Digital Accept Secure Integration

Developer Guide

© 2024. Cybersource Corporation. All rights reserved.

Cybersource Corporation (Cybersource) furnishes this document and the software described in this document under the applicable agreement between the reader of this document (You) and Cybersource (Agreement). You may use this document and/or software only in accordance with the terms of the Agreement. Except as expressly set forth in the Agreement, the information contained in this document is subject to change without notice and therefore should not be interpreted in any way as a guarantee or warranty by Cybersource. Cybersource assumes no responsibility or liability for any errors that may appear in this document. The copyrighted software that accompanies this document is licensed to You for use only in strict accordance with the Agreement. You should read the Agreement carefully before using the software. Except as permitted by the Agreement, You may not reproduce any part of this document, store this document in a retrieval system, or transmit this document, in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written consent of Cybersource.

Restricted Rights Legends

For Government or defense agencies: Use, duplication, or disclosure by the Government or defense agencies is subject to restrictions as set forth the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

For civilian agencies: Use, reproduction, or disclosure is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer Software Restricted Rights clause at 52.227-19 and the limitations set forth in Cybersource Corporation's standard commercial agreement for this software. Unpublished rights reserved under the copyright laws of the United States.

Trademarks

Authorize.Net, eCheck.Net, and The Power of Payment are registered trademarks of Cybersource Corporation. Cybersource and Cybersource Decision Manager are trademarks and/or service marks of Cybersource Corporation. Visa, Visa International, Cybersource, the Visa logo, the Cybersource logo, and 3-D Secure are the registered trademarks of Visa International in the United States and other countries. All other trademarks, service marks, registered marks, or registered service marks are the property of their respective owners.

Version: 24.03

Contents

Recent Revisions to This Document 7
About This Guide 9
Introducing Digital Accept Secure Integration Product Suite 10
Flex API 13
Establishing a Payment Session with a Capture Context 13
REST Example: Establishing a Payment Session with a Capture Context 14
Validating the JSON Web Token 15
Retrieving the Public Key ID 16
Retrieving the Public Key 16
JAVA Example: Validating the Transient Token 17
Populating the JSON Web Token with Customer Information 19
Constructing the JSON Payload 19
Generating a JSON Web Encryption Data Object 21
Populating the Token Request 25
Microform Integration v2 27
Getting Started 28
Creating the Server-Side Context 28
Validating the Capture Context 30
Setting Up the Client Side 32
Getting Started Examples 36
Styling 43
Events 47
Security Recommendations 49
PCI DSS Guidance 50
API Reference 50
Class: Field 50
Module: FLEX 58
Class: Microform 60
Class: MicroformError 63
Events 66
Global 69
Unified Checkout 74

Unified Checkout Flow.......... 75
- Enabling Unified Checkout in the Business Center.......... 77

Server-Side Set Up.......... 78
- Capture Context.......... 78

Client-Side Set Up.......... 81
- Loading the JavaScript Library and Invoking the Accept Function.......... 81
- Adding the Payment Application and Payment Acceptance.......... 82

Transient Tokens.......... 84
- Transient Token Format.......... 84
- Token Verification.......... 85

Authorizations with a Transient Token.......... 86
- Required Field for an Authorization with a Transient Token.......... 86
- REST Example: Authorization with a Transient Token.......... 87

Capture Context API.......... 89
- Required Fields for Requesting the Capture Context.......... 91
- REST Example: Requesting the Capture Context.......... 91

Payment Details API.......... 97
- Required Field for Retrieving Transient Token Payment Details.......... 99
- REST Example: Retrieving Transient Token Payment Details.......... 99

Unified Checkout Configuration.......... 100
- Enable Digital Payments.......... 101
- Manage Permissions.......... 103

Unified Checkout UI.......... 106
- Click to Pay UI.......... 107
- Google Pay UI.......... 109
- Manual Payment Entry UI.......... 110
- Pay with Bank Account UI.......... 114
- Paze UI.......... 121

JSON Web Tokens.......... 123

Supported Countries for Digital Payments.......... 124
- Supported Countries for Digital Payments A-D.......... 124
- Supported Countries for Digital Payments E-K.......... 126
- Supported Countries for Digital Payments L-R.......... 129
- Supported Countries for Digital Payments S-Z.......... 132

Supported Locales.......... 135

Reason Codes........ 137
Click to Pay Drop-In UI........ 138
Click to Pay Customer Workflows........ 139
Recognized Click to Pay Customer........ 140
Unrecognized Click to Pay Customer........ 142
Guest Customer........ 144
Click to Pay Drop-In UI Flow........ 146
Enabling Unified Checkout in the Business Center........ 148
Server-Side Set Up........ 149
Capture Context........ 149
Client-Side Set Up........ 152
Loading the JavaScript Library and Invoking the Accept Function........ 152
Adding the Payment Application and Payment Acceptance........ 153
Transient Tokens........ 155
Transient Token Format........ 155
Token Verification........ 156
Capture Context API........ 156
Required Fields for Requesting the Capture Context........ 159
REST Example: Requesting the Capture Context........ 159
Payment Details API........ 164
Required Field for Retrieving Transient Token Payment Details........ 166
REST Example: Retrieving Transient Token Payment Details........ 166
Payment Credentials API........ 167
Required Field for Retrieving Payment Credentials........ 170
REST Example: Retrieving Payment Credentials........ 170
Unified Checkout Configuration........ 172
Upload Your Encryption Key........ 172
Enable Click to Pay........ 175
Manage Permissions........ 177
Unified Checkout UI........ 180
JSON Web Tokens........ 182
Supported Countries for Click to Pay........ 182
Supported Locales........ 185
Processing Authorizations with a Transient Token........ 187
Authorization with a Transient Token........ 187

Required Field for an Authorization with a Transient Token 187
REST Interactive Example: Authorization with a Transient Token... 188
REST Example: Authorization with a Transient Token.. 188
Authorization and Creating TMS Tokens with a Transient Token...190
Required Fields for an Authorization and Creating TMS Tokens with a Transient Token... 190
REST Interactive Example: Authorization and Creating TMS Tokens with a Transient Token...192
REST Example: Authorization and Creating TMS Tokens with a Transient Token.................192
VISA Platform Connect: Specifications and Conditions for Resellers/Partners..........................196

Recent Revisions to This Document

24.03

Click to Pay Drop-In UI

Added Click to Pay Drop-In UI.

24.02

This revision contains only editorial changes and no technical updates.

24.01

Checkout API

Removed the Checkout API, as this method is deprecated.

Unified Checkout

Added a Unified Checkout card entry form diagram. See Unified Checkout Flow (on page 75).

Updated the capture context description and request example. See Capture Context API (on page 89) and REST Example: Requesting the Capture Context (on page 91).

23.05

Revised the Flex API section and enhanced the Introduction to Digital Accept content.

23.04

Flex API v2

Added the list of possible fields to capture and tokenize and added an example that includes all possible API fields for generating the capture context.

23.03

All Integration Products

Updated the overview. See Digital Accept Overview (on page).

Added payment examples.

23.02

Unified Checkout

Added Unified Checkout Integration as an option for digital acceptance.

About This Guide

This section describes how to use this guide and where to find further information.

Audience and Purpose

This document is written for merchants who want to enable Unified Checkout on their e-commerce page.

Conventions

This special statement is used in this document:

Important: An *Important* statement contains information essential to successfully completing a task or learning a concept.

Related Documentation

Visit the Cybersource documentation hub to find additional processor-specific versions of this guide and additional technical documentation.

Customer Support

For support information about any service, visit the Support Center:

http://support.cybersource.com

Introducing Digital Accept Secure Integration Product Suite

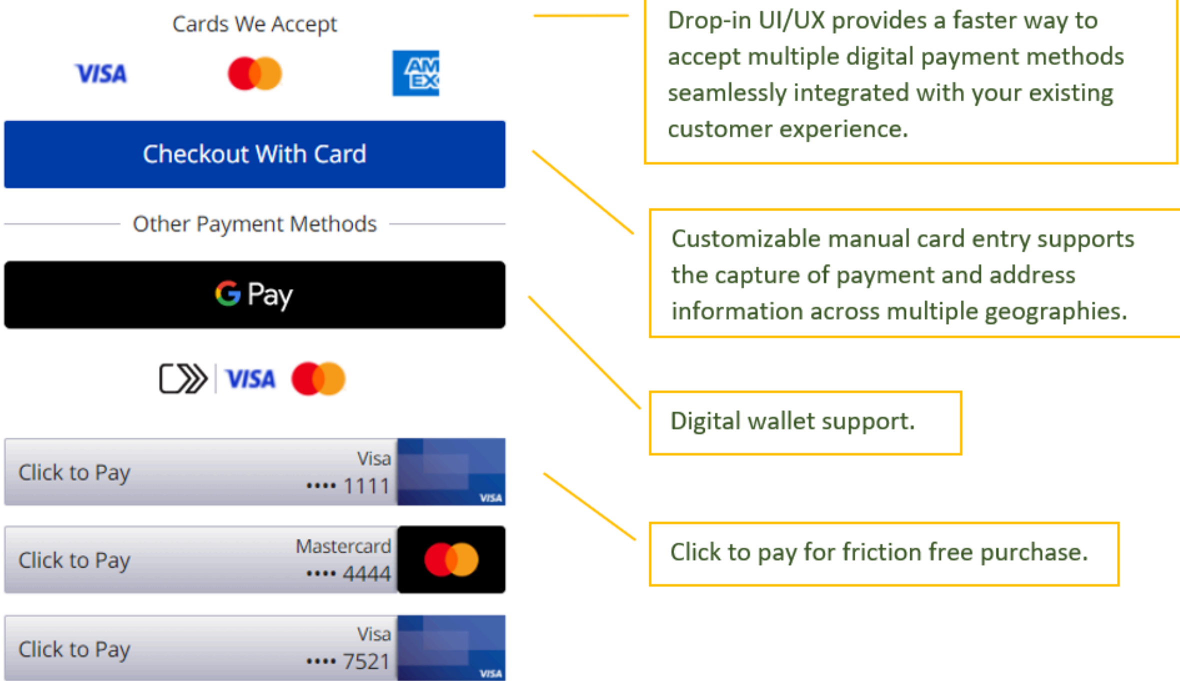
The Secure Integration Product Suite allows you to simplify the acceptance of sensitive customer payment information. When a customer enters their payment details on your webpage, app, or elsewhere, it is replaced with a transient token. Tokenization ensures that the card data can be transported securely, which limits your exposure and significantly reduces your Payment Card Industry Data Security Standard (PCI DSS) compliance burden.

The Secure Integration Product Suite consists of three products that can be used in a variety of scenarios: Unified Checkout, Microform Integration, and Flex API.

Unified Checkout

Unified Checkout is a pre-configured drop-in UI for accepting online payments. It supports multiple payment methods including traditional cards and digital wallets such as Google Pay and Visa Click to Pay. Because it is pre-configured with digital payment support, Unified Checkout enables you to go live faster and substantially reduce the development burden of accepting a multitude of payment options. This solution is ideal for sellers looking for a complete payment acceptance technology with support for multiple payment methods.

Unified Checkout Button Widget Interface



Unified Checkout includes these features:

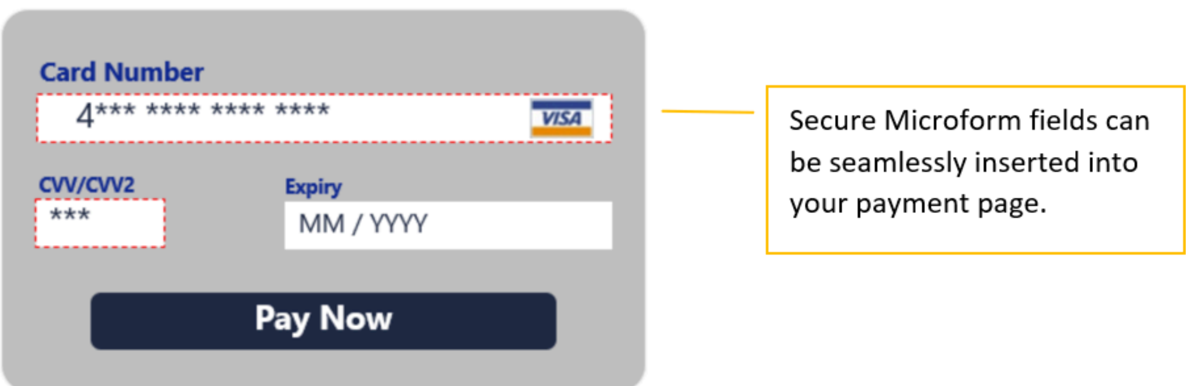
- Leading security technology
- Simple front-end integration
- Integrated with emerging digital standards
- Integrated with a range of payment methods
- Payment option presentation is optimized

For more information, see Unified Checkout (on page 74).

Microform Integration

Microform Integration is a payment card and card verification acceptance solution that can be embedded. Use it to securely accept payment information at your web page and have complete control over the look and feel of your payment form. Microform Integration captures the card number and card verification number fields from within your existing user interface. This solution is for sellers looking for a secure way to capture sensitive payment data from within their own customized payment form.

Microform Integration Payment Form Interface



Microform Integration includes these features:

- Leading security technology
- Seamlessly integrated into existing payment pages
- Fully customizable

For more information, see Microform Integration v2 (on page 27).

Flex API

Flex API can be used to securely capture and transport payment data between systems. This solution is ideal for Internet of Things (IoT) and third-party integrations. For more information, see Flex API (on page 13).

Digital Accept Product Comparison

This chart compares Digital Accept products and features.

Products and Features Comparison Chart

	Unified Checkout Integration	Microform Integration	Flex API
Drop-in UI	✓	✓	✗
Digital Wallet Support (Google Pay and Visa Click to Pay)	✓	✗	✗
Browser Based	✓	✓	✗
Complete Control of Look and Feel	✗	✓	✓
Platforms	Web only	Web only	All

Flex API

The Flex API enables merchants to securely accept customer payment information captured within a server-side application using a set of APIs. These APIs protect your customer's primary account number (PAN), card verification number (CVN), and other payment information by embedding it within a transient token. This allows payment data to be stored and transported and complies with the Payment Card Industry Data Security Standard (PCI DSS) policies and procedures. These transient tokens can be validated by the receiver to ensure the data integrity and protect against data injection attacks.

Warning: Flex API is intended for server-side applications only. Do not use the Flex API in client-side applications. To add secure payments directly into client-side code, use Unified Checkout.

How It Works

Follow these steps to capture payments using the Flex API:

1. Establish a payment session with a predefined customer context.
2. Validate the JSON Web Token.
3. Populate the JSON Web Token with customer information.

Customer Context

An important benefit of the Flex API is managing Personal Identifiable Information (PII). You can set up your customer context to include all PII associated with transactions, protecting this information from third parties.

Establishing a Payment Session with a Capture Context

To establish a payment session, include the API fields you plan to use in that session in the body of the request. The system then returns a JSON Web Token (JWT) that includes the capture context.

To determine which fields to include in your capture context, identify the personal information that you wish to isolate from the payment session.

Capture Context Fields

When making a session request, any fields that you request to be added to the capture context are required by default. However, you can choose to make a field optional by setting the `required` parameter to `false`.

For example, the following code snippet includes both required and optional fields:

```
"fields" : {
  "paymentInformation" : {
    "card" : {
      "number" : {
        },
      "securityCode" : {
        "required" : true
        },
    "expirationMonth" : {
      "number" : {
        "required" : false
        },
      "expirationYear" : {
        "required" : false
        }
      }
    }
  }
}
```

In this example, the **paymentInformation.card.number** and **paymentInformation.card.securityCode** fields are required and the **paymentInformation.card.expirationMonth** and **paymentInformation.card.expirationYear** fields are optional. The inclusion of the **paymentInformation.card.number** field in the request sets it as a required field and, therefore, you do not need to include the **paymentInformation.card.number.required** field.

Endpoint

Production: `GET https://api.cybersource.com/flex/v2/sessions`

Test: `GET https://apitest.cybersource.com/flex/v2/sessions`

REST Example: Establishing a Payment Session with a Capture Context

Production Endpoint: `GET https://api.cybersource.com/flex/v2/sessions`

Test Endpoint: `GET https://apitest.cybersource.com/flex/v2/sessions`

Request

```
{
  "fields" : {
    "paymentInformation" : {
      "card" : {
        "number" : { },
        "securityCode" : {
          "required" : false
        },
        "expirationMonth" : {
          "required" : false
        },
        "expirationYear" : {
          "required" : false
        },
        "type" : {
          "required" : false
        }
      }
    }
  }
}
```

Response to Successful Request

```
JWT is returned.
```

Validating the JSON Web Token

When the system has returned the transient JWT, validate the token's authenticity. Retrieve the public key signature that is part of the transient JWT and compare that signature with the public key returned from Cybersource.

Follow these steps to validate the key:

1. Retrieve the public key ID (`kid`) from the transient JWT header.
2. Retrieve the public key from Cybersource.
3. Validate the public key signature.

Retrieving the Public Key ID

A JSON Web Token (JWT) includes these three elements:

- Header
- Payload
- Signature

Each element is separated by a period (.) in this format: `header.payload.signature`.

The `kid` parameter within the JWT header is the public key ID. You use this ID to request the public key using the `/flex/v2/public-keys/{kid}` endpoint.

Decrypting the JWT Header

The JWT is Base64-encoded. You must decrypt the token before you can see the `kid` parameter.

Example: Header


```
eyJraWQiOiJ6dSIsImFsZyI6IlJTMjU2In0K
```


Example: Decrypting Header on the Command Line

```
echo 'eyJraWQiOiJ6dSIsImFsZyI6IlJTMjU2In0K' | base64 --decode
```

Example: Output

```
{"kid":"zu","alg":"RS256"}
```

Retrieving the Public Key

When you obtain the `kid` value from the JWT header, use that value to retrieve the public key. To retrieve the public key, send a request to the `/flex/v2/public-keys/{kid}` endpoint.

The public key is returned as a JSON Web Key (JWK).

Request

Endpoiont: `GET https://apitest.cybersource.com/flex/v2/public-keys/zu`

```
{}
```

Response to Successful Request


```
{
    "kty": "RSA",
    "use": "enc",
    "kid": "zu",

  "n": "ozmvkuGzWNHs9cEcC5PWwbG-dmSjPcoQFxEbqH_fBjkj_nfTTKshdiSq5ciulWEa_rrqQ2qwcSADNxtTzR
R1qfud-NvsM8Vlt

 T7xDuVVqPTZoWLKa0BWXgQQ-1mCm1KdGltYWccB0R1LoF-rb3DEEZySsHvqErYzYt4M_rqjEiK5Y9y1h3k1h5Yk4z
GLWchko3

 jiDS-pVevvWsQsN-Y3KuB19485G9P_MXLtfJWQ4wC4jlo9etdD_hgDfxX-hQy3wuwHfHifLdxvxiB8X5Is4m6DuY4
_7hS5RwX
          AjO1QSd-zUYZNT_2yWVR56_jyiZEiOdgIm9QtLPZCTKzqsXoqZQ",
    "e": "AQAB"
}
```


JAVA Example: Validating the Transient Token

The Java code below can be used to validate the transient token with the public key.

```
package com.cybersource.example.service;


import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.cybersource.example.config.ApplicationProperties;
import com.cybersource.example.domain.CaptureContextResponseBody;
import com.cybersource.example.domain.CaptureContextResponseHeader;
import com.cybersource.example.domain.JWK;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.math.BigInteger;
```

```java
import java.security.KeyFactory;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.RSAPublicKeySpec;
import java.util.Base64;
import java.util.Base64.Decoder;

@Service
@RequiredArgsConstructor
public class JwtProcessorService {

    @Autowired
    private final ApplicationProperties applicationProperties;
    @SneakyThrows
    public String verifyJwtAndGetDecodedBody(final String jwt) {
        // Parse the JWT response into header, payload, and signature
        final String[] jwtChunks = jwt.split("\\.");
        final Decoder decoder = Base64.getUrlDecoder();
        final String header = new String(decoder.decode(jwtChunks[0]));
        final String body = new String(decoder.decode(jwtChunks[1]));

        // Normally you'd want to cache the header and JWK, and only
 hit /flex/v2/public-keys/{kid} when the key rotates.
        // For simplicity and demonstration's sake let's retrieve it every time
        final JWK publicKeyJWK = getPublicKeyFromHeader(header);

        // Construct an RSA Key out of the response we got from the /public-keys endpoint
        final BigInteger modulus = new BigInteger(1, decoder.decode(publicKeyJWK.n()));
        final BigInteger exponent = new BigInteger(1, decoder.decode(publicKeyJWK.e()));
        final RSAPublicKey rsaPublicKey = (RSAPublicKey)
 KeyFactory.getInstance("RSA").generatePublic(new RSAPublicKeySpec(modulus, exponent));

        // Verify the JWT's signature using the public key
        final Algorithm algorithm = Algorithm.RSA256(rsaPublicKey, null);
        final JWTVerifier verifier = JWT.require(algorithm).build();

        // This will throw a runtime exception if there's a signature mismatch.
        verifier.verify(jwt);

        return body;
    }
    @SneakyThrows
    public String getClientVersionFromDecodedBody(final String jwtBody) {
        // Map the JWT Body to a POJO
        final CaptureContextResponseBody mappedBody = new
 ObjectMapper().readValue(jwtBody, CaptureContextResponseBody.class);

        // Dynamically retrieve the client library
        return mappedBody.ctx().stream().findFirst()
                .map(wrapper -> wrapper.data().clientLibrary())
```

```
                .orElseThrow();
    }

    @SneakyThrows
    private JWK getPublicKeyFromHeader(final String jwtHeader) {
        // Again, this process should be cached so you don't need to hit /public-keys
        // You'd want to look for a difference in the header's value (e.g. new key id
 [kid]) to refresh your cache
        final CaptureContextResponseHeader mappedJwtHeader =
                new ObjectMapper().readValue(jwtHeader,
 CaptureContextResponseHeader.class);

        final RestTemplate restTemplate = new RestTemplate();
        final ResponseEntity<String> response =
                restTemplate.getForEntity(
                        "https://" + applicationProperties.getRequestHost()
 + "/flex/v2/public-keys/" + mappedJwtHeader.kid(),
                        String.class);
        return new ObjectMapper().readValue(response.getBody(), JWK.class);
    }
}
```

Populating the JSON Web Token with Customer Information

As soon as the transient token is validated, you can add the customer's personal information to the token.

Follow these steps to populate the token:

1. Construct the JSON payload.
2. Generate the JSON Web Encryption (JWE) data object.

Constructing the JSON Payload

To construct the JSON payload, create a JSON dataset that includes these elements:

- `data:` The payload. This payload must include all required fields and can contain any or all of the optional fields in the transient token's capture context.
- `context:` The capture context from the transient token. The transient token's payload is the claimset
- `index:` Specifies the recipient key used.

The payload should follow this format:

```
{
      "data": {
      [Claim set field data]
      },
      "context": [Claimset (payload) extracted from the transient token],
      "index": 0 //In this case, there is only one recipient for the JWT, so this value
 must be set to 0.
      }
```

Example

```
{
  "data": {
    "paymentInformation": {
      "card": {
        "number": "4111111111111111",
        "expirationMonth": "12",
        "expirationYear": "2031",
        "type": "",
        "securityCode": ""
      }
    },
    "orderInformation": {
      "amountDetails": {
        "totalAmount": "102.21",
        "currency": "USD"
      },
      "billTo": {
        "firstName": "John",
        "lastName": "Doe",
        "address1": "1 Market St",
        "locality": "san francisco",
        "administrativeArea": "CA",
        "postalCode": "94105",
        "country": "US",
        "email": "test@cybs.com",
        "phoneNumber": "4158880000"
      }
    }
  },
  "context": "eyJraWQiOiIzZyIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2Vu
cyIsImRhdGE
iOiJyMlh5b2QxUk9SdUEyajFwUnA0cUpoQUFFSkFvUVN1QzZzZXFkVHpMaUJuTmZrMzljOXJQSHJnQTRsSEZ1QXRrS
0JiRmpqa0tH
```

```
V2tmNUVjNHhBRVBMTzc0b0NsdjhneUhueFJOb1E1dHYwVnpNYU5pOWNxd21EWmJReExxENW5pVk1SWGMiLCJvcmlnaW
4iOiJodHRwc
zovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbSIsImp3ayI6eyJrdHkiOiJSU0EiLCJlIjoiQVFBQiIsInVzZSI6ImV
uYyIsIm4iOi
JqYlA4dHpIX21FQUloYUdmcXJ3TEQtZHZsbTZSLXgySWVaVDNweUU2YXF2SkxkY0h4bzRQZktOSXpMZ0hfZEJVTjZE
NGxFc2dTY3N
oT1RVOVVGVVQyVERpZUlaMVJjNW5rclNub2lYcmR5MFJscUlrS3BCa2h1WXRsSWM4OTZQb3JYVENmUk45MmpXOXgzN
2dUUnRBc2l2
QXJQR2p0WGV4QnhaN29SWkFXRVY5Yy1FYVFybU55N2ZzTnJxdEZMR2xVbXdEQ05ONEVERXdjaWd3ck5JUlJQaHpPQk
J5UWFvenB6V
lhXSVctS3RRb2otSHFfTmk2YUN0MXkwdWVLZjFkZ0dyUHpibDV6WVNFYUJtM3gzdGZzTmM3MXVQbGJXZzY0LU83Snl
McFJWVU5UYn
R1NC1ONWNic0ZaMnZBeGYwWTdWRnRaclZiR0ZTRmFLQjZPWVdWVnciLCJraWQiOiIwOGlHZEN2Z2lCWEM4YXd6U0sz
WjRoUm9hbEl
KTzVvMSJ9fSwiY3R4IjpbeyJkYXRhIjp7InRhcmdldE9yaWdpbnMiOlsiaHR0cDovL2xvY2FsaG9zdDozMDAwIiwia
HR0cDovL2xv
Y2FsaG9zdDo1MDAwIl0sIm1mT3JpZ2luIjoiaHR0cHM6Ly90ZXN0ZmxleC5jeWJlcnNvdXJjZS5jb20ifSwidHlwZS
I6Im1mLTAuM
TEuMCJ9XSwiaXNzIjoiRmxleCBBUEkiLCJleHAiOjE2MDQ2MTc4MjgsImlhdCI6MTYwNDYxNjkyOCwianRpIjoiR1o
xb1dCbTVBbH
kzendwOCJ9.ZF9-CG_FvIQTMocIMwcBH6IMWBiFfl-ufPj0TdXFuTSpusL6fAsxnyxdlf6V6i6wO0PDgv6SY-2MWP-
Q600WAjFZfm
R1y3r13Tig9Ldql4WOp8zhIb6klLD01PYWeyXYZ0xqRQL0_eYTliDrV66P72PVX6DqCeoJFYnh_csEcAChmyBVRqI2
Gxd9zelALqB
NU6WeHiN8FT36xRHHruxRJ2hBCI_OE0p9haQjuD4qtfk9grfhnt2mFpiC4s0j0yHaHCgiVm5NPuPecpS7t47cjsSG6
PfIHNbBAjdI
VcNpmFFyH6sCLRplOgW0vPYw4nUOgtq7y_voHe_nOal6eHFr4A",
  "index": 0
}
```

Generating a JSON Web Encryption Data Object

The JSON web encryption (JWE) data object is built using these elements:

- `header`: Include the `kid` and `alg` parameters.
- `Content Encryption Key (CEK)`: The unique encryption key used to encrypt the token.
- `ciphertext`: The encrypted JSON payload.
- `initialization vector`: A Base64-encoded randomly generated number that is used along with a secret key to encrypt data.
- `authentication tag`: Created during the encryption, this tag enables the verifier to prove the integrity of the ciphertext and the header.

The payload should use this format:

```
header.cek.cyphertext.initialization_vector.auth_tag
```

For more information about JWE data objects, see RFC 7516.

Example

Important: Line breaks have been added for readability and formatting.


```
eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImVuYyI6IkEyNTZHQ00iLCJhbGciOiJ
SU0EtT0FFUCJ9.juQDhF5XcZ1rDbupn1nZ1qHhephzWpa8FumH4KrsD0yF1tCOD0L8WfpSyd5VGIewb4I1IipmS
B5vV0O3Cb6FrNLipjFq-oexFRwSK92NbB88ySFO-7FyvPddiqaQFkA81xn8nwdoHMwUsQuqe8Ts_krLsvYghmsc
xXKkwcEKqxoWbmD-yEfvKxGyHACLprAKLm-xusexaJLF42OTxYuEhzzrSe6MRll0zXuk2DAhtUL2oHCgu8P3shg
JBJqsOPcAFtwtLBRoDwlDt0ybOHjd34Svbpgf_3ncFnDkEQYe5QeElEHaB2a0Nbwo61I1UETfhedHQc8IMtDmVu
Kk9pgCTg.uWrwGp2jZxZd5wF0.oFzZ3I2ry77jf-3wB_2q8G-0tbYJWQj88NdzRmVNO34JbreX5WOCju7ntvN8h
83NJXEA_cQech2PEGIZV_tADBaLbSxJeitYKwaQhs_tRVrzrcd8Qhgs4OADfky2m310eV8bUG8D4GZBKRHL6ScL
f5p30b6Hoa5fDYsU7IHNyCReiaiGPExlY4luwL9QQxrfY2LTv74Pcqyh-B4byNxR5hTw3SJm7DT7YQLl6_-2ROq
JhJoweTdDJtmJoM-LxKEij2TLgHBdqso9f036dfn0SHLl1vG86C1-6DA9yFIZB3gLYnyom1jZuGxUOPXDojUfXo
0OpUj8OI6CnQWdhKpC9X19s8xAhIAUYYdvWrEqFfBzd9S-4E-ZdyUGfxG7fLQuLZKQJeYBbGCssLGSIXLOb15sK
OopIgqCTU7M5EN_F7zW0IwJ4-b8OVf_J80-hW1e043RlzBoMr3aGdXFIaLmVbEIzTNeZrulYTTWWLbQlcLTXqAM
0yFlKmIrpq55VruvVR8i_iju5MFzzTYuLut9ecvYbFFeUkUaUBihNXg4Np57Ix23gaJuMcPBgUqkH3nCTZQE7yQ
OynzO-lho_jAHy1xcwV_DJhhAJnACO5HUDAjVKmr-GKqxvDZWVzrqjFkPArX81eRSnn9Dr2Ahozehn9FTB37AJV
3BEC2i7WMvAbQE1EpPVGTdvVDhH2xlLAHqHTBeQakzY4e81h2L3EDCmdjx_yZdZOUUSG3mLQSp864OV5pHc2X22
ZRadGbrLwnA-m2W1oDZIzh2t5nZdJhePnNzHbNXTf0xWSklxdgJdfG52FVSH-cKiJQnDhmCH6nPVK7NKnL0vRuZ
-uuOa4PJQDoT2H8eSjpvo8fo9rwfLYmQJa042t7OSE95bER9k1oJTUm83LNA3bxhWk5en2UFgcip3z3KlOmFwPL
VNCpzitULzAEHwBJlrB0aGXkQi1bJMxo9XZNREnFyYAlX3-aruXIe47pwAyOEX-hd-3Y7UsxBVYB86se51q2-VU
ldR0zj6cwZvrTxhFM_gAsD0HisAGa6E3n3n3w1JAvjuZdHRoQqaT00YFmTdSbocmTOEUammYmBjagKKycOzgmoZ
SaYpffQl_R06tEZke6uhJrPQuTwLwivZMtnWE8O16VIRX4cG3OfzaRYs0GvPWumDlrSbM8FugMIEaUTng5T9Cdk
ixegRmszDELzNjNTJLe2WwxJG4Kb_1-yGMRlhFys4FEwVMk8AWJJRDpwG0jdmHkBz9l7z1PFdIcidbIpmgH7m5R
D6kwRSxaG_BJWDc2IkIFyNa2G_-gHjQh_utablUOL9CXxxFCKD9UHojtsHneFt1bhV2P_sfYYhtZo5XloKAAEXq
mOSY2boYyj0hMlKNuVqukrnWG6-bV-LBf9DvpYNKO9YeU6rYD_WOxSQlliqVvEK8n9xLCmQQKsK2Xj2WGh7wWTQ
TMh18hcsNENN3Loq9DofAbOrCXqdREAshxg_MOI5vGe0JvIR9Gj6kAhKGFf2DYBqMynbb9jWJnjCzFXBCqXXjTO
uCoZdzlV9RbLxIBOOojIfLfdtVLGKPLKizXaSQ8YrLiBATarkpO7WFSSF66lvezwDZlfDErA-0kij1n2poKqDLY
L3vNfX8vU33ef96VQc9I3auTpiWd0NLa5yw0RWREAjqa4pHYTEZDiLcD0vETt84_aon3U7co_8fAYrztokTIJ2O
RuhN_xA0rV1MbOZIwW6m-duqYLFLQlcwjxNwTdaberNy6bCg9otljd5l7nSbzZ6UpHrHDF02LrM41NmQUx9tZFH
ypYjFdgiKKgqk-kTe3pq6ithsTPvcDvDkNgCSb9H_X30qm2-0VXaGIcYBcmJdsbBt7VJuYVZ1I_2l4-_6glgvgQ
z9d5KaHyZeJimSXqOsbqUQzNKWC7_K81Z5XmqCPJByrOiROkO6iEe_poqRgVzHETHYmstAzUlgUvPD3XocZdlHu
PHArQe6GddVmxnhTDV1M0TmXwK03f0jGg7LMjWjU1k15X8xYZTk_HMo76IetUOdf9BIoaMBqMHJkk936uzjIeiW
1DbEb4ExLtpIeSoq_fnelAWoVEDMa_XoVkWCR5R7wTJjGyZKjJJkJ6UqYQguS9oO95MZp8N0Qa41wKCvztLbFKt
EU7sPz3pU5oUVbn9cZS7WCzCUNWGxb3PO0nTzPsP_MhD71JcuAEFSLS05m1hkoNiYe_6pmLv8Rrgp71kFsTOIOU
rcUvwdJRikDOLdNbO5b-_6HjczDPzx9PaM_Zn-34mfOQPthWAfum3YvpmthuKxAWfdBChZXe9oCMeBGewGl7mKM
h9H5SP6su5yw-IFe7iBd338LVVPjRXif1rNsU631YXBu9Lz-l6o4cuGuYPVHPhHf4lifFXvlvi702wD7fbYn3cZ
55_yGVJvcFPq6OMUGJUSy5ncj-n7a8-IcGmSFpMtgnMc1ycJa_0N1vtwyjm0WvdzkUrBNC_OoCmHlLaG3XTRenL
_WYhzxDUdQQBuSC3acFu28x3NL8cmR5iqy7sBGUKcwt_ogX9ZoQyFzUTFOw.QqKIuF8EnuhOTM8PvGEs8A
```

Example Java Code

This Java example includes the code that can be used to generate the JWE data object:

```
package com.cybersource.example.service;

import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.cybersource.example.config.ApplicationProperties;
import com.cybersource.example.domain.CaptureContextResponseBody;
import com.cybersource.example.domain.CaptureContextResponseHeader;
import com.cybersource.example.domain.JWK;
import com.fasterxml.jackson.databind.ObjectMapper;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

import java.math.BigInteger;
import java.security.KeyFactory;
import java.security.interfaces.RSAPublicKey;
import java.security.spec.RSAPublicKeySpec;
import java.util.Base64;
import java.util.Base64.Decoder;

@Service
@RequiredArgsConstructor
public class JwtProcessorService {

    @Autowired
    private final ApplicationProperties applicationProperties;
    @SneakyThrows
    public String verifyJwtAndGetDecodedBody(final String jwt) {
        // Parse the JWT response into header, payload, and signature
        final String[] jwtChunks = jwt.split("\\.");
        final Decoder decoder = Base64.getUrlDecoder();
        final String header = new String(decoder.decode(jwtChunks[0]));
        final String body = new String(decoder.decode(jwtChunks[1]));

        // Normally you'd want to cache the header and JWK, and only
 hit /flex/v2/public-keys/{kid} when the key rotates.
        // For simplicity and demonstration's sake let's retrieve it every time
        final JWK publicKeyJWK = getPublicKeyFromHeader(header);

        // Construct an RSA Key out of the response we got from the /public-keys endpoint
        final BigInteger modulus = new BigInteger(1, decoder.decode(publicKeyJWK.n()));
```

```
        final BigInteger exponent = new BigInteger(1, decoder.decode(publicKeyJWK.e()));
        final RSAPublicKey rsaPublicKey = (RSAPublicKey)
 KeyFactory.getInstance("RSA").generatePublic(new RSAPublicKeySpec(modulus, exponent));

        // Verify the JWT's signature using the public key
        final Algorithm algorithm = Algorithm.RSA256(rsaPublicKey, null);
        final JWTVerifier verifier = JWT.require(algorithm).build();

        // This will throw a runtime exception if there's a signature mismatch.
        verifier.verify(jwt);

        return body;
    }
    @SneakyThrows
    public String getClientVersionFromDecodedBody(final String jwtBody) {
        // Map the JWT Body to a POJO
        final CaptureContextResponseBody mappedBody = new
 ObjectMapper().readValue(jwtBody, CaptureContextResponseBody.class);

        // Dynamically retrieve the client library
        return mappedBody.ctx().stream().findFirst()
                .map(wrapper -> wrapper.data().clientLibrary())
                .orElseThrow();
    }

    @SneakyThrows
    private JWK getPublicKeyFromHeader(final String jwtHeader) {
        // Again, this process should be cached so you don't need to hit /public-keys
        // You'd want to look for a difference in the header's value (e.g. new key id
 [kid]) to refresh your cache
        final CaptureContextResponseHeader mappedJwtHeader =
                new ObjectMapper().readValue(jwtHeader,
 CaptureContextResponseHeader.class);

        final RestTemplate restTemplate = new RestTemplate();
        final ResponseEntity&lt;String> response =
                restTemplate.getForEntity(
                        "https://" + applicationProperties.getRequestHost()
 + "/flex/v2/public-keys/" + mappedJwtHeader.kid(),
                        String.class);
        return new ObjectMapper().readValue(response.getBody(), JWK.class);
    }
}
```

Populating the Token Request

When you have created the JWE data object, insert that object into the body of a request, and send it to the `token` endpoint.

