can be found in the customers wallet, you may then decrypt the response with the private key to extract the gift card number and pin.

Transaction Outcome

JSON


```
{
  "data": {
    "paymentInstrumentId": "81xxx",
    "paymentInstrumentType": "GIFT_CARD",
    "paymentToken": "ec9b****-****-****-****-a8ca4f*******",
    "status": "UNVERIFIED_PERSISTENT",
    "createdOn": "2017-11-06T08:38:09.890Z",
    "lastUpdated": "2017-11-06T19:38:09.860+11:00",
    "lastUsed": "2017-10-12T13:25:49.770+11:00",
    "primary": true,
    "allowed": true,
    "paymentInstrumentDetail": {
      "cardSuffix": "2517",
      "programName": "WISH Gift Card"
    }
  },
  "meta": {
    "cipherText": "INLh2cH2MtnTKQ1RxwwWQHiXUZ**********************"
  }
}
```


Where

- `cipherText` is encrypted and base64 encoded gift card data. You will need to decrypt and decode it (using base64 encoding) in order to extract the gift card number and pin.
- `paymentInstrumentId` is the payment token of the associated gift card saved in the customer's wallet.
- `paymentToken` is the payment token unique GUID of the associated gift card saved in the customer's wallet
- `status` is either **VERIFIED** or **UNVERIFIED_PERSISTENT**. Verified indicates that a successful verification or purchase has occurred using the instrument. Unverified indicates that the card has not yet been verified or used in a purchase.
- `lastUpdated` is the date the gift card information was last updated.
- `lastUsed` is the date the gift card was last used to make a payment.
- `allowed` indicates whether the gift card is an allowed payment method based on your merchant config with Wpay.
- `cardSuffix` provides the last 4 digits of the tokenized gift card for display purposes.
- `programName` is the gift card name given at the point of tokenization.

Sample of the gift card number and pin after decryption can be seen below.

JSON

```
plainText={
  "pinCode" : "3333",
  "cardNumber" : "628759191**********"
}
```

Manage a Customers Wallet

Your customers may wish to remove a saved instrument from their wallet after it has been saved. You can easily allow customers to remove saved instruments from their wallet using the delete instrument feature.

Delete an Instrument

cURL JavaScript Swift Kotlin

```
curl --location --request DELETE 'https://dev-api.wpay.com.au/wow/v1/pay/instore/customer/instruments/{{p
--header 'X-Api-Key: {{yourAPIKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
```

Once an instrument is successfully deleted from a customer's wallet it will no longer be returned when retrieving a customer's wallet.

> 🚧 **Instruments Linked to Payment Agreements**
>
> An instrument linked to an existing active payment agreement cannot be deleted from a customers wallet. The payment agreement will need to be updated to a new payment instrument or the payment agreement must be expired/deleted before the instrument can be removed.

Gift Card Balance Check

As part of utilising gift cards on your site and as part of the customers wallet you may wish to retrieve the current available balance for a gift card to display to a customer as well as calculate any split payments across the remaining balance of a gift card and another payment instrument type.

Balance by Gift Card Number & Pin

> 📘 **Single vs. Multiple Card Balance Check**
>
> When checking a card balance by gift card number and pin, the service is restricted to only check a single instrument. When checking balance using a tokenized gift card's instrument ID, multiple gift card balances can be processed in a single call.

cURL JavaScript Swift Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/giftcards/balance' \
--header 'accept: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--header 'Content-Type: application/json' \
--data-raw '{
    "giftCards": [
        {
            "cardNumber": "6280005563194014720",
            "pinCode": "5211"
        }
    ]
}'
```

Balance by Instrument ID

cURL JavaScript Swift Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/giftcards/balance' \
--header 'accept: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--header 'Content-Type: application/json' \
--data-raw '{
    "giftCardInstruments": [
        {
            "paymentInstrumentId": "163****"
        }
    ]
}'
```

Gift Card Balance Outcome

JSON

```
{
    "giftCardBalances": [
        {
            "cardNumber": "628000**********",
            "balance": 111.03,
            "expiryDay": "31",
            "expiryMonth": "12",
            "expiryYear": "2120",
            "expired": false
        }
    ]
}
```

Where:

- `balance` is the remaining balance available for use on the gift card
- `expiryDay`, `expiryMonth` & `expiryYear` provide the date on which the card is scheduled to expire
- `expired` is a boolean which indicates whether the card is considered an expired card

Retrieve Gift Card Balance using List Instruments

You can also retrieve the balance of multiple gift cards in your wallet using Get Payment Instruments List by adding `include=GC_BALANCE` optional query parameter. Refer to Retrieve a Customers Wallet for more information about List Instruments.

cURL | JavaScript | Swift | Kotlin

```
curl --location \
--request GET 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/instruments?include=GC_BAL
--header 'accept: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--header 'Content-Type: application/json' \
```

List Instrument including Gift Card Balance Outcome

JSON


```
{
    "data": {
        "creditCards": [],
        "giftCards": [
            {
                "paymentInstrumentId": "251****",
                "paymentToken": "a61aa56d-c9db-****-****-***********",
                "status": "VERIFIED",
                "createdOn": "2022-08-23T09:27:40.478+10:00",
                "lastUpdated": "2022-08-23T09:28:26.596+10:00",
                "lastUsed": "2022-08-23T09:28:26.105+10:00",
                "primary": true,
                "allowed": true,
                "programName": "Wish eGift Card",
                "cardSuffix": "8480",
                "stepUp": {
                    "type": "REQUIRE_PASSCODE",
                    "mandatory": false
                },
                "instrumentType": "GIFT_CARD",
                "balanceDetail": {
                    "balance": 709.45
                }
            },
            {
                "paymentInstrumentId": "251****",
                "paymentToken": "f896b20a-34c0-****-****-***********",
                "status": "UNVERIFIED_PERSISTENT",
                "createdOn": "2022-08-23T09:19:04.186+10:00",
                "lastUpdated": "2022-08-23T09:27:40.472+10:00",
                "primary": false,
                "allowed": true,
                "cardSuffix": "6401",
                "programName": "Wish eGift Card",
                "stepUp": {
                    "type": "REQUIRE_PASSCODE",
                    "mandatory": false
                },
                "instrumentType": "GIFT_CARD",
                "balanceDetail": {
                    "balance": 812.35
                }
            }
        ],
        "paymentAgreements": []
    },
    "meta": {}
}
```


Where:

- `paymentInstrumentId` is the payment token of the associated gift card saved in the customer's wallet.
- `paymentToken` is the payment token unique GUID of the associated gift card saved in the customer's wallet
- `status` is either VERIFIED or UNVERIFIED_PERSISTENT. Verified indicates that a successful verification or purchase has occurred using the instrument. Unverified indicates that the card has not yet been verified or used in a purchase.

- `lastUpdated` is the date the gift card information was last updated.
- `primary` indicates whether the gift card is the primary instrument in the customer's wallet to make payments.
- `allowed` indicates whether the gift card is an allowed payment method based on your merchant config with Wpay.
- `cardSuffix` provides the last 4 digits of the tokenized gift card for display purposes.
- `programName` is the gift card name given at the point of tokenization.
- `stepUp – mandatory` will align with the `stepUp – type` indicator.
- `instrumentType` is always GIFT_CARD for gift card instrument.
- `balanceDetail – balance` is the remaining balance available for use on the gift card.

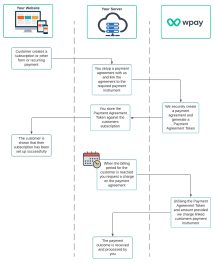
Recurring Payments

Overview

A payment agreement allows you to setup a recurring payment as part of a subscription or other similar service between your business and customer. These payment agreements allow you to easily charge the customer on the payment instrument linked to the payment agreement when required.

Currently, payment agreements are only supported for the credit/debit cards & PayPal methods when saved in the customer's wallet.

How it works



Payment Agreements

A payment agreement can be created utilising an allowed payment instrument type (currently only debit cards, credit cards and PayPal are supported instrument types for payment agreements) when saved in the customer's wallet. When one of these allowed payment instrument types has been tokenized in your customer's wallet you are able to create a payment agreement.

Create a Payment Agreement

Utilising a saved payment instrument you can create a payment agreement which will allow you to charge the agreement when required based on your billing cycle.

Payment agreements can only be created with instruments that are saved to the customer's wallet and cannot be created using single-use instruments.

> 📘 **Scheduled Payments**
>
> Note that metadata captured as part of the payment agreement such as the charge frequency and amounts do not result in charges automatically being processed by Wpay. As the merchant you will need invoke the charge at the desired date to process the payment.

cURL | JavaScript | Swift | Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/payments/
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw '{
    "data": {
                "clientReference": "UNIQUE_CLIENT_REFERENCE",
                "orderNumber": "UNIQUE_ORDER_NO",
        "customerRef": "12345",
        "description": "Description for Recurring Payment",
        "billingAddress": {
            "firstName": "John",
            "lastName": "Doe",
            "email": "jdoe@wpay.com.au",
            "company": "Wpay",
            "extendedAddress": "4th Floor",
            "streetAddress": "407 Elizabeth Street",
            "suburb": "Surry Hills",
            "stateOrTerritory": "NSW",
            "postalCode": "2010",
            "countryCode": "AU"
        },
        "paymentAgreement": {
            "paymentInstrumentId": "215726",
            "type": "RECURRING",
            "startDate": "2022-09-01T00:00:00.000+1100",
            "endDate": "2023-12-31T23:59:59.999+1100",
            "chargeFrequency": "WEEKLY",
            "chargeAmount": 25.99,
            "immediateCharge": true
        }
    },
    "meta": {
        "fraud": {
            "provider": "cybersource",
            "version": "CyberSourceTransaction_1.101",
            "format": "XML",
            "responseFormat": "XML",
            "message": "<?xml version=\"1.0\" encoding=\"Windows-1252\"?>\r\n<RequestMessage xmlns:xsd=\'
        },
        "challengeResponses": [
            {
                "instrumentId": "215726",
                "type": "STEP_UP",
                "token": "tokenise-stepup-token"
            }
        ]
    }
}'
```

Where:

- `clientReference` is your application-specific reference number. This number should uniquely identify the transaction in your system
- `orderNumber` is your order number of the transaction as generated during the order creation.

- `paymentInstrumentId` is the tokenized payment instrument which you want to link the payment agreement to. This is either from the frames SDK response or the list payment instruments response.
- `type` can be set to one of the following: ADHOC, INSTALLMENT or RECURRING. Depending on the value you specify, Wpay will ensure the correct data element values are then passed to the Issuers as per the latest scheme credential on file mandates.
  - ADHOC: payments can be charged at any time between the effective `startDate` and `endDate` (once or multiple times).
  - INSTALLMENT: payments processed in multiple installments during the valid period of the agreement.
  - RECURRING: recurring charges (e.g. weekly, monthly, etc) - normally used in subscription-based services.
- `startDate` is the date from which you want the payment agreement to be effective. This field cannot be set to a date in the past and if you would like the start date to be from NOW then you may leave this field out of the request and we will set the start date as a default to NOW.
- `endDate` is the last date to which you want the payment agreement to be effective. This date cannot be set to a date before the `startDate`. If your recurring payment has no end date then this field can be omitted or set to `null`.
- `chargeFrequency` can be captured as additional information when capturing a payment agreement, this will be returned when viewing a payment agreement so you can display details of the payment agreement back to your customers. Valid values are: `ADHOC` or `WEEKLY`, `MONTHLY` and `ANNUALLY`.
- `chargeAmount` is the amount for which the payment agreement is being set up, this will be returned when viewing a payment agreement so you can display details of the payment agreement back to your customers.
- `immediateCharge` is set to true where you wish to process a change on the payment agreement as part of the creation process. When this is set to true a charge request will be made on the payment agreement and the outcome of the charge returned in the response. This will default to false.
- `fraud` is the CyberSource fraud payload for the transaction. Please see the Fraud Detection page for more information on the fraud process and fraud payload.
- `challengeResponses` contains the payment challenge data such as the step-up token for CVV capture.

Payment Agreement Outcome

When a payment agreement has been successfully setup we will provide back the relevant information on your newly created payment agreement.

> 📘 **Payment Agreement Payment Token**
>
> The **paymentToken** is important as this will be required when editing, deleting or charging the payment agreement. This can be saved in your scheduling system for charging or retrieved from the customers saved payment agreements utilizing the view payment agreement functionality.

Immediate Charge False | Immediate Charge True


```
{
    "data": {
        "type": "RECURRING",
        "paymentInstrumentId": "215726",
        "paymentInstrumentType": "CREDIT_CARD",
        "startDate": "2022-08-31T23:00:00.000+10:00",
        "endDate": "2023-12-31T23:59:59.999+11:00",
        "chargeFrequency": "WEEKLY",
        "chargeAmount": 25.99,
        "scheme": "MASTERCARD",
        "expiryMonth": "01",
        "expiryYear": "23",
        "cardSuffix": "0407",
        "paymentToken": "e8605265-d983-4b3d-8a48-1204eb43ea6e",
        "description": "Description for Recurring Payment",
        "fraudResponse": {
            "clientId": "6474024752216768604008",
            "reasonCode": "480",
            "decision": "REVIEW",
            "riskInformation": [
                {
                    "name": "Review rule1",
                    "decision": "REVIEW"
                }
            ]
        }
    },
    "meta": {}
}
```


View a Payment Agreement

Utilising our get payment agreements functionality you are able to retrieve all payment agreements which you have setup for your customer or a single payment agreement. This can be used to display the payment agreements your customer has setup with you or to retrieve the `paymentToken` linked to the payment agreement.

View all payment agreements for a customer

cURL | JavaScript | Swift | Kotlin

```
curl --location --request GET 'https://dev-api.wpay.com.au/wow/v1/pay/instore/customer/payments/agreement
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw ''
```

This will return all the configured payment agreements for the customer

JSON


```
{
    "data": {
        "paymentAgreements": [
            {
                "paymentInstrumentId": "215726",
                "paymentToken": "e8605265-d983-4b3d-8a48-1204eb43ea6e",
                "status": "VERIFIED",
                "createdOn": "2021-10-01T11:31:49.574+10:00",
                "lastUpdated": "2021-10-01T11:31:49.574+10:00",
                "primary": false,
                "allowed": true,
                "endDate": "2023-12-31T23:59:59.999",
                "startDate": "2022-08-31T23:00",
                "chargeAmount": 25.99,
                "chargeCycle": "0",
                "type": "RECURRING",
                "chargeFrequency": "WEEKLY",
                "cardSuffix": "0407",
                "expiryMonth": "01",
                "expiryYear": "23",
                "scheme": "MASTERCARD",
                "expired": false,
                "updateURL": "https://dev-api.wpay.com.au/wow/v1/pay/paymentagreements/e8605265-d983-4b3
                "stepUp": {
                    "type": "CAPTURE_CVV",
                    "mandatory": true,
                    "url": "https://iframe.dev1.payments.woolworths.com.au/container-ws/getCaptureFrame/
                    "sessionId": "a1e23906-8889-47ef-a086-7b81bbc6c294"
                }
            },
            {
                "paymentInstrumentId": "215728",
                "paymentToken": "2e6f5c64-02fd-40b5-82af-27974093025e",
                "status": "VERIFIED",
                "createdOn": "2021-10-01T12:01:40.945+10:00",
                "lastUpdated": "2021-10-01T12:01:40.945+10:00",
                "primary": false,
                "allowed": true,
                "chargeAmount": 149.99,
                "type": "RECURRING",
                "endDate": "2023-12-31T23:59:59.999",
                "chargeFrequency": "MONTHLY",
                "startDate": "2022-08-31T23:00",
                "chargeCycle": "0",
                "cardSuffix": "0608",
                "expiryMonth": "02",
                "expiryYear": "23",
                "scheme": "VISA",
                "expired": false,
                "updateURL": "https://dev-api.wpay.com.au/wow/v1/pay/paymentagreements/2e6f5c64-02fd-40b5
                "stepUp": {
                    "type": "CAPTURE_CVV",
                    "mandatory": true,
                    "url": "https://iframe.dev1.payments.woolworths.com.au/container-ws/getCaptureFrame/
                    "sessionId": "a1e23906-8889-47ef-a086-7b81bbc6c294"
```


View a specific payment agreement

By specifying the `paymentToken` as part of the request URL you are able to retrieve the details for a specific payment agreement.

cURL JavaScript Swift Kotlin

