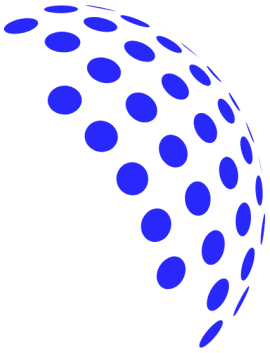

Inovio

Google Pay™ Payment Configuration

Client Setup User Guide

Version 1.0

Table of Contents

1 Overview

1.1 Client Documentation

This documentation is intended for merchants accounts that will process transactions with Google Pay and should be used as a guide on Google Pay configuration through the Portal and the client server side setup. By integrating Google Pay, you agree to Google's acceptable use policy and terms of service.

1.2 Revision History

Version	Date	Description
1.0	10 Jul 2025	Initial version

2 Google Pay Setup

2.1 Overview

This section discusses how to send credit card transactions to the Payment Gateway using Google Pay. In order to do so, the web pages which host the payment forms must have their domains registered. **Additionally, the Processor and Merchant Account being used must also support Google Pay.** Please contact your gateway support representative for more information.

Click here for an overview of Google Pay, Google Pay Integration Checklist, and Google Pay Branding Guidelines. For more information on Google Pay:

- List of payment methods that support Google Pay. Your list of allowedCardNetworks will be determined by your merchant account configuration.
- List of countries (regions) where you can use Google Pay. The Gateway supports the currencies enabled by your merchant account configuration.
- List of supported browsers.

2.2 Implementation Process Flow

Below are the necessary steps to utilize Google Pay properly within the Payment Gateway. Each item in the list is discussed in the following sections. Some example code is provided. Note that all code provided is for use as an example only, and does not represent code that can be used in your production environment.

- Fetch your Google Pay configuration data from the gateway
- Add the Google Pay library to your payment page
- Customizing the Google Pay Button
- Handling the Google Pay Loaded Event
- Presenting the Google Pay Button
- Handling the Google Pay Button Click
- Payment Authorization
- Submitting authorized payment data

NOTE: Real PAN must be used when testing in a production environment. Test cards will not work.

2.3 Fetch your Google Pay configuration data from the gateway

You will need to get your configuration data from the Payment Gateway in order to set up the Google Pay session after the customer clicks the Google Pay button. This comes from a request to the following endpoint:

Endpoint: **https://api.inoviopay.com/payment/googlepay.cfm**

- Requests must be made in JSON format.

Parameters:

Field Name	Description
REQ_USERNAME	API CREDENTIAL USERNAME
REQ_PASSWORD	API CREDENTIAL PASSWORD
DOMAIN_NAME	DOMAIN NAME WHERE THE PAYMENT PAGE IS HOSTED/SERVED FROM
CLIENT_ID	CLIENT ID
REQUEST_ACTION	REQUEST_ACTION = **GOOGLEPAYCONFIG** Used to instruct the endpoint to provide the proper Google Pay Configuration for your unique information

Note: All parameters are required

Sample Body Request

```
JSON
{
"request_login": "googlepayClient@inoviopay.com",
"request_password": "Test123",
"request_action": "GOOGLEPAYCONFIG",
"domain_name": "paymentpagedomain.com",
"client_id": "100"
}
```

Sample Response

The response will be a JSON-formatted string.

```
JSON
{
    "GOOGLEPAY_CLI_CONF_ID": 1,
    "REQ_ID": 2744,
    "CLIENT_ID": 100,
    "DOMAIN_LIST": [
        {
            "DOMAIN_NAME": "paymentpagedomain.com",
            "DOMAIN_STATUS": "ACTIVE"
        }
    ],
    "hostConfig": {
        "merchantId": "BC...JW",
        "environment*": "TEST",
        "merchantName": "Invovio Payments",
        "gatewayId*": "inoviopay",
        "gatewayMerchantId*": "string-string"
```

```
    }
}
```

This should be provided on your payment page in the parameter: merchantConfig.
* Google Pay field

2.4 Add the Google Pay library to your payment page

Import the JavaScript library into your payment page. Then, add an empty DIV element titled "gpay-container". This DIV acts as a placeholder where the Google Pay button will appear. You can place this DIV anywhere you want the button to show up on your website.