Production Endpoint: `GET https://api.cybersource.com/flex/v2/token`

Test Endpoint: `GET https://apitest.cybersource.com/flex/v2/token`

Request

Important: Line breaks have been added for readability and formatting.


```
{eyJraWQiOiIwOHBNSnRoMnFRazBGZDZNcWtHamZRS0FrOFZ0aDNncCIsImVuYyI6IkEyNTZHQ00iLCΩ
hbGciOiJSU0EtT0FFUCJ9.CXY9bqD1uFtK40xcJiENdI6vkKusaW8xa5kzWLfg1zyCgijwv1EYvZleqv
Un4VgNQPuj5cVHZLJJNIqR4EI-kAIULsSxnq5xeyEwIH0DX9suEIICAs8p9dDiUDts67lfzLsQvUHkdT
nk2z4dpnctz5DrF3YXlDOghkn3M74N2Fq_H8lp0C5e5uc8oE-B0jDWNjY4zpDZO3wFoSTKRjJZ6mALAJ
5tf-GAGGl1HxVIm4THRGud-tR1IqRpmx0RDNgBXe55JVhT7_5wA-9sOSk16ylricRqnI0BeKchB_B1Z6
v8K3pyl363EUDRSHj9TlG951h6Jcv_dpTYHbiqcx9kjA.c2M3S4GcXaQtSKB8.dCiqN1XaPb8owIz56z
zIEenXd7wlfJwWdXwj_n_rMsufiQXf3_nKSLJaH0B_3f0DEz_AIkXdfmfPkMtwxTZcBXvQVcgBv1I1wN
18FNEmEi059b0CD73ODPyXlx7NFnnNmsEeu90PQfe6C_vsnQuSMMBgYddeYn1yOmQDxmsRJjB8_fJckq
SnW91hP7HeJZny-s1EQHlYpg0CZkePCndgBGEG1BrQDfZc5iKbn4nRb9fW7XC_7OV0AjN-r2Wkf1jTI5
w6fZbmseqrpKBEsMKM44Vs_8cTyzbrDU2jome3U42fc8vMVYq2Z6Z_tMSOR_7Qrt8IErzR02E-24wO4d
qPolGhJKVFcvn3fGW590QOW0qc0Q4oMWFQemYgN_LdTWfEX5KAE-FVSRwcHQCHFAYO_SK4Tk4iTexitF
G4w4GXQsrKf1FhUZto6AGU62RCPzYuSY3TWosuAcYP5wVhMaajCd23sSSkdFR2uGtyru88UwEwVBpU5t
Ypc3_Je80LX6aZVJS73JDYky8IYhKLYEDIamI3bkIOFZUtGFWJ2ybM20JLIMGbkvV1_1wHEqpj6upCh3
JNC9rMatRjb4hsXwlzLLQ0BYizNDNgqkbIjM_uBbu692ymYNgbtfInP0Tt6I_am5_ZYsoj9X88mqOlvU
dM8-LYlrMA2hEXc5ALxh6cm2njMbUXxBFjrjEXko6znH9O5v8tzH0BhR154MWMrer8FXHtvrl7bbZQg9
ioRUsrRL6ubTLaFFeHfihAA9DK4qmiQtrMIHyIzIr5d7nPhZMHcgItGZS3jQUOu8_f-Md4QH9Hd6356k
sQut60MYFDqS0BOXNxeRCKA2eQtusP8LBJLHlJJSiBvjs_XzLDXei9uo22lPl8TpPHFmtxFuAb--fs0M
T7TOUQMaLAnJMCCdP95KGpIOixCVoE2mnmCBoebEC698hlV_93w2uslHN3nF2arFmn4V6qRsST2pMNfc
0c_3Oi80-g-e2IPgajmE-hQRo3BGqICFx0P4XC0GMtJhfYADhA9q60wjFSpJqCIcA0TJkuE8yP3i5hUl
e8ELPKr-OYQHUWN9LJM7c6ypDtSjtoehFC5-w54sMEExY9LiM3bI4VmYPIrwMdliD0iHhXJTFQBMDKiu
S0G0v4yPMsu5RSQn1Ow5rK8V4lA-Q4uVcJQOyGlZO1wTbIAVEFeEUKaApWiEGN9g0CewSSou09aLcQXe
Owiv8MaBvza4MHW-NGYfxCbTJpMYDpT1ax3L_Lht3xsPURupAJj-0_z3jdvNFV3Q6-DQfPD1gOcDqVKG
MHgnH3tUBi24_1hlJ8Mv9ji0YveaewQJgYFXHmJ11PLpRGE9jDLrPwXtdYUMpb-Jg4EZ6Ba4O_U4zly6
KUJFlxyJ9hh74CGgzbiVSz-OO7K1_7-zUPeI_gkFRfd2TaRB3PaqtW6w6-B50SGQpHkshLX_hbRKupcQ
nFeKbTd_BySBKIV0zBIGxsOGXyA0OeBygK-frFFE21dM3hbKEHvHKa2JsCgeeHztRPOi3747iT8v41Xa
2pV0PtzzKbDEJVPSFYm6B2pF9vX4uvWXS8DizsPm8CNwgzhUXYJBXxaXI498ZQzwuwBPPmx2ovJoN-rh
kVZzG4NkqVRRDLA-fcfuUkxCHzVTxdFi65LBQ-SJJ5_g4NMhWkpsvD5HbS3simIM3qke2GHeDz0V6MbT
ZNck4CJC0Qdh6ZTyQMILP12Q5SUnxhuHVQouFllJV14nm3SpifhiVkKao26sj7drn8x6TR5PhGylwys3
Z96fXG9cyBZGvne5Keigu5hLY7g0GQR8SQu989m55MRnWtFfESY08Qafg6jax54opR34K320PZtPyZgi
S0vX3TRI6QiKeI3_OW5phuqUgxnK-UhU259r9E3ckDuFM22IlZEXWWjmK0bQqwE_FGZHIxUfuXxzMmM_
I7BI6nQgxZ4KQR8ZmZOIDoPq7VdOSpIZ-7PqcJO7SE-LFFP4nGYMPeXVS3eLsOXqoRxoweho06HdKQS5
RoFC8srmO-LC-wxXHMowF_L63PDEY_ppO1YZnAZQHJatt337OCqvDrgw6SOsYxCTuroqJAaqzbcFUXBA
ZvI7JZ_df0f8fGLbyJmul2SmB-G_J0CKoFtr-fQ9GwJba1ERZHUzyBWF9-cK06lSyhbLxlD3Dl_KkmPp
```

```
piz8cGhrNSUfjNNi1CzTSxCmRCQK7Igv05Y9HVnu4SSTZi2NHqxFsEradx_9wO77ZHAQ6Mxsx0_xqk9L
19ooJBhgZXL8zsCouJWkLr1-sf5hBQO_zmyqDJFUyQzeFJhaeO8jn5xV0IBS9gEPfeogn5xP5-HLY3MQ
TpceBXobVvhfiTfKdaBkqEAUUdmAEuou6Jwwy24FbAugrhdjaXr2_5RLdmy7xuy6EGAs_T7HgkMgCrLk
r9w3zpTXSjiiBfqaoLFwUEvCFczeW33YUn0h05cjGfpw9lIE8nQ-A6Tv3TXKzrxIdRJWwGmUKE--fPi8
4LS0GmLLI_cB1_lKXKsTw9-Q-mEwk99PYr8L-W0Q0v_zlEVgq3LlGSshefKySXUKV4-CxtthRcMOZhw4
eKIMh4dtYuqlcmTaSK5YXtLIsc5bGcWAx0AM_KOx9EwlX_Ug4W71tFHanGQ-MXnoPG2atLJSmwODD2yW
ftB2zedcU3epXK83K9LZG3xoeYVh_j-9Xmd-ToaN4firdX4WhVU4h0rAOTBqgQm-pJ5U-NztXu2mdCgN
tx5ZwKIb1wzGTs8ZkbkeqJIXtPlFO1BRAq9NGcg2777ognoy21ehJZiPQTESRhe7wQ_Y0niIWylP9AV3
PJVY3Pk-GpRctZOc8WkBdTPhOyczVZs5GbBAsOeYweo9i3EKlVwloxIFMY6MQD7e300K2_OEyfq961gq
GYCJf4IzJsoP4zJAKBr71NPpqLKZbkJRPerzHwmDFCfoCfy9Sp7cHLBACwUMD32JIjIyVUC0Cjt8q0W5
zoszUDBnPNchII2mXWYfFxxc_bN_cdpuBCXW5R42u6p_J8OgQxLM7PCd9lQQ9WgS1cKG_1rabKdMIYKl
1eDi7DKK_FPBxEFbf9wMwXo2U0kaQEMEQbeLb-cMn60jiQ0pyPVMsMBFrvkiS3gLaDebu-03hShHg52C
CZsA66l_Y4ZXgNPZ5EeJeczUTftj_L827f_SDPX2m40LLeDh_8zs1EfRh2x-_PrFt2JGGZTjQ0WzDHpr
H6DWEGPCEokQqV1v3RGYaz58VcBptWS16dXZExnRA9M-Pf2hwjy32pjTodIvcT2AARbWDeb-oOMUXpGl
B1Cuk1hrqtpWES-N15ONPWRJ6VK8XWcarrz7x_LESs9pS8mrWLDNXIsFd0MUd6ZTEwlN5eaS_CtuQGcC
TIAMSSpt6DHDt24bVLIPjl9X3LzU3PgCei8wObEYOHNqsrLpM8Ps3Enuca6bbSFRT8h1pVedRSRWUN2V
4C6CROeTuid7P-PorYoV8McomHuVcPqS6kvIi5gPwi8T-pybnjyDPgcQ50JAYHWVqVw0EeC3hPMGxlU5
T9IWeC2qvhzSZ8-Iov2k3MnqNnhiLsxTuPVHNLnPhZ6UP-LHLE6vyA-4oSVQ2d5O0tiFOt4H3PQ8B-jD
zjFPEPQ-qv6K8fxtdNLja2beJyvO2v5ymYhCVjgL6DKLL4xD3JD30SJ4WmSKBPtzScFrBHitlJdyGEEt
xjYE9FLXeoJi4RplleOEXn6WH_7wqSxk9jGT78CeNIZCGZMavKUESG8oUF-vxoRX1sh1LXD26T_B3q6l
5TLaAiCF-STJI5_P99-8tWvzmdfDbXDYIAg6OMs94ohiOMhNccT-IH8AUQpauPLaX9V06w7bU28Qt8uq
SnkImQKbicr7LJ_MTIeqogfGjpnV9PWolWQ3QoKSb72Ed9OahV1mYl3fPFdMS8GKiKN1NI8sRPUbIM7D
8IOBfTZovesPcFhf80z9MP1IUXti9qpJ_T-axjhtMbZOKmQVCfoc0DP4h09vySPiRkwx7bjQZnCV6fZs
4qLrKxTxpy6mbihIKAM-v3eZMU4-UoV_mzWP_Q5nclH0jO19omLrFszXEXuIUrY1_7AUkNBiV7vjQ7F6
E7f4wQDjE1azCYwuULc7QiJ_Q5JrL5Ql_UY9iG0dkyLGA6XKUTbtZF01VgCOMuCQN677LmvXkkqGxlvY
WDpQq9TuwNzcnIUoE.Wb8jG4qNmCGq8M9cOTnfnQ
```


Response to Successful Request

```
JWT is returned.
```

Microform Integration v2

Microform Integration replaces the card number input field of a client application with a Cybersource-hosted field that accepts payment information securely and replaces it with a non sensitive token.

You can style this page to look and behave like any other field on your website, which might qualify you for PCI DSS assessments based on SAQ A.

Microform Integration provides the most secure method for tokenizing card data. Sensitive data is encrypted on the customer's device before HTTPS transmission to Cybersource. This method reduces the potential for man-in-the middle attacks on the HTTPS connection.

How It Works

The Microform Integration JavaScript library enables you to replace the sensitive card number input field with a secure iframe (hosted by Cybersource), which captures data on your behalf. This embedded field will blend seamlessly into your checkout process.

When captured, the card number is replaced with a mathematically irreversible token that only you can use. The token can be used in place of the card number for follow-on transactions in existing Cybersource APIs.

PCI Compliance

The least burdensome level of PCI compliance is SAQ A. To achieve this compliance, you must securely capture sensitive payment data using a validated payment provider.

To meet this requirement, Microform Integration renders secure iframes for the payment card and card verification number input fields. These iframes are hosted by Cybersource and payment data is submitted directly to Cybersource through the secure Flex API v2 suite, never touching your systems.

Browser Support

- Chrome 37 or later
- Edge 12 or later
- Firefox 34 or later

- Internet Explorer 11 or later
- Opera 24 or later
- Safari 10.1 or later

Getting Started

Microform Integration replaces the primary account number (PAN) or card verification number (CVN) field, or both, in your payment input form. It has two components:

- Server-side component to create a capture context request that contains limited-use public keys from the Flex API v2 suite.
- Client-side JavaScript library that you integrate into your digital payment acceptance web page for the secure acceptance of payment information.

Implementing Microform Integration is a three-step process:

1. Creating the Server-Side Capture Context (on page 28)
2. Setting Up the Client Side (on page 32)
3. Validating the Transient Token (on page 34)

Version Numbering

Microform Integration follows Semantic Versioning. Cybersource recommends referencing the latest major version, v2, to receive the latest patch and minor versions automatically. Referencing a specific patch version is not supported.

Upgrade Paths

Because of semantic versioning, every effort will be made to ensure that upgrade paths and patch releases are backwards-compatible and require no code change.

Creating the Server-Side Context

The first step in integrating with Microform Integration is developing the server-side code that generates the capture context. The capture context is a digitally signed JWT that provides authentication, one-time keys, and the target origin to the Microform Integration application. The target origin is the protocol, URL, and port number (if used) of the page on which you will host the microform. You must use the `https://` protocol unless you use `http://localhost`. For example, if you are serving Microform on example.com, the target origin is `https://example.com.`

You can also configure microform to filter out cards by designating the accepted card types.

Sample Microform Integration projects are available for download in the Flex samples on GitHub.

1. Send an authenticated POST request to `https://apitest.cybersource.com/microform/v2/sessions`. Include the target origin URL and at least one accepted card type in the content of the body of the request.
   For example:

   ```
   {
     "targetOrigins": ["https://www.example.com"],
     "allowedCardNetworks": ["VISA"],
     "clientVersion": "v2.0"
   }
   ```

   Optionally, you can include multiple target origins and a list of your accepted card types. For example:

   ```
   {
     "targetOrigins": ["https://www.example.com", "https://www.example.net"]
     "allowedCardNetworks": ["VISA",
       "MAESTRO",
       "MASTERCARD",
       "AMEX",
       "DISCOVER",
       "DINERSCLUB",
       "JCB",
       "CUP",
       "CARTESBANCAIRES",
       "CARNET"
      ],
     "clientVersion": "v2.0"
   }
   ```

2. Pass the capture context response data object to your front-end application. The capture context is valid for 15 minutes.

   See Example: Node.js REST Code Snippet (on page 36).

Important Security Note:

- Ensure that all endpoints within your ownership are secure with some kind of authentication so they cannot be called at will by bad actors.
- Do not pass the `targetOrigin` in any external requests. Hard code it on the server side.

Validating the Capture Context

The capture context that you generated is a JSON Web Token (JWT) data object. The JWT is digitally signed using a public key. The purpose is to ensure the validity of the JWT and confirm that it comes from Cybersource. When you do not have a key specified locally in the JWT header, you should follow best cryptography practices and validate the capture context signature.

To validate a JWT, you can obtain its public key. This public RSA key is in JSON Web Key (JWK) format. This public key is associated with the capture context on the Cybersource domain.

To get the public key of a capture context from the header of the capture context itself, retrieve the key ID associated with the public key. Then, pass the key ID to the `public-keys` endpoint.

Example

From the header of the capture context, get the key ID (`kid`) as shown in this example:

```
{
  "kid": "3g",
  "alg": "RS256"
}
```

Append the key ID to the endpoint `/flex/v2/public-keys/3g`. Then, call this endpoint to get the public key.

Important: When validating the public key, some cryptographic methods require you to convert the public key to PEM format.

Resource

Pass the key ID (kid), that you obtained from the capture context header, as a path parameter, and send a GET request to the `/public-keys` endpoint:

- Test: `https://apitest.cybersource.com/flex/v2/public-keys/{kid}`
- Production: `https://api.cybersource.com/flex/v2/public-keys/{kid}`

The resource returns the public key. Use this public RSA key to validate the capture context.

Example


```
eyJraWQiOiIzZyIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsIm
RhdGEiOiI2bUFLNTNPNVpGTUk5Y3RobWZmd2doQUFFRGNqNU5QYzcxelErbm8reDN6WStLOTVWQ2c5bThm
QWs4czlTRXBtT21zMmVhbEx5NkhHZ29oQ0JEWjVlN3ZUSGQ5YTR5a2tNRDlNVHhqK3ZoWXVDUmRDaDhVY1
dwVUNZWlZnbTE1UXVFMkEiLCJvcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbSIs
Imp3ayI6eyJrdHkiOiJSU0EiLCJlIjoiQVFBQiIsInVzZSI6ImVuYyIsIm4iOiJyQmZwdDRjeGlkcVZwT0
pmVTlJQXcwU1JCNUZqN0xMZjA4U0R0VmNyUjlaajA2bEYwTVc1aUpZb3F6R3ROdnBIMnFZbFN6LVRsSDdy
bVNTUEZIeTFJQ3BfZ0I3eURjQnJ0RWNEanpLeVNZSTVCVjNsNHh6Qk5CNzRJdnB2Smtqcnd3QVZvVU4wM1
RaT3FVc0pfSy1jT0xpYzVXV0ZhQTEyOUthWFZrZFd3N3c3LVBLdnMwNmpjeGwyV05STUIzTS1ZQ0xOb3FC
dkdCSk5oYy1uM1lBNU5hazB2NDdiYUswYWdHQXRfWEZ0ZGItZkphVUVUTW5WdW9fQmRhVm90d1NqUFNaOH
FMOGkzWUdmemp2MURDTUM2WURZRzlmX0tqNzJjTi1OaG9BRURWUlZyTUtiZ3QyRDlwWkJ1d2gzZlNfS3VR
clFWTVdPelRnT3AzT2s3UVFGZ1EiLCJraWQiOiIwOEJhWXMxbjdKTUhjSDh1bkcxc1NDUVdxN2VveWQ1Zy
J9fSwiY3R4IjpbeyJkYXRhIjp7InRhcmdldE9yaWdpbnMiOlsiaHR0cHM6Ly93d3cudGVzdC5jb20iXSwi
bWZPcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbSJ9LCJ0eXBlIjoibWYtMC4xMS
4wIn1dLCJpc3MiOiJGbGV4IEFQSSIsImV4cCI6MTYxNjc3OTA5MSwiaWF0IjoxNjE2Nzc4MTkxLCJqdGki
OiJ6SG1tZ25uaTVoN3ptdGY0In0.GvBzyw6JKl3b2PztHb9rZXawx2T817nYqu6goxpe4PsjqBY1qeTo19
R-CP_DkJXov9hdJZgdlzlNmRY6yoiziSZnGJdpnZ-pCqIlC06qrpJVEDob3O_efR9L03Gz7F5JlLOiTXSj
6nVwC5mRlcP032ytPDEx5TMI9Y0hmBadJYnhEMwQnn_paMm3wLh2v6rfTkaBqd8n6rPvCNrWMOwoMdoTeF
xku-d27jlA95RXqJWfhJSN1MFquKa7THemvTX2tnjZdTcrTcpgHlxi22w7MUFcnNXsbMouoaYiEdAdSlCZ
7LCXrS1Brdr_FWDp7v0uwqHm7OALsGrw8QbGTafF8w
```


Base64 decode the capture context to get the key ID (`kid`) from its header:


```
{
  "kid": "3g",
  "alg": "RS256"
}
```


Get its public key from `/flex/v2/public-keys/3g`:


```
{
    "kty":"RSA",
    "use":"enc",
    "kid":"3g",
    "n":"ir7Nl1Bj8G9rxr3co5v_JLkP3o9UxXZRX1LIZFZeckguEf7Gdt5kGFFfTsymKBesm3Pe
     8o1hwfkq7KmJZEZSuDbiJSZvFBZycK2pEeBjycahw9CqOweM7aKG2F_bhwVHrY4YdKsp
     _cSJe_ZMXFUqYmjk7D0p7clX6CmR1QgMl41Ajb7NHI23uOWL7PyfJQwP1X8HdunE6ZwK
     DNcavqxOW5VuW6nfsGvtygKQxjeHrI-gpyMXF0e_PeVpUIG0KVjmb5-em_Vd2SbyPNme
     nADGJGCmECYMgL5hEvnTuyAybwgVwuM9amyfFqIbRcrAIzclT4jQBeZFwkzZfQF7MgA6QQ",
     "e":"AQAB"
}
```

Introduction to JWT
JWT (signed) Specification
JWK Specification

Setting Up the Client Side

You can integrate Microform Integration with your native payment acceptance web page or mobile application.

Web Page

Initiate and embed Microform Integration into your payment acceptance web page.

1. Add the Microform Integration JavaScript library to your page by loading it directly from Cybersource. See Version Numbering (on page 28). You should do this dynamically per environment by using the asset path returned in the JWT from `/microform/v2/sessions`. For example:

   ```
   ctx": [
       {
         "data": {
           "clientLibrary":
    https://testflex.cybersource.com/microform/bundle/v2/flex-microform.min.js,
         ...
   ```

   - **Test**: `<script src="https://testflex.cybersource.com/microform/bundle/v2/flex-microform.min.js"></script>`
   - **Production**: `<script src="https://flex.cybersource.com/microform/bundle/v2/flex-microform.min.js"></script>`

2. Create the HTML placeholder objects to attach to the microforms.

   Microform Integration attaches the microform fields to containers within your HTML. Within your HTML checkout, replace the payment card and CVN tag with a simple container. Microform Integration uses the container to render an iframe for secured credit card input. The following example contains simple `div` tags to define where to place the PAN and CVN fields within the payment acceptance page: `<div id="number-container" class="form-control"></div>`. See Example: Checkout Payment Form (on page 36).

3. Invoke the Flex SDK by passing the capture context that was generated in the previous step to the microform object.

   ```
   var flex = new Flex(captureContext);
   ```

4. Initiate the microform object with styling to match your web page.

   After you create a new Flex object, you can begin creating your Microform. You will pass your baseline styles and ensure that the button matches your merchant page. `var microform = flex.microform({ styles: myStyles });`

5. Create and attach the microform fields to the HTML objects through the Microform Integration JavaScript library.

```
var number = microform.createField('number', { placeholder: 'Enter card
 number' });
            var securityCode = microform.createField('securityCode',
 { placeholder: '•••' });
            number.load('#number-container');
            securityCode.load('#securityCode-container');
```

6. Create a function for the customer to submit their payment information, and invoke the tokenization request to Microform Integration for the transient token.

Mobile Application

To initiate and embed Microform Integration into native payment acceptance mobile application, follow the steps for web page setup, and ensure that these additional requirements are met:

- The card acceptance fields of PAN and CVV must be hosted on a web page.
- The native application must load the hosted card entry form web page in a web view.

As an alternative, you can use the Mobile SDKs hosted on GitHub:

- iOS sample: https://github.com/Cybersource/flex-v2-ios-sample
- Android sample: https://github.com/CyberSource/flex-v2-android-sample

Transient Token Time Limit

Transient Token Time Limit

The sensitive data associated with the transient token is available for use only for 15 minutes or until one successful authorization occurs. Before the transient token expires, its data is still usable in other non-authorization services. After 15 minutes, you must prompt the customer to restart the checkout flow.

See Example: Creating the Pay Button with Event Listener (on page 38).

When the customer submits the form, Microform Integration securely collects and tokenizes the data in the loaded fields as well as the options supplied to the `createToken()` function. The month and year are included in the request. If tokenization succeeds, your callback receives the token as its second parameter. Send the token to your server, and use it in place of the PAN when you use supported payment services.

See Example: Customer-Submitted Form (on page 38).

Transient Token Response Format

The transient token is issued as a JSON Web Token (RFC 7519). A JWT is a string consisting of three parts that are separated by dots:

- Header
- Payload
- Signature

JWT example: `xxxxx.yyyyy.zzzzz`

The payload portion of the token is an encoded Base64url JSON string and contains various claims.

Important: The internal data structure of the JWT can expand to contain additional data elements. Ensure that your integration and validation rules do not limit the data elements contained in responses.

See Example: Token Payload (on page 40).

Validating the Transient Token

After receiving the transient token, validate its integrity using the public key embedded within the capture context created at the beginning of this flow. This verifies that Cybersource issued the token and that no data tampering occurred during transit. See Example: Capture Context Public Key (on page 41).

Use the capture context public key to cryptographically validate the JWT provided from a successful `microform.createToken` call. You might have to convert the JSON Web Key (JWK) to privacy-enhanced mail (PEM) format for compatibility with some JWT validation software libraries.

The Cybersource SDK has functions that verify the token response. You must verify the response to ensure that no tampering occurs as it passes through the cardholder device. Do so by using the public key generated at the start of the process.

See Example: Validating the Transient Token (on page 41).

Using the Transient Token

After you validate the transient token, you can use it in place of the PAN with payment services for 15 minutes. See Transient Token Time Limit (on page 33).

When the consuming service receives a request containing a transient token, it retrieves the tokenized data and injects the values into your request before processing, and none of the sensitive data is stored on your systems. In some scenarios, the `jti` value contained in the JWT transient token response must be extracted and used instead of the entire JWT.

Connection Method	Field
Simple Order API	tokenSource_transientToken
SCMP API	transient_token

REST API with Transient Token JSON Web Token	"tokenInformation": {
	
	"transientTokenJwt": "eyJraWQiOiIwNzRsM3p5M2xCRWN5d1gxcnhXNFFoUmJFNXJLN1NmQiIsImFsZyI6IlJTMjU2In0.eyJkYXRhIjp7ImV4cGlyYXRpb25ZZWFyIjoiMjAyMSIsIm51bWJlciI6IjQxMTExMVhYWFhYWDExMTEiLCJleHBpcmF0aW9uTW9udGgiOiIwNSIsInR5cGUiOiIwMDEifSwiaXNzIjoiRmxleC8wOCIsImV4cCI6MTU4ODcwMjkxNSwidHlwZSI6Im1mLTAuMTEuMCIsImlhdCI6MTU4ODcwMjAxNSwianRpIjoiMUU0Q0NMSUw4NFFXM1RPSTFBM0pUU1RGMTZGQUNVNkUwNU9VRVNGWlRQNUhIVkJDWTQwUTVFQjFBRUMzNDZBMCJ9.FB3b2r8mjtvqo3_k05sRIPGmCZ_5dRSZp8AIJ4u7NKb8E0-6ZOHDwEpxtOMFzfozwXMTJ3C6yBK9vFIPTIG6kydcrWNheE2Pfort8KbxyUxG-PYONY-xFnRDF841EFhCMC4nRFvXEIvlcLnSK6opUUe7myKPjpZI1ijWpF0N-DzZiVT8JX-9ZIarJq2OI0S61Y3912xLJUKi5c2VpRPQOS54hRr5GHdGJ2fV8JZ1gTuup_qLyyK7uE1VxI0aucsyH7yeF5vTdjgSd76ZJ1OUFi-3Ij5kSLsiX4j-D0T8ENT1DbB_hPTaK9o6qqtGJs7QEeW8abtnKFsTwVGrT32G2w"
	
	}

REST API with JSON Web Token ID	"tokenInformation": {
	
	"jti": "1E3GQY1RNKBG6IBD2EP93C43PIZ2NQ6SQLUIM3S16BGLHTY4IIEK5EB1AE5D73A4",
	
	}

See Example: Authorization with a Transient Token Using the REST API (on page 42).

Getting Started Examples

Example: Node.js REST Code Snippet

```
try {
 var instance = new cybersourceRestApi.KeyGenerationApi(configObj);
 var request = new cybersourceRestApi.GeneratePublicKeyRequest();

 request.encryptionType = 'RsaOaep256';
 request.targetOrigin = 'http://localhost:3000';
 var opts = [];
 opts['format'] = 'JWT';

 console.log('\n*************** Generate Key ********************* ');

 instance.generatePublicKey(request, opts, function (error, data, response) {
 if (error) {
  console.log('Error : ' + error);
  console.log('Error status code : ' + error.statusCode);
 }
 else if (data) {
  console.log('Data : ' + JSON.stringify(data));
  console.log('CaptureContext: '+data.keyId);
  res.render('index', { keyInfo: JSON.stringify(data.keyId)});
 }
  console.log('Response : ' + JSON.stringify(response));
  console.log('Response Code Of GenerateKey : ' + response['status']);
  callback(error, data);
 });

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

Back to Creating the Server-Side Context (on page 28)

Example: Checkout Payment Form

This simple payment form captures the name, PAN, CVN, month, and year, and a pay button for submitting the information.

```
<h1>Checkout</h1>
                <div id="errors-output" role="alert"></div>
                <form action="/token" id="my-sample-form" method="post">
                    <div class="form-group">
```

```
                        <label for="cardholderName">Name</label>
                        <input id="cardholderName" class="form-control"
name="cardholderName" placeholder="Name on the card">
                        <label id="cardNumber-label">Card Number</label>
                        <div id="number-container" class="form-control"></div>
                        <label for="securityCode-container">Security Code</label>
                        <div id="securityCode-container"
class="form-control"></div>
                    </div>

                    <div class="form-row">
                        <div class="form-group col-md-6">
                            <label for="expMonth">Expiry month</label>
                            <select id="expMonth" class="form-control">
                                <option>01</option>
                                <option>02</option>
                                <option>03</option>
                                <option>04</option>
                                <option>05</option>
                                <option>06</option>
                                <option>07</option>
                                <option>08</option>
                                <option>09</option>
                                <option>10</option>
                                <option>11</option>
                                <option>12</option>
                            </select>
                        </div>
                        <div class="form-group col-md-6">
                            <label for="expYear">Expiry year</label>
                            <select id="expYear" class="form-control">
                                <option>2021</option>
                                <option>2022</option>
                                <option>2023</option>
                            </select>
                        </div>
                    </div>

                    <button type="button" id="pay-button" class="btn
btn-primary">Pay</button>
                    <input type="hidden" id="flexresponse" name="flexresponse">
                </form>
```

Back to Setting Up the Client Side (on page 32).

Example: Creating the Pay Button with Event Listener

```
payButton.addEventListener('click', function() {

            // Compiling MM & YY into optional parameters
            var options = {
            expirationMonth: document.querySelector('#expMonth').value,
            expirationYear: document.querySelector('#expYear').value
            };
            //
            microform.createToken(options, function (err, token) {
              if (err) {
                // handle error
                console.error(err);
                errorsOutput.textContent = err.message;
              } else {
                // At this point you may pass the token back to your server as you
 wish.
                // In this example we append a hidden input to the form and submit
 it.
                console.log(JSON.stringify(token));
                flexResponse.value = JSON.stringify(token);
                form.submit();
              }
            });
          });
```

Back to Transient Token Time Limit (on page 33).

Example: Customer-Submitted Form

```
<script>
        // Variables from the HTML form
        var form = document.querySelector('#my-sample-form');
        var payButton = document.querySelector('#pay-button');
        var flexResponse = document.querySelector('#flexresponse');
        var expMonth = document.querySelector('#expMonth');
        var expYear = document.querySelector('#expYear');
        var errorsOutput = document.querySelector('#errors-output');

        // the capture context that was requested server-side for this transaction
        var captureContext = <%-keyInfo%> ;
        // custom styles that will be applied to each field we create using
 Microform
        var myStyles = {
          'input': {
```

```
            'font-size': '14px',
            'font-family': 'helvetica, tahoma, calibri, sans-serif',
            'color': '#555'
          },
          ':focus': { 'color': 'blue' },
          ':disabled': { 'cursor': 'not-allowed' },
          'valid': { 'color': '#3c763d' },
          'invalid': { 'color': '#a94442' }
        };
        // setup Microform
        var flex = new Flex(captureContext);
        var microform = flex.microform({ styles: myStyles });
        var number = microform.createField('number', { placeholder: 'Enter card
number' });
        var securityCode = microform.createField('securityCode', { placeholder:
'•••' });
        number.load('#number-container');
        securityCode.load('#securityCode-container');


        // Configuring a Listener for the Pay button
      payButton.addEventListener('click', function() {

        // Compiling MM & YY into optional paramiters
          var options = {
            expirationMonth: document.querySelector('#expMonth').value,
            expirationYear: document.querySelector('#expYear').value
          };
        //
          microform.createToken(options, function (err, token) {
            if (err) {
              // handle error
              console.error(err);
              errorsOutput.textContent = err.message;
            } else {
              // At this point you may pass the token back to your server as you
wish.
              // In this example we append a hidden input to the form and submit
it.
              console.log(JSON.stringify(token));
              flexResponse.value = JSON.stringify(token);
              form.submit();
            }
          });
        });
      </script>
```

Back to Transient Token Time Limit (on page 33).

Example: Token Payload

```
{
// token id to be used with Cybersource services
 "jti": "408H4LHTRUSHXQZWLKDIN22ROVXJFLU6VLU00ZWL8PYJOZQWGPS9CUWNASNR59K4",
// when the token was issued
 "iat": 1558612859,
 // when the token will expire
 "exp": 1558613759,
// info about the stored data associated with this token
// any sensitive data will be masked
 "data": {
 "number": "444433XXXXXX1111",
 "type": "001",
 "expirationMonth": "06",
 "expirationYear": "2025"
 }
}
```

Back to Transient Token Response Format (on page 34).

Example: Token Payload with Multiple Card Types

```
{
  "iss": "Flex/08",
  "exp": 1661350495,
  "type": "mf-2.0.0",
  "iat": 1661349595,
  "jti": "1C174LLWIFFR9OV0V0IJQOY0IB1JQP70ZNF4TBI3V6H3AIOY0W1T6306325F91C0",
  "content": {
    "paymentInformation": {
      "card": {
        "expirationYear": {
          "value": "2023"
        },
        "number": {
          "detectedCardTypes": [
            "042",
            "036"
          ],
          "maskedValue": "XXXXXXXXXXXX1800",
          "bin": "501767"
        },
        "securityCode": {},
        "expirationMonth": {
          "value": "01"
```

```
        }
      }
    }
  }
}
```

Back to Transient Token Response Format (on page 34).

Example: Capture Context Public Key

```
"jwk": {
                "kty": "RSA",
                "e": "AQAB",
                "use": "enc",
                "n":
 "3DhDtIHLxsbsSygEAG1hcFqnw64khTIZ6w9W9mZNl83gIyj1FVk-H5GDMa85e8RZFxUwgU_zQ0kHLtON
o8SB52Z0hsJVE9wqHNIRoloiNPGPQYVXQZw2S1BSPxBtCEjA5x_-bcG6aeJdsz_cAE7OrIYkJa5Fphg9_p
xgYRod6JCFjgdHj0iDSQxtBsmtxagAGHjDhW7UoiIig71SN-f-gggaCpITem4zlb5kkRVvmKMUANe4B36v
4XSSSpwdP_H5kv4JDz_cVlp_Vy8T3AfAbCtROyRyH9iH1Z-4Yy6T5hb-9y3IPD8vlc8E3JQ4qt6U46EeiK
PH4KtcdokMPjqiuQ",
                "kid": "00UaBe20jy9VkwZUQPZwNNoKFPJA4Qhc"    }
```

Back to Validating the Transient Token (on page 34).

Example: Validating the Transient Token

This example shows how to extract the signature key from the capture context and use the key to validate the transient token object returned from a successful microform interaction.

```
console.log('Response TransientToken: ' + req.body.transientToken);
                console.log('Response CaptureContext: ' +
 req.body.captureContext);

                //  Validating Token JWT Against Signature in Capture Context
                var capturecontext = req.body.captureContext;
                var transientToken = req.body.transientToken;

                // Extracting JWK in Body of Capture Context
                var ccBody = capturecontext.split('.')[1];
                console.log('Body: ' + ccBody);
                var atob = require('atob');
                var ccDecodedValue = JSON.parse( atob(ccBody));
                var jwk = ccDecodedValue.flx.jwk;
```

```
                  console.log('CaptureContext JWK: ' + JSON.stringify(jwk));

                  // Converting JWK to PEM
                  var jwkToPem = require('jwk-to-pem'),
                  jwt = require('jsonwebtoken');

                  var pem = jwkToPem(jwk);
                  //  Validating JWT
                  var validJWT = jwt.verify(transientToken, pem);
                  console.log('Validated Resposonse: ' + JSON.stringify(validJWT));
```

Back to Validating the Transient Token (on page 34).

Example: Authorization with a Transient Token Using the REST API

```
{
    "clientReferenceInformation": {
        "code": "TC50171_3"
    },
    "orderInformation": {
        "amountDetails": {
            "totalAmount": "102.21",
            "currency": "USD"
        },
        "billTo": {
            "firstName": "Tanya",
            "lastName": "Lee",
            "address1": "1234 Main St.",
            "locality": "Small Town",
            "administrativeArea": "MI",
            "postalCode": "98765-4321",
            "country": "US",
            "district": "MI",
            "buildingNumber": "123",
            "email": "tanyalee@example.com",
            "phoneNumber": "987-654-3210"
        }
    },
    "tokenInformation": {

  "transientTokenJwt": "eyJraWQiOiIwN0JwSE9abkhJM3c3UVAycmhNZkhuWE9XQlhwa1ZHTiIsImFsZyI6Il
JTMjU2In0.eyJkYXRhIjp7ImV4cGlyYXRpb25ZZWFyIjoiMjAyMCIsIm51bWJlciI6IjQxMTExMVhYWFhYWDExMTEi
LCJleHBpcmF0aW9uTW9udGgiOiIxMCIsInR5cGUiOiIwMDEifSwiaXNzIjoiRmxleC8wNyIsImV4cCI6MTU5MTc0Nj
AyNCwidHlwZSI6Im1mLTAuMTEuMCIsImlhdCI6MTU5MTc0NTEyNCwianRpIjoiMUMzWjdUTkpaVjI4OVM5MTdQM0JH
SFM1T0ZQNFNBRERCUUtKMFFKMzMzOEhRR0MwWTg0QjVFRTAxREU4NEZDQiJ9.cfwzUMJf115K2T9-wE_A_k2jZptXl
ovls8-fKY0muO8YzGatE5fu9r6aC4q7n0YOvEU6G7XdH4ASG32mWnYu-kKlqN4IY_cquRJeUvV89ZPZ5WTttyrgVH1
```

```
7LSTE2EvwMawKNYnjh0lJwqYJ51cLnJiVlyqTdEAv3DJ3vInXP1YeQjLX5_vF-OWEuZfJxahHfUdsjeGhGaaOGVMUZ
JSkzpTu9zDLTvpb1px3WGGPu8FcHoxrcCGGpcKk456AZgYMBSHNjr-pPkRr3Dnd7XgNF6shfzIPbcXeWDYPTpS4PNY
8ZsWKx8nFQIeROMWCSxIZOmu3Wt71KN9iK6DfOPro7w"
    }
}
```


Back to Using the Transient Token (on page 35).

Styling

Microform Integration can be styled to look and behave like any other input field on your site.

General Appearance

The `<iframe>` element rendered by Microform has an entirely transparent background that completely fills the container you specify. By styling your container to look like your input fields, your customer will be unable to detect any visual difference. You control the appearance using your own stylesheets. With stylesheets, there are no restrictions and you can often re-use existing rules.

Explicitly Setting Container Height

Typically, input elements calculate their height from font size and line height (and a few other properties), but Microform Integration requires explicit configuration of height. Make sure you style the height of your containers in your stylesheets.

Managed Classes

In addition to your own container styles, Microform Integration automatically applies some classes to the container in response to internal state changes.

Class	**Description**
`.flex-microform`	Base class added to any element in which a field has been loaded.
`.flex-microform-disabled`	The field has been disabled.
`.flex-microform-focused`	The field has user focus.
`.flex-microform-valid`	The input card number is valid.
`.flex-microform-invalid`	The input card number invalid.

Class	Description
`.flex-microform-autocomplete`	The field has been filled using an `autocomplete/autofill` event.

To make use of these classes, include overrides in your application's stylesheets. You can combine these styles using regular CSS rules. Here is an example of applying CSS transitions in response to input state changes:

```
.flex-microform {
  height: 20px;
  background: #ffffff;
  -webkit-transition: background 200ms;
  transition: background 200ms;
}

/* different styling for a specifc container */
#securityCode-container.flex-microform {
  background: purple;
}

.flex-microform-focused {
  background: lightyellow;
}

.flex-microform-valid {
  background: green;
}

.flex-microform-valid.flex-microform-focused {
  background: lightgreen;
}

.flex-microform-autocomplete {
  background: #faffbd;
}
```

Input Field Text

To style the text within the iframe element, use the JavaScript library. The `styles` property in the setup options accepts a CSS-like object that allows customization of the text. Only a subset of the CSS properties is supported.