```
curl --location --request GET 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/payments/a
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw ''
```

This will return the specified payment agreements details

JSON


```
{
    "data": {
        "paymentInstrumentId": "215728",
        "paymentToken": "2e6f5c64-02fd-40b5-82af-27974093025e",
        "status": "VERIFIED",
        "createdOn": "2021-10-01T12:01:40.945+10:00",
        "lastUpdated": "2021-10-01T12:01:40.945+10:00",
        "primary": false,
        "allowed": true,
        "chargeAmount": 149.99,
        "type": "RECURRING",
        "endDate": "2023-12-31T23:59:59.999",
        "chargeFrequency": "MONTHLY",
        "startDate": "2022-08-31T23:00",
        "chargeCycle": "0",
        "cardSuffix": "0608",
        "expiryMonth": "02",
        "expiryYear": "23",
        "scheme": "VISA",
        "expired": false,
        "updateURL": "https://dev-api.wpay.com.au/wow/v1/pay/paymentagreements/2e6f5c64-02fd-40b5-82af-27
        "stepUp": {
            "type": "CAPTURE_CVV",
            "mandatory": true,
            "url": "https://iframe.dev1.payments.woolworths.com.au/container-ws/getCaptureFrame/cvv/a3697
            "sessionId": "a36977a6-d9b9-43c7-a13d-7855c0e9658b"
        }
    },
    "meta": {}
}
```


Update a Payment Agreement

Existing payment agreements can be updated to utilise a different instrument in the customer's wallet as well as other information such as the charge frequency, amount, start and end date.

cURL JavaScript Swift Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/payments/
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--data-raw '{
    "data": {
        "clientReference": "00815690",
        "customerRef": "29332770",
        "orderNumber": "36915651",
        "billingAddress": {
            "firstName": "John",
            "lastName": "Doe",
            "email": "jdoe@wpay.com.au",
            "company": "Wpay",
            "extendedAddress": "4th Floor",
            "streetAddress": "407 Elizabeth Street",
            "suburb": "Surry Hills",
            "stateOrTerritory": "NSW",
            "postalCode": "2010",
            "countryCode": "AU"
        },
        "paymentAgreement": {
            "paymentInstrumentId": "1600183",
            "type": "RECURRING",
            "startDate": "2022-09-01T00:00:00.000+1100",
            "endDate": "2023-12-31T23:59:59.999+1100",
            "chargeFrequency": "MONTHLY",
            "chargeAmount": 149.99,
            "description": "Description for Recurring Payment"
        }
    },
    "meta": {
        "fraud": {
            "provider": "cybersource",
            "version": "CyberSourceTransaction_1.101",
            "format": "XML",
            "responseFormat": "XML",
            "message": "<?xml version=\"1.0\" encoding=\"Windows-1252\"?>\r\n<RequestMessage xmlns:xsd=\'
        },
        "challengeResponses": [
            {
                "instrumentId": "1600183",
                "type": "STEP_UP",
                "token": "tokenise-stepup-token"
            }
        ]
    }
}'
```

Delete a Payment Agreement

Should you wish to delete a payment agreement you can utilise our delete payment agreement functionality. This will remove the payment agreement from your customer and it can no longer be retrieved when viewing the payment agreement list or trying to retrieve the specific payment agreement.

🚧 Expiring a Payment Agreement

Should you or your customer wish to cancel a payment agreement, rather than deleting it, update the payment agreement and set the effective-to date to the date you wish to expire the agreement.

cURL JavaScript Swift Kotlin

```
curl --location --request DELETE 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/merchant/payment
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--data-raw ''
```

Charging Payment Agreements

A payment agreement can be charged when the client's billing schedule is reached by simply calling the charge API with the payment agreement token for the customer and an amount.

The charge will be processed against the payment instrument linked to the payment agreement.

📘 Billing Scheduler

Please note that the scheduling of the billing is not coordinated by Wpay and instead will need to be processed by you or another payments provider.

cURL JavaScript

```
curl --location --request PUT 'https://pt-api.wpay.com.au/wow/v1/pay/instore/merchant/payments/agreements
--header 'Content-Type: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
--data-raw '{
  "data": {
    "paymentToken": "ce3ec653-199b-48c2-####-############",
    "amount": 2.99,
    "clientReference": "UNIQUE_CLIENT_REFERENCE",
    "orderNumber": "UNIQUE_ORDER_NO",
     "transactionType": {
        "creditCard": "PURCHASE"
        },
    "customerRef": "UNIQUE_CUSTOMER_REFERENCE"
  },
  "meta": {
    "fraud": {
        "provider": "cybersource",
        "version": "CyberSourceTransaction_1.101",
        "format": "XML",
        "responseFormat": "XML",
        "message": "<?xml version=\"1.0\" encoding=\"Windows-1252\"?>\r\n<RequestMessage xmlns:xsd=\"http
    }
  }
}'
```

Where:

- `paymentToken` is the payment agreement token generated when the payment agreement is created.
- `amount` is the amount you wish to charge the specified payment agreement for.
- `clientReference` is your application-specific reference number. This number should uniquely identify the transaction in your system
- `orderNumber` is your order number of the transaction as generated during the order creation.
- `transactionType` is the method by which you want to process the transaction, either as a preauth or as a purchase.
- `fraud` is the CyberSource fraud payload for the transaction

> 🚧 **Restricted API's**
>
> The Payment Agreement API's are IP restricted and do not allow unauthenticated server side calls. Your servers will need to be whitelisted to allow charges to be processed.

Charge Outcome

Following the charge, an outcome of the transaction is returned with the status of the payment agreement charge as well as the transaction references.

JSON


```
{
    "data": {
        "transactionId": "0349cfd3-3daa-4981-965f-cfea9ef749ec",
        "merchantReferenceId": "40252433",
        "merchantId": "10001",
        "paymentRequestId": "e7bacfb6-a59c-4578-89b0-6d7a56c37f5f",
        "grossAmount": 6.99,
        "clientReference": "36952873",
        "executionTime": "2021-12-20T06:34:57.068Z",
        "type": "PAYMENT",
        "status": "APPROVED",
        "instruments": [
            {
                "paymentInstrumentId": "1966757",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "PAYMENT",
                        "executionTime": "2021-12-20T06:34:57.068Z",
                        "amount": 6.99,
                        "paymentTransactionRef": "1000000022049853",
                        "status": "APPROVED"
                    }
                ]
            }
        ],
        "subTransactions": [
            {
                "transactionReceipt": "1000000022049853",
                "paymentToken": "b5c65f37-a78c-45b5-b4e9-b77cdd3716e0",
                "paymentAgreement": {
                    "type": "RECURRING",
                    "paymentInstrumentId": "1966757",
                    "paymentInstrumentType": "CREDIT_CARD",
                    "startDate": "2021-12-20T17:34:51.227+11:00",
                    "endDate": "2023-12-31T23:59:59.999+11:00",
                    "chargeFrequency": "MONTHLY",
                    "chargeAmount": 6.99,
                    "expiryMonth": "11",
                    "cardSuffix": "1111",
                    "expiryYear": "22",
                    "scheme": "VISA"
                },
                "fraudResponse": {
                    "clientId": "6399820980916196203010",
                    "reasonCode": "100",
                    "decision": "ACCEPT"
                },
                "extendedTransactionData": [
                    {
                        "field": "bin",
                        "value": "444433"
                    },
                    {
                        "field": "stan",
```


Transaction Outcomes

The status of your payment request is provided in the response as an `externalServiceMessage`. The outcome of a payment agreement charge can either be **APPROVED** or **REJECTED**.

Transaction Reference

The transaction ID `transactionId` is returned as part of a charge payment agreement response and is a unique reference to the charge request. A payment can be made up of multiple sub-transactions each with a `paymentTransactionRef` which is the unique reference of the sub-transaction.

The `transactionId` and `paymentTransactionRef` are important as these will be a required piece of information when either completing a pre-authorised transaction, voiding the transaction or processing a refund.

Fraud Outcomes

Should you use the fraud services of Wpay the fraud outcome (`fraudResponse`) in the payment response will give you the outcome of the fraud decision manager check. This can be **Accept**, **Reject** or **Review**.

- **Accept**: No fraud detected. Advice is that payment can proceed.
- **Reject**: Fraud likely. Advice is that payment should be voided or refunded.
- **Review**: Fraud potential. Payment should be manually reviewed to determine fraud decision.

Ancillary Services

Merchant Profile

View Merchant Profile

Your merchant profile contains important information for how your merchant has been set up with Wpay and provides details that can be utilised in subsequent service calls meaning you don't need to store the information if you do not wish to.

cURL JavaScript

```
curl --location \
--request GET 'https://{{environment}}-api.wpay.com.au/wow/v1/pay/merchants/profile' \
--header 'accept: application/json' \
--header 'X-Api-Key: {{yourApiKey}}' \
```

Merchant Profile Outcome

Your currently configured merchant profile with Wpay will be returned showing important information such as:

- Allowed payment methods showing which payment methods are currently configured including additional details such as the allowed bins for gift cards and the allowed networks for credit cards.
- Token and key data for 3rd party payment providers such as PayPal.

JSON


```
{
  "allowedPaymentMethods": {
    "giftCard": {
      "allowedBins": [
        "600300",
        "628000"
      ],
      "serviceStatus": "ENABLED",
      "pinAlwaysRequired": false
    },
    "creditCard": {
      "allowedNetworks": [
        "AMEX",
        "MASTERCARD",
        "JCB",
        "VISA",
        "DINERS"
      ],
      "allowedTransactionTypes": [
        "PREAUTH",
        "PURCHASE"
      ],
      "serviceStatus": "ENABLED"
    },
    "payPal": {
      "clientToken": "ey**********************em=",
      "serviceStatus": "ENABLED"
    },
    "googlePay": {
      "publicKey": null,
      "publicKeyHash": null,
      "publicKeyExpiry": null,
      "merchantId": "11111",
      "merchantName": "Test Merchant",
      "creditCard": {
        "allowedNetworks": [
          "AMEX",
          "MASTERCARD",
          "VISA"
        ],
        "allowedTransactionTypes": [
          "PURCHASE",
          "PREAUTH"
        ]
      },
      "debitCard": {
        "allowedNetworks": [],
        "allowedTransactionTypes": [
          "PURCHASE",
          "PREAUTH"
        ]
      },
      "serviceStatus": "ENABLED"
    },
    "applePay": {
```

Transaction History

All transactions processed through Wpay can be retrieved along with the important transaction details you or your customer's may be interested in.

The transaction history can be retrieved for a specific customer, for all your transactions as a merchant across customers or a specific transaction.

Transaction History List

The transaction history lists return multiple transactions either for a specific customer or all your transactions for your merchant. These services utilise Pagination to keep the responses quick and minimise the data returned.

Customer's Transaction History List

The customer transaction history list will show all transactions for a specific customer.

cURL

```
curl --location --request GET 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/transactio
--header 'X-Api-Key: {{yourAPIKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
```

Merchant's Transaction History List

The merchant transaction history list will show all transactions for you as a merchant across customers.

cURL

```
curl --location --request GET 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/merchant/transactio
--header 'content-type: application/json' \
--header 'X-Api-Key: {{yourAPIKey}}' \
```

> 🚧 **Guest Customer's Transaction History**
>
> A guest customer's transaction history will be viewable through the merchant's transaction history list & transaction history detail API calls, however, the customer list and detail APIs will not return any transaction history data for a specific guest user as a guest users payments are anonymous.

Transaction History Detail

Using a transaction reference you are also able to get a single transaction and retrieve the transaction details for the specified transaction.

Customer's Transaction History Detail

The customer transaction history detail will retrieve the specified transaction for the customer and uses the bearer token to ensure that customers do not see other customers transactions.

cURL

```
curl --location --request GET 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/transactio
--header 'X-Api-Key: {{yourAPIKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
```

Merchant's Transaction History Detail

The merchant transaction history detail will retrieve the specified transaction for any transaction through you as the merchant.

cURL

```
curl --location --request GET 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/merchant/transactio
--header 'content-type: application/json' \
--header 'X-Api-Key: {{yourAPIKey}}' \
```

Transaction History Outcome

JSON


```
{
    "data": {
        "transactions": [
            {
                "type": "PAYMENT",
                "status": "APPROVED",
                "walletId": "28e76b0e-ee08-43cf-8cae-5bf21b3b6fbf",
                "grossAmount": 199.99,
                "instruments": [
                    {
                        "transactions": [
                            {
                                "type": "PAYMENT",
                                "amount": 199.99,
                                "status": "APPROVED",
                                "executionTime": "2021-10-06T01:41:35.790Z",
                                "paymentTransactionRef": "1000000007828200"
                            }
                        ],
                        "instrumentType": "CREDIT_CARD",
                        "paymentInstrumentId": "215914"
                    }
                ],
                "executionTime": "2021-10-06T01:41:35.321Z",
                "transactionId": "33095ed8-ccdb-40eb-88f8-f937d3ee625f",
                "clientReference": "9988776655443322",
                "subTransactions": [
                    {
                        "fraudResponse": {
                            "clientId": "6334844975146683503012",
                            "decision": "REJECT",
                            "reasonCode": "481"
                        },
                        "partialSuccess": false,
                        "paymentResponses": [
                            {
                                "threeDS": {
                                    "car": null,
                                    "sli": null,
                                    "dsTransID": null
                                },
                                "receiptData": {
                                    "scheme": "VISA",
                                    "cardSuffix": "0608",
                                    "expiryYear": "23",
                                    "expiryMonth": "02"
                                },
                                "paymentToken": "aaf5f76d-1685-42f1-a046-30dea061467f",
                                "externalServiceCode": "00",
                                "paymentInstrumentId": "215914",
                                "paymentInstrumentType": "CREDIT_CARD",
                                "paymentTransactionRef": "1000000007828200",
                                "externalServiceMessage": "APPROVED",
                                "extendedTransactionData": [
                                    {
```

Pagination

Where a service would return a large amount of information we've implemented pagination to keep responses quick and reduce the amount of data that you need to handle.

By default we will limit these services to return 25 records per page however this can be configured to return as many records as you require.

This can be done by specifying the page size and page number.

- `page-size` is the maximum number of records per page you wish to return
- `page` is the page number in the total returned records.

This is specified in the request URL for example:
`https://{{environment}}.wpay.com.au/wow/v1/pay/instore/merchant/transactions?page-size=2&page=1`

Services that support pagination will also indicate the total record count and records being viewed as part of the metadata, e.g:

JSON

```
"meta": {
        "recordCount": 2,
        "totalRecords": 155
    }
```

Where

- `recordCount` is the number of records returned on the specified page
- `totalRecords` is the total number of records which are available in the response.