JavaScript

```
<!DOCTYPE html>
<html lang="en">
 <script>
   function onGPayApiLoaded() {
     googlePayHandler.initialize({
       gpayButtonContainerId: 'gpay-container' // The ID of the div where the
button should appear
     });
   }
 </script>
  <body>
    <div id="gpay-container"></div>
    <script type="text/javascript" src="main.js"></script>
    <script async src="https://pay.google.com/gp/p/js/pay.js"
onload="onGPayApiLoaded()"></script>
  </body>
</html>
```

The is your designated spot for the button.
The script tag for pay.js asynchronously loads the Google Pay library.
The onload="onGooglePayLoaded()" attribute ensures that your onGooglePayLoaded() function (which checks if Google Pay is ready) runs as soon as the library finishes loading.

2.5 Customizing the Google Pay button

The Google Pay button should fit with your website's design and user experience. Consult the documentation from Google for how to achieve this goal. After the Google Pay API is loaded and ready, your site can display the Google Pay button. It is dynamically generated by the Google Pay library and placed inside the gpay-container you made.

JavaScript

```
const GPAY_BUTTON_CONTAINER_ID = 'gpay-container';
function renderGooglePayButton() {
  const button = getGooglePaymentsClient().createButton({
    buttonColor: 'default',
    buttonType: 'buy',
```

```
    buttonRadius: 4,
    buttonLocale: 'en',
    onClick: onGooglePaymentButtonClicked,
    allowedPaymentMethods: baseGooglePayRequest.allowedPaymentMethods,
  });
  document.getElementById(GPAY_BUTTON_CONTAINER_ID).appendChild(button);
}
```

The createButton() library method takes the ButtonOptions configuration argument that lets you define how you want the button to look and behave.

GPAY_BUTTON_CONTAINER_ID: Constant to hold the ID (gpay-container) of the HTML element where you want the button to appear.
renderGooglePayButton(): Function that is responsible for creating and adding the button.
getGooglePaymentsClient().createButton({...}): Uses the Google Pay client to generate the button. The ButtonOptions object then allows for customization:

- buttonColor: Choose the appearance (e.g., 'default', 'black', 'white').
- buttonType: Defines the text on the button (e.g., "Buy/Checkout with Google Pay").
- buttonRadius: Adjusts the roundness of the button's corners.
- buttonLocale: Sets the language for the button's text.
- onClick: onGooglePaymentButtonClicked: Prompts the button to call your onGooglePaymentButtonClicked function (which initiates the payment process) whenever a customer clicks on it.
- allowedPaymentMethods: Ensures the button only appears if there are payment methods available that match what you've configured in your baseGooglePayRequest.

document.getElementById(...).appendChild(button): Takes the button that Google Pay created and inserts it directly into the DIV element on your webpage, making it visible to the user.

2.6 Handling the Google Pay Loaded Event

The onGooglePayLoaded() function is called when the Google Pay API script has finished loading. In essence, onGooglePayLoaded() is the handshake with the Google Pay service, ensuring everything is in order before you present the payment option to your customers.

JavaScript

```
function onGooglePayLoaded() {
  const req = deepCopy(baseGooglePayRequest);
  getGooglePaymentsClient()
    .isReadyToPay(req)
    .then(function (res) {
      if (res.result) {
        renderGooglePayButton();
      } else {
        console.log('Google Pay is not ready for this user.');
      }
    })
```

```
    .catch(console.error);
}
```

2.7 Configuring Google Pay

The baseGooglePayRequest object defines the fundamental configuration for all Google Pay requests.

```javascript
const baseGooglePayRequest = {
  apiVersion: 2,
  apiVersionMinor: 0,
  allowedPaymentMethods: [
    {
      type: 'CARD',
      parameters: {
        allowedAuthMethods: ['PAN_ONLY', 'CRYPTOGRAM_3DS'],
        allowedCardNetworks: ['AMEX', 'DISCOVER', 'MASTERCARD', 'VISA'],
      },
      tokenizationSpecification: {
        type: "PAYMENT_GATEWAY",
        parameters: {
          "gateway": "inoviopay",
          "gatewayMerchantId": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
                }
      },
    },
  ],
  merchantInfo,
};
Object.freeze(baseGooglePayRequest);
let paymentsClient = null;
function getGooglePaymentsClient() {
  if (paymentsClient === null) {
    paymentsClient = new google.payments.api.PaymentsClient({
      environment: 'TEST', // Environment configuration
      merchantInfo,
      paymentDataCallbacks: {
        onPaymentAuthorized: onPaymentAuthorized,
        onPaymentDataChanged: onPaymentDataChanged,
      },
    });
  }
  return paymentsClient;
}
```

Key properties that shape the Google Pay experience:

- `apiVersion` and `apiVersionMinor`: These simply tell Google Pay which version of the Google Pay API you are using.
- `allowedPaymentMethods`: Array that declares what types of payments you accept.