```
var customStyles = {
  'input': {
    'font-size': '16px',
```

```
    'color': '#3A3A3A'
  },
  '::placeholder': {
    'color': 'blue'
  },
  ':focus': {
    'color': 'blue'
  },
  ':hover': {
    'font-style': 'italic'
  },
  ':disabled': {
    'cursor': 'not-allowed',
  },
  'valid': {
    'color': 'green'
  },
  'invalid': {
    'color': 'red'
  }
};

var flex = new Flex('..........');
// apply styles to all fields
var microform = flex.microform({ styles: customStyles });
var securityCode = microform.createField('securityCode');

// override the text color for for the card number field
var number = microform.createField('number', { styles: { input: { color:
 '#000' }}});
```

Supported Properties

The following CSS properties are supported in the `styles: { ... }` configuration hash. Unsupported properties are not added to the inner field, and a warning is output to the console.

- `color`
- `cursor`
- `font`
- `font-family`
- `font-kerning`
- `font-size`

- `font-size-adjust`
- `font-stretch`
- `font-style`
- `font-variant`
- `font-variant-alternates`
- `font-variant-caps`
- `font-variant-east-asian`
- `font-variant-ligatures`
- `font-variant-numeric`
- `font-weight`
- `line-height`
- `opacity`
- `text-shadow`
- `text-rendering`
- `transition`
- `-moz-osx-font-smoothing`
- `-moz-tap-highlight-color`
- `-moz-transition`
- `-o-transition`
- `-webkit-font-smoothing`
- `-webkit-tap-highlight-color`
- `-webkit-transition`

Events

You can subscribe to Microform Integration events and obtain them through event listeners. Using these events, you can easily enable your checkout user interface to respond to any state changes as soon as they happen.

Events

Event Name	Emitted When
`autocomplete`	Customer fills the credit card number using a browser or third-party extension. This event provides a hook onto the additional information provided during the `autocomplete` event.
`blur`	Field loses focus.
`change`	Field contents are edited by the customer. This event contains various data such as validation information and details of any detected card types.
`focus`	Field gains focus.
`inputSubmitRequest`	Customer requests submission of the field by pressing the Return key or similar.
`load`	Field has been loaded on the page and is ready for user input.
`unload`	Field is removed from the page and no longer available for user input.
`update`	Field configuration was updated with new options.

Some events may return data to the event listener's callback as described in the next section.

Subscribing to Events

Using the `.on()` method provided in the `microformInstance` object, you can easily subscribe to any of the supported events.

For example, you could listen for the `change` event and in turn display appropriate card art and display brand-specific information.

```
var secCodeLbl = document.querySelector('#mySecurityCodeLabel');
var numberField = flex.createField('number');

// Update your security code label to match the detected card type's terminology
numberField.on('change', function(data) {
  secCodeLbl.textContent = (data.card && data.card.length > 0) ?
 data.card[0].securityCode.name : 'CVN';
});

numberField.load('#myNumberContainer');
```

The `data` object supplied to the event listener's callback includes any information specific to the triggered event.

Card Detection

By default, Microform attempts to detect the card type as it is entered. Detection info is bubbled outwards in the `change` event. You can use this information to build a dynamic user experience, providing feedback to the user as they type their card number.

```
{
  "card": [
    {
      "name": "mastercard",
      "brandedName": "MasterCard",
      "cybsCardType": "002",
      "spaces": [ 4, 8, 12],
      "lengths": [16],
      "securityCode": {
        "name": "CVC",
        "length": 3
      },
      "luhn": true,
      "valid": false,
      "couldBeValid": true
    },
    /* other identified card types */
  ]
}
```

If Microform Integration is unable to determine a single card type, you can use this information to prompt the customer to choose from a possible range of values.

If **type** is specified in the `microformInstance.createToken(options,...)` method, the specified value always takes precedence over the detected value.

Autocomplete

By default, Microform Integration supports the autocomplete event of the **cardnumber** field provided by certain browsers and third-party extensions. An `autocomplete` event is provided to allow easy access to the data that was provided to allow integration with other elements in your checkout process.

The format of the data provided in the event might be as follows:

```
{
name: '_____',
expirationMonth: '__',
expirationYear: '____'
}
```

These properties are in the object only if they contain a value; otherwise, they are undefined. Check for the properties before using the event. The following example displays how to use this event to update other fields in your checkout process:

```
var number = microform.createField('number');
number.on('autocomplete', function(data) {
  if (data.name) document.querySelector('#myName').value = data.name;
  if (data.expirationMonth) document.querySelector('#myMonth').value =
 data.expirationMonth;
  if (data.expirationYear) document.querySelector('#myYear').value =
 data.expirationYear;
});
```

Security Recommendations

By implementing a Content Security Policy, you can make use of browser features to mitigate many cross-site scripting attacks.

The full set of directives required for Microform Integration is:

Security Policy Locations

Policy	Sandbox	Production
frame-src	https://testflex.cybersource.com/	https://flex.cybersource.com/
child-src	https://testflex.cybersource.com/	https://flex.cybersource.com/
script-src	https://testflex.cybersource.com/	https://flex.cybersource.com/

PCI DSS Guidance

Any merchant accepting payments must comply with the PCI Data Security Standards (PCI DSS). Microform Integration's approach facilitates PCI DSS compliance through self-assessment and the storage of sensitive PCI information.

Self Assessment Questionnaire

Microform Integration handles the card number input and transmission from within iframe elements served from Cybersource controlled domains. This approach can qualify merchants for SAQ A-based assessments. Related fields, such as card holder name or expiration date, are not considered sensitive when not accompanied by the PAN.

Storing Returned Data

Responses from Microform Integration are stripped of sensitive PCI information such as card number. Fields included in the response, such as card type and masked card number, are not subject to PCI compliance and can be safely stored within your systems. If you collect the CVN, note that it can be used for the initial authorization but not stored for subsequent authorizations.

API Reference

This reference provides details about the JavaScript API for creating Microform Integration web pages.

Class: Field

An instance of this class is returned when you add a Field to a Microform integration using microform.createField (on page 60). With this object, you can then interact with the Field to subscribe to events, programmatically set properties in the Field, and load it to the DOM.

Methods

`clear()`

Programmatically clear any entered value within the field.

Example

```
field.clear();
```

`dispose()`

Permanently remove this field from your Microform integration.

Example

```
field.dispose();
```

`focus()`

Programmatically set user focus to the Microform input field.

Example

```
field.focus();
```

`load(container)`

Load this field into a container element on your page.

Successful loading of this field will trigger a load event.

Parameters

Name	Type	Description
container	HTMLElement | string	Location in which to load this field. It can be either an HTMLElement reference or a CSS selector string that will be used to load the element.

Examples

Using a CSS selector

```
field.load('.form-control.card-number');
```

Using an HTML element

```
var container = document.getElementById('container');
field.load(container);
```

`off(type, listener)`

Unsubscribe an event handler from a Microform Field.

Parameter

Name	Type	Description
type	string	Name of the event you wish to unsubscribe from.
listener	function	The handler you wish to be unsubscribed.

Example

```
// subscribe to an event using .on() but keep a reference to the handler that was
 supplied.
var focusHandler = function() { console.log('focus received'); }
field.on('focus', focusHandler);

// then at a later point you can remove this subscription by supplying the same
 arguments to .off()
field.off('focus', focusHandler);
```

`on(type, listener)`

Subscribe to events emitted by a Microform Field. Supported eventTypes are:

- autocomplete
- blur
- change
- error
- focus
- inputSubmitRequest
- load
- unload
- update

Some events may return data as the first parameter to the callback otherwise this will be undefined. For further details see each event's documentation using the links above.

Parameters

Name	Type	Description
type	string	Name of the event you wish to subscribe to.
listener	function	Handler to execute when event is triggered.

Example

```
field.on('focus', function() {
  console.log('focus received'); });
```

`unload()`

Remove a the Field from the DOM. This is the opposite of a load operation.

Example

```
field.unload();
```

`update(options)`

Update the field with new configuration options. This accepts the same parameters as microform.createField(). New options will be merged into the existing configuration of the field.

Parameter

Name	Type	Description
options	object	New options to be merged with previous configuration.

Example

```
// field initially loaded as disabled with no placeholder
var number = microform.createField('number', { disabled: true });
number.load('#container');

// enable the field and set placeholder text
number.update({ disabled: false, placeholder: 'Please enter your card number' });
```

Events

`autocomplete`

Emitted when a customer has used a browser or third-party tool to perform an autocomplete/autofill on the input field. Microform will attempt to capture additional information from the autocompletion and supply these to the callback if available. Possible additional values returned are:

- name
- expirationMonth
- expirationYear

If a value has not been supplied in the autocompletion, it will be undefined in the callback data. As such you should check for its existence before use.

Examples

Possible format of data supplied to callback

```
{
  name: '_____',
  expirationMonth: '__',
  expirationYear: '____'
}
```

Updating the rest of your checkout after an autocomplete event

```
field.on('autocomplete', function(data) {
if (data.name) document.querySelector('#myName').value = data.name;
if (data.expirationMonth) document.querySelector('#myMonth').value =
 data.expirationMonth;
if (data.expirationYear) document.querySelector('#myYear').value =
 data.expirationYear;
});
```

`blur`

This event is emitted when the input field has lost focus.

Example

```
field.on('blur', function() {
  console.log('Field has lost focus');
});

// focus the field in the browser then un-focus the field to see your supplied
 handler execute
```

`change`

Emitted when some state has changed within the input field. The payload for this event contains several properties.

Type: object

Properties

Name	Type
card	object
valid	boolean
couldBeValid	boolean
empty	boolean

Examples

Minimal example:

```
field.on('change', function(data) {
  console.log('Change event!');
  console.log(data);
});
```

Use the card detection result to update your UI.

```
var cardImage = document.querySelector('img.cardDisplay');
var cardSecurityCodeLabel = document.querySelector('label[for=securityCode]');

// create an object to map card names to the URL of your custom images
var cardImages = {
  visa: '/your-images/visa.png',
  mastercard: '/your-images/mastercard.png',
  amex: '/your-images/amex.png',
  maestro: '/your-images/maestro.png',
  discover: '/your-images/discover.png',
  dinersclub: '/your-images/dinersclub.png',
  jcb: '/your-images/jcb.png'
};

field.on('change', function(data) {
  if (data.card.length === 1) {
    // use the card name to to set the correct image src
    cardImage.src = cardImages[data.card[0].name];
```

```
    // update the security code label to match the detected card's naming
 convention
    cardSecurityCodeLabel.textContent = data.card[0].securityCode.name;
  } else {
    // show a generic card image
    cardImage.src = '/your-images/generic-card.png';
  }
});
```

Use the card detection result to filter select element in another part of your checkout.

```
var cardTypeOptions = document.querySelector('select[name=cardType] option');

field.on('change', function(data) {
  // extract the identified card types
  var detectedCardTypes = data.card.map(function(c) { return c.cybsCardType; });

  // disable any select options not in the detected card types list
  cardTypeOptions.forEach(function (o) {
    o.disabled = detectedCardTypes.includes(o.value);
  });
});
```

Updating validation styles on your form element.

```
var myForm = document.querySelector('form');

field.on('change', function(data) {
  myForm.classList.toggle('cardIsValidStyle', data.valid);
  myForm.classList.toggle('cardCouldBeValidStyle', data.couldBeValid);
});
```

`focus`

Emitted when the input field has received focus.

Example

```
field.on('focus', function() {
  console.log('Field has received focus');
});

// focus the field in the browser to see your supplied handler execute
```

`inputSubmitRequest`

Emitted when a customer has requested submission of the input by pressing Return key or similar. By subscribing to this event you can easily replicate the familiar user experience of pressing enter to submit a form. Shown below is an example of how to implement this. The inputSubmitRequest handler will:

1. Call Microform.createToken() (on page 60).
2. Take the result and add it to a hidden input on your checkout.
3. Trigger submission of the form containing the newly created token for you to use server-side.

Example

```
var form = document.querySelector('form');
var hiddenInput = document.querySelector('form input[name=token]');

field.on('inputSubmitRequest', function() {
  var options = {
    //
  };

  microform.createToken(options, function(response) {
    hiddenInput.value = response.token;
    form.submit();
  });
});
```

`load`

This event is emitted when the field has been fully loaded and is ready for user input.

Example

```
field.on('load', function() {
  console.log('Field is ready for user input');
});
```

`unload`

This event is emitted when the field has been unloaded and no longer available for user input.

Example

```
field.on('unload', function() {
  console.log('Field has been removed from the DOM');
});
```

`update`

This event is emitted when the field has been updated. The event data will contain the settings that were successfully applied during this update.

Type: object

Example

```
 field.on('update', function(data) {
  console.log('Field has been updated. Changes applied were:');
  console.log(data);
});
```

Module: FLEX

Flex(captureContext)

`new Flex(captureContext)`

For detailed setup instructions, see Getting Started (on page 28).

Parameters:

Name	Type	Description
captureContext	String	JWT string that you requested via a server-side authenticated call before starting the checkout flow.

Example

Basic Setup

```
script
 src="https://
flex.cybersource.com/cybersource/assets/microform/0.11/flex-microform.min.js"></sc
ript>
<script>
  var flex = new Flex('header.payload.signature');
</script>
```

Methods

`microform(optionsopt) > {Microform}`

This method is the main setup function used to initialize Microform Integration. Upon successful setup, the callback receives a `microform`, which is used to interact with the service and build your integration. For details, see Class: Microform (on page 60).

Parameter

Name	Type	Description
options	Object	

Property

Name	Type	Attributes	Description
styles	Object	<optional>	Apply custom styling to all the fields in your integration.

Returns:

Type: Microform

Examples

Minimal Setup

```
var flex = new Flex('header.payload.signature');
var microform = flex.microform();
```

Custom Styling

```
var flex = new Flex('header.payload.signature');
var microform = flex.microform({
  styles: {
    input: {
      color: '#212529',
        'font-size': '20px'
    }
  }
});
```

Class: Microform

An instance of this class is returned when you create a Microform integration using `flex.microform`. This object allows the creation of Microform Fields. For details, see Module: Flex (on page 58).

Methods

`createField(fieldType, optionsopt) > {Field}`

Create a field for this Microform integration.

Parameters

Name	Type	Attributes	Description
fieldType	string		Supported values: • `number` • `securityCode`
options	object	<optional>	To change these options after initialization use `field.update()`.

Properties

Name	Type	Attributes	Default	Description
placeholder	string	<optional>		Sets the `placeholder` attribute on the input.
title	string	<optional>		Sets the title attribute on the input. Typically used to display tooltip text on hover.
description	string	<optional>		Sets the input's description for use by assistive technologies using the aria-describedby attribute.
disabled	Boolean	<optional>	false	Sets the `disabled` attribute on the input.
autoformat	Boolean	<optional>	true	Enable or disable automatic formatting of the input field. This is only supported for number fields and will automatically insert spaces based on the detected card type.

Name	Type	Attributes	Default	Description
maxLength	number	<optional>	3	Sets the maximum length attribute on the input. This is only supported for `securityCode` fields and may take a value of `3` or `4`.
styles	stylingOptions	<optional>		Apply custom styling to this field

Returns

Type: Field

Examples

Minimal Setup

```
var flex = new Flex('.........');
var microform = flex.microform();
var number = microform.createField('number');
```

Providing Custom Styles

```
var flex = new Flex('.........');
var microform = flex.microform();
var number = microform.createField('number', {
  styles: {
    input: {
      'font-family': '"Courier New", monospace'
    }
  }
});
```

Setting the length of a security code field

```
var flex = new Flex('.........');
var microform = flex.microform();
var securityCode = microform.createField('securityCode', { maxLength: 4 });
```

`createToken(options, callback)`

Request a token using the card data captured in the Microform fields. A successful token creation will receive a transient token as its second callback parameter.

Parameter

Name	Type	Description
options	object	Additional tokenization options.
callback	callback	Any error will be returned as the first callback parameter. Any successful creation of a token will be returned as a string in the second parameter.

Properties

Name	Type	Attributes	Description
type	string	<optional>	Three digit card type string. If set, this will override any automatic card detection.
expirationMonth	string	<optional>	Two digit month string. Must be padded with leading zeros if single digit.
expirationYear	string	<optional>	Four digit year string.

Examples

Minimal example omitting all optional parameters.

```
microform.createToken({}, function(err, token) {
  if (err) {
    console.error(err);
    return;
  }

  console.log('Token successfully created!');
  console.log(token);
});
```

*Override the **cardType** parameter using a select element that is part of your checkout.*

```
// Assumes your checkout has a select element with option values that
 are Cybersource card type codes:
// <select id="cardTypeOverride">
//   <option value="001">Visa</option>
//   <option value="002">Mastercard</option>
//   <option value="003">American Express</option>
//    etc...
// </select>

var options = {
  type: document.querySelector('#cardTypeOverride').value
```

```
};
microform.createToken(options, function(err, token) {
  // handle errors & token response
});
```

Handling error scenarios

```
microform.createToken(options, function(err, token) {
  if (err) {
    switch (err.reason) {
      case 'CREATE_TOKEN_NO_FIELDS_LOADED':
        break;
      case 'CREATE_TOKEN_TIMEOUT':
        break;
      case 'CREATE_TOKEN_NO_FIELDS':
        break;
      case 'CREATE_TOKEN_VALIDATION_PARAMS':
        break;
      case 'CREATE_TOKEN_VALIDATION_FIELDS':
        break;
      case 'CREATE_TOKEN_VALIDATION_SERVERSIDE':
        break;
      case 'CREATE_TOKEN_UNABLE_TO_START':
        break;
      default:
        console.error('Unknown error');
        break;
  } else {
    console.log('Token created: ', token);
  }
});
```

Class: MicroformError

This class defines how error scenarios are presented by Microform, primarily as the first argument to callbacks. See callback(erropt, nullable, dataopt, nullable) > {void} (on page 69).

Members

`(static, readonly)`Reason Codes - Field Load Errors

Possible errors that can occur during the loading or unloading of a field.

Properties

Name	Type	Description
FIELD_UNLOAD_ERROR	string	Occurs when you attempt to unload a field that is not currently loaded.
FIELD_ALREADY_LOADED	string	Occurs when you attempt to load a field which is already loaded.
FIELD_LOAD_CONTAINER_SELECTOR	string	Occurs when a DOM element cannot be located using the supplied CSS Selector string.
FIELD_LOAD_INVALID_CONTAINER	string	Occurs when an invalid container parameter has been supplied.
FIELD_SUBSCRIBE_UNSUPPORTED_EVENT	string	Occurs when you attempt to subscribe to an unsupported event type.
FIELD_SUBSCRIBE_INVALID_CALLBACK	string	Occurs when you supply a callback that is not a function.

`(static, readonly)`Reason Codes - Field object Creation

Possible errors that can occur during the creation of a Field object createField(fieldType, optionsopt) > {Field} (on page 60).

Properties

Name	Type	Description
CREATE_FIELD_INVALID_FIELD_TYPE	string	Occurs when you try to create a field with an unsupported type.
CREATE_FIELD_DUPLICATE	string	Occurs when a field of the given type has already been added to your integration.

`(static, readonly)`Reason Codes - Flex object Creation

Possible errors that can occur during the creation of a Flex object.

Properties

Name	Type	Description
CAPTURE_CONTEXT_INVALID	string	Occurs when you pass an invalid JWT.
CAPTURE_CONTEXT_EXPIRED	string	Occurs when the JWT you pass has expired.

`(static, readonly)`Reason Codes - Iframe validation errors

Possible errors that can occur during the loading of an iframe.

Properties

Name	Type	Description
IFRAME_JWT_VALIDATION_FAILED	string	Occurs when the iframe cannot validate the JWT passed.
IFRAME_UNSUPPORTED_FIELD_TYPE	string	Occurs when the iframe is attempting to load with an invalid field type.

`(static, readonly)`Reason Codes - Token creation

Possible errors that can occur during the request to create a token.

Properties

Name	Type	Description
CREATE_TOKEN_NO_FIELDS_LOADED	string	Occurs when you try to request a token, but no fields have been loaded.
CREATE_TOKEN_TIMEOUT	string	Occurs when the `createToken` call was unable to proceed.
CREATE_TOKEN_XHR_ERROR	string	Occurs when there is a network error when attempting to create a token.
CREATE_TOKEN_NO_FIELDS	string	Occurs when the data fields are unavailable for collection.
CREATE_TOKEN_VALIDATION_PARAMS	string	Occurs when there's an issue with parameters supplied to `createToken`.
CREATE_TOKEN_VALIDATION_FIELDS	string	Occurs when there's a validation issue with data in your loaded fields.
CREATE_TOKEN_VALIDATION_SERVERSIDE	string	Occurs when server-side validation rejects the `createToken` request.
CREATE_TOKEN_UNABLE_TO_START	string	Occurs when no loaded field was able to handle the `createToken` request.

`(nullable)correlationID :string`

The correlationId of any underlying API call that resulted in this error.

Type

String

`(nullable)details :array`

Additional error specific information.

Type

Array

`(nullable)informationLink :string`

A URL link to general online documentation for this error.

Type

String

`message :string`

A simple human-readable description of the error that has occurred.

Type

String

`reason :string`

A reason corresponding to the specific error that has occurred.

Type

String

Events

You can subscribe to Microform Integration events and obtain them through event listeners. Using these events, you can easily enable your checkout user interface to respond to any state changes as soon as they happen.

Events

Event Name	Emitted When
`autocomplete`	Customer fills the credit card number using a browser or third-party extension. This event provides a hook onto the additional information provided during the `autocomplete` event.
`blur`	Field loses focus.
`change`	Field contents are edited by the customer. This event contains various data such as validation information and details of any detected card types.
`focus`	Field gains focus.
`inputSubmitRequest`	Customer requests submission of the field by pressing the Return key or similar.
`load`	Field has been loaded on the page and is ready for user input.
`unload`	Field is removed from the page and no longer available for user input.
`update`	Field configuration was updated with new options.

Some events may return data to the event listener's callback as described in the next section.

Subscribing to Events

Using the `.on()` method provided in the `microformInstance` object, you can easily subscribe to any of the supported events.

For example, you could listen for the `change` event and in turn display appropriate card art and display brand-specific information.

```
var secCodeLbl = document.querySelector('#mySecurityCodeLabel');
var numberField = flex.createField('number');

// Update your security code label to match the detected card type's terminology
numberField.on('change', function(data) {
```

```
  secCodeLbl.textContent = (data.card && data.card.length > 0) ?
 data.card[0].securityCode.name : 'CVN';
});

numberField.load('#myNumberContainer');
```

The `data` object supplied to the event listener's callback includes any information specific to the triggered event.

Card Detection

By default, Microform attempts to detect the card type as it is entered. Detection info is bubbled outwards in the `change` event. You can use this information to build a dynamic user experience, providing feedback to the user as they type their card number.

```
{
  "card": [
    {
      "name": "mastercard",
      "brandedName": "MasterCard",
      "cybsCardType": "002",
      "spaces": [ 4, 8, 12],
      "lengths": [16],
      "securityCode": {
        "name": "CVC",
        "length": 3
      },
      "luhn": true,
      "valid": false,
      "couldBeValid": true
    },
    /* other identified card types */
  ]
}
```

If Microform Integration is unable to determine a single card type, you can use this information to prompt the customer to choose from a possible range of values.

If **type** is specified in the `microformInstance.createToken(options,...)` method, the specified value always takes precedence over the detected value.

Autocomplete

By default, Microform Integration supports the autocomplete event of the **cardnumber** field provided by certain browsers and third-party extensions. An `autocomplete` event is provided to allow easy access to the data that was provided to allow integration with other elements in your checkout process.

The format of the data provided in the event might be as follows:

```
{
name: '_____',
expirationMonth: '__',
expirationYear: '____'
}
```

These properties are in the object only if they contain a value; otherwise, they are undefined. Check for the properties before using the event. The following example displays how to use this event to update other fields in your checkout process:

```
var number = microform.createField('number');
number.on('autocomplete', function(data) {
  if (data.name) document.querySelector('#myName').value = data.name;
  if (data.expirationMonth) document.querySelector('#myMonth').value =
 data.expirationMonth;
  if (data.expirationYear) document.querySelector('#myYear').value =
 data.expirationYear;
});
```

Global

Type Definitions

`callback(erropt, nullable, dataopt, nullable) > {void}`

Microform uses the error-first callback pattern, as commonly used in Node.js.

If an error occurs, it is returned by the first `err` argument of the callback. If no error occurs, `err` has a null value and any return data is provided in the second argument.

Parameters

Name	Type	Attributes	Description
err	MicroformError. See Class: MicroformError (on page 63).	<optional> <nullable>	An Object detailing occurred errors, otherwise null.
data	*	<optional> <nullable>	In success scenarios, this is whatever data has been returned by the asynchronous function call, if any.

Returns

Type: void

Example

The following example shows how to make use of this style of error handling in your code:

```
foo(function (err, data) {
    // check for and handle any errors
    if (err) throw err;

    // otherwise use the data returned
    console.log(data);
});
```

`StylingOptions`

Styling options are supplied as an object that resembles CSS but is limited to a subset of CSS properties that relate only to the text within the iframe.

Supported CSS selectors:

- input
- ::placeholder
- :hover
- :focus
- :disabled
- valid
- invalid

Supported CSS properties:

- `color`
- `cursor`
- `font`
- `font-family`
- `font-kerning`
- `font-size`
- `font-size-adjust`
- `font-stretch`
- `font-style`
- `font-variant`
- `font-variant-alternates`
- `font-variant-caps`
- `font-variant-east-asian`
- `font-variant-ligatures`
- `font-variant-numeric`
- `font-weight`
- `line-height`
- `opacity`
- `text-shadow`
- `text-rendering`
- `transition`
- `-moz-osx-font-smoothing`
- `-moz-tap-highlight-color`

- `-moz-transition`
- `-o-transition`
- `-webkit-font-smoothing`
- `-webkit-tap-highlight-color`
- `-webkit-transition`

Any unsupported properties will not be applied and raise a `console.warn()`.

Properties

Name	Type	Attributes	Description
input	object	<optional>	Main styling applied to the input field.
::placeholder	object	<optional>	Styles for the ::placeholder pseudo-element within the main input field. This also adds vendor prefixes for supported browsers.
:hover	object	<optional>	Styles to apply when the input field is hovered over.
:focus	object	<optional>	Styles to apply when the input field has focus.
:disabled	object	<optional>	Styles applied when the input field has been disabled.
valid	object	<optional>	Styles applied when Microform detects that the input card number is valid. Relies on card detection being enabled.
invalid	object	<optional>	Styles applied when Microform detects that the input card number is invalid. Relies on card detection being enabled.

Example

```
const styles = {
  'input': {
    'color': '#464646',
    'font-size': '16px',
    'font-family': 'monospace'
  },
  ':hover': {
    'font-style': 'italic'
  },
  'invalid': {
```

```
    'color': 'red'
  }
};
```

Unified Checkout

Unified Checkout provides a single interface with which you can accept numerous types of digital payments.

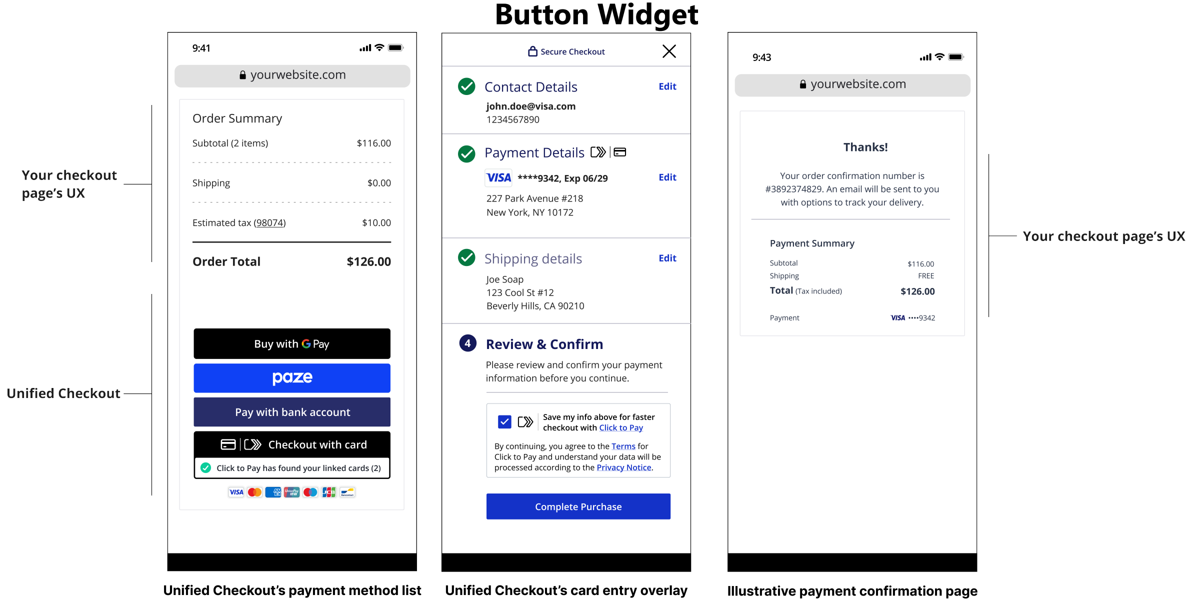
Unified Checkout consists of a server-side component and a client-side JavaScript library.

The server-side component authenticates your merchant identity and instructs the system to act within your payment environment. The response contains limited-use public keys. The keys are for end-to-end encryption and contain merchant-specific payment information that drives the interaction of the application. The client-side JavaScript library dynamically and securely places digital payment options onto your e-commerce page.

The provided JavaScript library enables you to securely accept many payment options within your e-commerce environment. Unified Checkout can be embedded seamlessly into your existing webpage, simplifying payment acceptance.

When a customer chooses a payment method from the button widget, Unified Checkout handles all of the interactions with the digital payment that was chosen. It also provides a response to your e-commerce system.

The figure below shows Unified Checkout with customer checkout payment options.



For examples of different payment method UIs through Unified Checkout, see Unified Checkout UI (on page 106).

Unified Checkout Flow

To integrate Unified Checkout into your platform, you must follow several integration steps. This section gives a high-level overview of how to integrate and launch Unified Checkout on your webpage and process a transaction using the data that Unified Checkout collects for you. You can find the detailed specifications of the APIs later in this document.

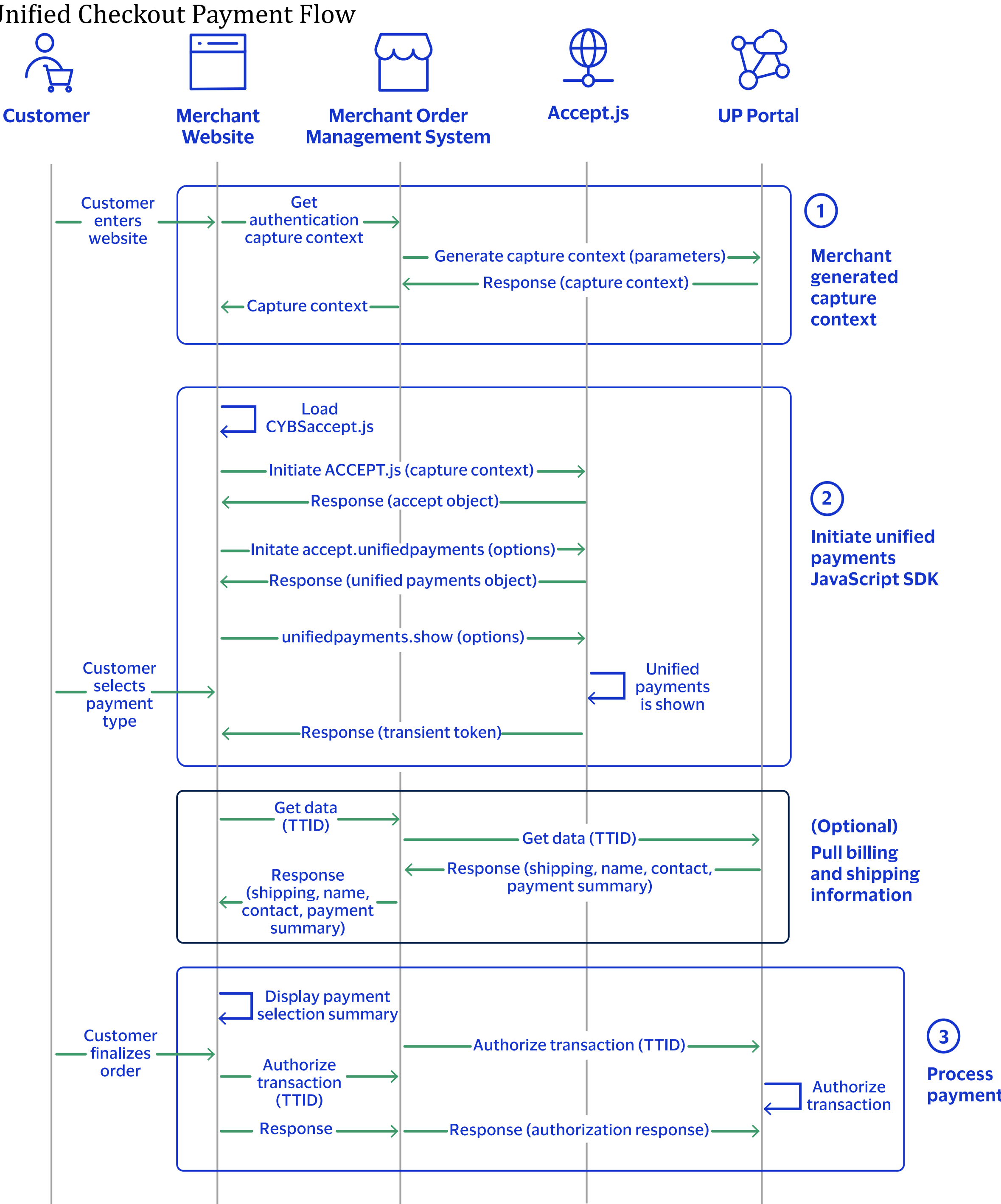
The integration flow consists of three events:

1. You send a server-to-server API request for a capture context. This request is fully authenticated and returns a JSON Web Token (JWT) that is necessary in order to invoke the frontend JavaScript library. For information on setting up the server side, see Server-Side Set Up (on page 78).
2. You invoke the Unified Checkout JavaScript library using the JWT response from the capture context request. For information on setting up the client side, see Client-Side Set Up (on page 81).
3. You process the payment.

If you want to retrieve the billing and shipping information captured by Unified Checkout, you can use the payment details API.

The figure below shows the Unified Checkout payment flow.

Unified Checkout Payment Flow



For more information on the specific APIs referenced, see these topics:

- Capture Context API (on page 89)
- Payment Details API (on page 97)

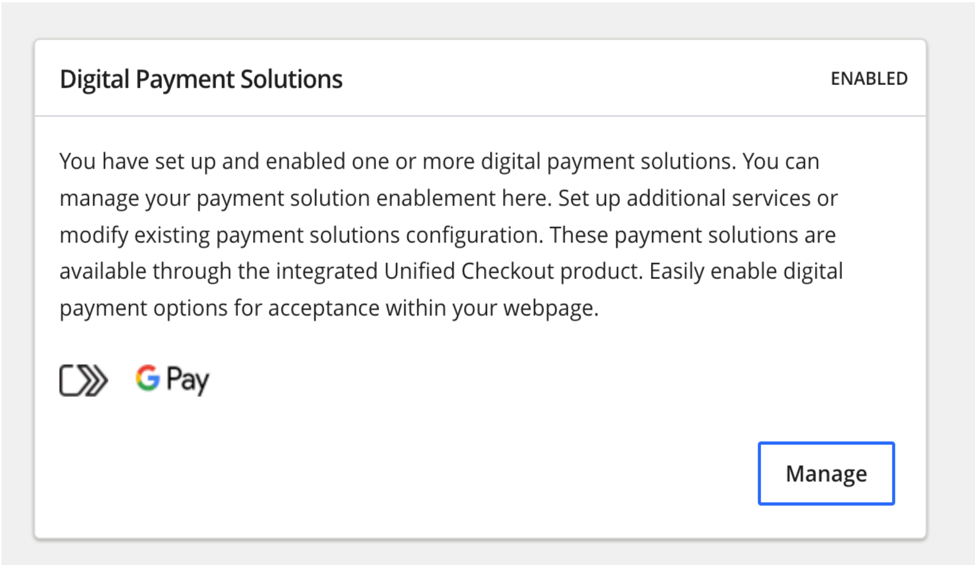
Enabling Unified Checkout in the Business Center

To begin using Unified Checkout, you must first ensure that your merchant ID (MID) is configured to use the service and that any payment methods you intend to use are properly set up.

1. Log in to the Business Center:
   Test URL: `https://businesscentertest.cybersource.com/ebc2`
   Production URL: `https://businesscenter.cybersource.com`
   If you are unable to access this page, contact your sales representative.

2. In the Business Center, go to the left navigation panel and choose **Payment Configuration** > **Unified Checkout**.

3. You can configure various payment methods such as Google Pay and Click to Pay. Click **Set up** and follow the instructions for your selected payment methods. When payment methods are enabled, they appear on the payment configuration page.



4. Click **Manage** to edit your existing payment method configurations or enroll in new payment methods as they are released.

Server-Side Set Up

This section contains the information you need to set up your server. Initializing Unified Checkout within your webpage begins with a server-to-server call to the sessions API. This step authenticates your merchant credentials, and establishes how the Unified Checkout frontend components will function. The sessions API request contains parameters that define how Unified Checkout performs.

The server-side component provides this information:

- A transaction-specific public key is used by the customer's browser to protect the transaction.
- An authenticated context description package that manages the payment experience on the client side. It includes available payment options such as card networks, payment interface styling, and payment methods.

The functions are compiled in a JSON Web Token (JWT) object referred to as the *capture context*. For information JSON Web Tokens, see JSON Web Tokens (on page 123).

Capture Context

The capture context request is a signed JSON Web Token (JWT) that includes all of the merchant-specific parameters. This request tells the frontend JavaScript library how to behave within your payment experience. For information on JSON Web Tokens, see JSON Web Tokens (on page 123).

You can define the payment cards and digital payments that you want to accept in the capture context. Use the **allowedCardNetworks** field to define the card types.

Available card networks for card entry:

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

Important: Click to Pay supports American Express, Mastercard, and Visa for saved cards.

Use the **allowedPaymentTypes** field to define the digital payment methods.

Example:

```
{
  "targetOrigins" : [ "https://www.test.com" ],
  "clientVersion" : "0.19",
  "allowedCardNetworks" : [ "VISA", "MASTERCARD", "AMEX" ],
  "allowedPaymentTypes" : [ "PANENTRY", "CLICKTOPAY", "GOOGLEPAY" ],
  "country" : "US",
  "locale" : "en_US",
  "captureMandate" : {
    "billingType" : "FULL",
    "requestEmail" : true,
    "requestPhone" : true,
    "requestShipping" : true,
    "shipToCountries" : [ "US", "GB" ],
    "showAcceptedNetworkIcons" : true
  },
  "orderInformation" : {
    "amountDetails" : {
      "totalAmount" : "1.01",
      "currency" : "USD"
    },
  }
}
```

This diagram shows how elements of the capture context request appear in the card entry form.

Anatomy of a Manual Card Entry Form


```
{
  "targetOrigins" : [ "https://the-up-demo.appspot.com" ],
  "clientVersion" : "0.19",
  "allowedCardNetworks" : [ "VISA", "MASTERCARD", "AMEX" ],
  "allowedPaymentTypes" : [ "CLICKTOPAY" ],
  "country" : "US",
  "locale" : "en_US",
  "captureMandate" : {
   "billingType" : "FULL",
   "requestEmail" : true,
   "requestPhone" : true,
   "requestShipping" : true,
   "shipToCountries" : [ "US", "GB" ],
   "showAcceptedNetworkIcons" : true
  },
  "orderInformation" : {
   "amountDetails" : {
    "totalAmount" : "1.01",
    "currency" : "USD"
   },
   "billTo" : {
    "address1" : "277 Park Avenue",
    "administrativeArea" : "NY",
    "buildingNumber" : "#218",
    "country" : "US",
    "district" : "district",
    "locality" : "New York",
    "postalCode" : "10172",
    "email" : "john.doe@visa.com",
    "firstName" : "John",
    "lastName" : "Doe",
    "middleName" : "F",
    "nameSuffix" : "Jr",
    "title" : "Mr",
    "phoneNumber" : "1234567890",
    "phoneType" : "phoneType"
   },
   "shipTo" : {
    "address1" : "123 Cool St",
    "administrativeArea" : "CA",
    "buildingNumber" : "#12",
    "country" : "US",
    "district" : "string",
    "locality" : "Beverly Hills",
    "postalCode" : "90210",
    "firstName" : "Joe",
    "lastName" : "Soap"
   }
  }
}
```


Secure checkout
1 Contact Details
Email address
john.doe@visa.com
Phone number
1234567890
Click to Pay will use this information to check if you have saved cards. A one-time passcode may be sent to confirm it's you. Message and data rates may apply. What is Click to Pay?
Continue
2 Payment Details
3 Shipping Details
3 Review and Confirm

Secure checkout
Contact Details Edit
john.doe@visa.com
1234567890
2 Payment Details
Card details
Card number
Expiry MM / YY
Security code
Billing address
First name John
Last name Doe
Address 277 Park Avenue
Address 2 #218
City New York
State New York
Zip code 10172
Country USA
Continue
3 Shipping Details
4 Review and Confirm

Secure checkout
Contact Details
john.doe@visa.com Edit
1234567890
Payment Details
VISA ****9342, Exp 06/29 Edit
227 Park Avenue #218
New York, NY 10172
3 Shipping Details
Same as billing address
First name Joe
Last name Soap
Address 123 Cool St
Address 2 #12
City Beverly Hills
State California
Zip code 90210
Country USA
Continue
4 Review and Confirm

Secure Checkout
Contact Details Edit
john.doe@visa.com
1234567890
Payment Details
VISA ****9342, Exp 06/29 Edit
227 Park Avenue #218
New York, NY 10172
Shipping details Edit
Joe Soap
123 Cool St #12
Beverly Hills, CA 90210
4 Review & Confirm
Please review and confirm your payment information before you continue.
Save my info above for faster checkout with Click to Pay
By continuing, you agree to the Terms for Click to Pay and understand your data will be processed according to the Privacy Notice.
Complete Purchase

For more information on requesting the capture context, see Capture Context API (on page 89).

Client-Side Set Up

This section contains the information you need to set up the client side. You use the Unified Checkout JavaScript library to add the payment interface to your e-commerce site. It has two primary components:

- The button widget, which lists the payment methods available to the customer.
- The payment acceptance page, which captures payment information from the cardholder. You can set up the payment acceptance page to be integrated with your webpage or added as a sidebar.

Follow these steps to set up the client:

1. Load the JavaScript library.
2. Initialize the accept object, the capture context JWT. For information JSON Web Tokens, see JSON Web Tokens (on page 123).
3. Initialize the unified payment object with optional parameters.
4. Show the button list or payment acceptance page or both.

The response to these interactions is a transient token that you use to retrieve the payment information captured by the UI.

Loading the JavaScript Library and Invoking the Accept Function

Use the client library asset path returned by the capture context response to invoke Unified Checkout on your page.

Get the JavaScript library URL dynamically from the capture context response. When decoded, it appears in the JSON parameter **clientLibrary** as:

```
https://apitest.cybersource.com/up/v1/assets/x.y.z/SecureAcceptance.js
```

When you load the library, the capture context that you received from your initial server-side request is used to invoke the accept function.

Important: Use the **clientLibrary** parameter value in the capture context response to obtain the Unified Checkout JavaScript library URL. This ensures that you are always using the most up-to-date library. Do not hard-code the Unified Checkout JavaScript library URL.

JavaScript Example: Initializing the SDK

```
<script
 src="https://apitest.cybersource.com/up/v1/assets/0.19.0/SecureAcceptance.js"></script>
<script>
  Accept('header.payload.signature').then(function(accept) {
    // use accept object
  });
</script>
```

In this example, `header.payload.signature` refers to the capture context JWT.

Adding the Payment Application and Payment Acceptance

After you initialize the Unified Checkout object, you can add the payment application and payment acceptance pages to your webpage. You can attach the Unified Checkout embedded tool and payment acceptance pages to any named element within your HTML. Typically, they are attached to explicit named `<div>` components that are replaced with Unified Checkout's `iframes`.

Important: If you do not specify a location for the payment acceptance page, it is placed in the side bar.

JavaScript Example: Setting Up with Full Sidebar

```
var authForm = document.getElementById("authForm");
var transientToken = document.getElementById("transientToken");

var cc = document.getElementById("captureContext").value;
var showArgs = {
  containers: {
    paymentSelection: "#buttonPaymentListContainer"
  }
};
Accept(cc)
  .then(function(accept) {
    return accept.unifiedPayments();
  })
  .then(function(up) {
    return up.show(showArgs);
  })
  .then(function(tt) {
```

```
    transientToken.value = tt;
    authForm.submit();
  });
```

JavaScript Example: Setting Up with the Embedded Component

The main difference between using an embedded component and the sidebar is that the **accept.unifiedPayments** object is set to `false`, and the location of the payment screen is passed in the containers argument.

```
var authForm = document.getElementById("authForm");
var transientToken = document.getElementById("transientToken");

var cc = document.getElementById("captureContext").value;
var showArgs = {
  containers: {
    paymentSelection: "#buttonPaymentListContainer",
    paymentScreen: "#embeddedPaymentContainer"
  }
};
Accept(cc)
    .then(function(accept) {
      // Gets the UC instance (e.g. what card brands I requested, any address information
 I pre-filled etc.)
      return accept.unifiedPayments();
    })
    .then(function(up) {
      // Display the UC instance
      return up.show(showArgs);
    })
    .then(function(tt) {
      // Return transient token from UC's UI to our app
      transientToken.value = tt;
      authForm.submit();
    }).catch(function(error) {
      //merchant logic for handling issues
      alert("something went wrong");
  });
```

Transient Tokens

The response to a successful customer interaction with Unified Checkout is a transient token. The transient token is a reference to the payment data collected on your behalf. Tokens allow secure card payments to occur without risk of exposure to sensitive payment information. The transient token is a short-term token that lasts 15 minutes. This reduces your PCI burden/responsibility and ensures that sensitive information is not exposed to your backend systems.

Transient Token Format

The transient token is issued as a JSON Web Token (JWT) (RFC 7519). For information on JSON Web Tokens, see JSON Web Tokens (on page 123).

The payload portion of the token is a Base64-encoded JSON string and contains various claims. This example shows a payload:

```
{
  "iss" : "Flex/00",
  "exp" : 1706910242,
  "type" : "gda-0.9.0",
  "iat" : 1706909347,
  "jti" : "1D1I2O2CSTMW3UIXOKEQFI4OQX1L7CMSKDE3LJ8B5DVZ6WBJGKLQ65BD6222D426",
  "content" : {
    "orderInformation" : {
      "billTo" : {
        // Empty fields present within this node indicate which fields were captured by
        // the application without exposing you to personally identifiable information
        // directly.
      },
      "amountDetails" : {
        // Empty fields present within this node indicate which fields were captured by
        // the application without exposing you to personally identifiable information
        // directly.
      },
      "shipTo" : {
        // Empty fields present within this node indicate which fields were captured by
        // the application without exposing you to personally identifiable information
        // directly.
      }
    },
    "paymentInformation" : {
      "card" : {
        "expirationYear" : {
          "value" : "2028"
        },
        "number" : {
```

```
          "maskedValue" : "XXXXXXXXXXXX1111",
          "bin" : "411111"
        },
        "securityCode" : { },
        "expirationMonth" : {
          "value" : "06"
        },
        "type" : {
          "value" : "001"
        }
      }
    }
  }
}
```


Token Verification

When you receive the transient token, you should cryptographically verify its integrity using the public key embedded within the capture context. Doing so verifies that Cybersource issued the token and that the data has not been tampered with in transit. Verifying the transient token JWT involves verifying the signature and various claims within the token. Programming languages each have their own specific libraries to assist. For an example in Java, see: Java Example in Github.

Authorizations with a Transient Token

This section provides the minimum information required in order to perform a successful authorization with a Unified Checkout transient token. Doong so eliminates the need to send sensitive payment data along with the request.

To send the transient token with a request, use the **tokenInformation.transientTokenJwt** field.

An API request made with a transient token looks like this:

```
"tokenInformation": {

  "transientTokenJwt": "eyJraWQiOiIwOG4zUnVsRTJGQXJDRktycVRkZFlkWGZSWFhMNXFoNSIsImFsZyI6Il
JTMjU2In0.eyJpc3MiOiJGbGV4LzA3IiwiZXhwIjoxNTk3MDg0ODk3LCJ0eXBlIjoiZ2RhLTAuMS4xIiwiaWF0Ijox
NTk3MDgzOTk3LCJqdGkiOiIxQzI2VlpSkVJUU1PTzVIMDUwNEtINDdJMEFNMklaRkM0M1Y1TDU0MUhCTE45Q09JM0w
3NUYzMTk0RTE5NkExIn0.SNm1VZaZr3DkTqUg9CdV0F5arRe-uQU9oUWPKfWIpbIzIPZutRokv5DSDcM7asZIKNJyN
IBx5DLsl_yQPrKgzhwQxZ8qbhto7cu3t-v8DHG2yO951plPQVQnj7x-vEDcXkLUL1F8sqY23R5HW-xSDAQ3AFLawCc
kn7Q2eudRGeuMhLWH742Gflf9Hz3KyKnmeNKA3o9yW2na16nmeVZaYGqbUSPVITdl5cMA0o9lEob8E3OQH0HHdmIsu
5uMA4x7DeBjfTKD1rQxFP3JBNVcv30AIMLkNcw0pHbtHDVzKBWxUVxvnm3zFEdiBuSAco2uWhC9zFqHrrp64ZvzxZq
oGA"
}
```

To retrieve non-sensitive data from a Unified Checkout transient token, use the `payment-details` endpoint. This data includes cardholder name and billing and shipping details. For more information, see Payment Details API (on page 97).

Important: Fields supplied directly in an API request supersede those that are also present in the transient token. For example, in the request below, the total amount might have been overridden because of a tax calculation.

Endpoint

Production: `POST https://api.cybersource.com/pts/v2/payments`

Test: `POST https://apitest.cybersource.com/pts/v2/payments`

Required Field for an Authorization with a Transient Token

tokenInformation.transientTokenJwt

REST Example: Authorization with a Transient Token

Endpoint:

- **Production:** `POST https://api.cybersource.com/pts/v2/payments`
- **Test:** `POST https://apitest.cybersource.com/pts/v2/payments`

Request

Important: The transient token may already contain information such as billing address and total amount. Any fields included in the request will supersede the information contained in the transient token.

```
{
  "tokenInformation": {

  "transientTokenJwt": "eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFsZyI6Il
JTMjU2In0.eyJpc3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTAuMS4wIiwiaWF0Ijox
NjE0NzkxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPRUhJVldBSURPkhLNjJJSFQxUVE
1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4TRIVZN-jpNbamcluAKVZ1zfdhbkrB6aNVW
ECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFfzTuq41sDhlIqB-UatveU_2ukPxLYl87EX9ytpx4zCJ
Vmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85tl2OHaajEc25tS1FwH3jDOfjAC8mu2MEk-Ew0-ukZ70Ce7Zaq4
cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bpvcKLr9k_8b9YnddIzp0p0JOCjXC_nuofQT7_x_-CQayx2czE0kD53HeNYC
5hQ"
  }
}
```

Response to Successful Request

```
{
    "_links": {
        "authReversal": {
            "method": "POST",
            "href": "/pts/v2/payments/6826225725096718703955/reversals"
        },
        "self": {
            "method": "GET",
            "href": "/pts/v2/payments/6826225725096718703955"
        },
        "capture": {
            "method": "POST",
            "href": "/pts/v2/payments/6826225725096718703955/captures"
```

```
        }
    },
    "clientReferenceInformation": {
        "code": "TC50171_3"
    },
    "id": "6826225725096718703955",
    "orderInformation": {
        "amountDetails": {
            "authorizedAmount": "102.21",
            "currency": "USD"
        }
    },
    "paymentAccountInformation": {
        "card": {
            "type": "001"
        }
    },
    "paymentInformation": {
        "tokenizedCard": {
            "type": "001"
        },
        "card": {
            "type": "001"
        },
        "customer": {
            "id": "AAE3DD3DED844001E05341588E0AD0D6"
        }
    },
    "pointOfSaleInformation": {
        "terminalId": "111111"
    },
    "processorInformation": {
        "approvalCode": "888888",
        "networkTransactionId": "123456789619999",
        "transactionId": "123456789619999",
        "responseCode": "100",
        "avs": {
            "code": "X",
            "codeRaw": "I1"
        }
    },
    "reconciliationId": "68450467YGMSJY18",
    "status": "AUTHORIZED",
    "submitTimeUtc": "2023-04-27T19:09:32Z"
    }
}
```

Capture Context API

This section contains the information you need to request the capture context using the capture context API.

The capture context request contains all of the merchant-specific parameters that tell the frontend JavaScript library how to behave within your payment experience.

The capture context is a signed JSON Web Token (JWT) containing this information:

- Merchant-specific parameters that dictate the customer payment experience for the current payment transaction.
- A one-time public key that secures the information flow during the current payment transaction.

For information on JSON Web Tokens, see JSON Web Tokens (on page 123).

The capture context is signed with long-lasting keys so that its authenticity can be validated.

You can define the payment cards and other application features in the capture context. Use the **allowedCardNetworks** field to define the card types. These are the available card networks:

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

Use the **allowedPaymentTypes** field to define the digital payment methods.

For more information on enabling and managing these digital payment methods, see these topics:

- Enabling Click to Pay (on page 102)
- Enrolling in Google Pay (on page 102)

Important:

When integrating with Cybersource APIs, Cybersource recommends that you dynamically parse the response for the fields that you are looking for. Additional fields may be added in the future.

You must ensure that your integration can handle new fields that are returned in the response. While the underlying data structures will not change, you must also ensure that your integration can handle changes to the order in which the data is returned. Cybersource uses semantic versioning practices, which enables you to retain backwards compatibility as new fields are introduced in minor version updates.

Endpoint

Production: `POST https://api.cybersource.com/up/v1/capture-contexts`

Test: `POST https://apitest.cybersource.com/up/v1/capture-contexts`

Required Fields for Requesting the Capture Context

Your capture context request must include these fields:

allowedPaymentTypes

clientVersion

country

locale

orderInformation.amountDetails.currency

orderInformation.amountDetails.totalAmount

targetOrigins

The URL in this field value must contain `https`.

For a complete list of fields you can include in your request, see the Cybersource REST API Reference.

REST Example: Requesting the Capture Context

Endpoint:

- **Production:** `POST https://api.cybersource.com/up/v1/capture-contexts`

  Test: `POST https://apitest.cybersource.com/up/v1/capture-contexts`

Request

```
{
  {
    "targetOrigins": [
      "https://unified-payments.appspot.com"
    ],
    "clientVersion": "0.19",
    "allowedCardNetworks" : [ "VISA", "MASTERCARD", "AMEX" ],
    "allowedPaymentTypes" : [ "CLICKTOPAY", "PANENTRY", "GOOGLEPAY" ],
    "country": "US",
    "locale": "en_US",
    "captureMandate": {
      "billingType": "FULL",
      "requestEmail": true,
      "requestPhone": true,
      "requestShipping": true,
      "shipToCountries": [
```

```
        "US",
        "UK"
      ],
      "showAcceptedNetworkIcons": true
    },
    "orderInformation": {
      "amountDetails": {
        "totalAmount": "21.00",
        "currency": "USD"
      },
      "billTo": {
        "address1": "1111 Park Street",
        "address2": "Apartment 24B",
        "administrativeArea": "NY",
        "country": "US",
        "district": "district",
        "locality": "New York",
        "postalCode": "00000",
        "company": {
          "name": "Visa Inc",
          "address1": "900 Metro Center Blvd",
          "administrativeArea": "CA",
          "buildingNumber": "1",
          "country": "US",
          "district": "district",
          "locality": "Foster City",
          "postalCode": "94404"
        },
        "email": "maya.tran@company.com",
        "firstName": "Maya",
        "lastName": "Tran",
        "middleName": "S",
        "title": "Ms",
        "phoneNumber": "1234567890",
        "phoneType": "phoneType"
      },
      "shipTo": {
        "address1": "Visa",
        "address2": "123 Main Street",
        "address3": "Apartment 102",
        "administrativeArea": "CA",
        "buildingNumber": "string",
        "country": "US",
        "locality": "Springfield",
        "postalCode": "99999",
        "firstName": "Joe",
        "lastName": "Soap"
      }
```

```
    }
  }
}
```

Successful Encrypted JWT Response to Request


```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsImRhdGEiOi
JHeUhXV0d5SG5lK2Fld1JsalVUaGJoQUFFQVZMbTR6QTA0UHBqaGFXOHVSZ2UvNFQweEtIbW9KUWNYaE1hd0RmVzVQ
VFBLNXBlZ05vRkVocnNacjdnbldLeHBRdTNWSm4vTDBjbmZOaTRSdjdlTElcdTAwM2QiLCJvcmlnaW4iOiJodHRwcz
ovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJqd2siOnsia3R5IjoiUlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJl
bmMiLCJuIjoibVhHbi1DbllDX1pkODVQdTJaaDluVDdZOUpQX1RjUV9BSzlBQTFHQkJfOFVXd2FHWEZIMGxfa2EwXz
V0ekFleU5uVWZLQ016WFFHV2dMZ2hnZXdLMjJzWlVXVTdDT0k4RkNTWktpUjBYRGJ2TTVZYkYxejk0TmNmWVJGc0p0
ZzhTbE1jY0stS00tOUFjdldYQWlxUEs0Mk5GZnlIVE5uX3BpVDdhZHRDMGFZQlhCdkw2WXFmcWM5bXBua05FQTJVN0
x5VWFyRy1rVFVIQW8xX2tjdW1tTEF1X1Y5OEQyMndsaHMtekhEcnFVTFhsNEdKSGF6WjNXVWJDWHc5c0o2dFowVmVn
X1Bpbnhmck9mazA0RWNaVlM5clBXWW1HRnA3V2NyR0FQTkRCQzFPZ0NKNW1mRmpMNEtpcVpVNURpTWFsbURGdzg5VV
p1bllBVWlrdUU1SURRIiwia2lkIjoiMDBDeWg5UHhhdDdCUkMwa0pXUG5hUVJsOU9jTGMzZVoifX0sImN0eCI6W3si
ZGF0YSI6eyJhbGxvd2VkUGF5bWVudFR5cGVzIjpbeyJwYWdlIjoxLCJ0eXBlIjoiUEFORU5UUlkifSx7InBhZ2UiOj
IsInR5cGUiOiJTUkNWSVNBIn0seyJwYWdlIjozLCJ0eXBlIjoiU1JDTUFTVEVSQ0FSRCJ9LHsicGFnZSI6NCwidHlw
ZSI6IlNSQ0FNRVgifSx7InBhZ2UiOjUsInR5cGUiOiJHT09HTEVQQVkifSx7InBhZ2UiOjYsInR5cGUiOiJBUFBMRV
BBWSJ9XSwicGF5bWVudENvbmZpZ3VyYXRpb25zIjp7IlNSQ1ZJU0EiOnsib3JpZ2luIjoiaHR0cHM6Ly9zYW5kYm94
LWFzc2V0cy5zZWN1cmUuY2hlY2tvdXQudmlzYS5jb20iLCJwYXRoIjoiL2NoZWNrb3V0LXdpZGdldC9yZXNvdXJjZX
MvanMvc3JjLWktYWRhcHRlci92aXNhU2RrLmpzIiwicGFuRW5jcnlwdGlvbktleSI6eyJrdHkiOiJSU0EiLCJlIjoi
QVFBQiIsInVzZSI6ImVuYyIsImtpZCI6IldaTEQzS0VBUFdJRThMS0pEMU0xMTNYMXExamZUZE5pNTI0al9aQWxLVm
tlanBxM0EiLCJuIjoic1pQSXVzRGY3eVFubmhCa1U5bXUxNFZPTzNDcnVpM2I3ckFmMktZZW9iVVJtWEExN2IxSlg5
amcwQ2QtdmdwbXV5VHJ4QlVTYy00YjAtVVBnU3dHRnFQV1VweDA4RXhxcndQRE92Rm9qQm91MndseXE4YmN5MFVzLU
JmZUN6U0U1bE1WZFNYVFhYWGNOcXUtcWIyMmpDQ0NKQUxweHNBcnNib01PWHNMZWRoM000WE5RNVhHQXRSZjdiLS11
VFk1RHI5S0xZeVV2WktBblkwNE1LSlBFTzU0WWlJRk01RFRBaE5PbXMwODlqZE1keC1VUklLSmpQVTItUnBIRzF1OE
xDRzAyOFJUSXBQc05iUmFudVM1VEFZX3pseERnYjFoS0ozNlliWkVOSExnOVBYVEJoZE9NbFU5MERUTGxmY2JMVGEt
RDdEZ2xqQWFXQ3V2ekxQYUd3In0sInBhcmFtZXRlcnMiOnsic3JjSW5pdGlhdG9ySWQiOiJSNDVOMzQzRDZLWFpSWU
1CSVhMSTIxeDgtWGtMaWh4Q21lcFMzaEFlUm91RWcwaTVVIiwic3JjaURwYUlkIjoiOTBhZDlhN2QtOTU5Ni00ZWQx
LWE3MTEtMmJjOTllM2JjNWZmIiwic3JjaVRyYW5zYWN0aW9uSWQiOiIzMWJkNTRjZi1hOGIyLTQwMTEtODQ0Ny1jYj
czZDM4OGU0NjYiLCJkcGFUcmFuc2FjdGlvbk9wdGlvbnMiOnsiZHBhTG9jYWxlIjoiZW5fVVMiLCJwYXlsb2FkVHlw
ZUluZGljYXRvciI6IkZVTEwiLCJyZXZpZXdBY3Rpb24iOiJjb250aW51ZSIsImRwYUFjY2VwdGVkQmlsbGluZ0NvdW
50cmllcyI6W10sImRwYUFjY2VwdGVkU2hpcHBpbmdDb3VudHJpZXMiOltdLCJkcGFCaWxsaW5nUHJlZmVyZW5jZSI6
IkFMTCIsImRwYVNoaXBwaW5nUHJlZmVyZW5jZSI6IkFMTCIsImNvbnN1bWVyTmFtZVJlcXVlc3RlZCI6dHJ1ZSwiY2
9uc3VtZXJFbWFpbEFkZHJlc3NSZXF1ZXN0ZWQiOnRydWUsImNvbnN1bWVyUGhvbmVOdW1iZXJSZXF1ZXN0ZWQiOnRy
dWUsInRyYW5zYWN0aW9uQW1vdW50Ijp7InRyYW5zYWN0aW9uQW1vdW50IjoiMS4wMSIsInRyYW5zYWN0aW9uQ3Vycm
VuY3lDb2RlIjoiVVNEIn0sInBheW1lbnRPcHRpb25zIjp7ImRwYUR5bmFtaWNEYXRhVHRsTWludXRlcyI6MTUsImR5
bmFtaWNEYXRhVHlwZSI6IlRBVlYiLCJkcGFQYW5SZXF1ZXN0ZWQiOmZhbHNlfX19fSwiU1JDTUFTVEVSQ0FSRCI6ey
JvcmlnaW4iOiJodHRwczovL3NhbmRib3guc3JjLm1hc3RlcmNhcmQuY29tIiwicGF0aCI6Ii9zZGsvc3Jjc2RrLm1h
c3RlcmNhcmQuanMiLCJwYW5FbmNyeXB0aW9uS2V5Ijp7Imt0eSI6IlJTQSIsImUiOiJBUUFCIiwidXNlIjoiZW5jIi
wia2lkIjoiMjAyMzAyMDcyMjM1MjEtc2FuZGJveC1mcGFuLWVuY3J5cHRpb24tc3JjLW1hc3RlcmNhcmQtaW50Iiwi
a2V5X29wcyI6WyJlbmNyeXB0Iiwid3JhcEtleSJdLCJhbGciOiJSU0EtT0FFUC0yNTYiLCJuIjoidDA2SThzamxTLX
Jyczd1Q2FnSDhldm9ldW1hUm92S3ppWlNJOVMyTjlJRFE5dFcyUGFwZlJhOUxjMUt2ZUVCRFZzMjdQa2hrVTVPeUhn
UDBpRWpUdUtWcHZoNTlUNGxhLW1CU0lsczdVZWNVUUxMYTBXa21idEw3ak5kbHRBNWZxN0FoY0FyNXFjYTk4OHFyTG
Q3SXlyOUUwQzNUeGJUOXRvMWlRY3B6OG9jWk9EUlhvaWRGQW5PVkw1WUdGbWxzcmVEYko0VmhzaTBwQWRjY1FjaWwt
eWRTZ3VyS0ItcnFLcHBiOWVwb211NFFVaDMzODJDdjhOb2JZbUYzb3M4bkdHZ0dQLWN5WG8wbnNLY1BBZ2ZybFF6b3
```

```
M3cUh4VU9yRmUyeF9sWjFHMUFFLVhya3J4akJ5czlxNTNHTVJTTkNROGMtX21jRjlwYnE0SFlCcy12RDVRIn0sInBh
cmFtZXRlcnMiOnsic3JjaVRyYW5zYWN0aW9uSWQiOiIzMWJkNTRjZi1hOGIyLTQwMTEtODQ0Ny1jYjczZDM4OGU0Nj
YiLCJzcmNpRHBhSWQiOiI5ODQ4Y2ZmNC1jODY0LTRmMTgtOWYwMy1hOGY1MGE2OTJlZGRfc3lzdGVtdGVzdCIsInNy
Y0luaXRpYXRvcklkIjoiNmY1ZDZjMDktZjhlMi00MzMwLWEzZGYtMjBiOWFkN2E0NTJiIiwiZHBhVHJhbnNhY3Rpb2
5PcHRpb25zIjp7InRyYW5zYWN0aW9uVHlwZSI6IlBVUkNIQVNFIiwiZHBhTG9jYWxlIjoiZW5fVVMiLCJkcGFBY2Nl
cHRlZFNoaXBwaW5nQ291bnRyaWVzIjpbXSwiY29uc3VtZXJFbWFpbEFkZHJlc3NSZXF1ZXN0ZWQiOnRydWUsImNvbn
N1bWVyUGhvbmVOdW1iZXJSZXF1ZXN0ZWQiOnRydWUsInRyYW5zYWN0aW9uQW1vdW50Ijp7InRyYW5zYWN0aW9uQW1v
dW50IjoiMS4wMSIsInRyYW5zYWN0aW9uQ3VycmVuY3lDb2RlIjoiVVNEIn0sImRwYUFjY2VwdGVkQmlsbGluZ0NvdW
50cmllcyI6W10sImRwYUJpbGxpbmdQcmVmZXJlbmNlIjoiRlVMTCIsImRwYVNoaXBwaW5nUHJlZmVyZW5jZSI6IkZV
TEwiLCJjb25zdW1lck5hbWVSZXF1ZXN0ZWQiOnRydWUsInBheWxvYWRUeXBlSW5kaWNhdG9yIjoiRlVMTCIsInBheW
1lbnRPcHRpb25zIjp7ImR5bmFtaWNEYXRhVHlwZSI6IkNBUkRfQVBQTElDQVRJT05fQ1JZUFRPR1JBTV9TSE9SVF9G
T1JNIn19fX0sIlNSQ0FNRVgiOnsib3JpZ2luIjoiaHR0cHM6Ly9xd3d3LmFleHAtc3RhdGljLmNvbSIsInBhdGgiOi
IvYWthbWFpL3JlbW90ZWNvbW1lcmNlL3NjcmlwdHMvYW1leFNESy0xLjAuMC5qcyIsInBhbkVuY3J5cHRpb25LZXki
Onsia3R5IjoiUlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJraWQiOiJzcmMtYW1leC1jYXJkLWVuYy0yMDI0Ii
wiYWxnIjoiUlNBLU9BRVAtMjU2IiwibiI6Im1FazBibUxDMlpRVy1hNEtYMW5EWTNaZlBMRnJIOHRuVXlJYjVrVEtn
emFlYWdpbWFINFhxUDRadzA1aWk2TXZkdk4wVDJweVNKUTRqb2toUEMySVdlbWlWUEc4ZkNQQk1KeHhqeTJFdTlvdG
Jpd0dSQkNneHdjdS1hY2pZYXVwVlB0RE43ZW5nSERkbk9nYXJsb0dyUFVNNklFRVpXX3ZFQjljU3JNX0JhOFNjQzhS
YWZnTlNZODFpeGF4UEE4Y09oQUF2ckxRN0toRTVReFN6SU1mcnpiMUxCWUdMNFlQQnVuZk5BMnczZnZMd2ZCbDJfLV
JGUkNVbVBFdjFOdVhxeG8xUk4wOGoydW44ZWljR3ZudDBndC0yMW5HcmJjNnhwcDdwWlkyb2otaGMwWlVsTnlFX2tK
cExxTNU9VWjhHZU9acDRxVlJ4aGtJUEd4RWVGLVFXaVNnOHVXazF4Nm5jdGhyTVVKWVYxSFB1OHRIa0pEbThBYS1Ec2
hQTmVpeERqX1ZGVkVTOFYteUlJUndnLVUyODJXUGIwVDJ0S1JYZG5qbE52Y2xCc0lfNFZ3ZzVjV0VoU2tTc3pVQXkx
UENTRm5rWjVJRU9yaGdfMFRwZTdhaU84dzVzUndOaFpuUnBKeUlzUHQtbE1Dbzd6cjg1QjJ2eGNvUGZmU1NwM0ZaIn
0sInBhcmFtZXRlcnMiOnsic3JjaVRyYW5zYWN0aW9uSWQiOiIzMWJkNTRjZi1hOGIyLTQwMTEtODQ0Ny1jYjczZDM4
OGU0NjYiLCJzcmNJbml0aWF0b3JJZCI6ImQyZTdkOTc1LWIwYWEtNGZhYS05YTUxLTY4MDAyMjkwZDc1NiIsImRwYU
RhdGEiOnsiZHBhTmFtZSI6InRlc3QgU2hvcCB3ZWJzaXRlIFJlZ2lzdHJhdGlvbiIsImRwYUxvZ29VcmkiOiJodHRw
Oi8vd3d3LnRlc3RzcmNyZWdpc3RyYXRpb24uY29tIiwiZHBhUHJlc2VudGF0aW9uTmFtZSI6InRlc3QgU2hvcCB3ZW
JzaXRlIFJlZ2lzdHJhdGlvbiIsImRwYVVyaSI6Imh0dHA6Ly93d3cudGVzdHNyY3JlZ2lzdHJhdGlvbi5jb20ifSwi
ZHBhVHJhbnNhY3Rpb25PcHRpb25zIjp7ImRwYUxvY2FsZSI6ImVuX1VTIiwiZHBhQWNjZXB0ZWRCaWxsaW5nQ291bn
RyaWVzIjpbXSwiZHBhQWNjZXB0ZWRTaGlwcGluZ0NvdW50cmllcyI6W10sImRwYUJpbGxpbmdQcmVmZXJlbmNlIjoi
QUxMIiwiZHBhU2hpcHBpbmdQcmVmZXJlbmNlIjoiQUxMIiwiY29uc3VtZXJOYW1lUmVxdWVzdGVkIjp0cnVlLCJjb2
5zdW1lckVtYWlsQWRkcmVzc1JlcXVlc3RlZCI6dHJ1ZSwiY29uc3VtZXJQaG9uZU51bWJlclJlcXVlc3RlZCI6dHJ1
ZSwicmV2aWV3QWN0aW9uIjoiY29udGludWUiLCJ0aHJlZURzUHJlZmVyZW5jZSI6Ik5PTkUiLCJwYXltZW50T3B0aW
9ucyI6W3siZHluYW1pY0RhdGFUeXBlIjoiRFlOQU1JQ19DQVJEX1NFQ1VSSVRZX0NPREUiLCJkcGFEeW5hbWljRGF0
YVR0bE1pbnV0ZXMiOiIxNSJ9XX19fSwiR09PR0xFUEFZIjp7ImNsaWVudExpYnJhcnkiOiJodHRwczovL3BheS5nb2
9nbGUuY29tL2dwL3AvanMvcGF5LmpzIiwicGF5bWVudE9wdGlvbnMiOnsiZW52aXJvbm1lbnQiOiJURVNUIn0sInBh
eW1lbnREYXRhUmVxdWVzdCI6eyJhcGlWZXJzaW9uIjoyLCJhcGlWZXJzaW9uTWlub3IiOjAsIm1lcmNoYW50SW5mby
I6eyJtZXJjaGFudElkIjoiQkNSMkRONFQ3RERZQlRUViIsIm1lcmNoYW50TmFtZSI6IlVuaWZpZWQgQ2hlY2tvdXQg
TWVyY2hhbnQifSwiYWxsb3dlZFBheW1lbnRNZXRob2RzIjpbeyJ0eXBlIjoiQ0FSRCIsInBhcmFtZXRlcnMiOnsiYW
xsb3dlZEF1dGhNZXRob2RzIjpbIlBBTl9PTkxZIiwiQ1JZUFRPR1JBTV8zRFMiXSwiYWxsb3dlZENhcmROZXR3b3Jr
cyI6WyJWSVNBIiwiTUFTVEVSQ0FSRCIsIkFNRVgiXSwiYmlsbGluZ0FkZHJlc3NSZXF1aXJlZCI6dHJ1ZSwiYmlsbG
luZ0FkZHJlc3NQYXJhbWV0ZXJzIjp7ImZvcm1hdCI6IkZVTEwiLCJwaG9uZU51bWJlclJlcXVpcmVkIjp0cnVlfX0s
InRva2VuaXphdGlvblNwZWNpZmljYXRpb24iOnsidHlwZSI6IlBBWU1FTlRfR0FURVdBWSIsInBhcmFtZXRlcnMiOn
siZ2F0ZXdheSI6ImN5YmVyc291cmNlIiwiZ2F0ZXdheU1lcmNoYW50SWQiOiJwc19ocGEifX19XSwidHJhbnNhY3Rp
b25JbmZvIjp7InRvdGFsUHJpY2VTdGF0dXMiOiJGSU5BTCIsInRvdGFsUHJpY2UiOiIxLjAxIiwiY291bnRyeUNvZG
UiOiJVUyIsImN1cnJlbmN5Q29kZSI6IlVTRCJ9LCJlbWFpbFJlcXVpcmVkIjp0cnVlLCJzaGlwcGluZ0FkZHJlc3NS
ZXF1aXJlZCI6dHJ1ZSwic2hpcHBpbmdBZGRyZXNzUGFyYW1ldGVycyI6eyJwaG9uZU51bWJlclJlcXVpcmVkIjp0cn
VlfX19LCJBUFBMRVBBWSI6eyJzZXNzaW9uUGF0aCI6Ii9mbGV4L3YyL2FwcGxlL3BheW1lbnQtc2Vzc2lvbnMiLCJt
ZXJjaGFudElkZW50aWZpZXIiOiJtZXJjaGFudC5jb20uY3liZXJzb3VyY2Uuc3RhZ2VmbGV4IiwiZGlzcGxheU5hbW
UiOiJVQyBUZXN0In19LCJjYXB0dXJlTWFuZGF0ZSI6eyJiaWxsaW5nVHlwZSI6IkZVTEwiLCJyZXF1ZXN0RW1haWwi
```