SDKs

Overview

Our software development kits are extremely simple to integrate with and enable you to capture and tokenize credit cards on your site in a safe, secure way and PCI compliant way.

Web SDK libraries

Javascript

Frames SDK

View on Github

Frames

> **ElementsSDK has been renamed to FramesSDK**
>
> Please see our changelog for more information.

Our Frames allow you to start accepting online payments quickly and easily. It's simple to integrate into your website checkout, accepts online payments from all major credit cards and is customisable to your brand through our styling options.

Capture Card Details Securely

Our frames allow your customers to capture their card payment details directly on your checkout page on your website or in your app and are highly customisable to the look and feel of your brand.

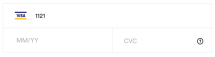
We process these and exchange them for a secure token which you can either use as a single-use for your payments or store this within your customer's wallet for easy checkout with saved cards.

This token will be used to request a payment, set up subscriptions and charge these subscriptions all without having to process or store any sensitive card information yourself.

Frames Option

Wpay provide easy to use single line or multi line frames for you to embed into your website or app. This will allow you to get the look and feel you want for your checkout page without sacrificing any of the functionality or security.

Multi Line Frame:



Single Line Frame:

How do Frames and Tokens work?

For more information on tokenizing a payment instrument, see Tokenizing a Payment Instrument.
For more information about how to integrate Frames into your site, see Integrate Frames.
For options on how to style and customise Frames to fit your website, see Frames Customisation and Styling.

Integrate Frames

How it works

To integrate Frames into your site there are a few steps that you need to follow:

1. Initialise the frames SDK and host the frames:
    i. Define a placeholder element where the capture elements will be inserted into the page. This element needs to have an `id` defined.
    ii. Start a new card capture action, the action will handle all interactions with your elements such as; creation, validation and submission. This call will need to be repeated between subsequent card captures.
    iii. Create the Frames capture element by calling the `createFramesControl` method on the action and passing in the element type and the `id` of the DOM element that you would like to attach it to.
2. Validate the captured data using events:
    i. Once the user has entered their credit card details you are going to want to validate that all the captured data is valid. To do so use the raised error events.
    ii. You are also going to want to validate that all the required information has been captured. To do so use the onBlur and onFocus events.
3. Tokenize the captured card:
    i. Once you have confirmed the data is valid you will want to submit the captured information and tokenize the card. To do this add a Submit button to the page calling the `submit` function on the action. This will run the card validation and submit the form if successful.
    ii. Once a card capture action has been successfully submitted you'll need to complete the action by calling the `complete` method on the action.

Initialising the Frames

Instantiate the Frames

TypeScript Kotlin Swift

```
import * as frames from '@wpay/frames';

const environment = 'pt-api';
const baseUrl = `https://${environment}.wpay.com.au/wow/v1`;

const apiKey = 'YOUR-API-KEY';
const framesApiBaseUrl = `${baseUrl}/pay/instore`;
const walletApiBaseUrl = `${baseUrl}/pay`;

//Instantiate the frames SDK, this will allow us to capture user card information.
const framesSDK = new frames.FramesSDK({
    apiKey: apiKey,
    authToken: `Bearer ${authorizationToken}`,
    apiBase: framesApiBaseUrl,
    logLevel: frames.LogLevel.DEBUG
});

// Once the page has loaded, initialise a new card capture action.
action = framesSDK.createAction(frames.ActionTypes.CaptureCard);
await action.start();
```

Where:

- the `package.json` file should contain the following dependencies:

```
  "@api-sdk-creator/axios-http-client": "^0.1.7",
  "@types/typescript": "^2.0.0",
  "@wpay/frames": "^2.0.3",
  "@wpay/sdk": "^1.8.7"
```

Card Capture Action Options

Save Card

By default, the card will attempt to save to the customer's wallet unless otherwise specified. If you would like to specify whether to save the card to the customer's wallet on tokenization the `save` property can be passed as part of the options when initialising the capture card action.

The 'save' option can be overridden during the tokenize step should you wish to change this after the frame has already been initialised. For example, where you provide a save check-box which the customer can select based on their preference to save the card to their wallet for easier future check-out or not. This is done during the completion call.

TypeScript Kotlin Swift

```
framesSDK.createAction(
  frames.ActionTypes.CaptureCard,
  { save: true }
);
```

Verify Card

Card verification is the process of performing a 1c pre-auth on the card at the point of tokenizing to ensure that the card is valid.

Card verification should be done when you are not wanting to immediately process a payment following the saving of a card. For example, in a use case where the customer can add and remove cards from their wallet without having to make an immediate payment.

> 🚧 **Verification Consumes the Step Up Token**
>
> When a 1c pre-auth verification is done as part of tokenizing a card it will consume the step up token generated as part of the card capture. Should you try and verify a card and immediately process a payment you will be required to capture the CVV again. Should you be processing a payment with a card immediately after tokenizing it you should set verify to **false**.

By default, the card verification is disabled on card capture. If you would like to capture a card and enforce verification the `verify` property can be passed as part of the options when initialising the capture card action.

TypeScript | Kotlin | Swift

```
framesSDK.createAction(
        frames.ActionTypes.CaptureCard,
  { verify: true }
);
```

An example of how to set multiple optional parameters (`save` and `verify`) when initiating the frame.

TypeScript | Kotlin | Swift

```
import * as frames from '@wpay/frames';

import { ApiTokenType } from '@wpay/sdk';
import * as frames from '@wpay/frames';

const apiKey = 'YOUR-API-KEY';
const authorizationToken: ApiTokenType = 'xxx'; // Obtained via serverside API token endpoint
const environment = 'pt-api'
const baseUrl = `https://${environment}.wpay.com.au/wow/v1`;
const framesApiBaseUrl = `${baseUrl}/pay/instore`;

//Instantiate the frames SDK, this will allow us to capture user card infromation.
const framesSDK = new frames.ElementsSDK(
    apiKey,
    `Bearer ${authorizationToken}`,
    framesApiBaseUrl,
    frames.LogLevel.DEBUG
);

framesSDK.createAction(
  frames.ActionTypes.CaptureCard,
  {
    save: true,
        verify: false
  }
);
```

Host the Frames

Multi Line Frame:

The multi-line frame is actually comprised of separate elements which can be arranged as required to achieve the desired look and feel for your site. In order, to generate the multi-line frame you will need to generate each element as shown here:

TypeScript HTML Kotlin Swift

```
// Link the HTML cardCapture ids: `cardCaptureCardNo`, `cardCaptureExpiry` and `cardCaptureCVV`
// with their respective frames SDK fields: `CardNo`, `CardExpiry` and `CardCVV`
action.createFramesControl('CardNo', 'cardCaptureCardNo', options);
action.createFramesControl('CardExpiry', 'cardCaptureExpiry', options);
action.createFramesControl('CardCVV', 'cardCaptureCVV', options);
```

Single Line Frame

The single-line frame is a single element that allows credit card capture but is limited in how the fields can be ordered. In order, to generate the single-line frame you will need to generate the card group as shown here:

TypeScript HTML Kotlin Swift

```
// Populate the HTML placeholder div id `cardCapturePlaceholder`
// with the card capture inputs fields using the `CardGroup`
action.createFramesControl('CardGroup', 'cardCapturePlaceholder');
```

Autofill for Card Capture fields using Single or Multi Line Frame

- Single and multi frame card capture support autofill out of the box with out the need for the developer to do anything extra.
- When using a signed certificate on a secure site autofill will work for the Card PAN, Expiry Month and Expiry Year.
- If using a card that is attached to a google pay wallet then autofill will populate the CVV.
- When using a development or local insecure environment you will need to use a self signed certificate to enable autofill to work.

Frame Events & Methods

Frames Events

The Card Capture Frames listens to onFocus and onBlur events fired from the frames SDK to display validation errors and to provide additional information to cater for your specific use case. For example, enabling a "pay" button once all fields are complete you need to know when controls are visited.

If you would like to listen in to these events you can do so by adding an event listener to the placeholder element in much the same way as you do for validation. e.g.

```
document .getElementById('cardCaptureCardNo') .addEventListener( Frames.FramesEventType.OnBlur, () => {
// Do something onBlur } );
```

In the below example the submit button is enabled when all the credit card fields have been visited and there is no current validation error:

TypeScript Kotlin Swift

```
import { createAxiosHttpClient } from '@api-sdk-creator/axios-http-client';
import {
  ApiTokenType, createCustomerSDK, WPayCustomerApi, WPayCustomerOptions
} from '@wpay/sdk';

const apiKey = 'YOUR-API-KEY';
const authorizationToken: ApiTokenType = 'xxx'; // Obtained via serverside API token endpoint
const environment = 'pt-api';
const walletApiBaseUrl = `https://${environment}.wpay.com.au/wow/v1/pay`;

let action: any;
let customerSDK: WPayCustomerApi;
let submitCardBtn: HTMLButtonElement;
let makePaymentBtn: HTMLButtonElement;

function setupForCardCapture() {
  //Instantiate the customer API
  const options: WPayCustomerOptions = {
    apiKey: apiKey,
    baseUrl: walletApiBaseUrl,
    accessToken: authorizationToken,
  };
  customerSDK = createCustomerSDK(createAxiosHttpClient, options);

  // Once the page has loaded, initialise a new card capture action.
  action = framesSDK.createAction(frames.ActionTypes.CaptureCard);
  await action.start();

  // Populate the placeholder div with the card capture inputs. In this case we
  // are using the 'CardGroup'
  action.createFramesControlcreateFramesControlcreateElement('CardGroup', 'cardCapturePlaceholder');

  // Add OnValidated Eventlistener which will cause the updateErrors
  // function to be called if a validation error is encoutned in the
  // frames SDK while entering a Credit Card details.
  document.getElementById('cardCapturePlaceholder')!
    .addEventListener(
      frames.FramesEventType.OnValidated,
      updateErrors
    );

  // Add OnFocus Eventlistener which is fired when a field is focused in the frames SDK.
  // This enables you to know if all fields have been visited and the credit card is ready
  // for submission.
  document.getElementById('cardCapturePlaceholder')!
    .addEventListener(
      frames.FramesEventType.OnFocus,
      setVisitedStatus
    );

  // Add OnBlur Eventlistener which is fired when a field is exited in the frames SDK.
  // This enables you to check for errors and visited fields to see if the credit card
  // is ready for submission.
  document.getElementById('cardCapturePlaceholder')!
    .addEventListener(
```

Where:

- `createAxiosHttpClient` is imported from the node module `@api-sdk-creator/axios-http-client` and is used to create `createCustomerSDK` (a custom instance of the @wpay/sdk)
- `updateErrors` is an error handling function an example implementation is provided in the Error Handling section below.

Frames Methods

Additional methods on the action that might be useful:

- The `errors` method will provide a list of errors in the event validation failed on one or more of the card capture fields.
- The `clear` method will allow the user to clear all of the card capture fields.

Error Handling

These the predetermined error events that get raised from the frames SDK:

Frames SDK Error returned	Error Description
Card No. Required	The card number is missing.
Invalid Card No	The card number is not valid and has failed length and LUHN checks.
Invalid Expiry	The expiry date is invalid and does not match the required date format.
Incomplete Expiry	The expiry date is missing or incomplete.
Expired card	The expiry date is in the past.
Invalid CVV	The CVV is missing or invalid.

The below example shows an application of the frames validations typically applied during the card capture process and applies logic to determine if all fields have been visited before enabling the Save button.

TypeScript Kotlin Swift

```
const errorMap: Map<string, string> = new Map([
    ['Card No. Required', 'Please enter a valid card number.'],
    ['Invalid Card No.', 'Please enter a valid card number.'],
    ['Invalid Expiry', 'Please enter a valid expiry.'],
    ['Incomplete Expiry', 'Please enter a valid expiry'],
    ['Expired card', 'The expiry entered is in the past. Please enter a valid expiry.'],
    ['Invalid CVV', 'Please enter a valid CVV.']
]);

async function updateErrors() {
  const errors = action.errors();
  if (errors !== undefined && errors.length > 0) {
    // Display the validation error that has occurred.
    document.getElementById('cardCaptureErrors')!.innerHTML =
      `${errorMap.get(errors[0]) ? errorMap.get(errors[0]) : errors[0]}`;
  } else {
    // No validation error has occurred so clear the any error message.
    document.getElementById('cardCaptureErrors')!.innerHTML = "";
  }
}

const visitedStatus: any = {}
let enableSaveButton = false;

async function setVisitedStatus(event: any) {
    // When the cursor enters a field set the vistedStatus for this field to true.
    if (event && event.detail && event.detail.control) {
        visitedStatus[event.detail.control] = true;
        checkVisitedStatus();
    };
}

async function checkVisitedStatus() {
    const keys = [ 'cardNo', 'cardExpiry', 'cardCVV'];
    for (const key of keys) {
        if (!visitedStatus[key]) return;
    }
    if (action.errors() === undefined || action.errors().length === 0) {
        enableSaveButton = true;
    } else {
        enableSaveButton = false;
    }
    (document.getElementById("submitCard")! as HTMLButtonElement).disabled = !enableSaveButton;
}
```

Tokenize a Captured Card

In order to tokenize a captured card, you will need to call the submit action on each of the created elements and then call the complete action to tokenize the card.

Depending on the action options set when creating the capture action the card will tokenize and potentially perform a 1c verification and save to the customer's wallet.

Choosing whether to `save` the card details captured during card tokenization

At the point that the card is ready to be tokenized you can specify whether you wish to save the card to the customer's wallet or not by passing in the optional `save` parameter to the `complete` action. If the optional `save` parameter is not passed then the value set when initialising the card capture action will be used.

Submit Card for tokenization

TypeScript Kotlin Swift

```
let saveCapturedCard = false;
 await action.submit();

 //If the optional parameter saveCapturedCard is not passed,
 //the value set when initialising the card capture action will be used.
 const completeResponse = await action.complete(saveCapturedCard);
```

Example of full card capture with CVV Step Up Frames Primed

TypeScript Kotlin Swift

```
let tokenizedInstrument: any;

async function captureCard(saveCapturedCard: boolean = false) {
    //Capture the card inputted by the user
    try {
        //Submit the card capture inputs for tokenization
        await action.submit();

        //If the optional parameter saveCapturedCard is not passed it will be true by default.
        const completeResponse = await action.complete(saveCapturedCard);

        if (completeResponse.paymentInstrument) {
            //A new instrument has been tokenized
            tokenizedInstrument = {
                paymentInstrumentId:
                    completeResponse.paymentInstrument.itemId,
                stepUpToken: completeResponse.stepUpToken,
            };
        } else {
            //The instrument was already found to exist
            tokenizedInstrument = {
                paymentInstrumentId: completeResponse.itemId,
                stepUpToken: completeResponse.stepUpToken,
            };
        }

        // Display the payment section
        submitCardBtn.disabled = true;
        document.getElementById('instrumentIdDisplay')!.innerText =
            tokenizedInstrument.paymentInstrumentId;
        displaySection('paymentSection');
        console.log('Instrument details: ', tokenizedInstrument);
    } catch (error) {
        console.log('Error during card capture: ', error);
    }
}
```

When a card is successfully tokenized the user will receive the new tokenized card information, should the card have already existed in the customers wallet then the existing card information will be returned.