    - parameters: Within the CARD type, you specify the details:
        - allowedAuthMethods: Defines the authentication methods your integration supports:
            - PAN_ONLY: Used for cards saved directly to a user's Google account. When used, Google Pay returns the actual card number.
                - To use 3D Secure with PAN_ONLY, submit your third party 3-D Secure details with the authorization.
                - The Payment Gateway will not prevent a transaction from being sent to the processor without 3DS on the transaction request.
            - CRYPTOGRAM_3DS: Used cards tokenized via the Google Wallet app. Instead of the real card number, a device-specific token is used. A 3-D Secure cryptogram is generated on the user's device, providing stronger security and often shifting liability for fraud to the card issuer. We recommend using this allowedAuthMethods only.
        - allowedCardNetworks: The list of major card networks accepted. You have to use the following API-specific values: 'AMEX', 'DISCOVER', 'MASTERCARD', 'VISA'. For more information, see the official Google Pay documentation: https://developers.google.com/pay/api/web/reference/request-objects#CardParameters
        -
    - tokenizationSpecification: Configuration for how payment credentials are secured and sent to your payment processor.
        - type: "PAYMENT_GATEWAY": Indicates that the payment gateway will handle the tokenization process on your behalf.
        - parameters: These are gateway-specific details essential for secure processing provided to you when you make a request for your configuration data from the gateway:
            - gateway: The name of your payment gateway (e.g., "inoviopay").
            - gatewayMerchantId: Your unique ID provided by the gateway.
- Object.freeze(baseGooglePayRequest): Prevents any accidental changes to your base Google Pay configuration after it is set.
- The paymentsClient variable is initially null. The getGooglePaymentsClient() function uses a common pattern called "lazy initialization." It creates the PaymentsClient instance only when ready, improving performance by not creating it prematurely.
- environment: The above example is set to 'TEST', which is perfect for development and debugging. Remember to change this to 'PRODUCTION' when you're ready to go live with real transactions!
- paymentDataCallbacks: Define the functions to handle events during the Google Pay flow:
    - onPaymentAuthorized: When a user is ready to finalize their purchase, the payment information is securely transmitted to the designated payment service provider for processing.
    - onPaymentDataChanged: Callback that fires if the user makes changes to their payment data (like selecting a different shipping address or payment method) within the Google Pay sheet, allowing you to dynamically update the order total or shipping options.

2.8 Handling the Google Pay Button Click

When a user clicks your Google Pay button, the onGooglePaymentButtonClicked() function swings into action. Its job is to gather all the specifics for the current transaction and then launch the Google Pay payment window.