```
OnRydWUsInJlcXVlc3RQaG9uZSI6dHJ1ZSwicmVxdWVzdFNoaXBwaW5nIjp0cnVlLCJzaGlwVG9Db3VudHJpZXMiOl
tdLCJzaG93QWNjZXB0ZWROZXR3b3JrSWNvbnMiOnRydWV9LCJvcmRlckluZm9ybWF0aW9uIjp7ImFtb3VudERldGFp
bHMiOnsidG90YWxBbW91bnQiOiIxLjAxIiwiY3VycmVuY3kiOiJVU0QifX0sInRhcmdldE9yaWdpbnMiOlsiaHR0cH
M6Ly90aGUtdXAtZGVtby5hcHBzcG90LmNvbSJdLCJpZnJhbWVzIjp7Im1jZSI6Ii9tY2UvaWZyYW1lLmh0bWwiLCJi
dXR0b25zIjoiL2J1dHRvbmxpc3QvaWZyYW1lLmh0bWwiLCJzcmMiOiIvc2VjdXJlLXJlbW90ZS1jb21tZXJjZS9zcm
MuaHRtbCIsImN0cCI6Ii9jdHAvY3RwLmh0bWwiLCJnb29nbGVwYXkiOiIvZ29vZ2xlcGF5L2dvb2dsZXBheS5odG1s
IiwiYXBwbGVwYXkiOiIvYXBwbGVwYXkvYXBwbGVwYXkuaHRtbCIsInBhemUiOiIvcGF6ZS9wYXplLmh0bWwifSwiY2
xpZW50VmVyc2lvbiI6IjAuMTkiLCJjb3VudHJ5IjoiVVMiLCJsb2NhbGUiOiJlbl9VUyIsImFsbG93ZWRDYXJkTmV0
d29ya3MiOlsiVklTQSIsIk1BU1RFUkNBUkQiLCJBTUVYIl0sImNyIjoiNmM0dUcyemFXdVBvbkxLM0R2NEwxVlJpTF
VOMkFVczY4QU84bVdaUTA0X1RNLVFDdDhNUDNTQklvcGQ2Y2NtOTdmSEo1QXViVzh6VFhJTW91TTRjQWFrbm80NktI
VndGRFpxQ0tfWTVwMEVzRHJmdFVTREFrZ21KZ0pNbHJ2cnYzTkpFOWdzcldBMl8zdDJBR2hQbEtfMU9rZyIsInNlcn
ZpY2VPcmlnaW4iOiJodHRwczovL3N0YWdldXAuY3liZXJzb3VyY2UuY29tIiwiY2xpZW50TGlicmFyeSI6Imh0dHBz
Oi8vc3RhZ2V1cC5jeWJlcnNvdXJjZS5jb20vdXAvdjEvYXNzZXRzLzAuMTkuMC9TZWN1cmVBY2NlcHRhbmNlLmpzIi
wibG9nZ2luZ1BhdGggiOiIvdXAvdjEvbG9nLWV2ZW50cyIsImFzc2V0c1BhdGggiOiIvdXAvdjEvYXNzZXRzLzAuMTku
MCIsImNsaWVudExpYnJhcnlJbnRlZ3JpdHkiOiJzaGEyNTYtWllDT2tucVh5bjRad3NyOFYwaE5OcjZaUitZYThJbH
NkdFplTkhPbDJYVVx1MDAzZCJ9LCJ0eXBlIjoiZ2RhLTAuOS4wIn1dLCJpc3MiOiJGbGV4IEFQSSIsImV4cCI6MTcx
MDk2NDc4MCwiaWF0IjoxNzEwOTYzODgwLCJqdGkiOiI4SWs4bHU2NEh3NmpDVDhsIn0.XWXmjiZZGyHWIhT1hbBnc2
xfhcYczpBYxhTn4g9NMt2utMaPR8wWcZ8TYDXd8HRLBWZkktkXxFFetJ4Tc6dQ4irZ6KmalWItWEUJpjN-5sLC4Qr1
gG1JOOH5_hK6n_1hnjcQeRUBg-MsCSRBE_MA6ROSZgyfc1_WwL0g1TQUiKN5SvaM_37ooimebPQfvYyXyR_6Zkn9fu
51w6NF_Qj0wtuQP4J4P3cgyZzzOFNKuHOwi7ISmyW6BcQXQrec577SRBfcMhhC3PBxl5OrXua4qUJ_qYbplA8P4n6f
2--onAYef3UXFHmc28eRiTEeN0l0P1Yj45CIotbuw36mZrnRPQ
```


Decrypted Capture Context Header


```
{
  "kid": "j4",
  "alg": "RS256"
}
```


Decrypted Capture Context Body with Selected Fields

```
{
  "flx" : {
    // filled with token metadata
  },
  "ctx" : [ {
    // filled with data related to your capture context request parameters
    "data" : {

  "clientLibrary" : "https://https://
apitest.cybersource.com/up/v1/assets/0.19.0/SecureAcceptance.js"
    },
    "type" : "gda-0.9.0"
  } ],
  "iss" : "Flex API",
```

```
  "exp" : 1710964780,
  "iat" : 1710963880,
  "jti" : "8Ik8lu64Hw6jCT8l"
}
```

Payment Details API

This section contains the information you need to retrieve the non-sensitive data associated with a Unified Checkout transient token and the payment details API. This API can be used to retrieve personally identifiable information, such as the cardholder name and billing and shipping details, without retrieving payment credentials; which helps ease the PCI compliance burden.

There are two methods of authentication:

- HTTP Signature Authentication
- JSON Web Token

Important:

When integrating with Cybersource APIs, Cybersource recommends that you dynamically parse the response for the fields that you are looking for. Additional fields may be added in the future.

You must ensure that your integration can handle new fields that are returned in the response. While the underlying data structures will not change, you must also ensure that your integration can handle changes to the order in which the data is returned. Cybersource uses semantic versioning practices, which enables you to retain backwards compatibility as new fields are introduced in minor version updates.

Endpoint

Production: `GET https://api.cybersource.com/up/v1/payment-details/{id}`

Test: `GET https://apitest.cybersource.com/up/v1/payment-details/{id}`

The `{id}` is the full JWT received from Unified Checkout as the result of capturing payment information. The transient token is a JWT object that you retrieved as part of a successful capture of payment information from a cardholder.

Required Field for Retrieving Transient Token Payment Details

Your request must include this field:

id

Set to the `{id}` of the full JWT received from Unified Checkout as the result of capturing payment information.

REST Example: Retrieving Transient Token Payment Details

Endpoint:

- **Production:** `GET https://api.cybersource.com/up/v1/payment-details/{id}`
- **Test:** `GET https://apitest.cybersource.com/up/v1/payment-details/{id}`

The `{id}` is the full JWT received from Unified Checkout as the result of capturing payment information. The transient token is a JWT object that you retrieved as part of a successful capture of payment information from a cardholder.

Request

```
GET https://apitest.cybersource.com/up/v1/payment-details/{id}
```

Response to Successful Request


```
{
  "paymentInformation": {
    "card": {
      "expirationYear": "2024",
      "number": "XXXXXXXXXXXX1111",
      "expirationMonth": "05",
      "type": "001"
    }
  },
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "21.00",
      "currency": "USD"
    },
    "billTo": {
      "lastName": "Lee",
      "country": "US",
```