New Card Capture Response JSON | Existing Card Capture Response JSON


```
{
  "status": {
        "responseText": "ACCEPTED",
    "responseCode": "00",
    "auditID": "3f9143d8-d8d1-46df-88b6-e21f0acb66f0",
    "txnTime": 1635230585570,
    "error": null,
    "esResponse": null
  },
  "paymentInstrument": {
    "itemId": "217011",
    "paymentToken": "08935b8a-4f61-46ca-9ba0-661e8474782e",
    "status": "UNVERIFIED_PERSISTENT",
    "created": 1635230585570,
    "bin": "360502",
    "suffix": "0913",
    "expiryMonth": "08",
    "expiryYear": "22",
    "nickname": "",
    "scheme": "DINERS"
  },
    "stepUpToken": "tokenise-stepup-token",
    "fraudResponse": {
    "fraudClientId": null,
    "fraudReasonCd": null,
    "fraudDecision": null
  }
}
```


The `itemId` can be as the tokenized instrument when making a payment. For more context you can refer to the API guide for Making a Payment.

Step Up Process

Capture CVV

When utilising saved cards from your customer's wallet you will usually want them to capture their CVV and have this CVV information included as part of the payment information. This is known as the step-up process and results in a `step up token` which can be included as part of your payload when making a payment as part of your challenge response.

Step Up Token

Where a step up token is required your customer will need to capture their CVV as part of a their payment request. The CVV capture will result in a step up token which is a UUID representation of the tokenized CVV which needs to be provided as part of a payment.

3DS Integration

If you are interested in 3DS2, please review 3D Secure (3DS)

How it works

1. Start a new card step up action
2. Add the CVV element to the page
3. Create your frames element - specify the element you would like to create and the id of the dom element that you would like to attach the element to
4. Create the Frames Controller for the CVV element
5. Submit the CVV and get the Step Up Token which can then be used to make a credit card payment

TypeScript HTML

```
/*Start a new card step up action referencing your paymentInstrumentID*/
let action = sdk.createAction(
    FRAMES.ActionTypes.StepUp,
    {
        paymentInstrumentId: <YOUR PAYMENT INSTRUMENT ID>,
        scheme: 'VISA'
    }
);
action.start();

/*This will initialise a new step up action.
This call will need to be repeated between subsequent step up token requests.*/

action.createFramesControl('CardCVV', 'cardCaptureCVV');

/* Same as in above example submit the CVV capture and make the payment */
submitCardBtn = document.getElementById('submitCard') as HTMLButtonElement;
makePaymentBtn = document.getElementById(
  'makePayment'
) as HTMLButtonElement;

// This is where action.submit() called as defined in the captureCard function above.
submitCardBtn.onclick = captureCard;

// This is where the makePayment function as defined above is linked to the button.
makePaymentBtn.onclick = makePayment;
```

Where:

- The scheme should be returned as part of the paymentInstrument when you call the complete method during the initial tokenisation `VISA`, `MASTERCARD`, `AMEX` and `DINERS` are all valid values.

The high-level flow is shown in the above example to capture a CVV and receive a step up token is:

- Initialise a new step up action. Note that this call will need to be repeated between subsequent step up token requests.
- Adding the CVV element to the page i.e. `<div id="cardCaptureCVV"></div>`. Note that the SDK attaches new elements to div placeholders within your page using the element id

- After adding your placeholder you can now create your frames element. When creating an element pass in the type of the element you would like to create and the id of the dom element that you would like to attach it to. i.e. `action.createFramesControl('CardCVV', 'cardCaptureCVV');` . Loading the page should now display the credit card capture element, displaying card, expiry date and CVV.
- Once the user has entered their CVV, you are going to want to submit and create the step-up token. To do this add a Submit button to the page calling the submit function on the action i.e. `<button onClick="async function() { await action.submit()}">Submit</button>`
- Once successfully submitted an action needs to be completed. Do so by calling complete i.e. `let stepUpResult = await action.complete();`

You should now have a step up token which can be used as part of Making a Payment where required. For more context you can refer to the API guide for Making a Payment.

Logging

If you would like to see what is going on inside of the SDK, you can enable logging using the SDK constructor. Simply set the log level you would like to see and you should be able to see the log output in the console window. The log level is universal so applies to both the SDK and IFrame content.

Log Levels

- NONE = 0,
- ERROR = 50,
- INFO = 100,
- DEBUG = 200

Frames Customisation and Styling

In order, to ensure seamless integration with your user experience the frames controls allow for a range of styling and customisation options. Styling can either be applied to the container via CSS or in the scenario you want to make styling changes inside the frame the styling can be injected into the elements at run time via the options configuration.

Customise the Frame container

To customise the Frame container to better fit into your site design you can define normal CSS classes targetting the container. An element has several classes that can be used as targets for styling:

- `woolies-element`
- `container`
- `error` (only applied when the element has been validated and reported an error)

Here is an example of how one might use these classes to customise the style of the elements:

CSS

```
.woolies-element.container {
    border: 1px solid #d9d9d9;
    margin-left: 5px;
    padding: 5px;
}

.woolies-element.error {
    border: 1px solid #D0021B;
    background-color: #FFECEE;
}
```

Customising the Frames controls

If you would like to style the internal aspects of the frames elements such as font-family/style/weight you can do so using the options object.

The options object allows you to either apply styling to all elements under the control of an action, or scope your changes to only the elements you want to change.

For instance this example would set the height of the frame elements top 40px and apply a font size of 30 pixels to all elements:

JavaScript

```
let options = {
        "height": "40px",
  "style": {
                "fontSize": "30px"
  }
}
```

You can also style individual frames. For example, if you wanted to the text within the Card Number field to be bold while making the other fields italic you could do so like this:

JavaScript

```
let options = {
    "cardNo": {
        "style": {
            "fontWeight": "bold",
            "fontStyle": "normal"
        }
    },
    "style": {
        "fontStyle": "italic"
    }
}
```

The `cardNo` element is a little unique in that it has a sub element type that is used to show an image based on card scheme. This element can also be targeted and has an additional property allowing you to choose which side of the element it is displayed on.

This example moves the card type to the right and sets the image width to 50px to fill out the space:

JavaScript

```
let options = {
    "cardNo": {
        "cardType": {
            "layout": "right",
            "style": {
                "width": "50px"
            }
        }
    }
}
```

Sometimes you want to make customisations that can't be inlined such as; styling the placeholder text or have a different colour on hover. You are able to do this by injecting CSS styling into the frame using the CSS property.

This example sets the placeholder colour to blue and changes it to green on hover:

JavaScript

```
let options = {
    "css" : `
        input::placeholder {
            color: blue;
        }

        input:hover::placeholder {
            color: green;
        }
    `
}
```

Testing

Test Card Numbers

> **! Scope of test cards**
>
> These test cards numbers only work if you are integrating to the APIs found here.
>
> If you are integrating via new APIs, these test cards will not work.

Test Cards for Successful Purchases

The following cards can be used to test successful purchases. To test error scenarios please see cards and scenarios listed under error scenario test cards.

> **! Real Cards**
>
> Genuine card numbers should not be used for testing and will produce an error in our test environment. To simulate payments, use any of the following test card details provided in the tables below.

Scheme	Number	MM/YY	CVV
Visa	4265581110647303	08/25	143
Visa	4265581900642308	11/25	608
Visa	4940521800534554	11/25	234
Mastercard	5128998786159203	08/25	963
Mastercard	5353181800226466	08/25	488
Mastercard	5313575350116622	12/25	123
AMEX	374245455400001	08/25	4455
AMEX	376045698745230	08/25	7788
AMEX	376445698745232	12/25	1234
Diners	36050200070913	08/25	368
Diners	36436513701486	08/25	234

3DS Test Cards

3DS test cards are specific to 3DSv2 validation, below the liability shift and response is outlined for each card in each scenario.

Expiry and CVV

Use the card number specified in the test with the card’s expiration date set to the month of January and the current year plus three. For example, for 2022, use 2025.

The CVV is not validated for 3DS

Scheme	Number	Scenario	Liability Shift	3DS Response
Visa	4456530000001005	Successful Frictionless Authentication	Yes	OK
Mastercard	5200000000001005	Successful Frictionless Authentication	Yes	OK
American Express	340000000001007	Successful Frictionless Authentication	Yes	OK
Visa	4456530000001013	Unsuccessful Frictionless Authentication	No	`AUTHENTICATION_FAILED`
Mastercard	5200000000001013	Unsuccessful Frictionless Authentication	No	`AUTHENTICATION_FAILED`
American Express	340000000001015	Unsuccessful Frictionless Authentication	No	`AUTHENTICATION_FAILED`
Visa	4456530000001021	Stand-In Frictionless Authentication	Yes	OK
Mastercard	5200000000001021	Stand-In Frictionless Authentication	Yes	OK
American Express	340000000001023	Stand-In Frictionless Authentication	Yes	OK
Visa	4456530000001039	Unavailable Frictionless Authentication	No	3DS Error: `3DS_003`
Mastercard	5200000000001039	Unavailable Frictionless Authentication	No	3DS Error: `3DS_003`
American Express	340000000001031	Unavailable Frictionless Authentication	No	3DS Error: `3DS_003`
Visa	4456530000001047	Rejected Frictionless Authentication	No	`AUTHENTICATION_FAILED`
Mastercard	5200000000001047	Rejected Frictionless Authentication	No	`AUTHENTICATION_FAILED`
American Express	340000000001049	Rejected Frictionless Authentication	No	`AUTHENTICATION_FAILED`

Scheme	Number	Scenario	Liability Shift	3DS Response
Visa	4456530000001054	Authentication not Available on Lookup (system error)	No	3DS Error: `3DS_003`
Mastercard	5200000000001054	Authentication not Available on Lookup (system error)	No	3DS Error: `3DS_003`
American Express	340000000001056	Authentication not Available on Lookup (system error)	No	3DS Error: `3DS_003`
Visa	4456530000001062	Enrollment error while attempting authentication	No	3DS Error: `3DS_003`
Mastercard	5200000000001062	Enrollment error while attempting authentication	No	3DS Error: `3DS_003`
American Express	340000000001064	Enrollment error while attempting authentication	No	3DS Error: `3DS_003`
Visa	4456530000001070	Time-Out	No	3DS Error: `3DS_007`
Mastercard	5200000000001070	Time-Out	No	3DS Error: `3DS_007`
American Express	340000000001072	Time-Out	No	3DS Error: `3DS_007`
Visa	4456530000001088	Bypassed Authentication	No	
Mastercard	5200000000001088	Bypassed Authentication	No	
American Express	340000000001080	Bypassed Authentication	No	
Visa	4456530000001096	Successful Step-Up Authentication	Yes	OK
Mastercard	5200000000001096	Successful Step-Up Authentication	Yes	OK
American Express	340000000001098	Successful Step-Up Authentication	Yes	OK
Visa	4456530000001104	Unsuccessful Step-Up Authentication	No	3DS Error: `3DS_006`
Mastercard	5200000000001104	Unsuccessful Step-Up	No	3DS Error: `3DS_006`

Scheme	Number	Scenario	Liability Shift	3DS Response
		Authentication		
American Express	340000000001106	Unsuccessful Step-Up Authentication	No	3DS Error: `3DS_006`
Visa	4456530000001112	Unavailable Step-Up Authentication	No	3DS Error: `3DS_003`
Mastercard	5200000000001112	Unavailable Step-Up Authentication	No	3DS Error: `3DS_003`
American Express	340000000001114	Unavailable Step-Up Authentication	No	3DS Error: `3DS_003`

Subscriptions and Recurring Payments

Scheme	Number	MM/YY	CVV	Scenario
AMEX	373953192377918	11/27	1234	**Active Subscription** - Greater Than $100 **Inactive Subscription** - Insufficient Funds (Less Than $100)
Mastercard	5599995345720574	11/27	123	**Active Subscription** - Greater Than $100 **Inactive Subscription** - Insufficient Funds (Less Than $100)
Visa	4560049630217702	11/27	123	**Active Subscription** - Greater Than $100 **Inactive Subscription** - Insufficient Funds (Less Than $100)

Error Scenario Test Cards

> **! Scope of test cards**
>
> These test cards numbers only work if you are integrating to the APIs found here.
>
> If you are integrating via new APIs, these test cards will not work.

The following cards can be used to test error outcomes on purchases. To test cards which will result in a successful transaction please see cards listed under test card numbers.

Card Type	Number	Expiry (MM/YY)	CVV	Description
AMEX	373953192353000	05/26	1234	Do Not Honour
AMEX	373953192355005	04/26	1234	Lost card, Pick up
AMEX	373953192354008	03/26	1234	Suspected Fraud
AMEX	373953192377710	01/26	1234	Refer to Card Issuer
AMEX	373953192377728	02/26	1234	CVV Validation Error
AMEX	373953192377736	06/26	1234	Card Over limit
AMEX	373953192358801	07/26	1234	Insufficient funds
AMEX	373953192377777	08/26	1234	Expired card
Mastercard	5453010000011015	05/26	123	Do Not Honour
Mastercard	5453010000012013	04/26	123	Lost card, Pick up
Mastercard	5453010000013011	03/26	123	Suspected Fraud
Mastercard	5453010000017012	07/26	123	Insufficient funds
Mastercard	5453010000018887	08/26	123	Expired card
Mastercard	5453010000011072	01/26	123	Refer to Card Issuer
Mastercard	5453010000011080	02/26	123	CVV Validation Error
Mastercard	5453010000011098	06/26	123	Card Over limit
Visa	4530303000277649	05/26	123	Do Not Honour
Visa	4530303000287648	04/26	123	Lost card, Pick up
Visa	4530303000297647	03/26	123	Suspected Fraud
Visa	4530303000297688	07/26	123	Insufficient funds
Visa	4530303000288885	08/26	123	Expired card
Visa	4530303000297605	01/26	123	Refer to Card Issuer
Visa	4530303000297613	02/26	123	CVV Validation Error
Visa	4530303000297621	06/26	123	Card Over limit
Diners	36436513702500	05/26	123	Do Not Honour
Diners	36436513703508	04/26	123	Lost card, Pick up
Diners	36436513704506	03/26	123	Suspected Fraud
Diners	36436513704704	07/26	123	Insufficient funds
Diners	36436513706667	08/26	123	Expired card
Diners	36436513702559	01/26	123	Refer to Card Issuer
Diners	36436513702567	02/26	123	CVV Validation Error
Diners	36436513702575	06/26	123	Card Over limit

Risk Management

PCI Compliance

What is PCI DSS Compliance?

PCI DSS or Payment Card Industry Data Security Standards (PCI DSS) are standards for security policies, technologies and ongoing processes. This ensures that merchants and financial institutions protect their payment systems from breaches and theft of cardholder data.

Any organisation involved with the processing, handling and storage of card data must comply with these standards.

Wpay is certified for PCI DSS as a Level 1 Service Provider which is the highest standard set by the payment card industry to ensure that credit card data is; processed, stored and transmitted in a secure environment.

How to ensure PCI DSS Compliance

PCI DSS compliance is a shared responsibility between your business and Wpay so whenever accepting and transmitting card information it is important to ensure that this is done in alignment with the PCI standards.

By using one of our integration options such as our Frames and SDK's it ensures you are able to accept payments without ever handling sensitive card data.

Fraud Detection

As part of our solutions, Wpay offers comprehensive fraud detection and management services to all our merchants. With Wpay Fraud services, you can run rules during your payment transaction to identify fraud. This can help to help reduce costs, capture revenue, and filter out risky transactions.

We partner with Sift and CyberSource to offer you a safe payment experience by keeping your business secure from growing fraud threats. With Sift, the fraud check is performed *prior to* payments whilst Cybersource fraud detection happens *after* payments.