JavaScript

```
// @namespace googlePayHandler
// @description A self-contained handler for a professional and secure Google
Pay integration.
// This script manages fetching configuration from a secure backend, setting up
the Google Pay client, and handling the entire payment lifecycle.
const googlePayHandler = {
  // Default settings can be overridden by the customer during initialization.
  config: {
    gpayButtonContainerId: 'gpay-container',
    currencyCode: 'USD',
    countryCode: 'US',
    environment: 'TEST', // Should be 'PRODUCTION' for live transactions
  },
  // --- Internal State ---
  paymentsClient: null,
  merchantConfig: null,
  // CRITICAL: Fetches the merchant configuration from the CUSTOMER'S backend.
  // This is a vital security measure. API credentials should NEVER be exposed
in client-side code.
  // Your customer must implement a server endpoint that securely communicates
with your API.
  // @returns {Promise<Object|null>} A promise that resolves with the merchant
configuration or null on failure.
  async fetchMerchantConfigFromServer() {
    // --- CUSTOMER ACTION REQUIRED ---
    // This URL must point to an endpoint on YOUR CUSTOMER'S server.
    // Their server is responsible for making the secure, server-to-server call
to the InovioPay API.
    const customerBackendUrl =
'https://api.customer-website.com/get-payment-config';
    try {
      const response = await fetch(customerBackendUrl);
      if (!response.ok) {
        throw new Error(`Network response was not ok: ${response.statusText}`);
      }
      const configData = await response.json();
      console.log("Successfully fetched merchant configuration.");
      return configData;
```

```
    } catch (error) {
      console.error("Fatal Error: Could not retrieve merchant configuration
from the server.", error);
      // Optional: Display a user-friendly error message in the UI.
      // e.g.,
document.getElementById(this.config.gpayButtonContainerId).innerText = "Payment
system unavailable.";
      return null;
    }
  },
  // Initializes the Google Pay client if it doesn't already exist.
  // @returns {google.payments.api.PaymentsClient}
  getGooglePaymentsClient() {
    if (this.paymentsClient === null) {
      if (!this.merchantConfig) {
        throw new Error("Cannot initialize Google Payments client without
merchant configuration.");
      }
      this.paymentsClient = new google.payments.api.PaymentsClient({
        environment: this.config.environment,
        merchantInfo: {
            merchantId: this.merchantConfig.HOST_CONFIG.merchantId,
            // Customer will set their own display name
            merchantName: this.merchantConfig.HOST_CONFIG.merchantName ||
'Sample Merchant',
        },
        paymentDataCallbacks: {
          onPaymentAuthorized: this.onPaymentAuthorized,
          onPaymentDataChanged: this.onPaymentDataChanged,
        },
      });
    }
    return this.paymentsClient;
  },
  // Main entry point. Initializes the Google Pay flow.
  // @param {Object} [userConfig={}] - Customer-specific configuration to
override the defaults.
  async initialize(userConfig = {}) {
    // 1. Merges customer's configuration with defaults.
    this.config = { ...this.config, ...userConfig };
    // 2. Securely fetches the configuration from the customer's backend.
    this.merchantConfig = await this.fetchMerchantConfigFromServer();
```

```
    if (!this.merchantConfig) {
      console.error("Google Pay initialization failed: Merchant configuration
is missing or could not be fetched.");
      return;
    }
      const isReadyToPayRequest = this.createBaseRequest();
   this.getGooglePaymentsClient()
      .isReadyToPay(isReadyToPayRequest)
      .then((isReadyToPayRequest) => {
        if (isReadyToPayRequest.result) {
          this.renderGooglePayButton();
        } else {
          console.log('Google Pay is not ready for this user.');
        }
      })
      .catch(console.error);
  },
  // Creates and appends the Google Pay button to the configured container.
  renderGooglePayButton() {
    const container =
document.getElementById(this.config.gpayButtonContainerId);
    if (!container) {
        console.error(`Google Pay button container with ID
"${this.config.gpayButtonContainerId}" was not found in the DOM.`);
        return;
    }
    // Clears any previous content (e.g. error messages)
    container.innerHTML = '';
    const button = this.getGooglePaymentsClient().createButton({
      onClick: this.onGooglePaymentButtonClicked.bind(this),
      allowedPaymentMethods: this.createBaseRequest().allowedPaymentMethods,
    });
    container.appendChild(button);
  },
  // Creates the base payment request object required by the Google Pay API.
  // @returns {Object} The Google Pay base request object.
  createBaseRequest() {
    return {
      apiVersion: 2,
      apiVersionMinor: 0,
      allowedPaymentMethods: [{
        type: 'CARD',
```

```
        parameters: {
          allowedAuthMethods: ['PAN_ONLY', 'CRYPTOGRAM_3DS'],
          allowedCardNetworks: ['AMEX', 'DISCOVER', 'INTERAC', 'JCB',
'MASTERCARD', 'VISA'],
        },
        tokenizationSpecification: {
          type: 'PAYMENT_GATEWAY',
          parameters: {
            gateway: this.merchantConfig.HOST_CONFIG.gatewayId,
            gatewayMerchantId: this.merchantConfig.MERCH_IDENTIFIER,
            // These values should come from your secure config fetch
          },
        },
      }],
    };
  },
  // Handles the click event from the Google Pay button.
  onGooglePaymentButtonClicked() {
    const paymentDataRequest = {
      ...this.createBaseRequest(),
      transactionInfo: {
        countryCode: this.config.countryCode,
        currencyCode: this.config.currencyCode,
        totalPriceStatus: 'FINAL',
        totalPrice: '10.00',
      },
      merchantInfo: {
        merchantId: this.merchantConfig.MERCH_IDENTIFIER,
        merchantName: this.merchantConfig.MERCHANT_NAME || 'Sample Merchant'
      },
      // The callbackIntents must match the callbacks provided in
getGooglePaymentsClient().
      // We provided onPaymentAuthorized and onPaymentDataChanged.
      // To handle dynamic shipping, you would add 'SHIPPING_ADDRESS' and
'SHIPPING_OPTION' and then build out the logic in the onPaymentDataChanged
callback.
      callbackIntents: ['PAYMENT_AUTHORIZATION'],
    };
    console.log('Requesting payment data...', paymentDataRequest);
    this.getGooglePaymentsClient()
      .loadPaymentData(paymentDataRequest)
```

```
      .catch(err => console.error("loadPaymentData error:", err)); // Catches
user cancellation or other errors.
  },
  // Callback for when payment is successfully authorized by the user.
  // @param {Object} paymentData - The authorized payment data from Google,
including the token.
  // @returns {Promise<Object>} A promise resolving with the final transaction
result.
  onPaymentAuthorized(paymentData) {
    return new Promise((resolve) => {
      console.log('Payment authorized. Response from Google:', paymentData);
      const paymentToken =
paymentData.paymentMethodData.tokenizationData.token;
      // --- CUSTOMER ACTION REQUIRED ---
      // 1. Send this `paymentToken` to your server. NEVER do this from the
client side
      // 2. On your server, use this token to make the final "charge" or
"auth/capture" call to the Payment Gateway API. Never do this from the client
side.
      // 3. Based on the response from the Payment Gateway, resolve with the
final state.
      console.log("Action required: Send this payment token to your server for
processing:", paymentToken);
      // Fetch your server here.
      resolve({ transactionState: 'SUCCESS' });
      // Example of resolving with an error if the Payment Gateway declines the
transaction:
      /*
      resolve({
        transactionState: 'ERROR',
        error: {
          intent: 'PAYMENT_AUTHORIZATION',
          message: 'Your payment was declined by the issuer. Please try another
card.',
          reason: 'PAYMENT_DATA_INVALID',
        },
      });
      */
    });
  },
}
```

```
  // Optional callback for when payment data changes (e.g., shipping address or
options).
  // @param {Object} intermediatePaymentData
  // @returns {Promise<Object>}
  function onPaymentDataChanged(intermediatePaymentData) {
    return new Promise((resolve) => {
      console.log('Intermediate payment data changed:',
intermediatePaymentData);
      // This is where you would implement dynamic updates, such as
recalculating
      // shipping costs or taxes based on the user's selected address.
      // For this example, we do nothing and resolve an empty object.
      resolve({});
    });
  }
```