```
      "firstName": "Tanya",
      "email": "tanyalee@example.com"
    },
    "shipTo": {
      "locality": "Small Town",
      "country": "US",
      "administrativeArea": "CA",
      "address1": "123 Main Street",
      "postalCode": "98765"
    }
  }
}
```

Unified Checkout Configuration

This section contains information necessary to configure Unified Checkout in the Business Center:

- Enable Digital Payments (on page 101)
- Manage Permissions (on page 103)

Enable Digital Payments

To enable digital payments on Unified Checkout, you must first register for each digital payment method that you would like enabled on your page. This enablement process sends the appropriate information to the digital payment systems and registers your page with each system.

Enable digital payments for Unified Checkout in the Business Center. A list of these available digital payment methods offered by Unified Checkout should be visible:

- Apple Pay
- Click to Pay
- Google Pay

For more information on enabling and managing these digital payment methods, see these topics:

- Enabling Click to Pay (on page 102)
- Enrolling in Google Pay (on page 102)

Enabling Click to Pay

Click to Pay is a digital payment solution that allows customers to pay with their preferred card network and issuer without entering their card details on every website. Customers can use Visa, Mastercard, and American Express cards to streamline their purchase experience. Click to Pay provides a fast, secure, and consistent checkout experience across devices and browsers.

Follow these steps to enable in Click to Pay on Unified Checkout:

1. Navigate to **Payment Configuration > Unified Checkout**.
2. In the Click to Pay section, click **Set Up**.
3. Enter your business name and website URL.
4. Click **Submit**.
   You can now accept digital payments with Click to Pay.

Enrolling in Google Pay

Google Pay is a digital payment product offered by Google through Chrome browsers and Android devices.

Follow these steps to enroll in Google Pay on Unified Checkout:

1. Navigate to **Payment Configuration > Unified Checkout**.
2. In the Google Pay section, click **Set Up**.
3. Enter your business name.
4. Click **Submit**.
   You can now accept digital payments with Google Pay.

Manage Permissions

Portfolio administrators can set permissions for new or existing Business Center user roles for Unified Checkout. Administrators retain full read and write permissions. They enable you to regulate access to specific pages and specify who can access, view, or amend digital products within Unified Checkout.

Portfolio administrators must apply the appropriate user role permission for any existing or newly created Business Center user roles for Unified Checkout. For information on managing permissions as a portfolio administrator, see Managing Permissions as a Portfolio Administrator (on page 105).

If you are a transacting merchant, you might find that your permissions are restricted. If your permissions are restricted, a message appears indicating that you do not have access, or buttons might appear gray. To make changes to your digital products within Unified Checkout that have restricted permissions, contact your portfolio administrator's customer support representative. For more information, see Managing Permissions as a Direct Merchant (on page 104).

Managing Permissions as a Direct Merchant

Follow these steps to configure and manage user permissions in the Business Center for Unified Checkout as a direct merchant:

1. On the left navigation panel, navigate to **Account Management**.
2. Click **Roles** to display a list of your user roles.
3. Click the pencil icon next to the user role that you want to update.
4. Click **Payment Configuration Permission**.
5. Select the relevant permission for the specific user role you are editing. You can select from these Unified Checkout permissions:
    - Unified Checkout View
    - Unified Checkout Manage

Important: If you are a transacting merchant without view permissions, Unified Checkout will still appear on the navigation bar, however, a *no access* message appears when you access Unified Checkout.

If you are a transacting merchant with view permissions but not management permissions, you can access the Unified Checkout screens and view the different payment methods configurations, however, you cannot edit or enroll new products.

Managing Permissions as a Portfolio Administrator

Follow these steps to configure and manage user permissions in the Business Center for Unified Checkout as a portfolio administrator:

1. On the left navigation panel, navigate to **Account Management**.
2. Click **Roles** to see a list of your user roles.
3. Click the pencil icon next to the user role that you want to update.
4. Click **Payment Configuration Permission**.
5. Select the relevant permission for the specific user role you are editing. You can choose from these Unified Checkout permissions:
    - Unified Checkout View
    - Unified Checkout Manage
    - Unified Checkout Portfolio View (available for portfolio users only)
    - Unified Checkout Portfolio Manage (available for portfolio users only)

Important: If all permissions are left unselected, the user has restricted permission. A *no access* message appears when the user tries to access the Unified Checkout digital product enablement pages. The user is advised to contact a customer representative.

If a portfolio user has view permissions and does not have a management role, they can access the Unified Checkout pages, but they cannot modify toggles for different digital payments.

Unified Checkout UI

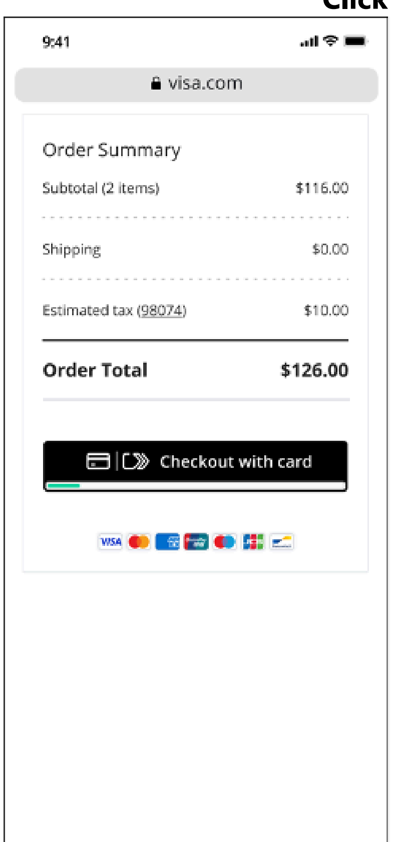
Completing a payment with Unified Checkout requires the customer to navigate through a sequence of interfaces. This section includes examples of the interfaces your customers can expect when completing a payment with these payment methods on Unified Checkout:

- Click to Pay UI (on page 107)
- Google Pay UI (on page 109)
- Manual Payment Entry UI (on page 110)
- Pay with Bank Account UI (on page 114)
- Paze UI (on page 121)

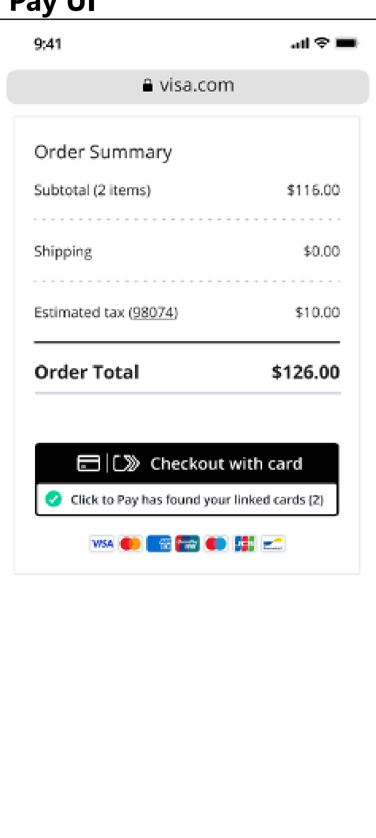
Click to Pay UI

These screen captures show the sequence of events your customer can expect when completing a payment with Click to Pay.

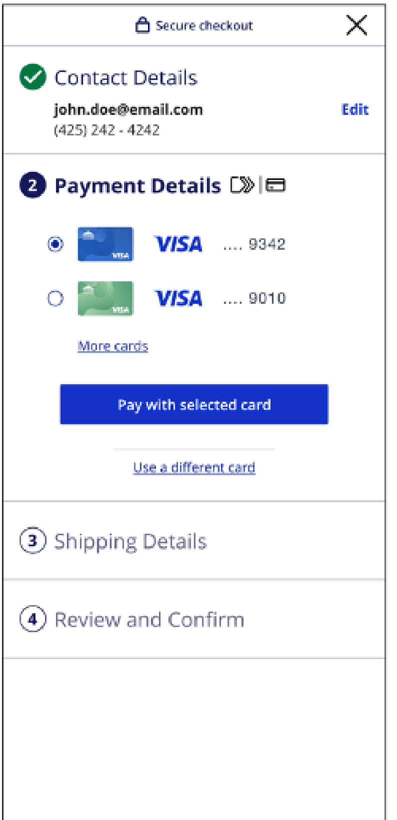
Click to Pay UI



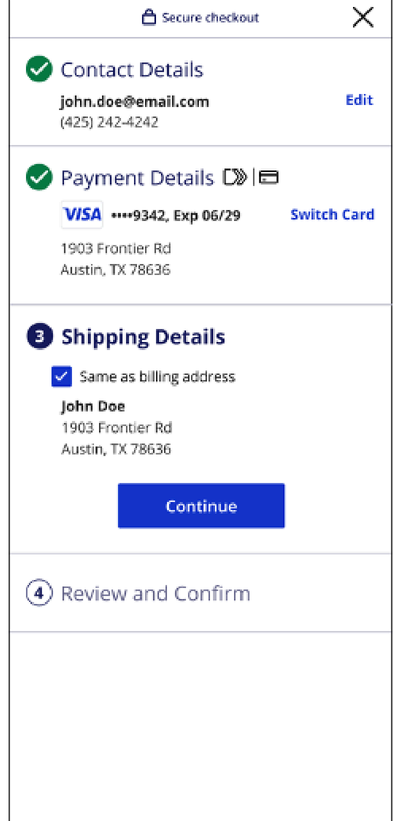
Click to Pay loader animation



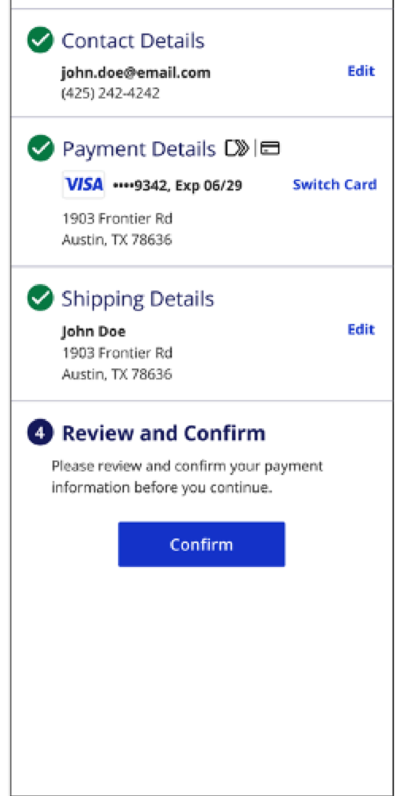
Click to Pay recognized user



Click to Pay saved cards



Click to Pay saved cards

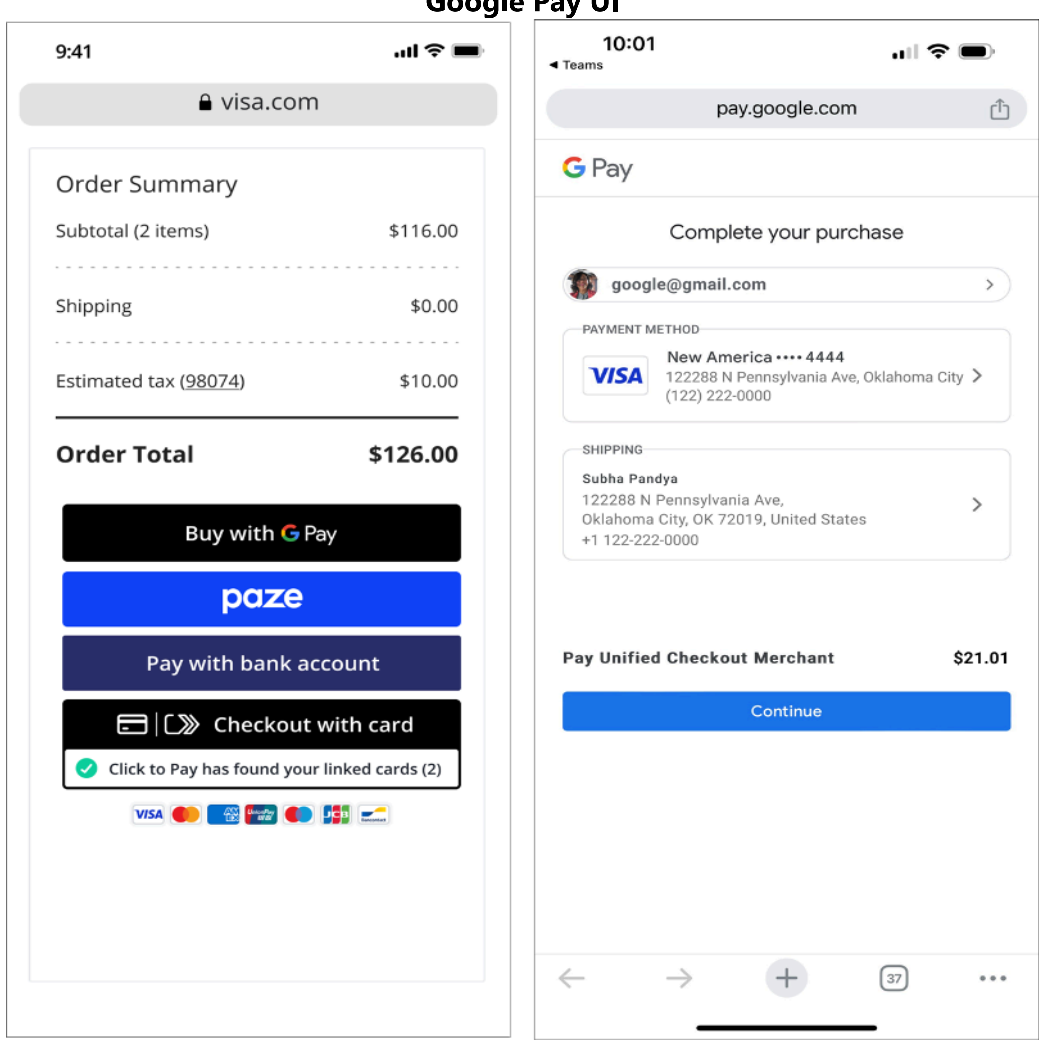


Review screen

Google Pay UI

These screen captures show the sequence of events your customer can expect when completing a payment with Google Pay.

Google Pay UI

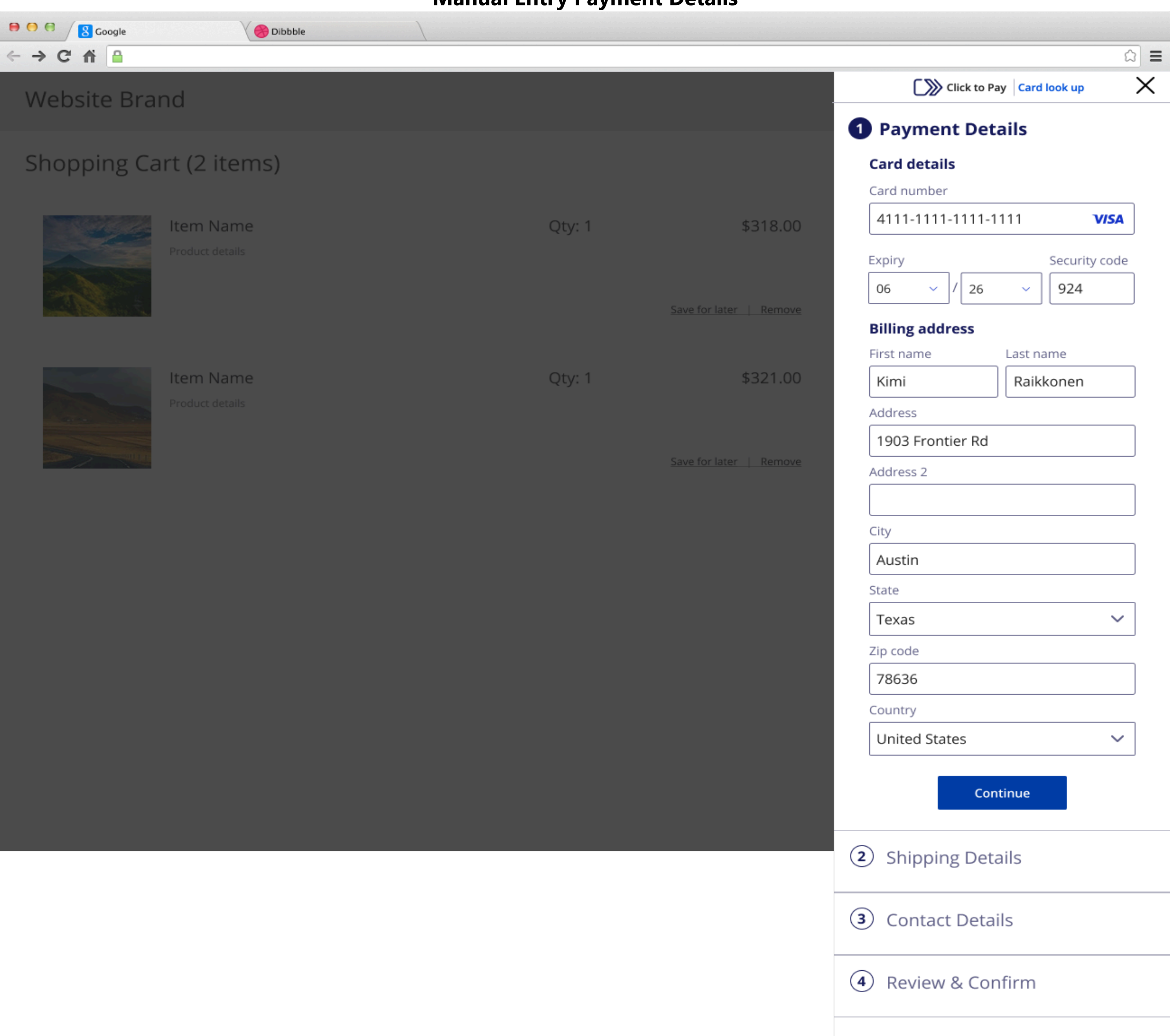


Google Pay in the payment method list **Google Pay checkout**

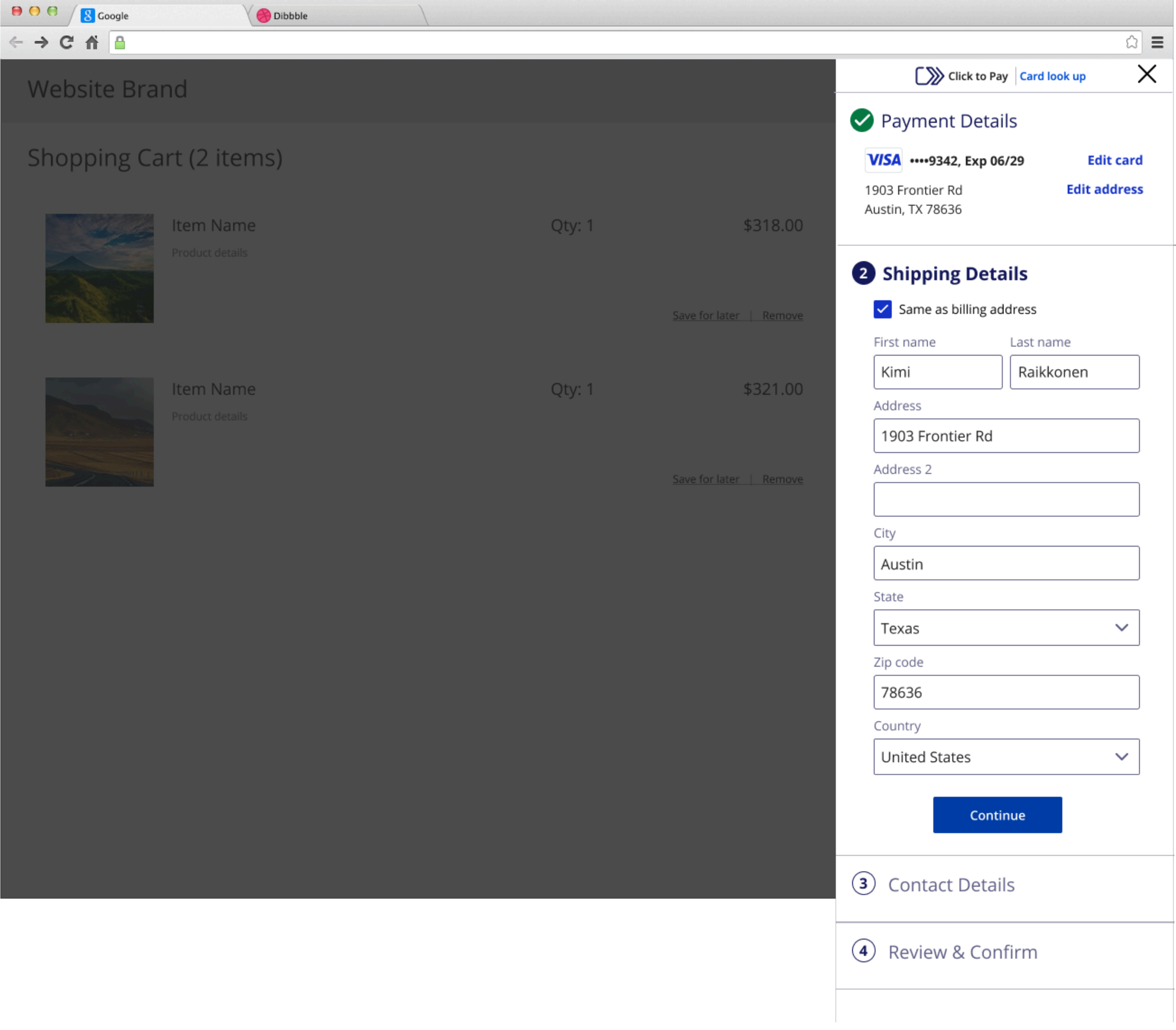
Manual Payment Entry UI

These screen captures show the sequence of events your customer can expect when completing a payment by manually entering payment, shipping, and contact information.

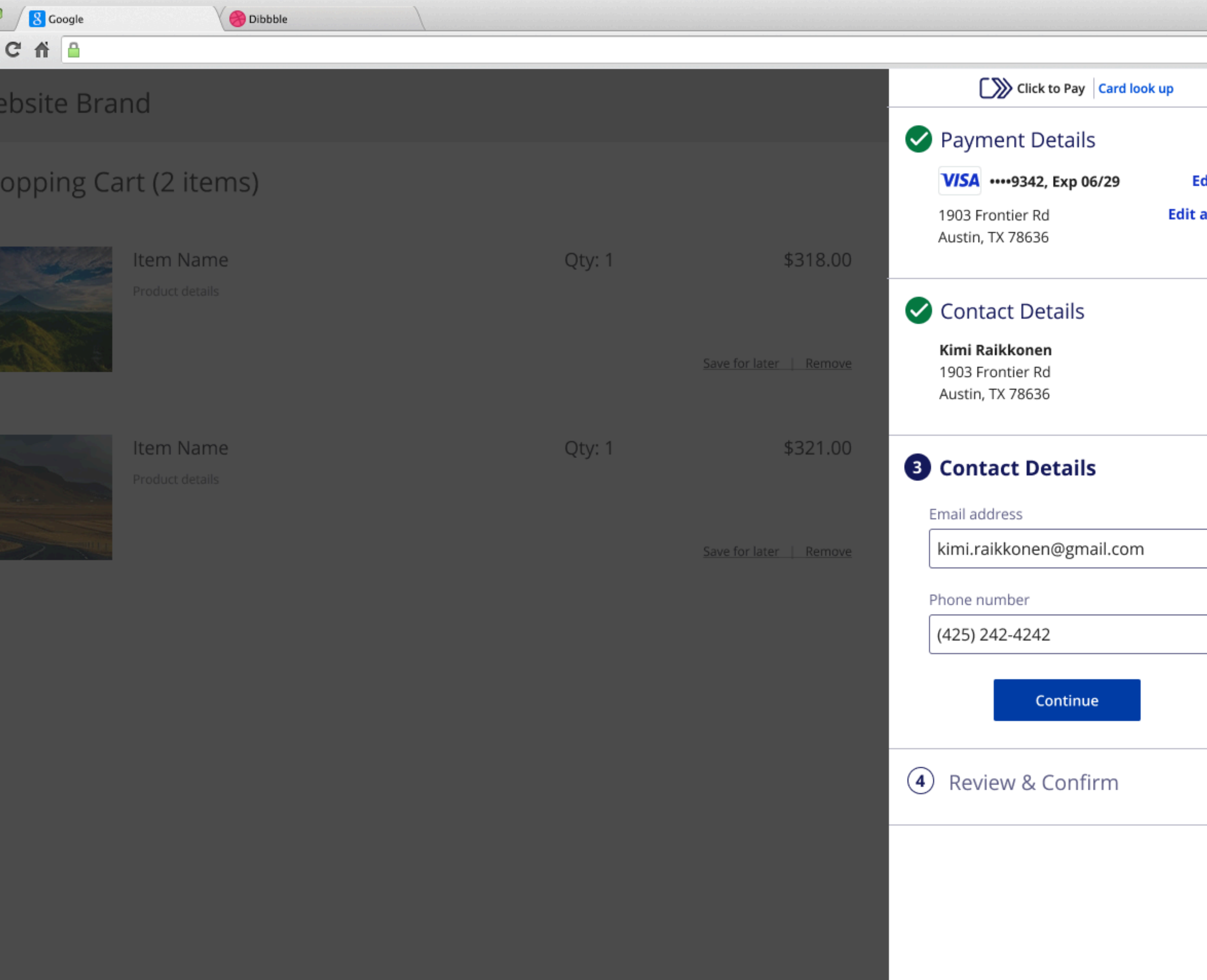
Manual Entry Payment Details



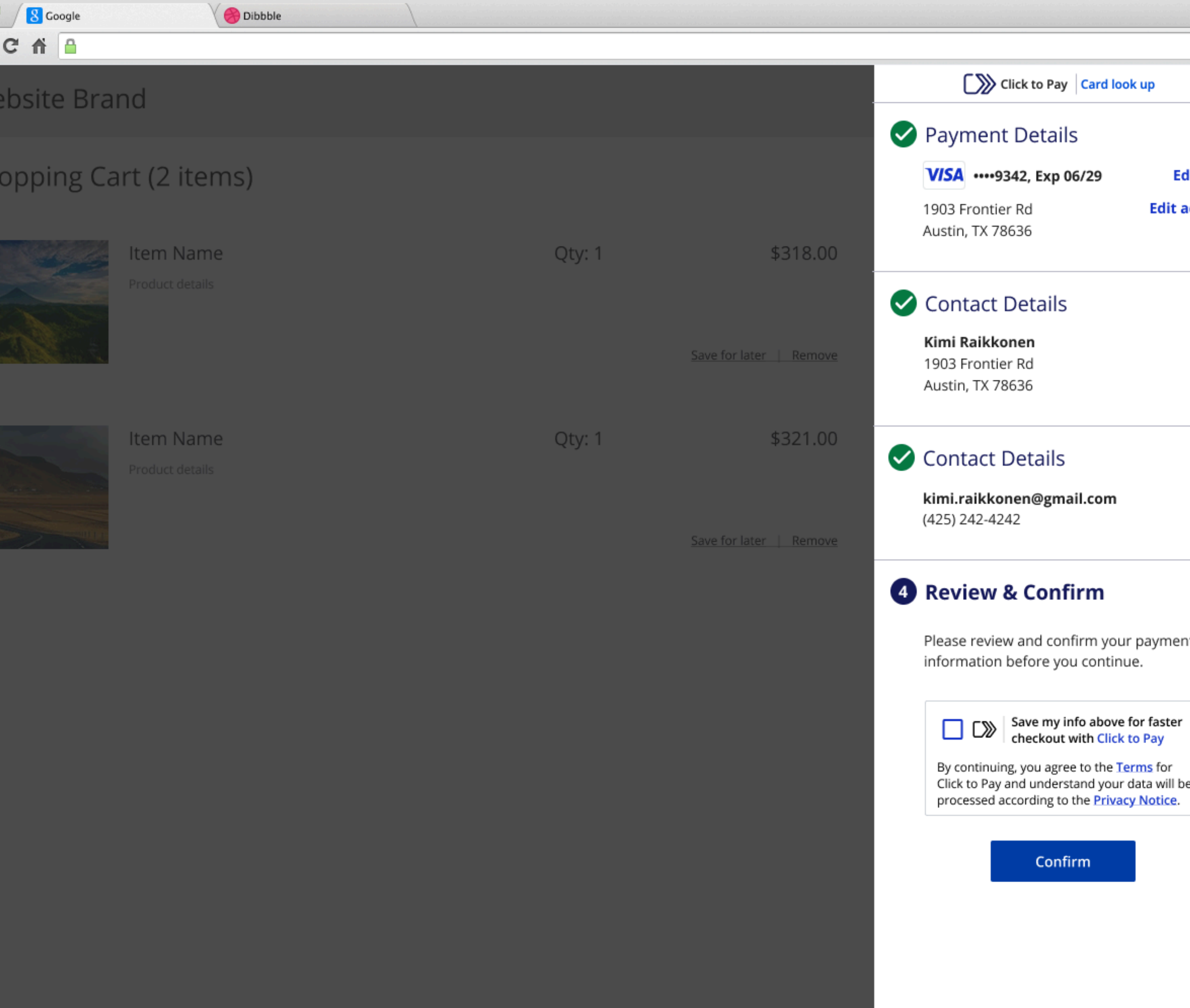
Manual Entry Shipping Details



Manual Entry Contact Details



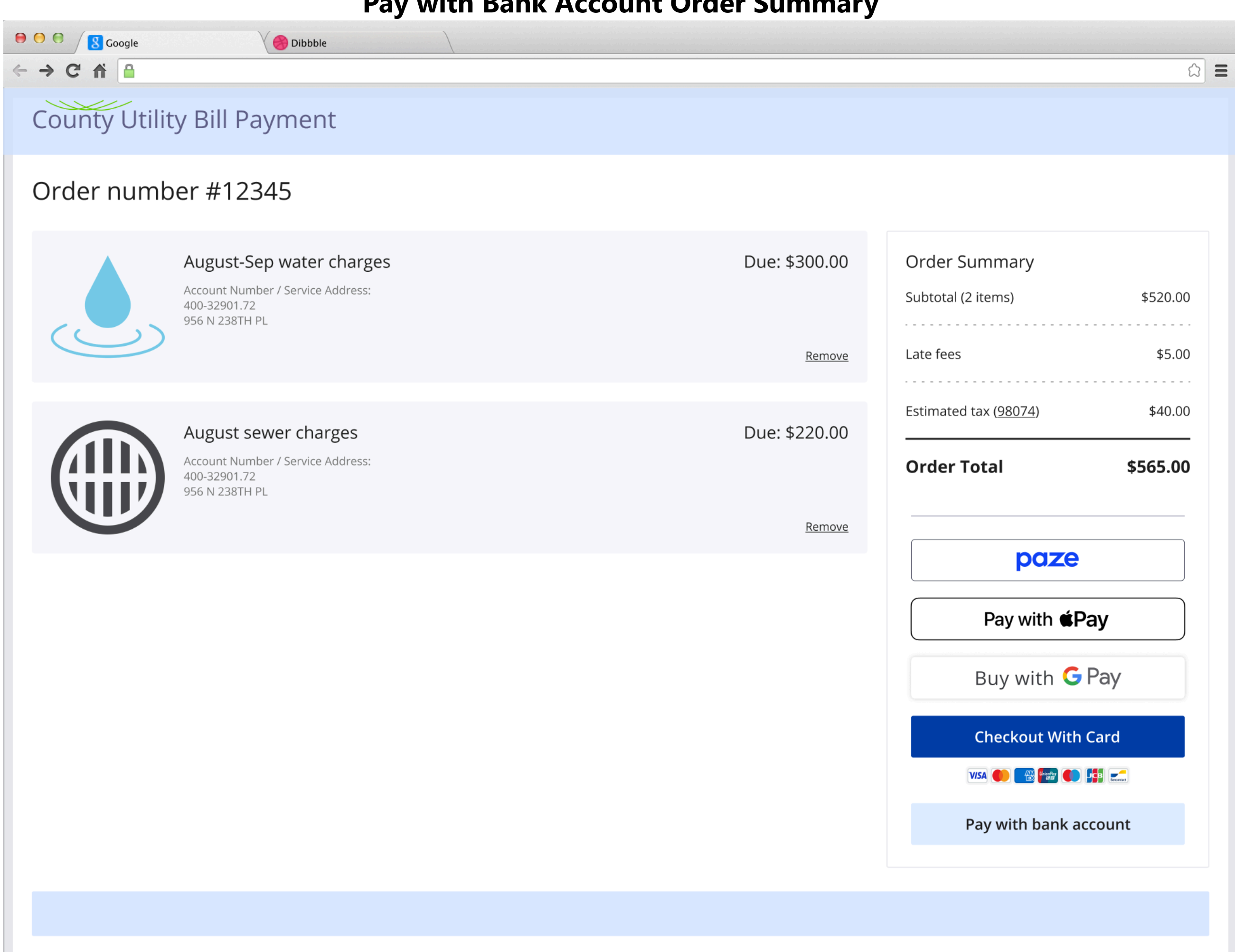
Manual Entry Review and Confirm



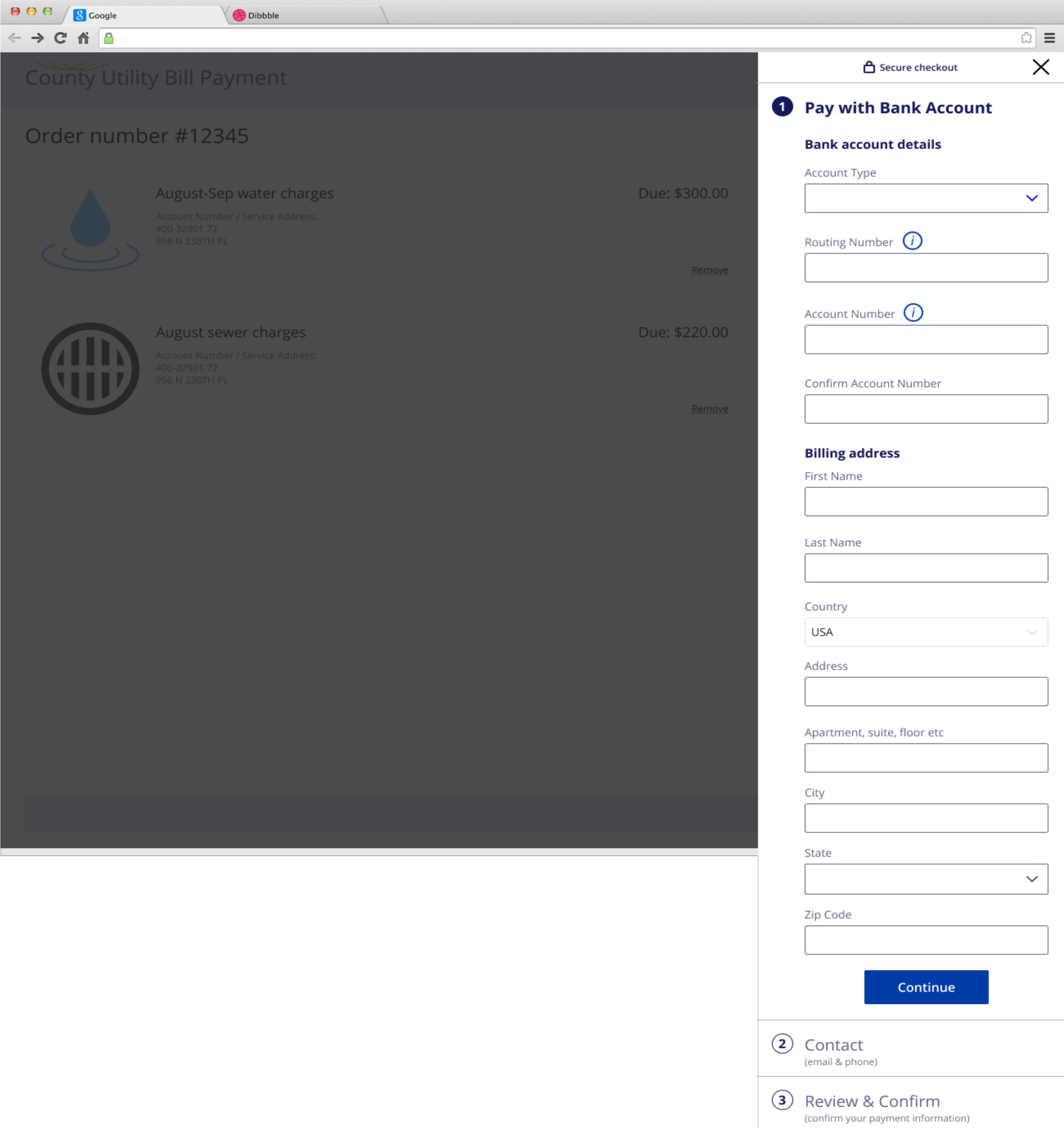
Pay with Bank Account UI

These screen captures show the sequence of events your customer can expect when completing a payment with a bank account.

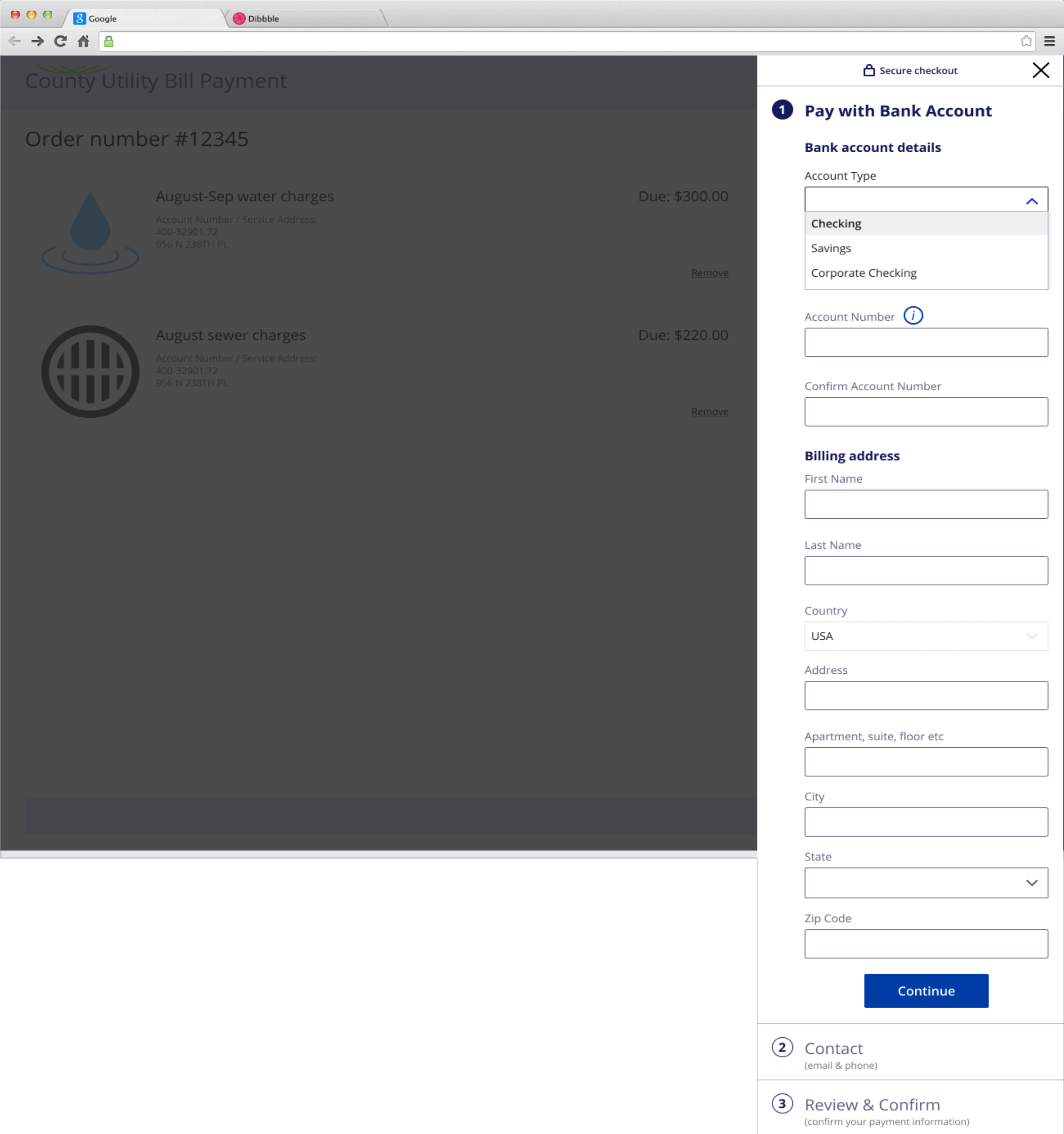
Pay with Bank Account Order Summary



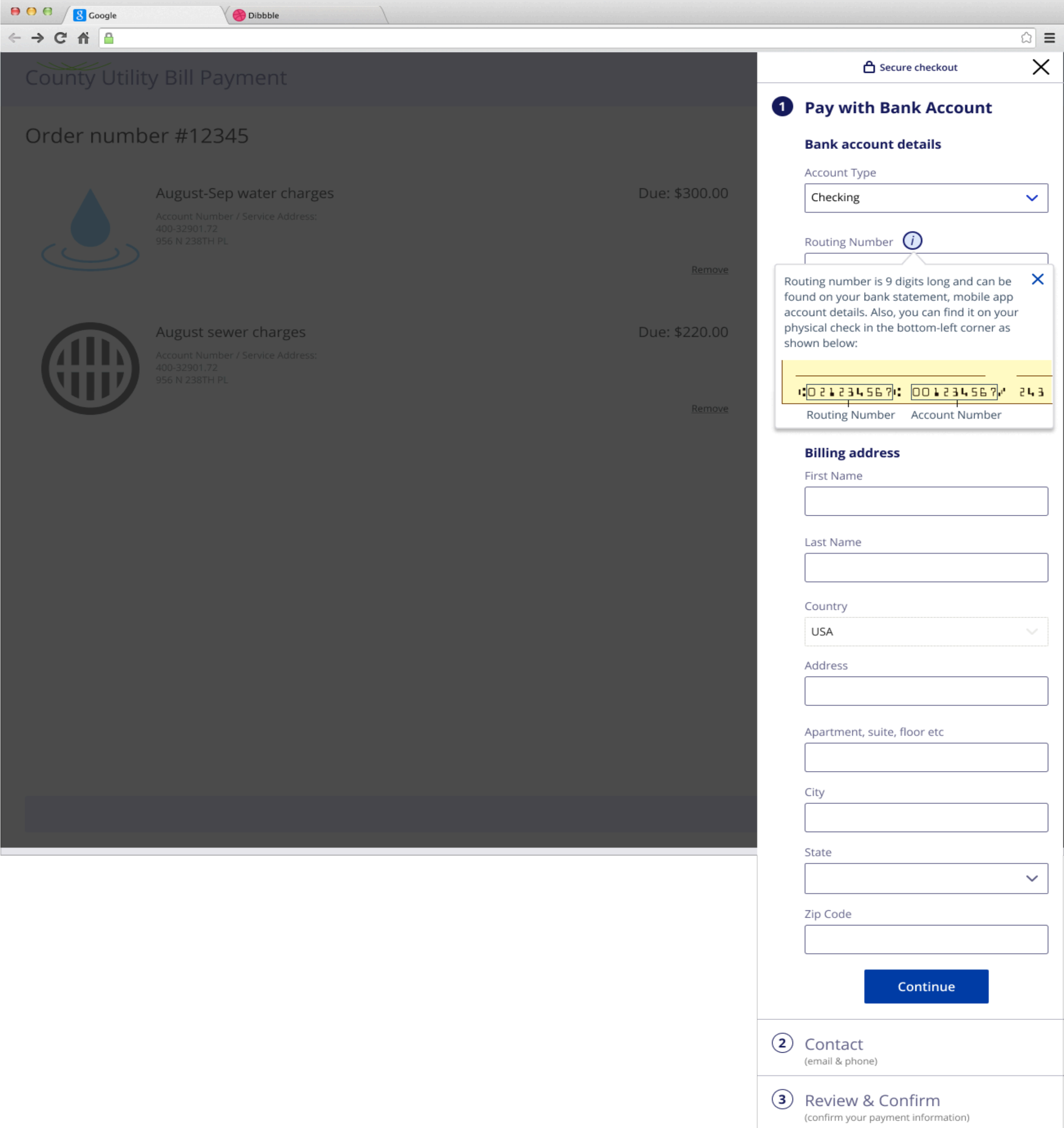
Pay with Bank Account Checkout



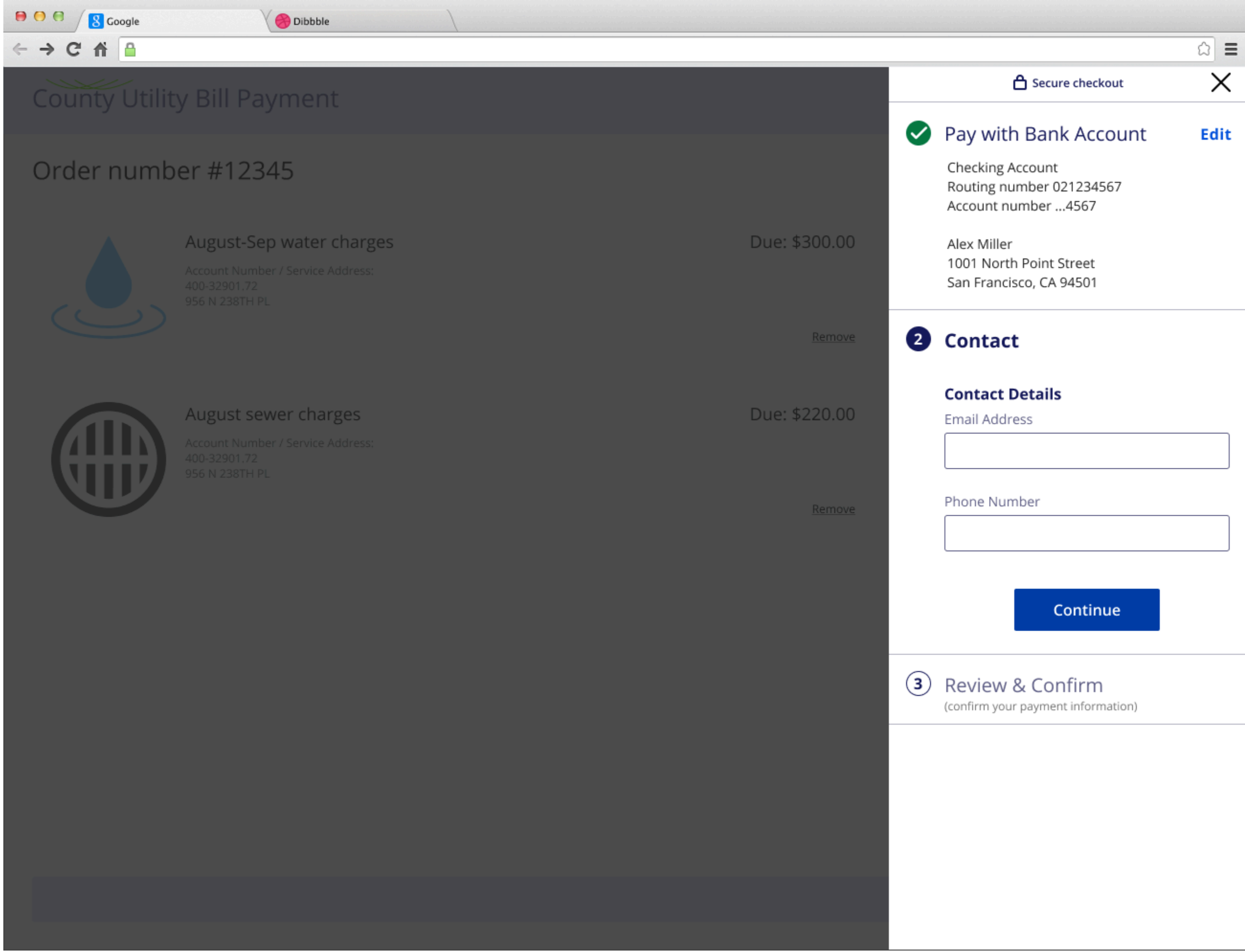
Pay with Bank Account Checking Account



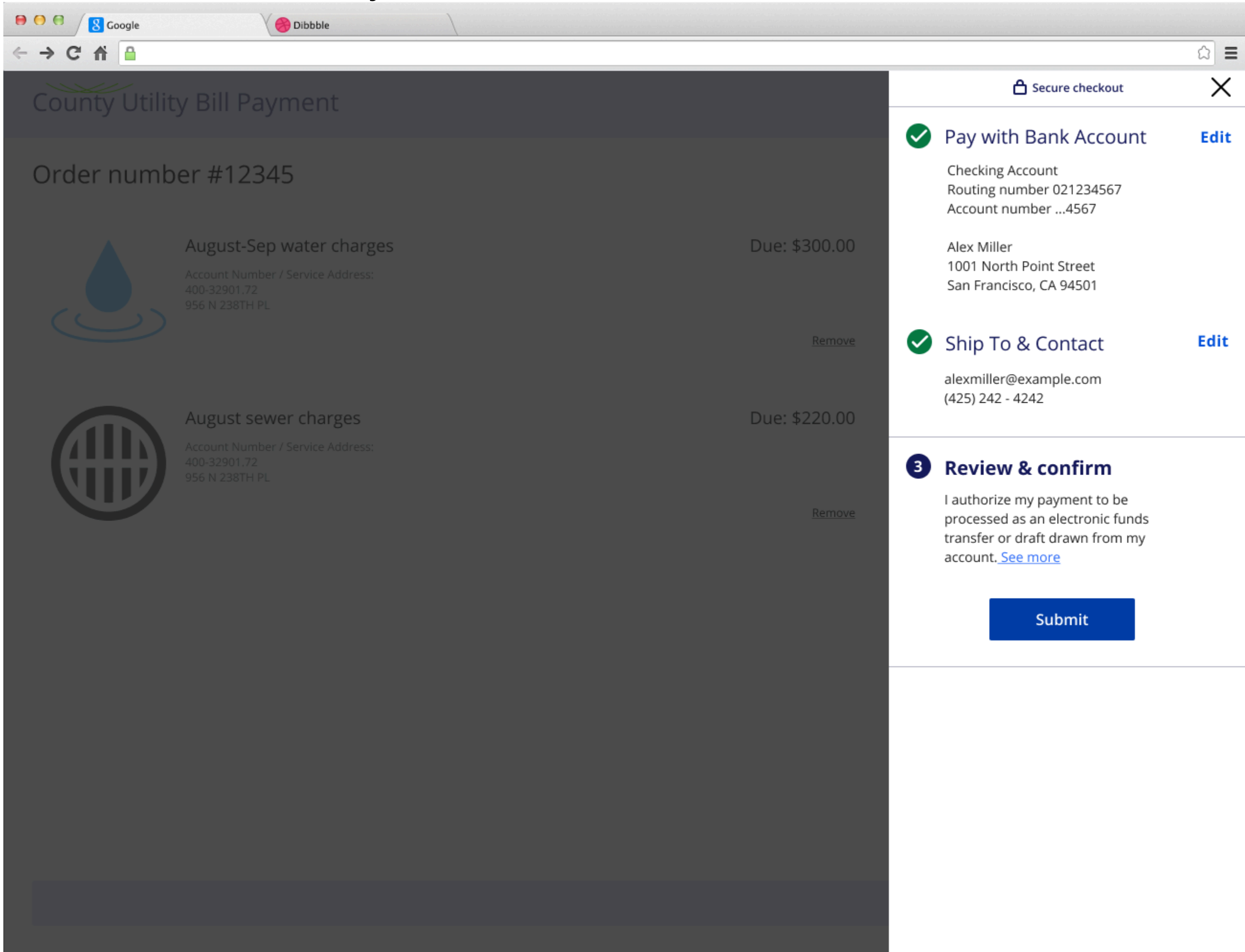
Pay with Bank Account Routing Number



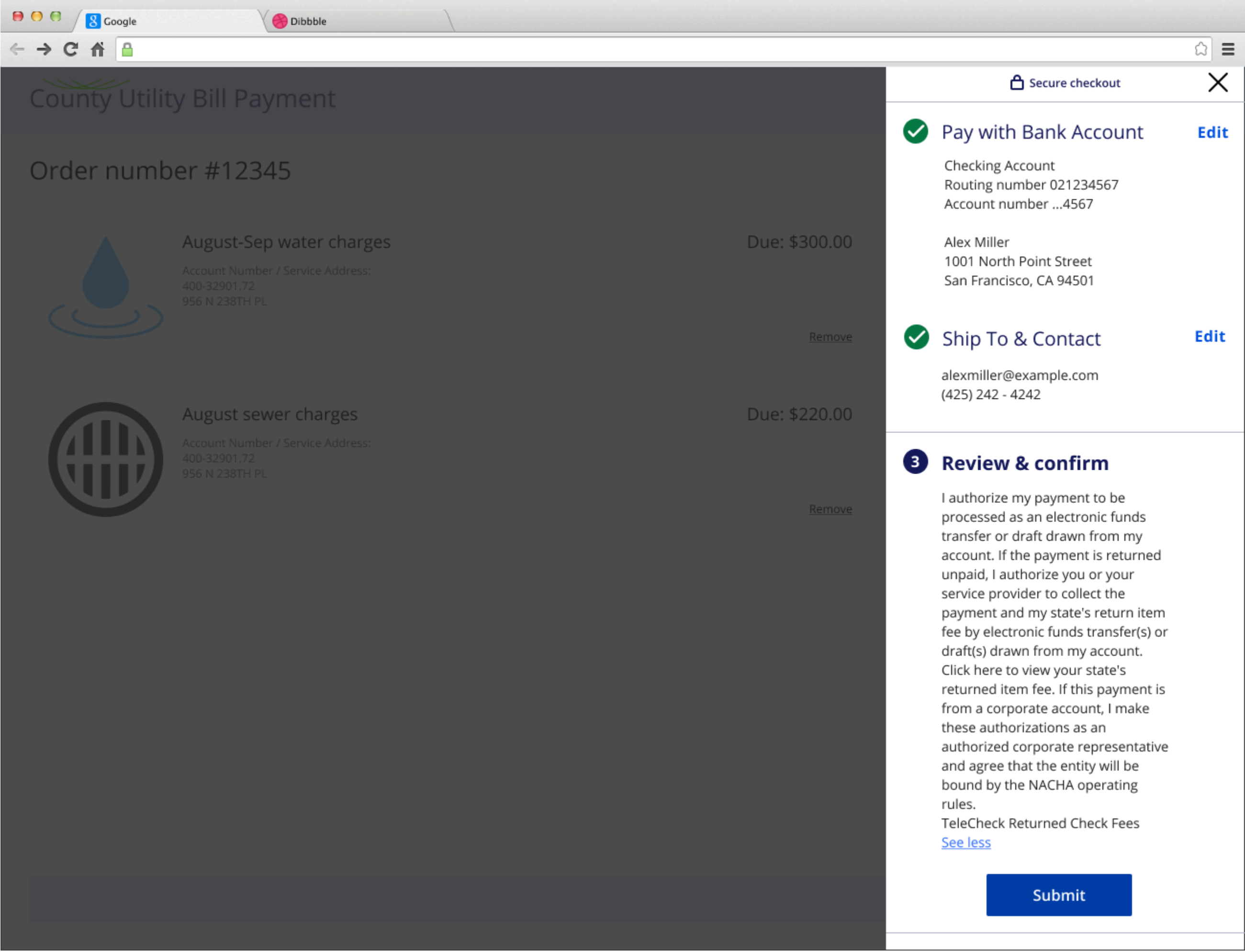
Pay with Bank Account Contact Details



Pay with Bank Account Review and Confirm



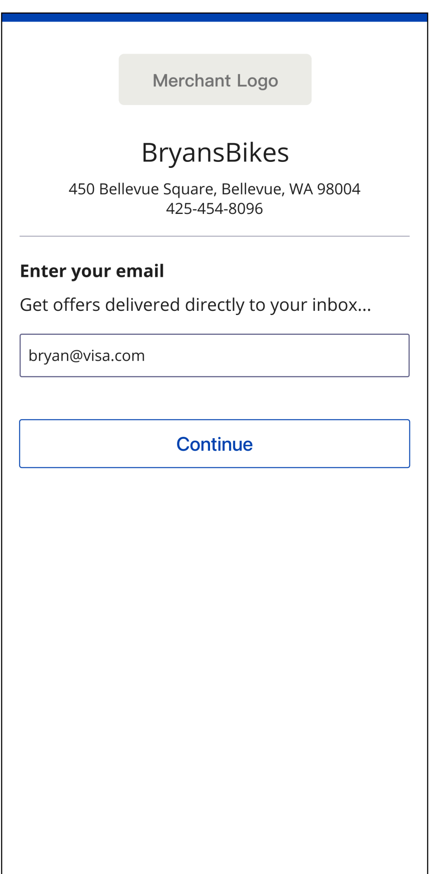
Pay with Bank Account Review and Confirm Disclaimer



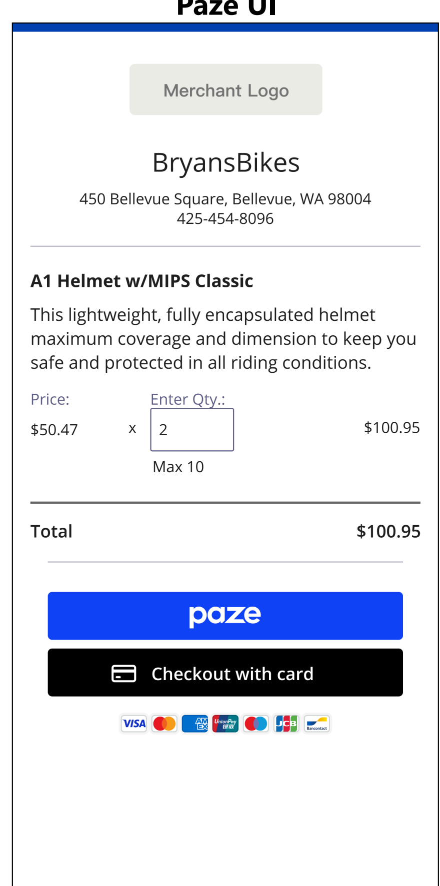
Paze UI

These screen captures show the sequence of events your customer can expect when completing a payment with Paze.

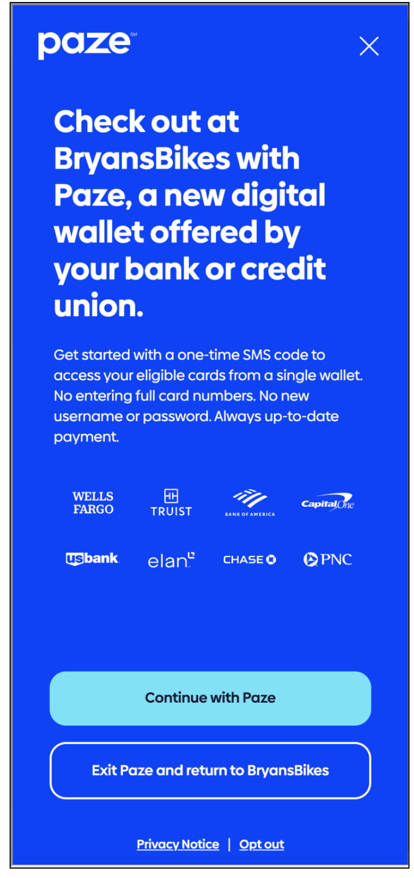
Paze UI



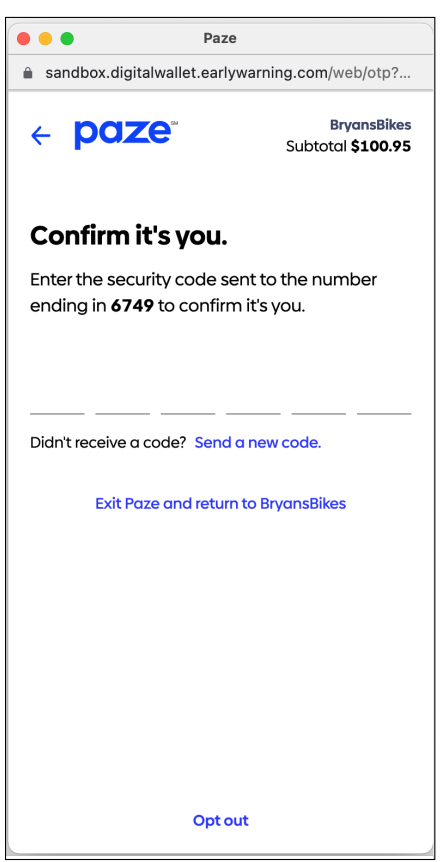
Collecting an email for Paze lookup



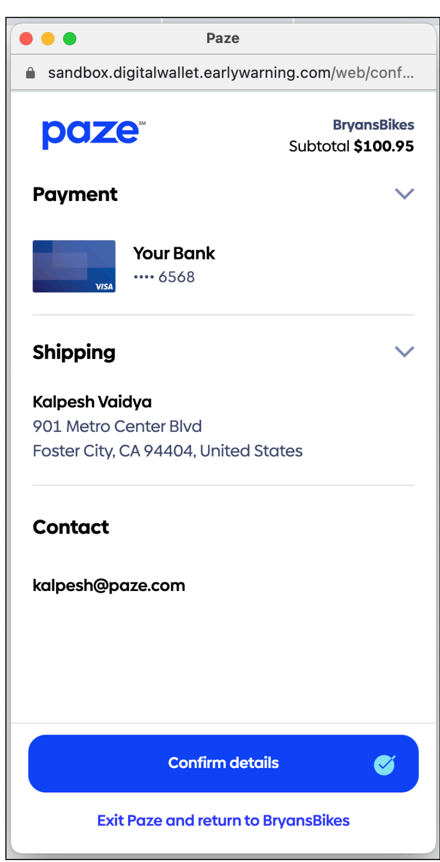
Paze in the payment method list



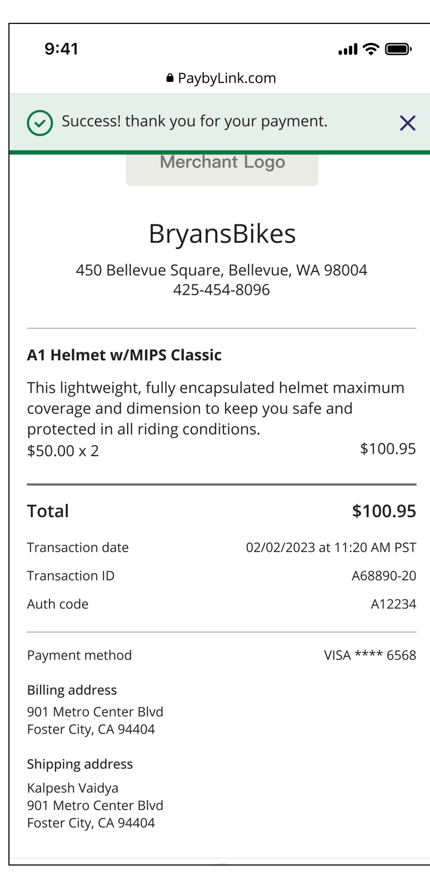
Paze explainer



Paze one-time password



Paze payment method selection



Paze payment confirmation

JSON Web Tokens

JSON Web Tokens (JWTs) are digitally signed JSON objects based on the open standard RFC 7519. These tokens provide a compact, self-contained method for securely transmitting information between parties. These tokens are signed with an RSA-encoded public/private key pair. The signature is calculated using the header and body, which enables the receiver to validate that the content has not been tampered with. Token-based applications are best for applications that use browser and mobile clients.

A JWT takes the form of a string, consisting of three parts separated by dots:

- Header
- Payload
- Signature

This example shows a JWT:

```
xxxxx.yyyyy.zzzzz
```

Supported Countries for Digital Payments

Apple Pay, Click to Pay, and Google Pay are supported in different countries. See these topics for the lists of the countries that support digital payments:

- Supported Countries for Digital Payments A-D (on page 124)
- Supported Countries for Digital Payments E-K (on page 126)
- Supported Countries for Digital Payments L-R (on page 129)
- Supported Countries for Digital Payments S-Z (on page 132)

Supported Countries for Digital Payments A-D

Supported Countries (A through D)

Country	Apple Pay	Click to Pay	Google Pay
Afghanistan	✗	✗	✓
Albania	✗	✗	✓
Algeria	✗	✗	✓
Andorra	✗	✗	✓
Angola	✗	✗	✓
Antigua and Barbuda	✗	✗	✓
Argentina	✓	✓	✓
Armenia	✓	✗	✓
Australia	✓	✓	✓
Austria	✓	✓	✓
Azerbaijan	✓	✗	✓
Bahamas	✗	✗	✓
Bahrain	✓	✗	✓
Bangladesh	✗	✗	✓

Supported Countries (A through D) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Barbados	✗	✗	✓
Belarus	✓	✗	✓
Belgium	✓	✗	✓
Brazil	✓	✓	✓
Belize	✗	✗	✓
Benin	✗	✗	✓
Bhutan	✗	✗	✗
Bolivia	✗	✗	✓
Bosnia and Herzegovina	✗	✗	✓
Botswana	✗	✗	✓
Brunei Darussalam	✗	✗	✓
Bulgaria	✓	✗	✗
Burkina Faso	✗	✗	✓
Burundi	✗	✗	✓
Cambodia	✗	✗	✓
Cameroon	✗	✗	✓
Canada	✓	✓	✓
Cape Verde	✗	✗	✓
Central African Republic	✗	✗	✓
Chad	✗	✗	✓
Chile	✓	✓	✓
China	✓	✓	✗

Supported Countries (A through D) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Colombia	✓	✓	✓
Comoros	✗	✗	✓
Costa Rica	✓	✓	✓
Côte d'Ivoire	✗	✗	✓
Croatia	✓	✗	✓
Cyprus	✓	✗	✓
Czech Republic	✓	✓	✓
Democratic Republic of the Congo	✗	✗	✓
Denmark	✓	✓	✓
Djibouti	✗	✗	✓
Dominica	✗	✗	✓
Dominican Republic	✗	✓	✓

Supported Countries for Digital Payments E-K

Supported Countries (E through K)

Country	Apple Pay	Click to Pay	Google Pay
Ecuador	✗	✓	✓
Egypt	✗	✗	✓
El Salvador	✓	✓	✓
Equatorial Guinea	✗	✗	✓
Eritrea	✗	✗	✓
Estonia	✓	✗	✓
Eswatini	✗	✗	✓

Supported Countries (E through K) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Ethiopia	✗	✗	✓
Faroe Islands	✓	✗	✗
Fiji	✗	✗	✓
Finland	✓	✓	✓
France	✓	✓	✓
Gabon	✗	✗	✓
Gambia	✗	✗	✓
Georgia	✓	✗	✓
Germany	✓	✓	✓
Ghana	✗	✗	✓
Gibraltar	✗	✗	✗
Greece	✓	✗	✓
Greenland	✓	✗	✗
Guernsey	✓	✗	✗
Grenada	✗	✗	✓
Guatemala	✓	✓	✓
Guinea	✗	✗	✓
Guinea-Bissau	✗	✗	✓
Guyana	✗	✗	✓
Haiti	✗	✗	✓
Honduras	✓	✓	✓
Hong Kong	✓	✓	✓
Hungary	✓	✓	✓

Supported Countries (E through K) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Iceland	✓	✗	✓
India	✗	✓	✓
Indonesia	✗	✓	✗
Iraq	✗	✗	✓
Ireland	✓	✓	✓
Isle of Man	✓	✗	✗
Israel	✓	✗	✓
Italy	✓	✓	✓
Jamaica	✗	✗	✓
Japan	✓	✗	✓
Jersey	✓	✗	✗
Jordan	✓	✓	✓
Kazakhstan	✓	✗	✓
Kenya	✗	✗	✓
Kiribati	✗	✗	✓
Kuwait	✓	✓	✓
Kyrgyzstan	✗	✗	✓

Supported Countries for Digital Payments L-R

Supported Countries (L through R)

Country	Apple Pay	Click to Pay	Google Pay
Laos	✗	✗	✓
Latvia	✓	✗	✓
Lebanon	✗	✗	✓
Lesotho	✗	✗	✓
Liberia	✗	✗	✓
Libya	✗	✗	✓
Liechtenstein	✓	✗	✓
Lithuania	✓	✗	✓
Luxembourg	✓	✗	✓
Macau	✓	✗	✓
Madagascar	✗	✗	✓
Malawi	✗	✗	✓
Malaysia	✓	✓	✓
Maldives	✗	✗	✓
Mali	✗	✗	✓
Malta	✓	✗	✓
Marshall Islands	✗	✗	✓
Mauritania	✗	✗	✓
Mauritius	✗	✗	✓
Mexico	✓	✓	✓
Micronesia, Federated States of	✗	✗	✓
Moldova	✓	✗	✓

Supported Countries (L through R) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Monaco	Yes	No	Yes
Mongolia	No	No	Yes
Montenegro	Yes	No	Yes
Morocco	No	No	Yes
Mozambique	No	No	Yes
Myanmar	No	No	Yes
Namibia	No	No	Yes
Nauru	No	No	Yes
Nepal	No	No	Yes
Netherlands	Yes	Yes	Yes
New Zealand	Yes	Yes	Yes
Nicaragua	No	Yes	Yes
Niger	No	No	Yes
Nigeria	No	No	Yes
North Macedonia	No	No	Yes
Norway	Yes	Yes	Yes
Oman	No	No	Yes
Pakistan	No	No	Yes
Palau	No	No	Yes
Palestinian Territories	Yes	No	Yes
Panama	Yes	Yes	Yes
Papua New Guinea	No	No	Yes
Paraguay	No	Yes	Yes

Supported Countries (L through R) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Peru	✓	✓	✓
Philippines	✗	✗	✗
Poland	✓	✓	✓
Portugal	✓	✗	✓
Qatar	✓	✓	✓
Republic of the Congo	✗	✗	✓
Romania	✓	✗	✓
Rwanda	✗	✗	✓

Supported Countries for Digital Payments S-Z

Supported Countries (S through Z)

Country	Apple Pay	Click to Pay	Google Pay
Saint Kitts and Nevis	✗	✗	✓
Saint Lucia	✗	✗	✓
Saint Vincent and the Grenadines	✗	✗	✓
Samoa	✗	✗	✓
San Marino	✓	✗	✓
Sao Tome and Principe	✗	✗	✓
Saudi Arabia	✓	✓	✓
Senegal	✗	✗	✓
Serbia	✓	✗	✓
Seychelles	✗	✗	✓
Sierra Leone	✗	✗	✓
Singapore	✓	✓	✓
Slovakia	✓	✓	✓
Slovenia	✓	✗	✓
Solomon Islands	✗	✗	✓
Somalia	✗	✗	✓
South Africa	✓	✓	✓
Korea, Republic of (South)	✓	✗	✓
South Sudan	✗	✗	✓
Spain	✓	✓	✓
Sri Lanka	✗	✗	✓

Supported Countries (S through Z) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Sudan	✗	✗	✓
Suriname	✗	✗	✓
Sweden	✓	✓	✓
Switzerland	✓	✓	✓
Switzerland -Italian	✗	✗	✗
Taiwan	✓	✗	✓
Tajikistan	✗	✗	✓
Tanzania	✗	✗	✓
Thailand	✗	✗	✗
Timor-Leste	✗	✗	✓
Togo	✗	✗	✓
Tonga	✗	✗	✓
Trinidad and Tobago	✗	✗	✓
Tunisia	✗	✗	✓
Turkey	✗	✗	✓
Turkmenistan	✗	✗	✓
Tuvalu	✗	✗	✓
Uganda	✗	✗	✓
Ukraine	✓	✓	✓
United Arab Emirates	✓	✓	✓
United Kingdom	✓	✓	✓
United States	✓	✓	✓
Uruguay	✗	✓	✓

Supported Countries (S through Z) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Uzbekistan	✗	✗	✓
Vanuatu	✗	✗	✓
Vatican City (Holy See)	✓	✗	✓
Venezuela	✗	✗	✓
Vietnam	✓	✗	✓
Yemen	✗	✗	✓
Zambia	✗	✗	✓
Zimbabwe	✗	✗	✓

Supported Locales

The locale field within the capture context request consists of an ISO 639 language code, an underscore (_), and an ISO 3166 region code. The locale controls the language in which the application is rendered. The following locales are supported:

- ar_AE
- ca_ES
- cs_CZ
- da_DK
- de_AT
- de_DE
- el_GR
- en_AU
- en_CA
- en_GB
- en_IE
- en_NZ
- en_US
- es_AR
- es_CL
- es_CO
- es_ES
- es_MX
- es_PE
- es_US
- fi_FI
- fr_CA
- fr_FR

- he_IL
- hr_HR
- hu_HU
- id_ID
- it_IT
- ja_JP
- km_KH
- ko_KR
- lo_LA
- ms_MY
- nb_NO
- nl_NL
- pl_PL
- pt_BR
- ru_RU
- sk_SK
- sv_SE
- th_TH
- tl_PH
- tr_TR
- vi_VN
- zh_CN
- zh_HK
- zh_MO
- zh_SG
- zh_TW

Reason Codes

A Unified Checkout request response returns one of the following reason codes:

Reason Codes

Reason Code	Description
200	Successful response.
201	Capture context created.
400	Bad request. Possible **reason** values: • CAPTURE_CONTEXT_EXPIRED • CAPTURE_CONTEXT_INVALID • CREATE_TOKEN_TIMEOUT • CREATE_TOKEN_XHR_ERROR • INVALID_APIKEY • SDK_XHR_ERROR • SHOW_LOAD_CONTAINER_SELECTOR • SHOW_LOAD_INVALID_CONTAINER • SHOW_PAYMENT_TIMEOUT • SHOW_TOKEN_TIMEOUT • SHOW_TOKEN_XHR_ERROR • UNIFIEDPAYMENT_PAYMENT_PARAMITERS • UNIFIEDPAYMENTS_VALIDATION_FIELDS • UNIFIEDPAYMENTS_VALIDATION_PARAMS
404	The specified resource not found in the system.
500	Unexpected server error.

Click to Pay Drop-In UI

The Click to Pay Drop-In UI powered by Unified Checkout provides an interface for easy acceptance of Click to Pay payments from Visa, Mastercard, and American Express cards. Throughout this guide we refer to both *Click to Pay Drop-In UI* and *Unified Checkout*.

Click to Pay Drop-In UI consists of a server-side component and a client-side JavaScript library.

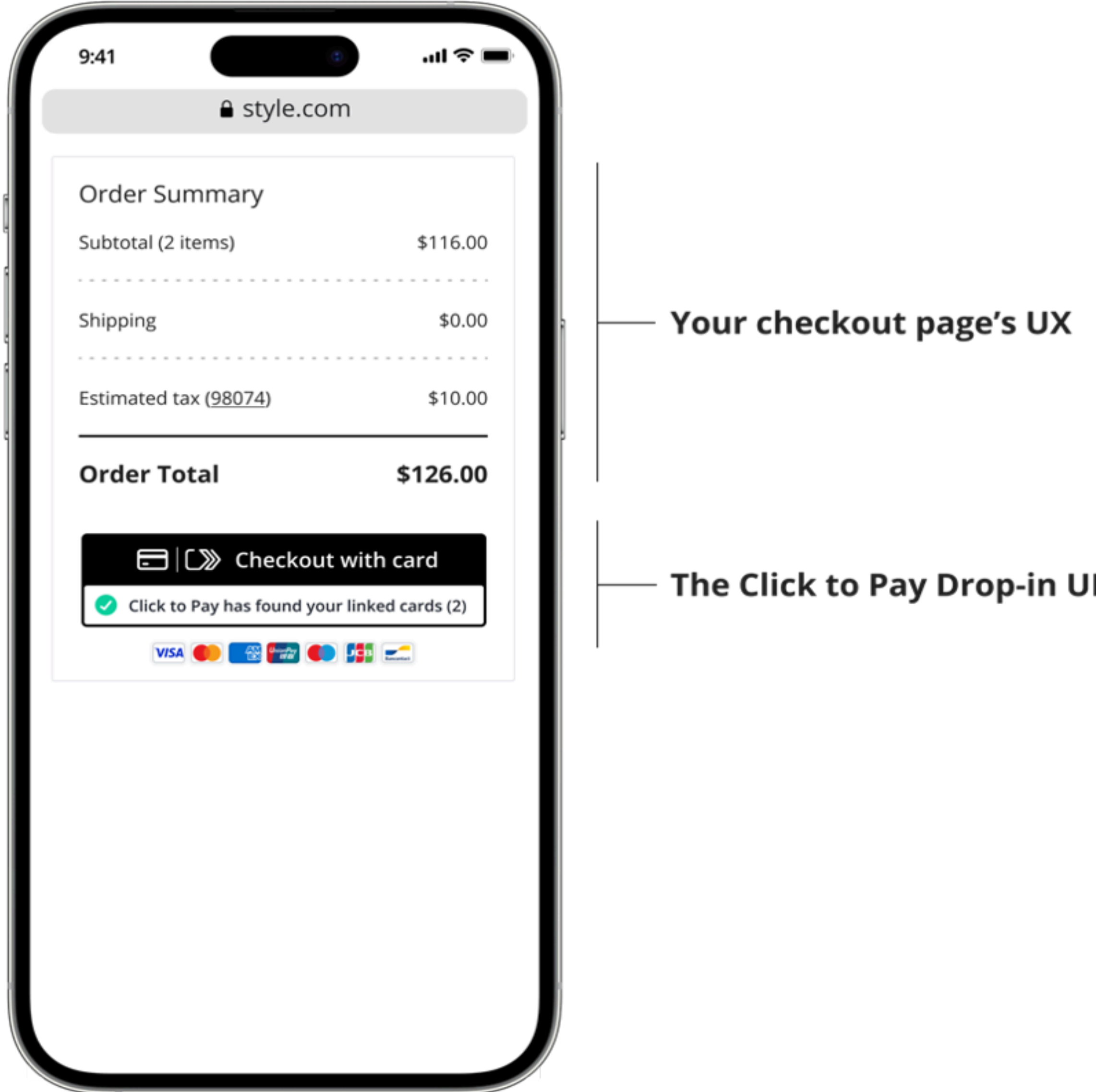
The server-side component authenticates your merchant identity and instructs the system to act within your payment environment. The response contains limited-use public keys. The keys are used for end-to-end encryption and contain merchant-specific payment information that drives the interaction of the application. The client-side JavaScript library dynamically and securely places digital payment options into your e-commerce page.

The provided JavaScript library enables you to place a payment application within your e-commerce environment. This embedded component offers Click to Pay and card entry to your customers.

Whether a customer uses a stored Click to Pay card or enters their payment information manually, the Click to Pay Drop-In UI handles all user interactions and provides a response to your e-commerce system.

The figure below shows the Click to Pay Drop-In UI for a recognized user.

Embedded Component



Click to Pay Customer Workflows

This section provides an overview of the Click to Pay Drop-In UI user experience. The Click to Pay Drop-In UI is designed to provide customers with a friction-free payment experience across many payment experiences. The user experience has been optimized for mobile use and performs equally well on mobile and desktop devices. Click to Pay recognizes customers as follows:

- The customer is a recognized Click to Pay customer.
- The customer is not recognized but is a Click to Pay customer.
- The customer is a guest at checkout.

These workflows show you the pages a customer encounters based on their status:

- Recognized Click to Pay Customer (on page 140)
- Unrecognized Click to Pay Customer (on page 142)
- Guest Customer (on page 144)

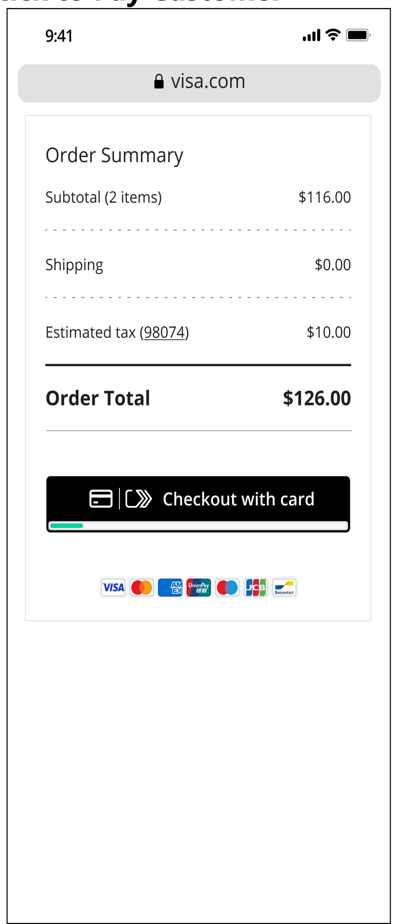
Recognized Click to Pay Customer

This section provides an overview of the Click to Pay Drop-In UI recognized experience. This interaction occurs when a customer’s device is recognized by the Click to Pay Drop-In UI.

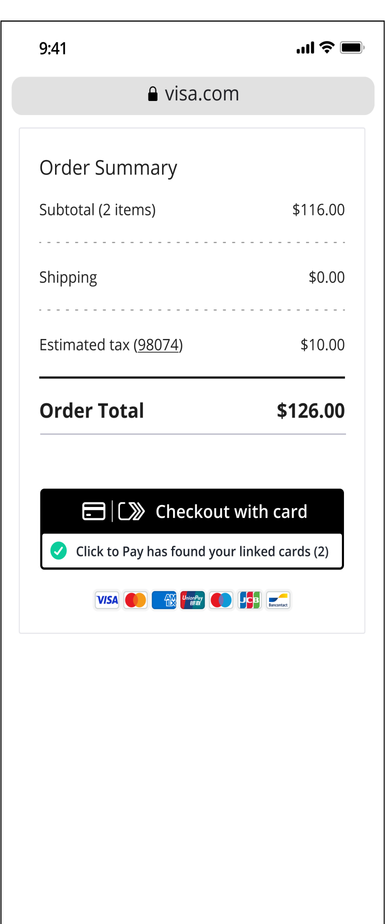
A customer's device is recognized under these conditions:

- When the customer has used Click to Pay on their device through any Click to Pay channel.
- If the customer chose to have their device remembered during a previous transaction.

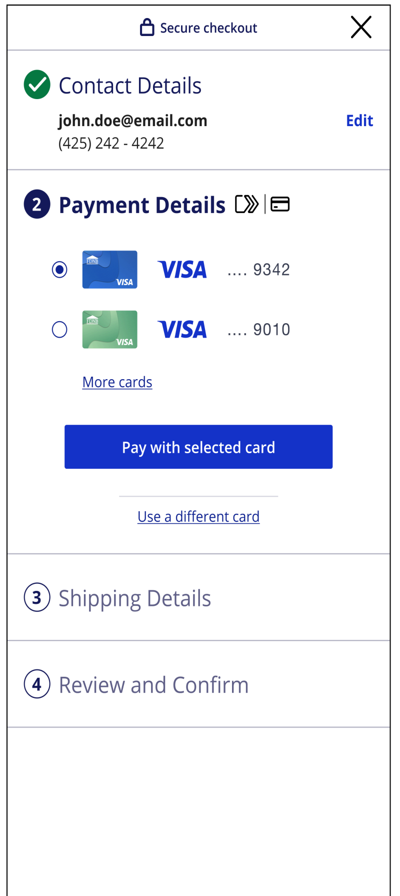
Recognized Click to Pay Customer



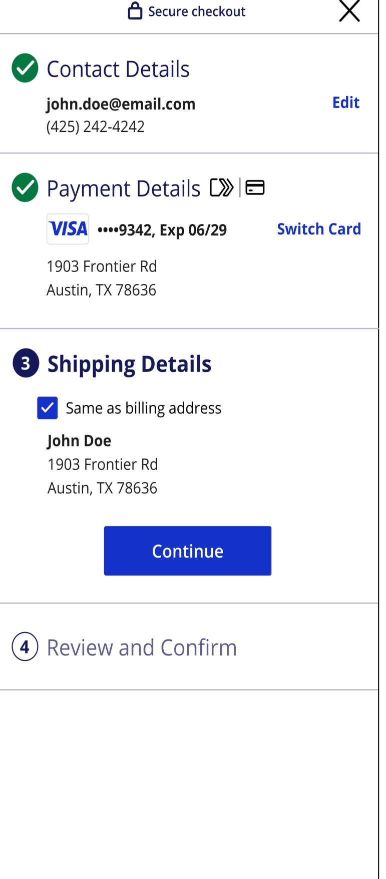
Click to Pay loader animation



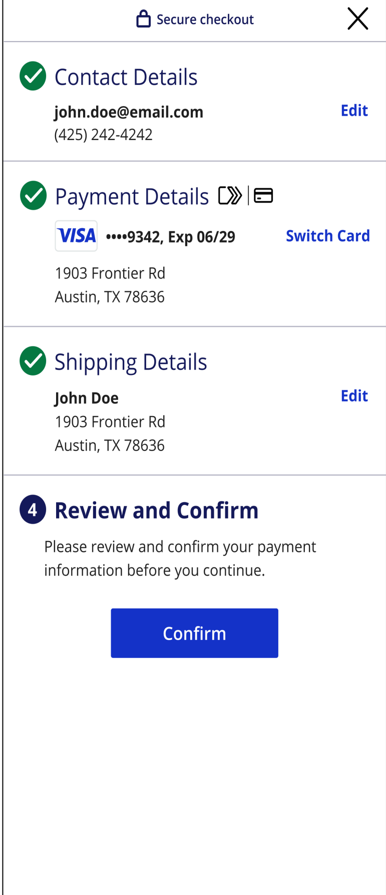
Click to Pay recognized user



Click to Pay saved cards



Click to Pay saved cards

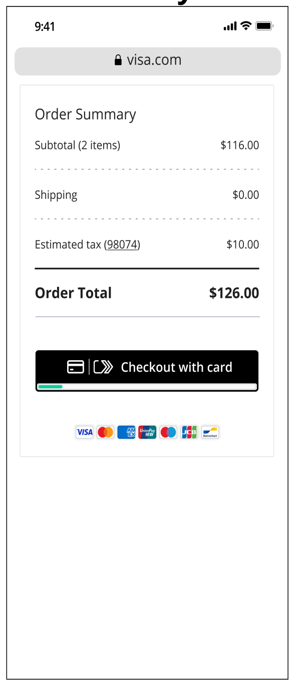


Review screen

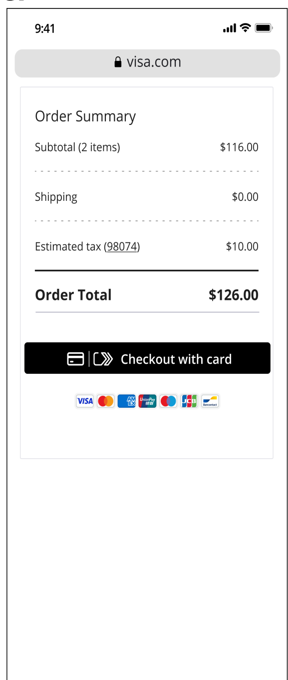
Unrecognized Click to Pay Customer

This section provides an overview of the Click to Pay Drop-In UI unrecognized experience. This interaction occurs when a customer's device is not recognized by the Click to Pay Drop-In UI. This condition occurs when the customer has a Click to Pay account but has not used it on their device previously.

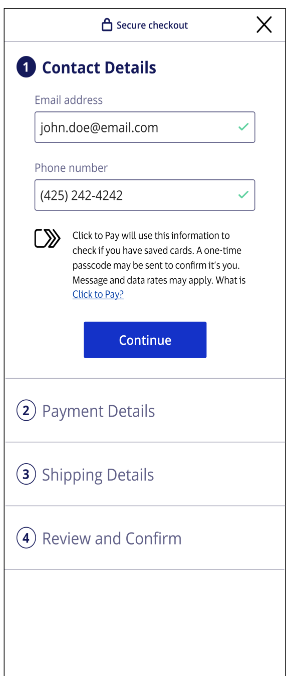
Unrecognized Click to Pay Customer



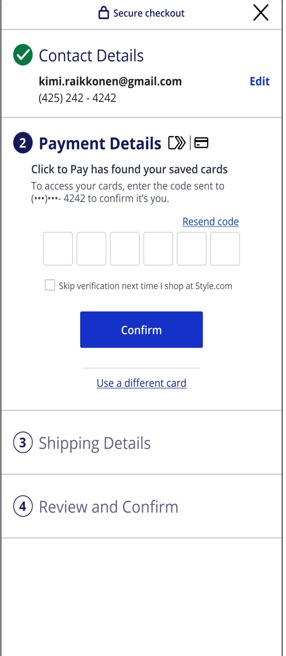
Click to Pay loader animation



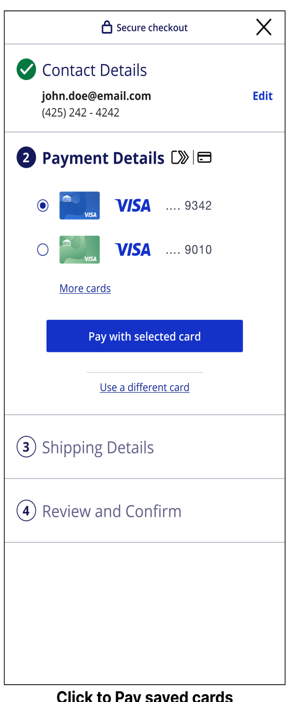
Click to Pay unrecognized user



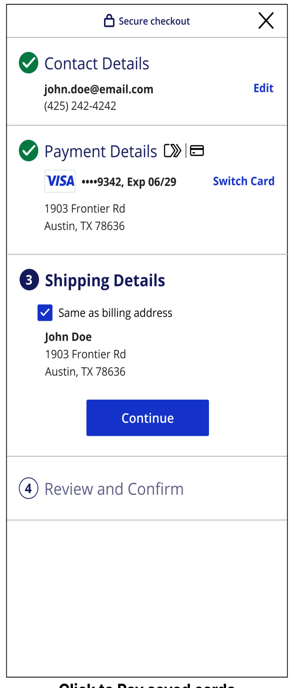
Identity lookup based on email provided



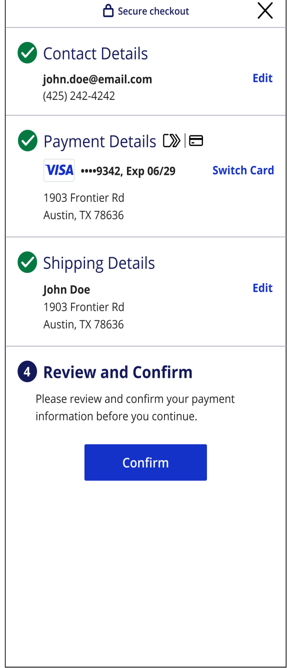
One-time password



Click to Pay saved cards



Click to Pay saved cards

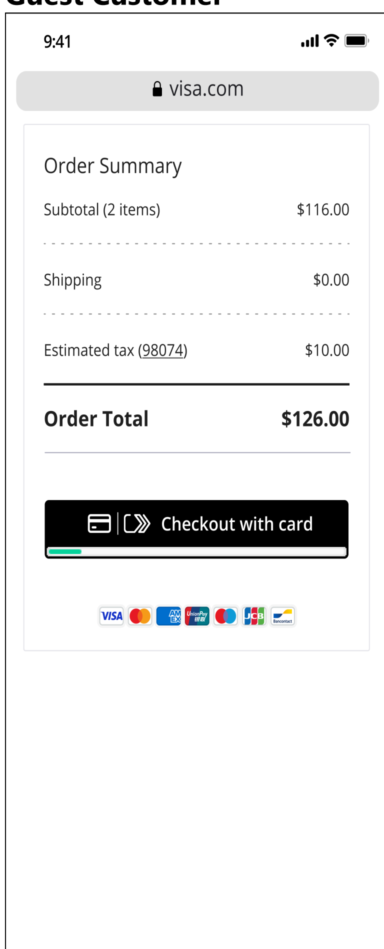


Review screen

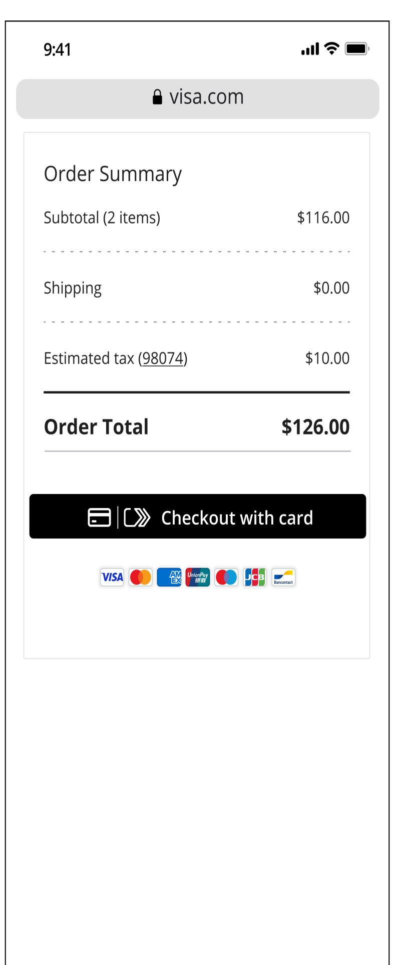
Guest Customer

This section provides an overview of the Click to Pay Drop-In UI guest experience. This interaction occurs when the customer has not created a Click to Pay account, or their issuer has not provisioned their card into Click to Pay.

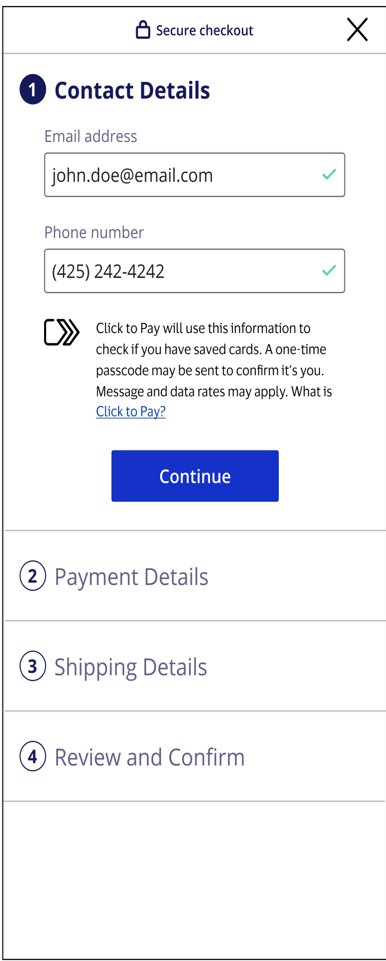
Guest Customer



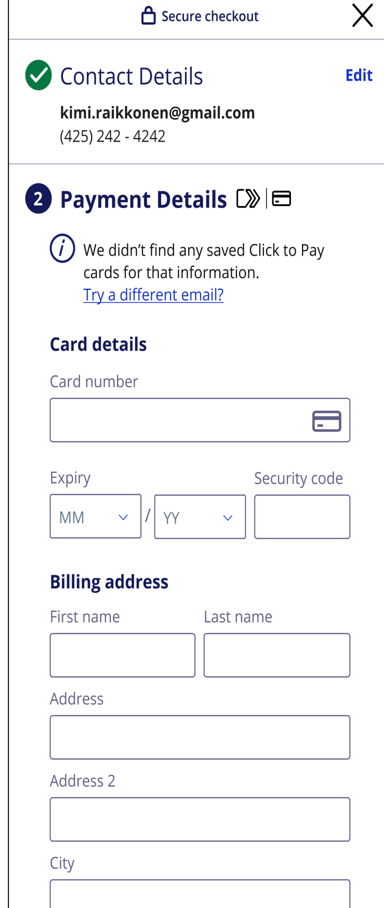
Click to Pay loader animation

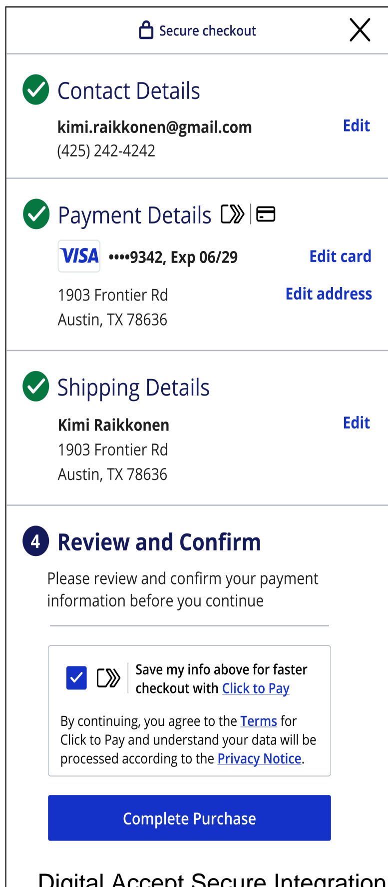


Click to Pay recognized user



Identity lookup based on email provided





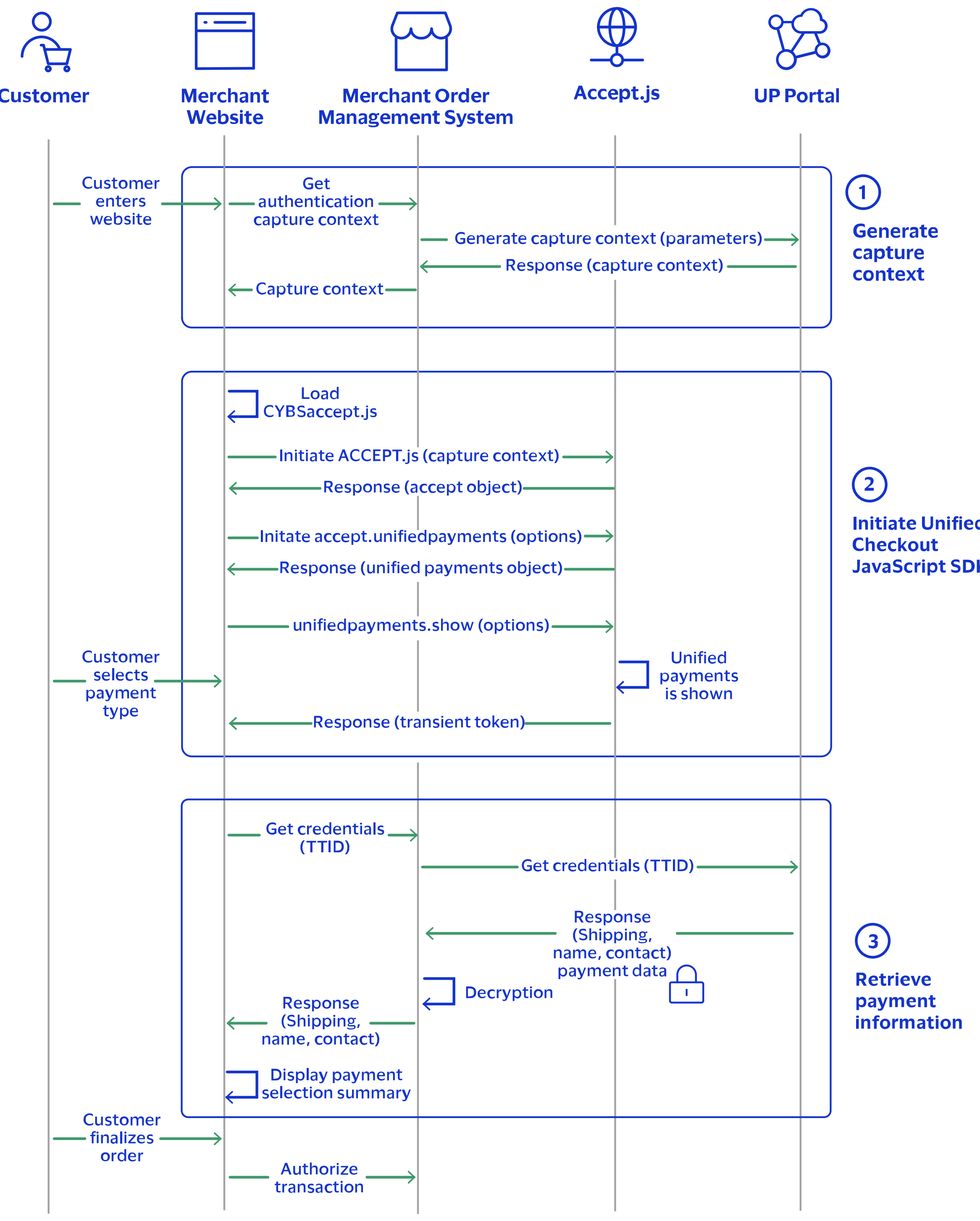
Click to Pay Drop-In UI Flow

To integrate Unified Checkout into your platform, you must follow several integration steps. This section gives a high-level overview of how to integrate and launch Unified Checkout on your webpage and process a transaction using the data that Unified Checkout collects for you. You can find the detailed specifications of the APIs later in this document.

1. You send a server-to-server API request for a capture context. This request is fully authenticated and returns a JSON Web Token (JWT) that is necessary to invoke the frontend JavaScript library. For information on setting up the server side, see Server-Side Set Up (on page 149).
2. You invoke the Unified Checkout JavaScript library using the JWT response from the capture context request. For information on setting up the client side, see Client-Side Set Up (on page 81).
3. You use the response from the Click to Pay Drop-In UI to retrieve payment credentials for payment processing or other steps.

The figure below illustrates the system's payment flow.

Click to Pay Payment Flow



For more information on the specific APIs referenced, see these topics:

- Capture Context API (on page 156)
- Payment Details API (on page 97)
- Payment Credentials API (on page 167)

Enabling Unified Checkout in the Business Center

To begin using the Click to Pay Drop-In UI powered by Unified Checkout, you must first ensure that your merchant ID (MID) is configured to use the service and that Click to Pay is properly set up.

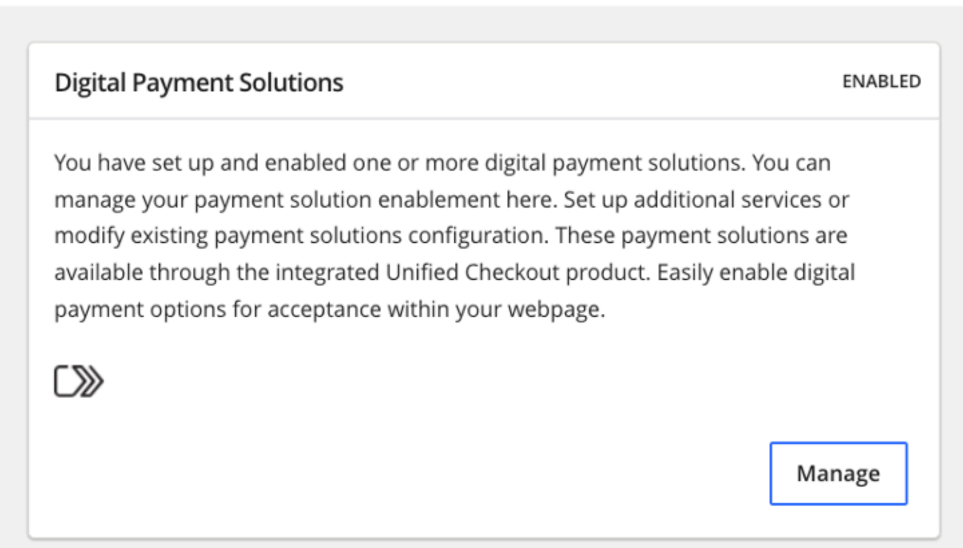
1. Log in to the Business Center:
   Test URL: `https://businesscentertest.cybersource.com/ebc2`
   Production URL: `https://businesscenter.cybersource.com`

2. In the Business Center, go to the left navigation panel and choose **Payment Configuration** > **Unified Checkout**.

3. Click **Setup** and follow the instructions to enroll your business in Click to Pay. When Click to Pay is enabled, it appears on the payment configuration page.

Payment Configuration

Unified Checkout



4. Click **Manage** to alter your Click to Pay enrollment details. For more information on registering for Click to Pay, see Enable Click to Pay (on page 175).

Server-Side Set Up

This section contains the information you need to set up your server. Initializing Unified Checkout within your webpage begins with a server-to-server call to the sessions API. This step authenticates your merchant credentials, and establishes how the Unified Checkout frontend components will function. The sessions API request contains parameters that define how Unified Checkout performs.

The server-side component provides this information:

- A transaction-specific public key is used by the customer's browser to protect the transaction.
- An authenticated context description package that manages the payment experience on the client side. It includes available payment options such as card networks, payment interface styling, and interaction methods.

The functions are compiled in a JSON Web Token (JWT) object referred to as the *capture context*. For information JSON Web Tokens, see JSON Web Tokens (on page 123).

Capture Context

The capture context request is a signed JSON Web Token (JWT) that includes all of the merchant-specific parameters. This request tells the frontend JavaScript library how to behave within your payment experience. For information on JSON Web Tokens, see JSON Web Tokens (on page 123).

You can define the payment cards and digital payments that you want to accept in the capture context. Use the **allowedCardNetworks** field to define the card types.

Available card networks for card entry:

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

Important: Click to Pay supports American Express, Mastercard, and Visa for saved cards.

Use the **allowedPaymentTypes** field to define the digital payment methods.

Example:

```
{
          "targetOrigins" : [ "https://www.test.com" ],
          "clientVersion" : "0.19",
          "allowedCardNetworks" : [ "VISA", "MASTERCARD", "AMEX" ],
          "allowedPaymentTypes" : ["CLICKTOPAY"],
          "country" : "US",
          "locale" : "en_US",
          "captureMandate" : {
          "billingType" : "FULL",
          "requestEmail" : true,
          "requestPhone" : true,
          "requestShipping" : true,
          "shipToCountries" : [ "US", "GB" ],
          "showAcceptedNetworkIcons" : true
          },
          "orderInformation" : {
          "amountDetails" : {
          "totalAmount" : "1.01",
          "currency" : "USD"
          },
          }
          }
```

This diagram shows how elements of the capture context request appear in the card entry form.

Anatomy of a Manual Card Entry Form


```json
{
  "targetOrigins" : [ "https://the-up-demo.appspot.com" ],
  "clientVersion" : "0.19",
  "allowedCardNetworks" : [ "VISA", "MASTERCARD", "AMEX" ],
  "allowedPaymentTypes" : [ "CLICKTOPAY" ],
  "country" : "US",
  "locale" : "en_US",
  "captureMandate" : {
    "billingType" : "FULL",
    "requestEmail" : true,
    "requestPhone" : true,
    "requestShipping" : true,
    "shipToCountries" : [ "US", "GB" ],
    "showAcceptedNetworkIcons" : true
  },
  "orderInformation" : {
    "amountDetails" : {
      "totalAmount" : "1.01",
      "currency" : "USD"
    },
    "billTo" : {
      "address1" : "277 Park Avenue",
      "administrativeArea" : "NY",
      "buildingNumber" : "#218",
      "country" : "US",
      "district" : "district",
      "locality" : "New York",
      "postalCode" : "10172",
      "email" : "john.doe@visa.com",
      "firstName" : "John",
      "lastName" : "Doe",
      "middleName" : "F",
      "nameSuffix" : "Jr",
      "title" : "Mr",
      "phoneNumber" : "1234567890",
      "phoneType" : "phoneType"
    },
    "shipTo" : {
      "address1" : "123 Cool St",
      "administrativeArea" : "CA",
      "buildingNumber" : "#12",
      "country" : "US",
      "district" : "string",
      "locality" : "Beverly Hills",
      "postalCode" : "90210",
      "firstName" : "Joe",
      "lastName" : "Soap"
    }
  }
}
```


Secure checkout

1 Contact Details

Email address
john.doe@visa.com

Phone number
1234567890

Click to Pay will use this information to check if you have saved cards. A one-time passcode may be sent to confirm it's you. Message and data rates may apply. What is Click to Pay?

Continue

2 Payment Details

3 Shipping Details

3 Review and Confirm

Secure checkout

Contact Details Edit
john.doe@visa.com
1234567890

2 Payment Details

Card details

Card number

Expiry MM / YY Security code

Billing address

First name John Last name Doe

Address 277 Park Avenue

Address 2 #218

City New York

State New York

Zip code 10172

Country USA

Continue

3 Shipping Details

4 Review and Confirm

Secure checkout

Contact Details
john.doe@visa.com Edit
1234567890

Payment Details
VISA ****9342, Exp 06/29 Edit
227 Park Avenue #218
New York, NY 10172

3 Shipping Details

Same as billing address

First name Joe Last name Soap

Address 123 Cool St

Address 2 #12

City Beverly Hills

State California

Zip code 90210

Country USA

Continue

4 Review and Confirm

Secure Checkout

Contact Details Edit
john.doe@visa.com
1234567890

Payment Details
VISA ****9342, Exp 06/29 Edit
227 Park Avenue #218
New York, NY 10172

Shipping details Edit
Joe Soap
123 Cool St #12
Beverly Hills, CA 90210

4 Review & Confirm

Please review and confirm your payment information before you continue.

Save my info above for faster checkout with Click to Pay

By continuing, you agree to the Terms for Click to Pay and understand your data will be processed according to the Privacy Notice.

Complete Purchase

For more information on requesting the capture context, see Capture Context API (on page 156).

Client-Side Set Up

This section contains the information you need to set up the client side. You use the Unified Checkout JavaScript library to integrate with your e-commerce website. It has two primary components:

- The button widget, which lists the payment methods available to the customer.
- The payment acceptance page, which captures payment information from the cardholder. You can integrate the payment acceptance page with your webpage or add it as a sidebar.

The Unified Checkout JavaScript library supports Click to Pay and manual card entry payment methods.

Follow these steps to set up the client:

1. Load the JavaScript library.
2. Initialize the accept object the capture context JWT. For information JSON Web Tokens, see JSON Web Tokens (on page 123).
3. Initialize the unified payment object with optional parameters.
4. Show the button list or payment acceptance page or both.

The response to these interactions is a transient token that you use to retrieve the payment information captured by the UI.

Loading the JavaScript Library and Invoking the Accept Function

Use the client library asset path returned by the capture context response to invoke Unified Checkout on your page.

Get the JavaScript library URL dynamically from the capture context response. When decoded, it appears in the JSON parameter **clientLibrary** as:

```
https://apitest.cybersource.com/up/v1/assets/x.y.z/SecureAcceptance.js
```

When you load the library, the capture context that you received from your initial server-side request is used to invoke the accept function.

Important: Use the **clientLibrary** parameter value in the capture context response to obtain the Unified Checkout JavaScript library URL. This ensures that you are always using the most up-to-date library. Do not hard-code the Unified Checkout JavaScript library URL.

JavaScript Example: Initializing the SDK

```
<script
 src="https://apitest.cybersource.com/up/v1/assets/0.19.0/SecureAcceptance.js"></script>
<script>
  Accept('header.payload.signature').then(function(accept) {
    // use accept object
  });
</script>
```

In this example, `header.payload.signature` refers to the capture context JWT.

Adding the Payment Application and Payment Acceptance

After you initialize the Unified Checkout object, you can add the payment application and payment acceptance pages to your webpage. You can attach the Unified Checkout embedded tool and payment acceptance pages to any named element within your HTML. Typically, they are attached to explicit named `<div>` components that are replaced with Click to Pay Drop-In UI `iframes`.

Important: If you do not specify a location for the payment acceptance page, it is placed in the sidebar.

JavaScript Example: Setting Up with Full Sidebar

```
var authForm = document.getElementById("authForm");
var transientToken = document.getElementById("transientToken");

var cc = document.getElementById("captureContext").value;
var showArgs = {
  containers: {
    paymentSelection: "#buttonPaymentListContainer"
  }
};
Accept(cc)
  .then(function(accept) {
    return accept.unifiedPayments();
  })
  .then(function(up) {
    return up.show(showArgs);
  })
  .then(function(tt) {
```

```
    transientToken.value = tt;
    authForm.submit();
  });
```

JavaScript Example: Setting Up with the Embedded Component

The main difference between using an embedded component and the sidebar is that the **accept.unifiedPayments** object is set to `false`, and the location of the payment screen is passed in the containers argument.