> 📘 **Fraud Services**
>
> Wpay also provides acquiring services and additional offerings to merchants who would be looking to outsource fraud monitoring services. Talk to our customer success representative to find out more about this.

Cybersource

Cybersource Decision Manager utilises machine learning capabilities to detect and prevent fraud whilst reducing payment frictions for the good transactions. The fraud check process occurs *after* the payment has been processed as the outcome of the payment will form part of the data that the Cybersource Decision Manager uses to determine if an order may be fraudulent. Wpay passes back both the outcome of the payment and the fraud checking process to the user. Possible fraud checking responses may be **Accept**, **Reject**, **Review**, or **Unexpected error**.

Decision	Reason Code	Description
Accept	100	No fraud detected. Advice is that payment can proceed.
Review	480	Fraud potential. Payment should be manually reviewed to determine fraud decision.
Reject	101	The request is missing one or more fields. Resend the request with the correct information.
Reject	102	One or more fields in the request contains invalid data.
Reject	481	Fraud likely. Advice is that payment should be voided or refunded.
Unexpected error	UN99	Failed to perform fraud check due to unexpected error.

The user may then determine the desired customer experience and how they wish to proceed with the transaction based on the payment and fraud outcomes i.e. if the outcome is **Reject (481)** you may wish to cancel the order and refund/void the payment.

Should you require the use of Cybersource Decision Manager, we will need to configure this for you along with any of your merchant-specific rules during the onboarding process. Once successfully set up you can submit a request to make a payment with the fraud payload included to trigger the fraud-checking process.

Fraud Payload

During payments or payment instrument verification where the fraud payload is present you can pass in the fraud payload with the required fields. The fraud payload is passed as a BLOB object and can be passed as both XML or ZIP BASE64 encoded.

Field Definition

Field	Description	Mandatory / Data Type
merchantID	Your Cybersource merchant ID which will be provided to you when set up with Cybersource.	Yes String (30)
merchantReferenceCode	Unique merchant-generated order reference or tracking number for each transaction. Typically this would be the Client Reference you provided as part of the payment transaction.	Yes String (50)
Bill To - firstName	The first name of the customer paying for the good/service	Yes String (60)
Bill To - lastName	The last name of the customer paying for the good/service	Yes String (60)
Bill To - street1	The street address of the customer paying for the good/service	Yes String (60)
Bill To - city	The city of the customer paying for the good/service	Yes String (50)
Bill To - state	The state of the customer paying for the good/service. Use the 2-3 digit ISO state code.	Yes String (3)
Bill To - postalCode	The postal code of the customer paying for the good/service.	Yes String (10)
Bill To - country	The country of the customer paying for the good/service. Use the two-character ISO country codes.	Yes String (2)
Bill To - email	The email of the customer paying for the good/service including the full domain name.	Yes String (255)
Bill To - ipAddress	The IP address of the customer paying for the good/service reported by your web server using socket information.	No String (45)
Bill To - dateOfBirth	The date of birth of the customer paying for the good/service. Use the format: YYYYMMDD.	No
Bill To - customerID	The customer identifier of the customer paying for the good/service. This is typically the same value provided as the shopper ID when identifying the customer.	No
Ship To - firstName	The first name of the customer receiving for the good/service	No String (60)
Ship To - lastName	The last name of the customer receiving for the good/service	No String (60)
Ship To - street1	The street address of the customer receiving the good/service	No String (60)
Ship To - city	The city of the customer receiving the good/service	No

Field	Description	Mandatory / Data Type
		String (50)
Ship To - state	The state of the customer receiving the good/service. Use the 2-3 digit ISO state code.	No String (3)
Ship To - postalCode	The postal code of the customer receiving the good/service.	No String (10)
Ship To - country	The country of the customer receiving the good/service. Use the two-character ISO country codes.	No String (2)
Ship To - phoneNumber	The phone number of the customer receiving the good/service. Add the country code at the beginning of the phone number, if possible. Otherwise, the billing country is used to determine the country code. Do not use dashes, spaces, or parentheses.	No String (15)
Ship To - email	The email of the customer receiving the good/service.	
Item - unitPrice	The unit price for the good/service being purchased. This value cannot be negative. You can include a decimal point (.), but you cannot include any other special characters.	Yes String (15)
Item - quantity	The quantity of the good/service being purchased	No Integer (10)
Item - productName	The name of the good/service being purchased.	No String (255)
Item - productSKU	Identification code (SKU) for the good/service being purchased.	No String (255)
Purchase Totals - currency	The currency of the good/service being purchased. Use the ISO currency codes.	Yes String (5)
Purchase Totals - grandTotalAmount	The total value of the basket/order for the goods/services being purchased. Must be greater than or equal to zero and must equal the total amount of each line item including the tax amount. Your request must include either this field or item_#_unitPrice.	No Decimal (15)
merchantDefinedData -1-60	60 available fields where merchant specific data can be specified based on your merchant specific rules set up with Cybersource	No
afsService run	Whether to include the Cybersource afsService run in your request. This field can be set to either true or	Yes Boolean

Field	Description	Mandatory / Data Type
	false. We suggest always setting this to true to enable fraud scoring which is required in many rules.	
deviceFingerprintID	The session ID for the fingerprint can use any string that you are already generating, such as an order number or web session ID. The string can contain uppercase and lowercase letters, digits, and these special characters: hyphen (-) and underscore (_).	No

Fraud Payload Example

An example of what the fraud payload in its XML form will look like:

XML

```
<?xml version="1.0" encoding="Windows-1252"?>
<RequestMessage xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi= "http://www.w3.org/2001/XMLSchema
<merchantID>woolworths_online</merchantID>
<merchantReferenceCode>18SJBB-26IO8JUN</merchantReferenceCode>

<billTo>
<firstName>Jane</firstName>
<lastName>Doe</lastName>
<street1>407 ELIZABETH STREET</street1>
<city>SURRY HILLS</city>
<state>NSW</state>
<postalCode>2199</postalCode>
<country>AU</country>
<phoneNumber>0400000000</phoneNumber>
<email>jane@doe.com.au</email>
<ipAddress>203.39.218.236</ipAddress>
<dateOfBirth>1970-01-01</dateOfBirth>
<customerID>123456</customerID>
</billTo>

<shipTo>
<firstName>Jane</firstName>
<lastName>Doe</lastName>
<phoneNumber>0400000000</phoneNumber>
<email>jane@doe.com.au</email>
</shipTo>

<item id="0">
<unitPrice>7.59</unitPrice>
<quantity>2</quantity>
<productName>Mccain Protein Plus Frozen Meal Satay Chicken</productName>
<productSKU>483660</productSKU>
 </item>
 <item id="1">
 <unitPrice>2.00</unitPrice>
 <quantity>2</quantity>
 <productName>Habee Savers Needles Household Repair</productName>
 <productSKU>159489</productSKU>
 </item>
  <item id="2">
 <unitPrice>6.60</unitPrice>
 <quantity>5</quantity>
 <productName>Chicken Breast Fillet Skinless Small</productName>
 <productSKU>118963</productSKU>
  </item>
 <item id="3">
 <unitPrice>5.43</unitPrice>
 <quantity>5</quantity>
 <productName>Chicken Drumsticks </productName>
 <productSKU>169014</productSKU>
  </item>
 <item id="4">
 <unitPrice>3.50</unitPrice>
 <quantity>4</quantity>
 <productName>Chicken Thigh Cutlets Skinless</productName>
```

Example of the fraud payload passed as part of a payment:

JSON


```
{
  "data": {
    "transactionType": {
      "creditCard": "PREAUTH",
      "giftCard": "PURCHASE",
      "payPal": "PURCHASE",
      "googlePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      },
      "applePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      }
    },
    "clientReference": "UNIQUE_CLIENT_REFERENCE",
    "orderNumber": "UNIQUE_ORDER_NO",
    "payments": [
      {
        "paymentInstrumentId": "213553",
        "amount": 10.5
      },
       {
        "paymentInstrumentId": "215319",
        "amount": 6.5
      }
    ]
  },
  "meta": {
    "fraud": {
        "provider": "cybersource",
        "version": "CyberSourceTransaction_1.101",
        "format": "XML",
        "responseFormat": "XML",
        "message": "<?xml version=\"1.0\" encoding=\"Windows-1252\"?>\r\n<RequestMessage xmlns:xsd=\"http
    },
    "challengeResponses": [
      {
        "instrumentId": "213553",
        "type": "STEP_UP",
        "token": "55bda344-c0ec-####-####-############"
      }
    ]
  }
}
```


Example of the fraud payload passed as part of a payment:

Where:

- `format` is the input format of the `message` being passed in. This can be **XML** or **ZIP_BASE_64_ENCODED**.
- `responseFormat` is the output format in which you will receive the fraud response. This can be **XML** or **ZIP_BASE_64_ENCODED**.
- `message` is the fraud payload provided in a single field in the specified format.

Sift

Wpay has partnered with Sift to leverage their machine learning capabilities to detect and prevent fraud. Sift makes risk predictions using your own data and data from across Sift's global network to identify patterns across thousands of device, user, network, and transactional signals.

Whilst Cybersource evaluates orders *after* payments, Sift Payment Protection checks for frauds *before* payments, providing the flexibility for you to take actions on the transaction.

Similar to Cybersource, Sift returns three different possible fraud responses - **Accept**, **Reject**, or flagging an order for **Review**.

Decision	Reason Code	Description
Accept	100	No fraud detected. Payment to proceed after fraud check.
Review	480	Fraud potential. Payment to proceed whilst the transaction is flagged for review in Sift. The fraud team should manually review the transaction to determine the next course of actions to take.
Reject	481	Fraud likely. Payment blocked and a Reject response returned to you to take further actions.

You may then determine the desired customer experience and how you wish to proceed with the transaction based on fraud outcomes e.g. if the outcome is **Reject (481)** you may wish to block the user from your store.

Merchant Configuration

To utilise Sift capabilities, we will need to configure this for you along with any of your merchant-specific rules during the onboarding process. Once successfully set up you can submit a request to make a payment with Sift fraud payload included to trigger the fraud-checking process.

Step 1: Create a Sift Merchant Account

To begin using Sift as your fraud screening solution you first need to setup a Sift instance which will be specific to your organisation. Your Wpay account management representative will be able to support you through the steps of setting up the Sift instance as part of your integration process.

Step 2: Loading historical data into Sift

Sift uses a machine learning algorithm to perform fraud scoring for transactions. It's recommended by Sift to backfill at least 3 - 6 months of historical data to get the best performance from the platform from day one. Sift outlines how to complete this process on the following page. As an optional step you might also wish to backfill the decisions of your current fraud engine. Your Wpay account management representative will be able to support you through the steps of backfilling your historical data.

Step 3: Config API Keys for Sandbox & PROD

The Sift solution requires your Sift API-key to be set up in APIGEE to enable the connection between the Wpay Platform and Sift. This step needs to be completed for both the non-PROD and PROD environments. Your Wpay account management representative will be able to support you through the process.

Step 4: Configuration of the Sift Rules

Once the above steps have been completed it's now time to setup your fraud rules in Sift. Your Wpay account management representative will be able to support you through the steps configuring your fraud rules.

> 📘 **Fraud Rule Considerations**
>
> If you are already using an existing fraud screening provider for fraud scoring you will need to work with your account management representative to perform an analysis to ensure your existing rules can be imported into Sift. Alternatively, if you aren't using an existing fraud screening provider, you will need to define and implement your fraud rules.

Step 5: Integration with the Wpay Platform

Now that the pre-requisite steps have been completed you are now ready to integrate with the Wpay platform to use Sift as your fraud screening provider. The Sift payload has been designed as structured JSON with its own schema in order to make the information easier to read by your developers.

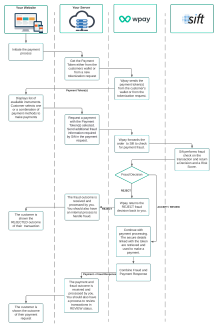
> 📘 **Customer Experience Considerations**
>
> Merchant must also update their orchestration logic and customer experience to handle a payment declined due to fraud before the payment occurs. Therefore, a new screen might be required to tell the customer their payment was unsuccessful but not tipping of a potential fraudster that it was due to fraud screening being utilised. The current process for Cybersource is post the payment being processed the merchant can reverse the transaction by either:
>
> 1. Void the transaction, if processed as a pre-auth.
> 2. Refund the transaction, if processed as a purchase.

Step 6: Embed the Sift snippets

Description	Documentation	Notes
JavaScript Snippet for all web traffic (Front-End)	JavaScript Snippet	**Where to deploy?** On all customer facing pages on your website **Before Login** Set the $session_id field **After Login** Set the $user_id field (should match the $user_id on REST API). Maintain the $session_id **Important** Disable the JS snippet for ALL Internal User Activity i.e. admins, analysts making bookings/orders on the behalf of users etc
Mobile SDK Overview	Mobile SDK Overview	N/A
Mobile SDK for mobile apps (Front-end)	iOS SDK	**Size** 66 KB including dependencies **Permissions** Access to Internet (Required), Location (Optional), Gyroscope (Optional) **OS Support** iOS 10+ **Data Usage** ~6kb of data per minute of active app use; App State + Device Information Collected and Sent via SDK App State sent once every minute, Device Info sent once every hour or whenver it changes **Installation** Cocoapods + Carthage Installation OR via Github repo
Mobile SDK for mobile apps (Front-end)	Android SDK	**Size** 4.5 MB total with all dependencies (3 MB without common libraries) **Permissions** Access to Internet (Required), Fine Location (Optional), Coarse Location (Optional)

Description	Documentation	Notes
		OS Support Support for Jelly Bean 4.1.x (Android API 16+) **Data Usage** Uses ~6kb of data per minute of active app use; App State + Device Information Collected and Sent via SDK App State sent once every minute, Device Info sent once every hour or whenver it changes **Installation** Maven or Jcenter Integration OR via Github repo

High Level Flow



Fraud Payload

The Fraud payload for Sift will be sent as part of the payment please refer to Making a Payment.