The googlePayHandler is designed to make Google Pay work smoothly and securely.
Default Settings: It has some basic settings like the currency (USD), country (US), and it is in "TEST" mode right now (it would be "PRODUCTION" for live transactions).
Getting Merchant Information: The system must get important setup information from the store's hidden computer server. Never put sensitive payment details directly on the website itself. The store's server talks securely to the Gateway. If it can't get this information, Google Pay can't start.
Setting Up Google Pay: Once the googlePayHandler has the store's information, it sets up the connection to Google Pay.
Checking if You Can Pay with Google Pay: The system checks if the customers device and browser are ready to use Google Pay. If both are true, it shows the Google Pay button. If not, it will hide the button.
Showing the Google Pay Button: If everything is ready, the googlePayHandler puts the Google Pay button on the page where you have told it to appear.
Preparing Your Payment Request: When you click the Google Pay button, it creates a detailed request for your payment. See Configuring Google Pay.
Opening the Google Pay Window: This request is then sent to Google Pay, which makes the familiar payment window pop up on your screen.
Getting a Secure "Token": If you successfully complete your payment in the Google Pay window, the system receives a paymentToken. Think of this as a highly secure, encrypted code that represents your payment.
Security Step: It's super important that this paymentToken is never handled directly on your web browser or device. It's immediately sent to the store's secure backend server. This server is the only place that should process this token to complete the payment with the payment company (like InovioPay). Trying to do this on your device would be a huge security risk!
Finalizing the Payment: The store's server uses this paymentToken to make the final "charge" or "authorize" transaction with the payment gateway. Based on whether that's successful, your payment is completed.