Field	Description	Mandatory / Data Type
schemaId	The ID of the previously configured schema that will be used to validate the contents of the fraud payload. The schema ID will be given back to the merchant during their setup process.	Yes String
sessionId	The user's current session ID.	No String
orderId	The ID for tracking this order in your system.	No, but strongly recommended to improve fraud scoring. String
userEmail	Email of the user creating this order.	No String
amount	Total transaction amount.	No, but strongly recommended to improve fraud scoring. String
currency	ISO-4217 currency code for the amount.	No String
sellerUserId	The seller's user ID for marketplace.	No String
verificationPhoneNumber	Phone number of the user. This phone number will be used to send One-Time Password (OTP) when required. The phone number should be in E.164 format including + and a country code.	No String
shippingTrackingNumbers	Shipping tracking number(s) for the shipment of the product(s).	No Array of String
billingAddress - firstName	The first name of the customer paying for the good/service	No, but strongly recommended to improve fraud scoring. String
billingAddress - lastName	The last name of the customer paying for the good/service	No, but strongly recommended to improve fraud scoring. String
billingAddress - email	The email of the customer paying for the good/service	No String
billingAddress - phone	The phone number of the customer paying for the good/service. Provide the phone number as a string starting with the country code. Use E.164 format or send in the standard national format of number's origin. For example: "+61433666666"	No String
billingAddress - streetAddress	The street address of the customer paying for the good/service	No, but strongly recommended to improve

Field	Description	Mandatory / Data Type
		fraud scoring. String
billingAddress - extendedAddress	The extended address of the customer paying for the good/service	No String
billingAddress - suburb	The suburb of the customer paying for the good/service	No, but strongly recommended to improve fraud scoring. String
billingAddress - stateOrTerritory	The state of the customer paying for the good/service	No, but strongly recommended to improve fraud scoring. String
billingAddress - postalCode	The postal code of the customer paying for the good/service	No, but strongly recommended to improve fraud scoring. String
billingAddress - countryCode	The country of the customer paying for the good/service. Use the two-character ISO-3166 country codes.	No, but strongly recommended to improve fraud scoring. String
orderFrom - storeId	The customer's internal identifier for the specific physical location providing the good or service.	No String
orderFrom - storeAddress - name	The full name associated with the store address providing the good or service.	No String
orderFrom - storeAddress - address1	The address first line of the store providing the good or service.	No String
orderFrom - storeAddress - address2	The address second line of the store providing the good or service.	No String
orderFrom - storeAddress - suburb	The city of the store providing the good or service.	No String
orderFrom - storeAddress - postalCode	The postal code of the store providing the good or service.	No String
orderFrom - storeAddress - stateOrTerritory	The suburb of the store providing the good or service.	No String
orderFrom - storeAddress - countryCode	The ISO-3166 country code of the store providing the good or service.	No String
orderFrom - storeAddress - phone	The phone of the store providing the good or service.	No String
brandName	Name of the brand of product or service being purchased.	No String
siteDomain	Domain being interfaced with. Use fully qualified domain name.	No String

Field	Description	Mandatory / Data Type
siteCountry	Country the company is providing service from. Use ISO-3166 country code.	No String
shippingAddress - firstName	The first name associated with the address where the product is shipped to.	No, but strongly recommended to improve fraud scoring. String
shippingAddress - lastName	The last name associated with the address where the product is shipped to.	No, but strongly recommended to improve fraud scoring. String
shippingAddress - email	The customer's email associated with the address where the product is shipped to.	No String
shippingAddress - phone	The customer's phone associated with the address where the product is shipped to.	No String
shippingAddress - streetAddress	The street address of the customer where the product is shipped to.	No, but strongly recommended to improve fraud scoring. String
shippingAddress - extendedAddress	The extended address of the customer where the product is shipped to.	No String
shippingAddress - suburb	The suburb of the customer where the product is shipped to.	No, but strongly recommended to improve fraud scoring. String
shippingAddress - stateOrTerritory	The state of the customer where the product is shipped to.	No, but strongly recommended to improve fraud scoring. String
shippingAddress - postalCode	The postal code of the customer where the product is shipped to.	No, but strongly recommended to improve fraud scoring. String
shippingAddress - countryCode	The ISO-3166 country code of the customer where the product is shipped to.	No, but strongly recommended to improve fraud scoring. String
expeditedShipping	A flag to indicate whether the user requested priority/expedited shipping on their order.	No Boolean
shippingMethod	The method of delivery to the user.	No Allowed values: [`electronic`, `physical`]

Field	Description	Mandatory / Data Type
shippingCarrier	Shipping carrier for the shipment of the product.	No String
shippingTrackingNumbers	Shipping tracking number(s) for the shipment of the product(s).	No Array of String
basketData - itemId	The item's unique identifier of good/service sold by your business.	No String
basketData - description	The item description	No String
basketData - quantity	The quantity of the item.	No String
basketData - price	The item unit price	No String
basketData - sku	If the item has a Stock-keeping Unit ID (SKU), provide it here.	No String
basketData - brand	The brand name of the item.	No String
basketData - category	The category this item is listed under in your business. e.g., "kitchen appliance", "menswear > pants".	No String
basketData - currencyCode	ISO-4217 currency code for the price.	No String
basketData - upc	If the item has a Universal Product Code (UPC), provide it here.	No String
basketData - isbn	If the item is a book with an International Standard Book Number (ISBN), provide it here.	No String
basketData - manufacturer	Name of the item's manufacturer.	No String
basketData - tags	The tags used to describe this item in your business. e.g., "funny", "halloween".	No Array of String
basketData - color	The color of the item.	No String
basketData - size	The size of the item.	No String
promotion - promotionId	The ID within your system that you use to represent this promotion. This ID is ideally unique to the promotion across users.	No String
promotion - description	Promotion description	No String
promotion - status	Promotion status	No Allowed values

Field	Description	Mandatory / Data Type
		[`success` , `failure`]
promotion - failureReason	Reason why adding a promotion fails.	No Allowed values [`already_used` , `invalid_code` , `not_applicable` , `expired` success`]
promotion - discount - currencyCode	ISO-4217 currency code for the discount amount.	No String
promotion - discount - percentageOff	The percentage discount. If the discount is 10% off, you would send "0.1".	No String
promotion - discount - amount	The amount of the discount that the promotion offers.	No String
promotion - discount - minimumPurchaseAmount	The minimum amount someone must spend in order for the promotion to be applied.	No String
mobileApp - operatingSystem	*Choose either mobileApp or browser, not both.* The operating system on which application is running. (e.g. iOS, Android)	No String
mobileApp - osVersion	The operating system version on which application is running. (e.g. 10.3.1, 7.1.1)	No String
mobileApp - deviceManufacturer	The manufacturer of the device on which application is running. (e.g. Samsung, Apple, LG)	No String
mobileApp - deviceModel	The model of the device on which application is running. (e.g. SM-G920x, iPhone8,1)	No String
mobileApp - deviceUniqueId	The unique ID of the device on which application is running. For iOS, send the IFV identifier. For Android, send the Android ID.	No String
mobileApp - appName	The name of your application.	No String
mobileApp - appVersion	The version of your application. Our accepted format is numbers separated by periods.	No String
mobileApp - clientLanguage	The language the application content is being delivered in. Use ISO-3166 format for country codes. Examples: "en", "en-us, de", "fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5", etc.	No String
browser - userAgent	*Choose either mobileApp or browser, not both.*	Yes if browser is not null / empty

Field	Description	Mandatory / Data Type
		String
browser - acceptLanguage	The language(s) that the client is requesting the site content be delivered in. Use ISO-3166 format for country codes. Examples: "en", "en-us, de", "fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5", etc.	No String
browser - contentLanguage	The language(s) of the user that the delivered site content is intended for. Use ISO-3166 format for country codes. Examples: "en", "en-us, de", "fr-CH, fr;q=0.9, en;q=0.8, de;q=0.7, *;q=0.5", etc.	No String

Fraud Payload Example

Example of the fraud payload passed as part of a payment:

JSON


```
{
  "data": {
    "transactionType": {
      "creditCard": "PREAUTH",
      "giftCard": "PURCHASE",
      "payPal": "PURCHASE",
      "googlePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      },
      "applePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      }
    },
    "clientReference": "ORDER-28168441",
    "orderNumber": "CUST_ORDER-123654",
    "payments": [
      {
        "paymentInstrumentId": "124****",
        "amount": 14.99
      }
    ]
  },
  "meta": {
    "fraud": {
        "provider": "sift",
        "version": "sift_1.101",
        "format": "JSON",
        "responseFormat": "JSON",
        "message": "",
        "payload": {
            "schemaId": "66847d9f-07c5-4647-****-************",
            "sessionId": "gigtleqddo84l8cm15qe4il",
            "orderId": "ORDER-28168441",
            "userEmail": "accept@accept.com",
            "amount": "14.99",
            "currency": "AUD",
            "billingAddress": {
                "firstName": "John",
                "lastName": "Sena",
                "email": "john.sena@mail.com",
                "phone": "0470623177",
                "extendedAddress": "4th Floor",
                "streetAddress": "407 Elizabeth Street",
                "suburb": "Surry Hills",
                "stateOrTerritory": "NSW",
                "postalCode": "2765",
                "countryCode": "AU"
            },
            "orderFrom": {
                "storeId": "1234",
                "storeAddress": {
                    "name": "Toongabbie",
                    "address1": "15 Aurelia Street",
```


Example of Fraud Response - Accept

JSON


```
{
    "data": {
        "transactionId": "aaff67e0-e078-42b2-****-************",
        "paymentRequestId": "f55bee99-5b75-41ad--****-************",
        "type": "PAYMENT",
        "status": "APPROVED",
        "grossAmount": 14.99,
        "executionTime": "2022-08-11T10:08:13.808Z",
        "merchantId": "WpayTestAPM",
        "merchantReferenceId": "5bhygswbhwx",
        "clientReference": "2042f2be-699e-418c-****-************",
        "fraudCheckProvider": "sift",
        "instruments": [
            {
                "paymentInstrumentId": "2499531",
                "instrumentType": "CREDIT_CARD",
                "transactions": [
                    {
                        "type": "PREAUTH",
                        "executionTime": "2022-08-11T10:08:20.918Z",
                        "paymentTransactionRef": "100000002673****",
                        "status": "APPROVED",
                        "amount": 14.99
                    }
                ]
            }
        ],
        "subTransactions": [
            {
                "transactionReceipt": "100000002673****",
                "partialSuccess": false,
                "fraudResponse": {
                    "clientId": "sift_test_acct_id",
                    "riskScore": 0.5544041033169816,   // risk score returned by Sift
                    "reasonCode": "100",
                    "decision": "ACCEPT",   // Decision = ACCEPT
                    "riskInformation": [
                        {
                            "app": "decision",
                            "name": "Accept User",
                            "state": "running",
                            "decision": "accept_user_payment_abuse"
                        },
                        {
                            "app": "decision",
                            "name": "Accept Order",
                            "state": "running",
                            "decision": "accept_order_payment_abuse"
                        }
                    ]
                },
                "paymentResponses": [
                    {
                        "paymentInstrumentId": "249****",
                        "paymentToken": "3ff50323-d8aa-4188-****-************",
```


Example of Reject Fraud Response

JSON


```
{
    "data": {
        "transactionId": "54738bcd-9160-44de-****-************",
        "paymentRequestId": "2a6c9a56-b292-4e13-****-************",
        "type": "PAYMENT",
        "status": "REJECTED",
        "rollback": "NOT_REQUIRED",
        "grossAmount": 0.01,
        "executionTime": "2022-08-11T23:50:39.262Z",
        "merchantId": "WpayTestAPM",
        "merchantReferenceId": "lvjmy6ik75",
        "clientReference": "lvjmy6ik75",
        "fraudCheckProvider": "sift",
        "instruments": [
            {
                "paymentInstrumentId": "249****",
                "instrumentType": "CREDIT_CARD",
                "transactions": []
            }
        ],
        "subTransactions": [
            {
                "fraudResponse": {
                    "clientId": "sift_test_acct_id",
                    "riskScore": 0.6722685731793858,   // Fraud scoring
                    "reasonCode": "481",
                    "decision": "REJECT",   // Decision = REJECT
                    "riskInformation": [
                        {
                            "app": "decision",
                            "name": "Block User",
                            "state": "running",
                            "decision": "block_user_payment_abuse"
                        },
                        {
                            "app": "decision",
                            "name": "Accept Order",
                            "state": "running",
                            "decision": "accept_order_payment_abuse"
                        }
                    ]
                }
            }
        ]
    },
    "meta": {}
}
```


Example of Review Fraud Response

JSON


```
{
    "data": {
        "transactionId": "0f562574-3aa9-45f9-****-************",
        "paymentRequestId": "116da4f9-1bcc-46c9-****-************",
        "type": "PAYMENT",
        "status": "APPROVED",
        "grossAmount": 0.5,
        "executionTime": "2022-08-12T02:10:55.957Z",
        "merchantId": "WpayTestAPM",
        "merchantReferenceId": "4a18dd32-efc2-4242-****-************",
        "clientReference": "010219b3-4e44-48e2-****-************",
        "fraudCheckProvider": "sift",
        "instruments": [
            {
                "paymentInstrumentId": "249****",
                "instrumentType": "APPLE_PAY",
                "transactions": [
                    {
                        "type": "PREAUTH",
                        "executionTime": "2022-08-12T02:10:57.728Z",
                        "paymentTransactionRef": "100000002674****",
                        "status": "APPROVED",
                        "amount": 0.5
                    }
                ]
            }
        ],
        "subTransactions": [
            {
                "transactionReceipt": "100000002674****",
                "partialSuccess": false,
                "fraudResponse": {
                    "clientId": "sift_test_acct_id",
                    "riskScore": 0.6492781559924097,  // Risk scoring
                    "reasonCode": "480",
                    "decision": "REVIEW",   // Decision = REVIEW
                    "riskInformation": [
                        {
                            "app": "review queue",
                            "name": "Manual Review - User",
                            "state": "running",
                            "decision": "review"
                        },
                        {
                            "app": "review queue",
                            "name": "Manual Review - Order",
                            "state": "running",
                            "decision": "review"
                        }
                    ]
                },
                "paymentResponses": [
                    {
                        "paymentInstrumentId": "249****",
                        "paymentToken": "affc24b1-3a03-4209-****-************",
```

3D Secure (3DS)

What is 3D Secure

3DS helps prevent fraud for online payments where a card is not physically present. Customers are quickly verified providing a fast and secure payment experience. By enabling the service you receive additional protection against customer disputes and chargebacks for fraud.

3DS version 2 has introduced a significant improvement in the customer experience in comparison to the older 3DS version 1. In most cases, a transaction can be frictionlessly verified without any further customer interaction by using additional data about the customer and card that was not available as part of 3DSv1. However, some transactions cannot be completed frictionlessly, in this case, the customer's card issuer will instruct Wpay to actively verify the customer through what is called challenge verification.

> 📘 **Merchant configuration required**
>
> In order to use 3DS with Wpay your merchant needs to be configured & enabled for 3DS within our system. Please contact us to have your merchant set up.

Wpay supports 3DSv2 for American Express, Mastercard and Visa. The 3DS brand acceptance marks of these schemes can be found in the following location:

- American Express
- Mastercard
- Visa

Liability shift

The primary benefit for merchants using 3DS is a liability shift. Once a customer has been verified using 3DS (either via the frictionless or challenge flow) the card issuer holds liability for the transaction.

Should a transaction be fraudulent the card issuer is responsible as they have verified their cardholder through this process.

Frictionless flow

The majority of transactions follow this path, a customer and their transaction is verified and authenticated behind the scenes between Wpay and the customer's card issuer. The customer often isn't even aware additional verification has taken place.

Challenge flow

If a card Issuer determines the transaction risk to be above a certain threshold they may request additional authentication from the customer. The exact step-up mechanism is controlled by the issuer and common methods include:

- biometric via banking apps
- one time passcode via SMS or email

The challenge flow is invoked for a minority of transactions.

What version of 3DS do we support

WPay supports the following versions of 3DS

- 2.1.0
- 2.2.0

How does it work

1. When a customer is ready to checkout they pay using their card.
2. We take information about the transaction and send this to the card's issuer for authentication.
3. The Issuer's 3DS provider determines transaction risk.
4. If the risk is low, the transaction is marked as verified (frictionlessly) and authentication is complete
5. If the risk is higher, the Issuer may prompt the cardholder to verify their identity (challenge flow), once verified the authentication is complete
6. The payment is processed to the card schemes with the authentication results and liability is shifted to the issuer.
7. Upon successful authentication, we submit the transaction for processing.

3DS Payment Integration

3DS Payment Authentication allows you to authenticate a payment request using 3DS. This guide will outline the requirements for authenticating a payment using 3DS through Wpay and will highlight where the 3DS payment flow and information differs from the normal guide for Making a Payment.

The typical flow for making a payment with 3DS authentication is as follows:

Step 1 - Make a Payment Request:

1. Ensure you have a payment instrument obtained either from the customer's wallet (for saved instruments) or from tokenizing a payment instrument (for new instruments) which is supported for 3DS authentication.
2. Make a payment request and include the `requires3DS` flag in the payment request to indicate that you wish to apply 3DS authentication to the payment
3. Check the payment response to determine if 3DS authentication is required as part of the payment.
    i. Should the outcome of the payment provide an error outcome indicating `3DS TOKEN REQUIRED` then you will need to authenticate the payment via 3DS utilising the Frames SDK. See **Step 2**.

Step 2 - Request 3DS Authentication:

1. Utilizing the Frames SDK request 3DS authentication for the requested payment. The outcome of the 3DS authentication will either be frictionless or a challenge required.
2. Should the 3DS process be successful you will be provided with the 3DS `token` and `reference` data which will be required when making the subsequent payment request.

Step 3 - Make a Final Payment Request including the 3DS Data:

1. Include the `requires3DS` flag as well as the 3DS `token` and `reference` data in the payment request and process the payment

2. The 3DS data will be authenticated with the issuer to ensure that it is valid and the payment will then be processed.

1. Make a Payment Request

1.1. Request Payment with 3DS Authentication

Make a payment request and include the `merchantPayload` and set the `requires3DS` flag to true.

Example of required `merchantPayload`:

JSON

```
"merchantPayload": {
  "payload": {
    "requires3DS": true
  },
  "schemaId": "0a221353-b26c-4848-9a77-4a8bcbacf228"
}
```

Example of payment request with included `merchantPayload`:

cURL JavaScript

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/payments
--header 'X-Api-Key: {{yourAPIKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--header 'x-guest: false' \
--header 'Content-Type: application/json' \
--data-raw '{
  "data": {
    "transactionType": {
      "creditCard": "PREAUTH",
      "giftCard": "PURCHASE",
      "payPal": "PURCHASE",
      "googlePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      },
      "applePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      }
    },
    "merchantPayload": {
      "payload": {
        "requires3DS": true
      },
      "schemaId": "0a221353-b26c-4848-9a77-4a8bcbacf228"
    },
    "clientReference": "UNIQUE_CLIENT_REFERENCE",
    "orderNumber": "UNIQUE_ORDER_NO",
    "payments": [
      {
        "paymentInstrumentId": "213553",
        "amount": 10.5
      },
       {
        "paymentInstrumentId": "215319",
        "amount": 6.5
      }
    ]
  },
  "meta": {
    "fraud": {
        "provider": "cybersource",
        "version": "CyberSourceTransaction_1.101",
        "format": "XML",
        "responseFormat": "XML",
        "message": "<?xml version=\"1.0\" encoding=\"Windows-1252\"?>\r\n<RequestMessage xmlns:xsd=\"http
    },
    "challengeResponses": [
      {
        "instrumentId": "213553",
        "type": "STEP_UP",
        "token": "55bda344-c0ec-####-####-############"
      }
    ]
  }
```

1.2. Check Payment Response to Determine if 3DS Authentication is Required

JSON


```
{
    "data": {
        "transactionId": "9b8e30bb-c8bb-4762-8a47-2233e59c21d7",
        "paymentRequestId": "1037fca0-9118-4664-9f63-696ecbcfe44d",
        "type": "PAYMENT",
        "status": "REJECTED",
        "rollback": "NOT_REQUIRED",
        "grossAmount": 50.5,
        "executionTime": "2022-02-21T08:51:54.908Z",
        "merchantReferenceId": "82799438",
        "clientReference": "80085011",
        "instruments": [
            {
                "paymentInstrumentId": "2159648",
                "instrumentType": "CREDIT_CARD",
                "transactions": []
            }
        ],
        "subTransactions": [
            {
                "errorCode": "3DS_001",
                "errorMessage": "3DS TOKEN REQUIRED",
                "threeDS": {
                    "paymentInstrumentId": "2159648",
                    "sessionId": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiIxMDM3ZmNhMC05MTE4LTQ2N
                }
            }
        ]
    },
    "meta": {}
}
```


Where:

- The `"errorCode": "3DS_001"` indicate that a 3DS token is required and that 3DS authentication should be requested.
- The `threeDS` - `sessionId` is required when completing authentication via 3DS in the next step.

2. Request 3DS2 Authentication

Using the `sessionId` obtained in the previous step you will use the Wpay Frames SDK to request 3DS authentication from your customer's issuing bank.

JavaScript

```
if (paymentResponse.status === 'APPROVED') {
  paymentComplete = true;
  paymentStatus = 'Payment Complete';
} else {
  // If error code is 3DS_001 then failure was due to a 3DS challenge
  if (paymentResponse.subTransactions[0].errorCode === '3DS_001') {
    const sessionId = paymentResponse.subTransactions[0].threeDS.sessionId;

    // Perform 3DS authentication
    const threeDSResponse = await capture3DS(sessionId, this.selectedInstrument, FRAMES.ActionTypes.Valic

    if (threeDSResponse.threeDSData &&
        (threeDSResponse.threeDSData.status === 'AUTHENTICATION_SUCCESSFUL' ||
         threeDSResponse.threeDSData.ActionCode === 'SUCCESS')) {
      challengeResponses.push(
        threeDSResponse.challengeResponse,
      );
    } else {
      paymentFailed = true;

      this.paymentStatus =
        (threeDSResponse.threeDSData && threeDSResponse.threeDSData.Payment &&
         threeDSResponse.threeDSData.Payment.ExtendedData &&
         threeDSResponse.threeDSData.Payment.ExtendedData.ChallengeCancel === '01') ?
        'Payment Failed - 3DS verification cancelled by user' :
        'Payment Failed - 3DS verification failed';
    }
  } else {
    paymentFailed = true;
    paymentStatus = 'Payment Failed';
  }
}
```

Where:

- `cardCaptureResponse.errorCode` will show any errors encoutered during 3DS authentication as defined in the section Error Codes

2.1. Authenticate payment via 3DS

Usually a customer will be authenticated frictionlessly meaning that no customer authentication is required, however in some cases the card issuer may require additional authentication. In this case a challenge-response is requested from the customer. This usually takes the form of an OTP via SMS.

Where a challenge-response is generated from the customer's issuer you will render a 3DS Challenge Model. While the content is controlled by the issuer there are some modifications that can be made to better fit this model into your existing CX.

Controlling the visibility of the 3DS Challenge Model.

The Model can be controlled in a couple of ways, size and the spinner.

JavaScript HTML CSS

```
let showSpinner = true;

private async capture3DS(sessionId, paymentInstrumentId, actionType) {
  const enrollmentRequest = {
    sessionId,
    paymentInstrumentId,
    threeDS: {
      consumerAuthenticationInformation: {
        acsWindowSize: settings.customer.acsWindowSize,
      },
    },
  };

  // Create a new payment validation frames action
  const action = framesSDK.createAction(actionType, enrollmentRequest);

  // Start the action, creating a new JWT and initialising cardinal
  await action.start();

  // Set the placeholder for the challenge IFrame to be injected
  action.createFramesControl('3DSValidation', 'overlay');

  const elementHandle = document.getElementById('overlay');

  const renderEventListener = () => {
    this.showSpinner = false;
    this.show3DS = true;
  };

  const closeEventListener = () => {
    this.showSpinner = true;
    this.show3DS = false;
  };

  // Add the event listeners for OnRender and OnClose for the 3DS challenge response
  elementHandle.addEventListener(FRAMES.FramesCardinalEventType.OnRender, renderEventListener);
  elementHandle.addEventListener(FRAMES.FramesCardinalEventType.OnClose, closeEventListener);

  // Check card enrolment, allowing cardinal show issuer challenge
  const authorizationResponse = await action.complete();

  // Romove the event listeners for OnRender and OnClose for the 3DS challenge response
  elementHandle.removeEventListener(FRAMES.FramesCardinalEventType.OnRender, renderEventListener);
  elementHandle.removeEventListener(FRAMES.FramesCardinalEventType.OnClose, closeEventListener);

  // 3DS check complete, use returned information to provide a challenge response within the payment endp
  console.log(`3DS authorization complete: ${JSON.stringify(authorizationResponse)}`);

  if (actionType === FRAMES.ActionTypes.ValidatePayment) {
    paymentAuthentication = authorizationResponse;
  } else if (actionType === FRAMES.ActionTypes.ValidateCard) {
    cardValidation = authorizationResponse;
  }

  return authorizationResponse;
```

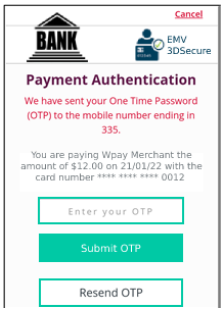
Where:

- `acsWindowSize` allows you to control the size of the 3DS challenge window and has values defined in the FAQs
- After validation of the card entered by the user the `showSpinner` is set to try showing a spinner while we wait for the issuer to return

Show the 3DS payment challenge response modal

When `action.complete` is called and the `renderEventListener` is fired, causing:

- the `showSpinner` to be set to `false`, causing this spinner to be hidden and
- the `show3DS` variable will be set to `true`, causing the Challenge Response Modal to be displayed like the one shown below:



Sample 3DS2 OTP screen for a $12 payment

Once the challenge is authenticated (in the above example this means the one time password that has been texted to the customer is entered and submitted), the `closeEventListener` is fired, setting the `show3DS` variable to `false` hiding the challenge-response iFrame.

Example of 3DS Challenge Response:

Below is an example of the response from the 3DS authentication request made through the Frames SDK. You can take the response as is and use it when making a payment as part of the `challengeResponses` array in the next step.

JSON

```
{
  "type": "3DS-frictionless",
  "instrumentId": "213553",
  "token": "{{challengeResponseToken}}",
  "reference": "{{ServerJWT}}"
}
```

3. Make a Final Payment Request including the 3DS Data

Now that we've completed our 3DS authentication we can add the required 3DS data to the `challengeResponses` and attempt to reprocess the payment.

cURL JavaScript

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/instore/customer/payments
--header 'X-Api-Key: {{yourAPIKey}}' \
--header 'Authorization: Bearer {{yourBearerToken}}' \
--header 'x-guest: false' \
--header 'Content-Type: application/json' \
--data-raw '{
  "data": {
    "transactionType": {
      "creditCard": "PREAUTH",
      "giftCard": "PURCHASE",
      "payPal": "PURCHASE",
      "googlePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      },
      "applePay": {
        "creditCard": "PREAUTH",
        "debitCard": "PURCHASE"
      }
    },
    "merchantPayload": {
      "payload": {
        "requires3DS": true
      },
      "schemaId": "0a221353-b26c-4848-9a77-4a8bcbacf228"
    },
    "clientReference": "UNIQUE_CLIENT_REFERENCE",
    "orderNumber": "UNIQUE_ORDER_NO",
    "payments": [
      {
        "paymentInstrumentId": "213553",
        "amount": 10.5
      },
       {
        "paymentInstrumentId": "215319",
        "amount": 6.5
      }
    ]
  },
  "meta": {
    "fraud": {
        "provider": "cybersource",
        "version": "CyberSourceTransaction_1.101",
        "format": "XML",
        "responseFormat": "XML",
        "message": "<?xml version=\"1.0\" encoding=\"Windows-1252\"?>\r\n<RequestMessage xmlns:xsd=\"http
    },
    "challengeResponses": [
      {
        "instrumentId": "213553",
        "type": "STEP_UP",
        "token": "55bda344-c0ec-####-####-############"
      },
      {
        "type": "3DS-frictionless",
```

Where:

- `type` is either 3DS or 3DS-frictionless depending on whether the frictionless or normal 3DS authenticationflow was applied.
- `token` is the challengeResponseToken received from the 3DS authentication
- `reference` is the ServerJWT received from the 3DS authentication

3DS Payment Success Response

Should the payment be successful you will receive an approved payment. The only difference between the payment response for a successful payment made with vs without a 3DS authenticationis the inclusion of the `threeDSData` as part of the response payload.

JSON


```
{
    "paymentRequest": {
        "expiryTime": "2021-11-17T08:27:32.309Z",
        "grossAmount": 12.4,
        "usesRemaining": 1,
        "merchantPayload": {
            "payload": {
                "requires3DS": true
            },
            "schemaId": "0a221353-b26c-4848-9a77-4a8bcbacf228"
        },
        "paymentRequestId": "cdbf72ab-03f6-4445-98dc-ec79bce91f17",
        "merchantReferenceId": "9c92a1f2-89c2-47f4-a1d5-dafe6e89bb50"
    },
    "cardValidation": {},
    "cardCapture": {},
    "paymentAuthentication": {
        "threeDSData": {
            "id": "6371373770786754404005",
            "submitTimeUtc": "2021-11-17T08:22:57Z",
            "status": "AUTHENTICATION_SUCCESSFUL",
            "clientReferenceInformation": {
                "code": "cdbf72ab-03f6-4445-98dc-ec79bce91f17"
            },
            "consumerAuthenticationInformation": {
                "acsTransactionId": "f3166c8b-f4c7-484e-99d8-f153fcd14976",
                "authenticationTransactionId": "UsFCe2c0h6cmh8S94sg0",
                "ecommerceIndicator": "spa",
                "eciRaw": "02",
                "paresStatus": "Y",
                "specificationVersion": "2.2.0",
                "threeDSServerTransactionId": "c271835d-d874-413b-846b-17b481bba171",
                "ucafAuthenticationData": "Y2FyZGluYWxjb21tZXJjZWF1dGg=",
                "ucafCollectionIndicator": "2",
                "veresEnrolled": "Y",
                "directoryServerTransactionId": "25050f2a-752d-4b7c-a2c8-7c0e8a379e17"
            }
        },
        "challengeResponse": {
            "type": "3DS-frictionless",
            "instrumentId": "1506694",
            "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiJjZGJmNzJhYi0wM2Y2LTQ0NDUtOThkYy1lY
            "reference": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiJjZGJmNzJhYi0wM2Y2LTQ0NDUtOThkY
        }
    },
    "transactions": [
       {
            "type": "PAYMENT",
            "status": "APPROVED",
            "merchantId": "petculture",
            "grossAmount": 12.4,
            "instruments": [
               {
                    "transactions": [
                       {
```


3DS Payment Error Response

Should an error occur either due to payment validation or due to 3DS an error response will be returned as per the below:

JSON


```
{
    "data": {
        "transactionId": "9b8e30bb-c8bb-4762-8a47-2233e59c21d7",
        "paymentRequestId": "1037fca0-9118-4664-9f63-696ecbcfe44d",
        "type": "PAYMENT",
        "status": "REJECTED",
        "rollback": "NOT_REQUIRED",
        "grossAmount": 50.5,
        "executionTime": "2022-02-21T08:51:54.908Z",
        "merchantReferenceId": "82799438",
        "clientReference": "80085011",
        "instruments": [
            {
                "paymentInstrumentId": "2159648",
                "instrumentType": "CREDIT_CARD",
                "transactions": []
            }
        ],
        "subTransactions": [
            {
                "errorCode": "3DS_001",
                "errorMessage": "3DS TOKEN REQUIRED",
                "threeDS": {
                    "paymentInstrumentId": "2159648",
                    "sessionId": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJqdGkiOiIxMDM3ZmNhMC05MTE4LTQ2N
                }
            }
        ]
    },
    "meta": {}
}
```


Where

- `errorCode` will give the error received during the payment request. A full list of error codes can be found here.
- `errorMessage` will give a description of the error received.