Handling Changes (Optional): If you change your shipping address within Google Pay, the system can automatically update the shipping costs or taxes.

2.9 Payment Authorization

The onPaymentAuthorized() function is a critical part of your Google Pay setup. It's called automatically after the user has successfully gone through the Google Pay sheet and given their approval for the payment. This is your moment to take that secure payment information and send it to your payment processor.

```javascript
function onPaymentAuthorized(paymentData) {
  return new Promise(function (resolve, reject) {
    // Write the data to console for debugging
    console.log('onPaymentAuthorized', paymentData);
     // --- CUSTOMER ACTION REQUIRED ---
      // 1. Send this `paymentToken` to your server.
      // 2. On your server, use this token to make the final "charge" or
"auth/capture" call to the InovioPay gateway API. NEVER do this from the client
side.
      // 3. Based on the response from the InovioPay gateway, resolve with the
final state.
    const paymentAuthorizationResult = { transactionState: 'SUCCESS' };
     // Example of resolving with an error if your gateway declines the
payment:
    /**
    const paymentAuthorizationResult ={
            transactionState: 'ERROR',
            error: {
              intent: 'PAYMENT_AUTHORIZATION',
              message: 'Insufficient funds',
              reason: 'PAYMENT_DATA_INVALID',
            },
          };
    */
    resolve(paymentAuthorizationResult);
  });
}
```

Receiving Payment Data (paymentData): When this function is called, Google Pay hands you the paymentData object. Inside this object is the secure payment token (located at paymentData.paymentMethodData.tokenizationData.token). This token is what represents the customer's payment method without exposing their actual card details.

The Critical Backend Step: The most important part of this function, though commented out in the example, is where you'd send this payment token to your own secure server. Your server is then responsible for:

- Sending the token to the Payment Gateway (e.g., InovioPay) to authorize and capture the funds.
- Receiving the processing result from the Payment Gateway.

Returning a Promise: Just like onPaymentDataChanged(), the onPaymentAuthorized (paymentData) function returns a Promise. The Promise lets you perform asynchronous operations and then resolve with the final paymentAuthorizationResult when you have it.

In essence, onPaymentAuthorized() is the bridge between the customer approving the payment in Google Pay and your system charging their card through your payment processor.

2.10 Submitting authorized payment data

Here is an example of a request to authorize a payment with Google Pay:

```
None
https://[domain]/payment/pmt_service.cfm?request_action=CCAUTHCAP&
li_count_1=1&li_prod_id_1=111&li_value_1=49.95req_username=GATEWAY
USER&req_password=GATEWAY_PASS&site_id=11111&request_response_format=JSON&reque
st_api_version=4.6&request_currency=USD&PMT_WALLET=googlepay&PMT_WALLET_CRYPTOG
RAM=xxxxxxx
```

2.10.1 Important requirements for Google Pay authorization requests

You must NOT include any of the following parameters, which will cause the Payment Gateway to reject your request:

```
None
PMT_NUMB
PMT_KEY
PMT_EXPIRY
TOKEN_GUID
PMT_ID
PMT_LAST4
PMT_ID_XTL
BILL_ADDR_STATE
BILL_ADDR_ZIP
BILL_ADDR_COUNTRY
BILL_ADDR_CITY
BILL_ADDR
```

Other gateway parameters can be included, but MUST match the values used in the Google Pay session.

- li_value_1 must match the value of paymentRequest.total in the merchantConfig data.
- request_currency must match the value of paymentRequest.currencyCode in the merchantConfig data

2.11 Rebill and Credentials on File

The Payment Gateway response will return a CUST_ID and PMT_ID on a successful authorization using a Google Pay token (in the same way that it does for standard transactions).

To authorize on the generated Google Pay payment record, the same CUST_ID and PMT_ID should be passed in on all subsequent rebills or unscheduled merchant initiated card-on-file transaction requests.