Additional Information

- A set of 3DS specific test cards are available to validate the full range of results available from issuers
- The list of 3DS error codes are listed here
- 3DS information (e.g. frame sizing) is included on our FAQs

3DS Card Capture Integration

3DS Card Capture allows you to have 3DS Authentication performed prior to saving the card into the Customer Wallet for future use.

Steps involved in performing a card capture include:

1. Create a 3DS enabled card capture action
2. Perform initial card capture
3. Check card capture response
   3.1 If we require 3DS, the the action will fail with 3DS_001
   3.1.1 Perform 3DS Card Capture
   In order to provide the 3DS evidence to the tokenisation process, we need to capture 3DS
   3.1.2 If 3DS capture was successful we will have a challenge response,
   3.1.3 If 3DS capture was unsuccessful there is no point trying to tokenise again.
   3.2 If we have an `paymentInstrumentId` then the capture was successful
4. Perform 3DS Card Capture
   4.1 Control the visibility of the spinner and 3DS challenge response modal
   4.2 Show the 3DS Card Capture challenge response

1. Create a 3DS enabled card capture action

To initialise a 3DS card capture for a non-frictionless card you specify a `threeDS` option when creating the `CaptureCard` action via the Frames SDK as shown in the code snippet below:

JavaScript | TypeScript

```
import * as FRAMES from '@wpay/frames';

const framesSDK = new FRAMES.FramesSDK(
  {
    apiKey: process.env.VUE_APP_API_KEY,
    authToken: process.env.VUE_APP_ACCESS_TOKEN,
    apiBase: `${process.env.VUE_APP_BASE_URL}/instore`,
    logLevel: FRAMES.LogLevel.DEBUG,
  },
);

const captureCardAction = framesSDK.createAction(
  FRAMES.ActionTypes.CaptureCard,
  {
    verify: false,
    save: true,
    threeDS: {
      requires3DS: true,
    },
  },
);
```

2. Perform initial card capture

The card capture and iFrame validation of card fields follow the same steps as documented in our iFrames integration guide: Host the Frames

The difference is that the response from the `cardCaptureAction.complete()` is tested to see whether a 3DS card capture is required:

HTML CSS JavaScript TypeScript

```
<div class="container">

  <!-- Card Capture iFrame place holder -->
  <div id="cardGroupPlaceholder"></div>

</div
```

3. Check card capture response

JavaScript TypeScript

```
// Intialise value of the card capture success return variable
let preconditionsMet = false;

// 3.1 If we require 3DS, the the action will fail with 3DS_001
if (cardCaptureResponse.errorCode === '3DS_001') {

    // 3.1.1 Perform 3DS Card Capture
    // In order to provide the 3DS evidence to the tokenisation process, we need to capture 3DS
    const authorizationResponse =
        await capture3DS(
            cardCaptureResponse.token,
            selectedInstrument,
            FRAMES.ActionTypes.ValidateCard);

    // 3.1.2 If 3DS capture was successful we will have a challenge response,
    if (authorizationResponse.challengeResponse) {
        cardCaptureResponse =
          await captureCardAction.complete(
            saveCard, [authorizationResponse.challengeResponse]);
        preconditionsMet = true;
    } else {

        // 3.1.3 If 3DS capture was unsuccessful there is no point trying to tokenise again.
        paymentStatus = 'Payment Failed - There was an issue during card capture';
    }
} else if (cardCaptureResponse.itemId || cardCaptureResponse.paymentInstrument.itemId) {

    // 3.2 If we have an paymentInstrumentId then the capture was successful
    preconditionsMet = true;
}
```

Where:

- `cardCaptureResponse.errorCode` has values as defined in the section 3DS error codes
- `saveCard` is a boolean value of `true` to save the card, `false` to not save the card or `undefined` if the we want to use the value set when initialising the card caption action

4. Perform 3DS Card Capture

4.1 Control the visibility of the spinner and 3DS challenge response modal

Controlling the visibility of the 3DS Card Capture Challenge Response Model. After validation of the card entered by the user the `showSpinner` is set to try showing the spinner.

HTML CSS JavaScript TypeScript

```
<div class="container">
  <!--
       3DS iFrame Challenge response placeholer.
       Includes a Vue.js class which controls the visible of this model,
       based on the value of the show3DS variable.
  -->
  <div
       id="overlay"
       class="overlay"
       style=""
       v-bind:class="{ hidden: !this.show3DS }">
  </div>

  <!-- Card Capture iFrame place holder -->
  <div id="cardGroupPlaceholder"></div>

  <!--
      Display the Spinner component, when
        the paymentDisabled variable is true or
        the showSpinner variable is true.
  -->
  <div class="processing"
       v-bind:class="{ hidden: !this.paymentDisabled || this.showSpinner === false }">
    <Spinner/>
  </div>
</div>
```

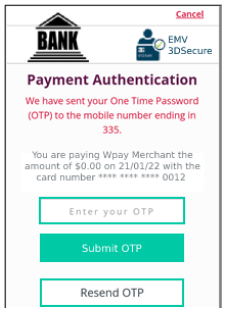
where:

- `acsWindowSize` has values defined in What are the 3DS Model challenge response screen size values ?

4.2 Show the 3DS Card Capture challenge response modal

When `action.complete` is called and the `renderEventListener` is fired, causing:

- the `showSpinner` to be set to `false`, causing this spinner to be hidden and
- the `show3DS` variable will be set to `true`, causing the Challenge Response Modal to be displayed like the one shown below:



Sample 3DS2 OTP screen for card capture (hence the $0.00)

Once the challenge is validated (in the above example this means the one time password that has been texted to the customer is entered and submitted) , the `closeEventListener` is fired, setting the `show3DS` variable to `false` hiding the challenge response iFrame.

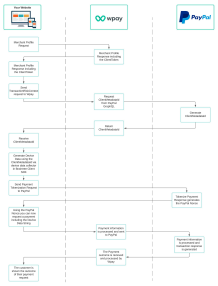
Additional Information

- A set of 3DS specific test cards are available to validate the full range of results available from issuers
- The list of 3DS error codes are listed here
- 3DS information (e.g. frame sizing) is included on our FAQs

PayPal Seller Protection

PayPal Seller Protection ensures your eligible sales are protected against unauthorised payments and transactions reversed due to suspicion of fraud. If a buyer claims they didn't receive an item, your eligible sale is protected when you provide proof of shipment or fulfilment. [1]

High Level Flow



How it works

1. Merchant's website to get the `clientToken` from the merchant's profile.
2. Merchant's website to send `TransactionRiskContext` request to a new API endpoint to get `ClientMetadataId`. In the backend, Wpay will forward the request to PayPal.
3. Merchant's website then generates `device data` using `ClientMerchantID` via the device data collector in BT Client SDK.
4. The other process to generate a Checkout instance, PayPal tokenization to get `Nonce` from PayPal and Tokenization process with Wpay remain the same.
5. When making a payment, the `device data` string will be added to the existing Making a Payment request payload to connect the payment transaction with `TransactionRiskContext` required for PayPal seller's protection.

Required Data Attributes

Below are the data attributes that you are required to provide Wpay so they can be passed to PayPal so Seller Protection can be applied to the transaction.

📘 Fields to be passed

PayPal advises to pass as many fields as possible to increase the effectiveness of their risk algorithms for PayPal Seller Protection.

Sender Profile - The fields in this section don't need to be included in the data transmission if either of these conditions exists:

- The merchant does not require the user to create a merchant account; that is, the user can perform the transaction through a "Guest Checkout" **OR**
- The merchant offers PayPal at the cart as a "Shortcut" or as "Checkout with PayPal", so that PayPal provides all the consumer information.

Delivery Information - This field is required for intangible goods only; otherwise, optional

Data Field Name	Description	Data Type	Format	Sample
sender_account_id	Unique identifier of the buyer account on the partner / merchant platform	string	Alphanumeric	A12345N343
sender_first_name	First name registered with the buyer's partner/merchant account	string	Alphanumeric	John
sender_last_name	Last name registered with the buyer's partner/merchant account	string	Alphanumeric	Smith
sender_email	Email address registered with the buyer's partner/merchant account	string	E.123 - Email Address	john@sample.com
sender_phone	Phone number (national notation) registered with the buyer's partner/merchant account	string	E.123 - Telephone Number (National Notation)	(042) 1123 4567
sender_country_code	Country code registered with the buyer's partner/merchant account	string	ISO Alpha-2 Country Code	US
sender_create_date	Date of creation of the buyer's account on the partner/merchant platform	date	ISO 8601 date format	2012-12-09T19:14:55.277-0:00
dg_delivery_method	Delivery method for an intangible item if there is an associated email/phone. It acts as the shipping address for an intangible.	string	{email, phone}	email
highrisk_txn_flag	Flag for high-risk items such as gift cards / anything cash equivalent	Boolean	Boolean (0 or 1)	0
vertical	Transaction level vertical flag for partner/merchant's transactions that are in several verticals	string	Alphanumeric	Retail

Retrieving the ClientMetadataID

To retrieve the `ClientMetadataId` use the createtransactionriskcontext endpoint to generate the `TransactionRiskContext` and send this information to Wpay who will retrieve this value on your behalf from PayPal.

cURL JavaScript Swift Kotlin

```
curl --location --request POST 'https://{{environment}}.wpay.com.au/wow/v1/pay/paypal/createtransactionri
--header 'x-api-key: {{yourApiKey}}' \
--header 'authorization: Bearer {{yourBearerToken}}' \
--header 'Content-Type: application/json' \
--data-raw '{
     "senderAccountId":"A123N23424",
     "senderFirstName":"John",
     "senderLastName":"Smith",
     "senderEmail":"John@sample.com",
     "senderPhone":"0444444444",
     "senderCountryCode":"AU",
     "senderCreateDate":"2022-06-09T02:01:41.041Z",
     "dgDeliveryMethod":"email",
     "highriskTxnFlag":true,
     "vertical":"Retail"
}'
```

TransactionRiskContext Response

JSON

```
{
  "clientMetadataId": "082029267ff97ffed8c089*********"
}
```

Making a Payment

To make a payment using PayPal Seller Protection using the Wpay Platform please follow Making a Payment

References

1. PayPal Seller Protection

Support

Glossary

We've provided the key terms used throughout the documentation for easy reference.

Term	Description
API Key	The X-Api-Key which is provided in the API headers is used to uniquely identify you as a merchant in our system. This key is highly sensitive and should be kept safe.
Bearer Token	A Bearer Token is an obfuscated string (not intended to have any meaning to clients using it). Some servers will issue tokens that are a short string of hexadecimal characters, while others may use structured tokens such as JSON Web tokens.
Payment Token	The payment token is a way to uniquely and securely identify a payment instrument which has been stored in the vault. Payment tokens are automatically issued during the tokenization process and will be used when processing payments, setting up payment agreements etc in place of sensitive card data.
Payment Instrument	A payment instrument when referred to throughout our guides and references is used for any payment method which has been tokenized. For example Credit Cards, Gift Cards and PayPal are all considered Payment Instruments.
Step Up Token	A step up token is a temporary unique and secure reference to a captured CVV. As per PCI DSS we are not allowed to store CVV details in our vault so where you as a merchant require a CVV to be captured for credit card payments we will return a step up token which should be used when making the payment. Please note that step up tokens are single use and once used to attempt to process a payment will be deleted and a new Step Up Token will be required.

FAQs

Can we tokenize a credit or debit card without the iframes?

No, due to PCI-DSS compliance it is required that all handling of the credit card information is done through the iFrames.

Are any rate limits applied to your API's?

Rate limiting is applied to many of our critical API's including our tokenizing, payments, gift card balance checks, and iFrame initialization services. These services are limited to 8 requests per customer per minute, meaning that a single customer on your site cannot process more than 8 requests for the same API within 60 seconds. We may set these rate limits to prevent system abuse, however, these can be adjusted based on your needs for your merchant.

Is it safe to expose your API key in the front end application / browser?

It is preferable that the API key is not exposed to the public through front-end apps for security reasons. The common way is using a back-end server as a relay to fetch the API results and pass them on to your front-end. If for some reason you have to make an API call from the front end, there are ways to hide the API keys like keeping them as environment variables.

Wpay uses a combination of the API Key, Bearer Tokens and IP Whitelisting for security measures. When you sign up with us and receive the API keys, you also need to specify a list of IPs that are authorized to be used with this key. Therefore, even if your API key is found or stolen, only your servers will be able to use it.

What are the 3DS Model challenge response screen size values?

The screen size value `acsWindowSize` which is past to Cardinal for the size of the iFrame modal challenge response.

Value	Size
01	250x400
02	390x400
03	500x600
04	600x400
05	Full Page

Error Codes

High Level Error Codes

HTTP Status Code	Error Code	Error Detail
200 - OK	NA	The request was successful.
400 - Bad Request	AP99 RV## BI##	Bad input data was received
401 - Unauthorised	AP01 AP02	Invalid API Key or Access Token
403 - Forbidden	AP99	Access not allowed to the requested resource
429 - Too Many Requests	AP99	Rate limiting threshold Reached due to too many requests within a limited time frame
500 - Server Error	(Multiple)	External gateway error
501 - Business Logic Validation	BV##	Data provided conflicts with business logic validation rules
502 - Bad Gateway	IS## AP99	Internal system error
503 - Service Unavailable	AP99	The service you are calling is currently unavailable
504 - Gateway Timeout	AP99	The service you are calling was not able to respond within the timeout window

Detailed Error Codes

External System Errors

Errors that occur when Wpay is interacting with external systems will be prefixed with **ES**.

Error Code	Error Description
ES36	External Gateway Timeout
ES52	Payment Transaction Declined
ES53	Technical Failure
ES91	Card Issuer Unavailable
ES94	Duplicate Transaction
ES105	Payment Instrument Expired
ES112	Invalid Transaction
ES120	Invalid Gift Card Details
ES122	Amount Is Greater Than Preauth Amount
ES130	Not Supported By Merchant
ES151	Insufficient Funds
ES470	Account Locked Or Closed
ES611	Problem Retrieving The Gift Card Balance
ES3000	Processor Network Unavailable

Business Validation/Bad Input Errors

Errors that occur when the data Wpay is receiving from you violates business validations are prefixed with **BV** and where bad input data is found they are prefixed with **BI**.

Error Code	Error Description
BV57	Illegal Step Up Token Found
BV58	Expired Step Up Token Found
BV59	Step Up Token Required
BI02	Invalid Account Or Password
BI08	Invalid Data Found In Request
BI18	Unsupported Transaction Type
BI19	Mandatory Field Value Not Found
BI22	Invalid Field Value Found
BI24	No Matching Record Found
BI33	Transaction Type Not Found
BI34	Unsupported Fraud Version
BI55	Original Payment Transaction Not Found

Token & Merchant Data Errors

Errors that occur when invalid token or merchant configuration data is found are prefixed with **AP**

Error Code	Error Description
AP01	Invalid API Key or API Product Match Not Found
AP02	Invalid or Expired Access Token
AP04	Merchant Configuration Not Found
AP05	Invalid User Linked to Access Token
AP99	Unknown Error

Catch All Error Code

`UN99` is used where an error has not been mapped to a specific code or where an unknown error has occurred. Please see the error details for further information.

Error Code	Error Description
UN99	Unknown / Unmapped Error

3DS Error Codes

3DS specific errors

errorCode	message
3DS_001	3DS Token Required
3DS_002	Invalid session
3DS_003	3DS Validation Failed
3DS_004	Unsupported 3DS Version
3DS_005	3DS Service Unavailable
3DS_006	3DS Authentication Failed
3DS_007	3DS Validation Timeout
3DS_100	Merchant does not support 3DS
3DS_500	3DS Unknown Error