```
var authForm = document.getElementById("authForm");
var transientToken = document.getElementById("transientToken");

var cc = document.getElementById("captureContext").value;
var showArgs = {
  containers: {
    paymentSelection: "#buttonPaymentListContainer",
    paymentScreen: "#embeddedPaymentContainer"
  }
};
Accept(cc)
    .then(function(accept) {
      // Gets the UC instance (e.g. what card brands I requested, any address information
 I pre-filled etc.)
      return accept.unifiedPayments();
    })
    .then(function(up) {
      // Display the UC instance
      return up.show(showArgs);
    })
    .then(function(tt) {
      // Return transient token from UC's UI to our app
      transientToken.value = tt;
      authForm.submit();
    }).catch(function(error) {
      //merchant logic for handling issues
      alert("something went wrong");
  });
```

Transient Tokens

The response to a successful customer interaction with the Click to Pay Drop-In UI is a transient token. The transient token is a reference to the payment data collected on your behalf. Tokens enable secure card payments without risking exposure to sensitive payment information. The transient token is a short-term token with a duration of 15 minutes.

Transient Token Format

The transient token is issued as a JSON Web Token (JWT) (RFC 7519). For information on JSON Web Tokens, see JSON Web Tokens (on page 123).

The payload portion of the token is a Base64-encoded JSON string and contains various claims. This example shows a payload:

```
{
  "iss" : "Flex/00",
  "exp" : 1706910242,
  "type" : "gda-0.9.0",
  "iat" : 1706909347,
  "jti" : "1D1I2O2CSTMW3UIXOKEQFI4OQX1L7CMSKDE3LJ8B5DVZ6WBJGKLQ65BD6222D426",
  "content" : {
    "orderInformation" : {
      "billTo" : {
        // Empty fields present within this node indicate which fields were captured by
        // the application without exposing you to personally identifiable information
        // directly.
      },
      "amountDetails" : {
        // Empty fields present within this node indicate which fields were captured by
        // the application without exposing you to personally identifiable information
        // directly.
      },
      "shipTo" : {
        // Empty fields present within this node indicate which fields were captured by
        // the application without exposing you to personally identifiable information
        // directly.
      }
    },
    "paymentInformation" : {
      "card" : {
        "expirationYear" : {
          "value" : "2028"
        },
        "number" : {
          "maskedValue" : "XXXXXXXXXXXX1111",
```

```
          "bin" : "411111"
        },
        "securityCode" : { },
        "expirationMonth" : {
          "value" : "06"
        },
        "type" : {
          "value" : "001"
        }
      }
    }
  }
}
```

Token Verification

When you receive the transient token, you should cryptographically verify its integrity using the public key embedded within the capture context. Doing so verifies that Cybersource issued the token and that the data has not been tampered with in transit. Verifying the transient token JWT involves verifying the signature and various claims within the token. Programming languages each have their own specific libraries to assist. For an example in Java, see: Java Example in Github.

Capture Context API

This section contains the information you need to request the capture context using the capture context API.

The capture context request is a signed JSON Web Token (JWT) that includes all of the merchant-specific parameters. This request tells the frontend JavaScript library how to behave within your payment experience. For information on JSON Web Tokens, see JSON Web Tokens (on page 123).

You can define the payment cards that you want to accept in the capture context. Use the **allowedCardNetworks** field to define the card types.

Available card networks for card entry:

- American Express
- Diners Club
- Discover

- JCB
- Mastercard
- Visa

For more information on enabling and managing Click to Pay, see Enabling Click to Pay (on page 102).

Important:

When integrating with Cybersource APIs, Cybersource recommends that you dynamically parse the response for the fields that you are looking for. Additional fields may be added in the future.

You must ensure that your integration can handle new fields that are returned in the response. While the underlying data structures will not change, you must also ensure that your integration can handle changes to the order in which the data is returned. Cybersource uses semantic versioning practices, which enables you to retain backwards compatibility as new fields are introduced in minor version updates.

Endpoint

Production: `POST https://api.cybersource.com/up/v1/capture-contexts`

Test: `POST https://apitest.cybersource.com/up/v1/capture-contexts`

Required Fields for Requesting the Capture Context

Your capture context request must include these fields:

allowedPaymentTypes

clientVersion

country

locale

orderInformation.amountDetails.currency

orderInformation.amountDetails.totalAmount

targetOrigins

The URL in this field value must contain `https`.

For a complete list of fields you can include in your request, see the Cybersource REST API Reference.

REST Example: Requesting the Capture Context

Endpoint:

- **Production:** `POST https://api.cybersource.com/up/v1/capture-contexts`

  Test: `POST https://apitest.cybersource.com/up/v1/capture-contexts`

```
{
  {
    "targetOrigins": [
      "https://unified-payments.appspot.com"
    ],
    "clientVersion": "0.19",
    "allowedCardNetworks" : [ "VISA", "MASTERCARD", "AMEX" ],
    "allowedPaymentTypes" : [ "CLICKTOPAY" ],
    "country": "US",
    "locale": "en_US",
    "captureMandate": {
      "billingType": "FULL",
      "requestEmail": true,
      "requestPhone": true,
      "requestShipping": true,
      "shipToCountries": [
        "US",
        "UK"
      ],
```

```
      "showAcceptedNetworkIcons": true
    },
    "orderInformation": {
      "amountDetails": {
        "totalAmount": "21.00",
        "currency": "USD"
      },
      "billTo": {
        "address1": "1111 Park Street",
        "address2": "Apartment 24B",
        "administrativeArea": "NY",
        "country": "US",
        "district": "district",
        "locality": "New York",
        "postalCode": "00000",
        "company": {
          "name": "Visa Inc",
          "address1": "900 Metro Center Blvd",
          "administrativeArea": "CA",
          "buildingNumber": "1",
          "country": "US",
          "district": "district",
          "locality": "Foster City",
          "postalCode": "94404"
        },
        "email": "maya.tran@company.com",
        "firstName": "Maya",
        "lastName": "Tran",
        "middleName": "S",
        "title": "Ms",
        "phoneNumber": "1234567890",
        "phoneType": "phoneType"
      },
      "shipTo": {
        "address1": "Visa",
        "address2": "123 Main Street",
        "address3": "Apartment 102",
        "administrativeArea": "CA",
        "buildingNumber": "string",
        "country": "US",
        "locality": "Springfield",
        "postalCode": "99999",
        "firstName": "Joe",
        "lastName": "Soap"
      }
    }
  }
}
```

Successful Encrypted JWT Response to Request


```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsImRhdGEiOi
JHeUhXV0d5SG5lK2Fld1JsalVUaGJoQUFFQVZMbTR6QTA0UHBqaGFXOHVSZ2UvNFQweEtIbW9KUWNYaE1hd0RmVzVQ
VFBLNXBlZ05vRkVocnNacjdnbldLeHBRdTNWSm4vTDBjbmZOaTRSdjdlTElcdTAwM2QiLCJvcmlnaW4iOiJodHRwcz
ovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJqd2siOnsia3R5IjoiUlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJl
bmMiLCJuIjoibVhHbi1DbllDX1pkODVQdTJaaDluVDdZOUpQX1RjUV9BSzlBQTFHQkJfOFVXd2FHWEZIMGxfa2EwXz
V0ekFleU5uVWZLQ016WFFHV2dMZ2hnZXdLMjJzWlVXVTdDT0k4RkNTWktpUjBYRGJ2TTVZYkYxejk0TmNmWVJGc0p0
ZzhTbE1jY0stS00tOUFjdldYQWlxUEs0Mk5GZnlIVE5uX3BpVDdhZHRDMGFZQlhCdkw2WXFmcWM5bXBua05FQTJVN0
x5VWFyRy1rVFVIQW8xX2tjdW1tTEF1X1Y5OEQyMndsaHMtekhEcnFVTFhsNEdKSGF6WjNXVWJDWHc5c0o2dFowVmVn
X1Bpbnhmck9mazA0RWNaVlM5clBXWW1HRnA3V2NyR0FQTkRCQzFPZ0NKNW1mRmpMNEtpcVpVNURpTWFsbURGdzg5VV
p1bllBVWlrdUU1SURRIiwia2lkIjoiMDBDeWg5UHhhdDdCUkMwa0pXUG5hUVJsOU9jTGMzZVoifX0sImN0eCI6W3si
ZGF0YSI6eyJhbGxvd2VkUGF5bWVudFR5cGVzIjpbeyJwYWdlIjoxLCJ0eXBlIjoiUEFORU5UUlkifSx7InBhZ2UiOj
IsInR5cGUiOiJTUkNWSVNBIn0seyJwYWdlIjozLCJ0eXBlIjoiU1JDTUFTVEVSQ0FSRCJ9LHsicGFnZSI6NCwidHlw
ZSI6IlNSQ0FNRVgifSx7InBhZ2UiOjUsInR5cGUiOiJHT09HTEVQQVkifSx7InBhZ2UiOjYsInR5cGUiOiJBUFBMRV
BBWSJ9XSwicGF5bWVudENvbmZpZ3VyYXRpb25zIjp7IlNSQ1ZJU0EiOnsib3JpZ2luIjoiaHR0cHM6Ly9zYW5kYm94
LWFzc2V0cy5zZWN1cmUuY2hlY2tvdXQudmlzYS5jb20iLCJwYXRoIjoiL2NoZWNrb3V0LXdpZGdldC9yZXNvdXJjZX
MvanMvc3JjLWktYWRhcHRlci92aXNhU2RrLmpzIiwicGFuRW5jcnlwdGlvbktleSI6eyJrdHkiOiJSU0EiLCJlIjoi
QVFBQiIsInVzZSI6ImVuYyIsImtpZCI6IldaTEQzS0VBUFdJRThMS0pEMU0xMTNYMXExamZUZE5pNTI0al9aQWxLLVm
tlanBxM0EiLCJuIjoic1pQSXVzRGY3eVFubmhCa1U5bXUxNFZPTzNDcnVpM2I3ckFmMktZZW9iVVJtWEExN2IxSlg5
amcwQ2QtdmdwbXV5VHJ4QlVTYy00YjAtVVBnU3dHRnFQV1VweDA4RXhxcndQRE92Rm9qQm91MndseXE4YmN5MFVzLU
JmZUN6U0U1bE1WZFNYVFhYWGNOcXUtcWIyMmpDQ0NKQUxweHNBcnNib01PWHNMZWRoM000WE5RNVhHQXRSZjdiLS11
VFk1RHI5S0xZeVV2WktBblkwNE1LSlBFTzU0WWlJRk01RFRBaE5PbXMwODlqZE1keC1VUklLSmpQVTItUnBIRzF1OE
xDRzAyOFJUSXBQc05iUmFudVM1VEFZX3pseERnYjFoS0ozNlliWkVOSExnOVBYVEJoZE9NbFU5MERUTGxmY2JMVGEt
RDdEZ2xqQWFXQ3V2ekxQYUd3In0sInBhcmFtZXRlcnMiOnsic3JjSW5pdGlhdG9ySWQiOiJSNDVOMzQzRDZLWFpSWU
1CSVhMSTIxeDgtWGtMaWh4Q21lcFMzaEFlUm91RWcwaTVVIiwic3JjaURwYUlkIjoiOTBhZDlhN2QtOTU5Ni00ZWQx
LWE3MTEtMmJjOTllM2JjNWZmIiwic3JjaVRyYW5zYWN0aW9uSWQiOiIzMWJkNTRjZi1hOGIyLTQwMTEtODQ0Ny1jYj
czZDM4OGU0NjYiLCJkcGFUcmFuc2FjdGlvbk9wdGlvbnMiOnsiZHBhTG9jYWxlIjoiZW5fVVMiLCJwYXlsb2FkVHlw
ZUluZGljYXRvciI6IkZVTEwiLCJyZXZpZXdBY3Rpb24iOiJjb250aW51ZSIsImRwYUFjY2VwdGVkQmlsbGluZ0NvdW
50cmllcyI6W10sImRwYUFjY2VwdGVkU2hpcHBpbmdDb3VudHJpZXMiOltdLCJkcGFCaWxsaW5nUHJlZmVyZW5jZSI6
IkFMTCIsImRwYVNoaXBwaW5nUHJlZmVyZW5jZSI6IkFMTCIsImNvbnN1bWVyTmFtZVJlcXVlc3RlZCI6dHJ1ZSwiY2
9uc3VtZXJFbWFpbEFkZHJlc3NSZXF1ZXN0ZWQiOnRydWUsImNvbnN1bWVyUGhvbmVOdW1iZXJSZXF1ZXN0ZWQiOnRy
dWUsInRyYW5zYWN0aW9uQW1vdW50Ijp7InRyYW5zYWN0aW9uQW1vdW50IjoiMS4wMSIsInRyYW5zYWN0aW9uQ3Vycm
VuY3lDb2RlIjoiVVNEIn0sInBheW1lbnRPcHRpb25zIjp7ImRwYUR5bmFtaWNEYXRhVHRsTWludXRlcyI6MTUsImR5
bmFtaWNEYXRhVHlwZSI6IlRBVlYiLCJkcGFQYW5SZXF1ZXN0ZWQiOmZhbHNlfX19fSwiU1JDTUFTVEVSQ0FSRCI6ey
JvcmlnaW4iOiJodHRwczovL3NhbmRib3guc3JjLm1hc3RlcmNhcmQuY29tIiwicGF0aCI6Ii9zZGsvc3Jjc2RrLm1h
c3RlcmNhcmQuanMiLCJwYW5FbmNyeXB0aW9uS2V5Ijp7Imt0eSI6IlJTQSIsImUiOiJBUUFCIiwidXNlIjoiZW5jIi
wia2lkIjoiMjAyMzAyMDcyMjM1MjEtc2FuZGJveC1mcGFuLWVuY3J5cHRpb24tc3JjLW1hc3RlcmNhcmQtaW50Iiwi
a2V5X29wcyI6WyJlbmNyeXB0Iiwid3JhcEtleSJdLCJhbGciOiJSU0EtT0FFUC0yNTYiLCJuIjoidDA2SThzamxTLX
Jyczd1Q2FnSDhldm9ldW1hUm92S3ppWlNJOVMyTjlJRFE5dFcyUGFwZlJhOUxjMUt2ZUVCRFZzMjdQa2hrVTVPeUhn
UDBpRWpUdUtWcHZoNTlUNGxhLW1CU0lsczdVZWNVUUxMYTBXa21idEw3ak5kbHRBNWZxN0FoY0FyNXFjYTk4OHFyTG
Q3SXlyOUUwQzNUeGJUOXRvMWlRY3B6OG9jWk9EUlhvaWRGQW5PVkw1WUdGbWxzcmVEYko0VmhzaTBwQWRjY1FjaWwt
eWRTZ3VyS0ItcnFLcHBiOWVwb211NFFVaDMzODJDdjhOb2JZbUYzb3M4bkdHZ0dQLWN5WG8wbnNLY1BBZ2ZybFF6b3
M3cUh4VU9yRmUyeF9sWjFHMUFFLVhya3J4akJ5czlxNTNHTVJTTkNROGMtX21jRjlwYnE0SFlCcy12RDVRIn0sInBh
cmFtZXRlcnMiOnsic3JjaVRyYW5zYWN0aW9uSWQiOiIzMWJkNTRjZi1hOGIyLTQwMTEtODQ0Ny1jYjczZDM4OGU0Nj
YiLCJzcmNpRHBhSWQiOiI5ODQ4Y2ZmNC1jODY0LTRmMTgtOWYwMy1hOGY1MGE2OTJlZGRfc3lzdGVtdGVzdCIsInNy
Y0luaXRpYXRvcklkIjoiNmY1ZDZjMDktZjhlMi00MzMwLWEzZGYtMjBiOWFkN2E0NTJiIiwiZHBhVHJhbnNhY3Rpb2
5PcHRpb25zIjp7InRyYW5zYWN0aW9uVHlwZSI6IlBVUkNIQVNFIiwiZHBhTG9jYWxlIjoiZW5fVVMiLCJkcGFBY2Nl
cHRlZFNoaXBwaW5nQ291bnRyaWVzIjpbXSwiY29uc3VtZXJFbWFpbEFkZHJlc3NSZXF1ZXN0ZWQiOnRydWUsImNvbn
```

```
N1bWVyUGhvbmVOdW1iZXJSZXF1ZXN0ZWQiOnRydWUsInRyYW5zYWN0aW9uQW1vdW50Ijp7InRyYW5zYWN0aW9uQW1v
dW50IjoiMS4wMSIsInRyYW5zYWN0aW9uQ3VycmVuY3lDb2RlIjoiVVNEIn0sImRwYUFjY2VwdGVkQmlsbGluZ0NvdW
50cmllcyI6W10sImRwYUJpbGxpbmdQcmVmZXJlbmNlIjoiRlVMTCIsImRwYVNoaXBwaW5nUHJlZmVyZW5jZSI6IkZV
TEwiLCJjb25zdW1lck5hbWVSZXF1ZXN0ZWQiOnRydWUsInBheWxvYWRUeXBlSW5kaWNhdG9yIjoiRlVMTCIsInBheW
1lbnRPcHRpb25zIjp7ImR5bmFtaWNEYXRhVHlwZSI6IkNBUkRfQVBQTElDQVRJT05fQ1JZUFRPR1JBTV9TSE9SVF9G
T1JNIn19fX0sIlNSQ0FNRVgiOnsib3JpZ2luIjoiaHR0cHM6Ly9xd3d3LmFleHAtc3RhdGljLmNvbSIsInBhdGgiOi
IvYWthbWFpL3JlbW90ZWNvbW1lcmNlL3NjcmlwdHMvYW1leFNESy0xLjAuMC5qcyIsInBhbkVuY3J5cHRpb25LZXki
Onsia3R5IjoiUlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJraWQiOiJzcmMtYW1leC1jYXJkLWVuYy0yMDI0Ii
wiYWxnIjoiUlNBLU9BRVAtMjU2IiwibiI6Im1FazBibUxDMlpRVy1hNEtYMW5EWTNaZlBMRnJIOHRuVXlJYjVrVEtn
emFlYWdpbWFINFhxUDRadzA1aWk2TXZkdk4wVDJweVNKUTRqb2toUEMySVdlbWlWUEc4ZkNQQk1KeHhqeTJFdTlvdG
Jpd0dSQkNneHdjdS1hY2pZYXVwVlB0RE43ZW5nSERkbk9nYXJsb0dyUFVNNklFRVpXX3ZFQjljU3JNX0JhOFNjQzhS
YWZnTlNZODFpeGF4UEE4Y09oQUF2ckxRN0toRTVReFN6SU1mcnpiMUxCWUdMNFlQQnVuZk5BMnczZnZMd2ZCbDJfLV
JGUkNVbVBFdjFOdVhxeG8xUk4wOGoydW44ZWljR3ZudDBndC0yMW5HcmJjNnhwcDdwWlkyb2otaGMwWlVsTnlFX2tK
cExxTNU9VWjhHZU9acDRxVlJ4aGtJUEd4RWVGLVFXaVNnOHVXazF4Nm5jdGhyTVVKWVYxSFB1OHRIa0pEbThBYS1Ec2
hQTmVpeERqX1ZGVkVTOFYteUlJUndnLVUyODJXUGIwVDJ0S1JYZG5qbE52Y2xCc0lfNFZ3ZzVjV0VoU2tTc3pVQXkx
UENTRm5rWjVJRU9yaGdfMFRwZTdhaU84dzVzUndOaFpuUnBKeUlzUHQtbE1Dbzd6cjg1QjJ2eGNvUGZmU1NwM0ZaIn
0sInBhcmFtZXRlcnMiOnsic3JjaVRyYW5zYWN0aW9uSWQiOiIzMWJkNTRjZi1hOGIyLTQwMTEtODQ0Ny1jYjczZDM4
OGU0NjYiLCJzcmNJbml0aWF0b3JJZCI6ImQyZTdkOTc1LWIwYWEtNGZhYS05YTUxLTY4MDAyMjkwZDc1NiIsImRwYU
RhdGEiOnsiZHBhTmFtZSI6InRlc3QgU2hvcCB3ZWJzaXRlIFJlZ2lzdHJhdGlvbiIsImRwYUxvZ29VcmkiOiJodHRw
Oi8vd3d3LnRlc3RzcmNyZWdpc3RyYXRpb24uY29tIiwiZHBhUHJlc2VudGF0aW9uTmFtZSI6InRlc3QgU2hvcCB3ZW
JzaXRlIFJlZ2lzdHJhdGlvbiIsImRwYVVyaSI6Imh0dHA6Ly93d3cudGVzdHNyY3JlZ2lzdHJhdGlvbi5jb20ifSwi
ZHBhVHJhbnNhY3Rpb25PcHRpb25zIjp7ImRwYUxvY2FsZSI6ImVuX1VTIiwiZHBhQWNjZXB0ZWRCaWxsaW5nQ291bn
RyaWVzIjpbXSwiZHBhQWNjZXB0ZWRTaGlwcGluZ0NvdW50cmllcyI6W10sImRwYUJpbGxpbmdQcmVmZXJlbmNlIjoi
QUxMIiwiZHBhU2hpcHBpbmdQcmVmZXJlbmNlIjoiQUxMIiwiY29uc3VtZXJOYW1lUmVxdWVzdGVkIjp0cnVlLCJjb2
5zdW1lckVtYWlsQWRkcmVzc1JlcXVlc3RlZCI6dHJ1ZSwiY29uc3VtZXJQaG9uZU51bWJlclJlcXVlc3RlZCI6dHJ1
ZSwicmV2aWV3QWN0aW9uIjoiY29udGludWUiLCJ0aHJlZURzUHJlZmVyZW5jZSI6Ik5PTkUiLCJwYXltZW50T3B0aW
9ucyI6W3siZHluYW1pY0RhdGFUeXBlIjoiRFlOQU1JQ19DQVJEX1NFQ1VSSVRZX0NPREUiLCJkcGFEeW5hbWljRGF0
YVR0bE1pbnV0ZXMiOiIxNSJ9XX19fSwiR09PR0xFUEFZIjp7ImNsaWVudExpYnJhcnkiOiJodHRwczovL3BheS5nb2
9nbGUuY29tL2dwL3AvanMvcGF5LmpzIiwicGF5bWVudE9wdGlvbnMiOnsiZW52aXJvbm1lbnQiOiJURVNUIn0sInBh
eW1lbnREYXRhUmVxdWVzdCI6eyJhcGlWZXJzaW9uIjoyLCJhcGlWZXJzaW9uTWlub3IiOjAsIm1lcmNoYW50SW5mby
I6eyJtZXJjaGFudElkIjoiQkNSMkRONFQ3RERZQlRUViIsIm1lcmNoYW50TmFtZSI6IlVuaWZpZWQgQ2hlY2tvdXQg
TWVyY2hhbnQifSwiYWxsb3dlZFBheW1lbnRNZXRob2RzIjpbeyJ0eXBlIjoiQ0FSRCIsInBhcmFtZXRlcnMiOnsiYW
xsb3dlZEF1dGhNZXRob2RzIjpbIlBBTl9PTkxZIiwiQ1JZUFRPR1JBTV8zRFMiXSwiYWxsb3dlZENhcmROZXR3b3Jr
cyI6WyJWSVNBIiwiTUFTVEVSQ0FSRCIsIkFNRVgiXSwiYmlsbGluZ0FkZHJlc3NSZXF1aXJlZCI6dHJ1ZSwiYmlsbG
luZ0FkZHJlc3NQYXJhbWV0ZXJzIjp7ImZvcm1hdCI6IkZVTEwiLCJwaG9uZU51bWJlclJlcXVpcmVkIjp0cnVlfX0s
InRva2VuaXphdGlvblNwZWNpZmljYXRpb24iOnsidHlwZSI6IlBBWU1FTlRfR0FURVdBWSIsInBhcmFtZXRlcnMiOn
siZ2F0ZXdheSI6ImN5YmVyc291cmNlIiwiZ2F0ZXdheU1lcmNoYW50SWQiOiJwc19ocGEifX19XSwidHJhbnNhY3Rp
b25JbmZvIjp7InRvdGFsUHJpY2VTdGF0dXMiOiJGSU5BTCIsInRvdGFsUHJpY2UiOiIxLjAxIiwiY291bnRyeUNvZG
UiOiJVUyIsImN1cnJlbmN5Q29kZSI6IlVTRCJ9LCJlbWFpbFJlcXVpcmVkIjp0cnVlLCJzaGlwcGluZ0FkZHJlc3NS
ZXF1aXJlZCI6dHJ1ZSwic2hpcHBpbmdBZGRyZXNzUGFyYW1ldGVycyI6eyJwaG9uZU51bWJlclJlcXVpcmVkIjp0cn
VlfX19LCJBUFBMRVBBWSI6eyJzZXNzaW9uUGF0aCI6Ii9mbGV4L3YyL2FwcGxlL3BheW1lbnQtc2Vzc2lvbnMiLCJt
ZXJjaGFudElkZW50aWZpZXIiOiJtZXJjaGFudC5jb20uY3liZXJzb3VyY2Uuc3RhZ2VmbGV4IiwiZGlzcGxheU5hbW
UiOiJVQyBUZXN0In19LCJjYXB0dXJlTWFuZGF0ZSI6eyJiaWxsaW5nVHlwZSI6IkZVTEwiLCJyZXF1ZXN0RW1haWwi
OnRydWUsInJlcXVlc3RQaG9uZSI6dHJ1ZSwicmVxdWVzdFNoaXBwaW5nIjp0cnVlLCJzaGlwVG9Db3VudHJpZXMiOl
tdLCJzaG93QWNjZXB0ZWROZXR3b3JrSWNvbnMiOnRydWV9LCJvcmRlckluZm9ybWF0aW9uIjp7ImFtb3VudERldGFp
bHMiOnsidG90YWxBbW91bnQiOiIxLjAxIiwiY3VycmVuY3kiOiJVU0QifX0sInRhcmdldE9yaWdpbnMiOlsiaHR0cH
M6Ly90aGUtdXAtZGVtby5hcHBzcG90LmNvbSJdLCJpZnJhbWVzIjp7Im1jZSI6Ii9tY2UvaWZyYW1lLmh0bWwiLCJi
dXR0b25zIjoiL2J1dHRvbmxpc3QvaWZyYW1lLmh0bWwiLCJzcmMiOiIvc2VjdXJlLXJlbW90ZS1jb21tZXJjZS9zcm
MuaHRtbCIsImN0cCI6Ii9jdHAvY3RwLmh0bWwiLCJnb29nbGVwYXkiOiIvZ29vZ2xlcGF5L2dvb2dsZXBheS5odG1s
```

```
IiwiYXBwbGVwYXkiOiIvYXBwbGVwYXkvYXBwbGVwYXkuaHRtbCIsInBhemUiOiIvcGF6ZS9wYXplLmh0bWwifSwiY2
xpZW50VmVyc2lvbiI6IjAuMTkiLCJjb3VudHJ5IjoiVVMiLCJsb2NhbGUiOiJlbl9VUyIsImFsbG93ZWRDYXJkTmV0
d29ya3MiOlsiVklTQSIsIk1BU1RFUkNBUkQiLCJBTUVYIl0sImNyIjoiNmM0dUcyemFXdVBvbkxLM0R2NEwxVlJpTF
VOMkFVczY4QU84bVdaUTA0X1RNLVFDdDhNUDNTQklvcGQ2Y2NtOTdmSEo1QXViVzh6VFhJTW91TTRjQWFrbm80NktI
VndGRFpxQ0tfWTVwMEVzRHJmdFVTREFrZ21KZ0pNbHJ2cnYzTkpFOWdzcldBMl8zdDJBR2hQbEtfMU9rZyIsInNlcn
ZpY2VPcmlnaW4iOiJodHRwczovL3N0YWdldXAuY3liZXJzb3VyY2UuY29tIiwiY2xpZW50TGlicmFyeSI6Imh0dHBz
Oi8vc3RhZ2V1cC5jeWJlcnNvdXJjZS5jb20vdXAvdjEvYXNzZXRzLzAuMTkuMC9TZWN1cmVBY2NlcHRhbmNlLmpzIi
wibG9nZ2luZ1BhdGgiOiIvdXAvdjEvbG9nLWV2ZW50cyIsImFzc2V0c1BhdGgiOiIvdXAvdjEvYXNzZXRzLzAuMTku
MCIsImNsaWVudExpYnJhcnlJbnRlZ3JpdHkiOiJzaGEyNTYtWllDT2tucVh5bjRad3NyOFYwaE5OcjZaUitZYThJbH
NkdFplTkhPbDJYVVx1MDAzZCJ9LCJ0eXBlIjoiZ2RhLTAuOS4wIn1dLCJpc3MiOiJGbGV4IEFQSSIsImV4cCI6MTcx
MDk2NDc4MCwiaWF0IjoxNzEwOTYzODgwLCJqdGkiOiI4SWs4bHU2NEh3NmpDVDhsIn0.XWXmjiZZGyHWIhT1hbBnc2
xfhcYczpBYxhTn4g9NMt2utMaPR8wWcZ8TYDXd8HRLBWZkktkXxFFetJ4Tc6dQ4irZ6KmalWItWEUJpjN-5sLC4Qr1
gG1JOOH5_hK6n_1hnjcQeRUBg-MsCSRBE_MA6ROSZgyfc1_WwL0g1TQUiKN5SvaM_37ooimebPQfvYyXyR_6Zkn9fu
51w6NF_Qj0wtuQP4J4P3cgyZzzOFNKuHOwi7ISmyW6BcQXQrec577SRBfcMhhC3PBxl5OrXua4qUJ_qYbplA8P4n6f
2--onAYef3UXFHmc28eRiTEeN0l0P1Yj45CIotbuw36mZrnRPQ
```


Decrypted Capture Context Header


```
{
          "kid": "j4",
          "alg": "RS256"
          }
```


Decrypted Capture Context Body with Selected Fields


```
{
          "flx" : {
          // filled with token metadata
          },
          "ctx" : [ {
          // filled with data related to your capture context request parameters
          "data" : {

  "clientLibrary" : "https://https://
apitest.cybersource.com/up/v1/assets/0.19.0/SecureAcceptance.js"
          },
          "type" : "gda-0.9.0"
          } ],
          "iss" : "Flex API",
          "exp" : 1710964780,
          "iat" : 1710963880,
          "jti" : "8Ik8lu64Hw6jCT8l"
          }
```

Payment Details API

This section contains the information you need to retrieve the non-sensitive data associated with a Unified Checkout transient token and the payment details API. This API can be used to retrieve personally identifiable information, such as the cardholder name and billing and shipping details, without retrieving payment credentials; which helps ease the PCI compliance burden.

There are two methods of authentication:

- HTTP Signature Authentication
- JSON Web Token

Important:

When integrating with Cybersource APIs, Cybersource recommends that you dynamically parse the response for the fields that you are looking for. Additional fields may be added in the future.

You must ensure that your integration can handle new fields that are returned in the response. While the underlying data structures will not change, you must also ensure that your integration can handle changes to the order in which the data is returned. Cybersource uses semantic versioning practices, which enables you to retain backwards compatibility as new fields are introduced in minor version updates.

Endpoint

Production: `GET https://api.cybersource.com/up/v1/payment-details/{id}`

Test: `GET https://apitest.cybersource.com/up/v1/payment-details/{id}`

The `{id}` is the full JWT received from Unified Checkout as the result of capturing payment information. The transient token is a JWT object that you retrieved as part of a successful capture of payment information from a cardholder.

Required Field for Retrieving Transient Token Payment Details

Your payment credentials request must include this field:

id

The `{id}` is the full JWT received from Unified Checkout as the result of capturing payment information.

REST Example: Retrieving Transient Token Payment Details

Endpoint:

- **Production:** `GET https://api.cybersource.com/up/v1/payment-details/{id}`
- **Test:** `GET https://apitest.cybersource.com/up/v1/payment-details/{id}`

The `{id}` is the full JWT received from Unified Checkout as the result of capturing payment information. The transient token is a JWT object that you retrieved as part of a successful capture of payment information from a cardholder.

Request

```
GET https://apitest.cybersource.com/up/v1/payment-details/{id}
```

Response to Successful Request


```
{
  "paymentInformation": {
    "card": {
      "expirationYear": "2024",
      "number": "XXXXXXXXXXXX1111",
      "expirationMonth": "05",
      "type": "001"
    }
  },
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "21.00",
      "currency": "USD"
    },
    "billTo": {
      "lastName": "Lee",
      "country": "US",
```

```
      "firstName": "Tanya",
      "email": "tanyalee@example.com"
    },
    "shipTo": {
      "locality": "Small Town",
      "country": "US",
      "administrativeArea": "CA",
      "address1": "123 Main Street",
      "postalCode": "98765"
    }
  }
}
```

Payment Credentials API

This section contains the information you need to retrieve the full payment credentials collected by the Unified Checkout tool using the payment credentials API. The payment information is returned in a redundantly signed and encrypted payment object. It uses the JSON Web Tokens (JWTs) as the data standard for communicating this sensitive data.

Important: Payment information returned by the `payment-credentials` endpoint will contain Personal Identifiable Information (PII). Retrieving this sensitive information requires your system to comply with PCI security standards. For more information on PCI security standards, see: https://www.pcisecuritystandards.org/

The response is returned using a JWE data object that is encrypted with your public key created during the Unified Checkout tool's integration. For more information, see Upload Your Encryption Key (on page 172).

To decrypt the JWE response, use your private key created during the Unified Checkout tool's integration. The decrypted content is a JWS data object containing a JSON payload. This payload can be validated with the Unified Checkout public signature key.

Important:

When integrating with Cybersource APIs, Cybersource recommends that you dynamically parse the response for the fields that you are looking for. Additional fields may be added in the future.

You must ensure that your integration can handle new fields that are returned in the response. While the underlying data structures will not change, you must also ensure that your integration can handle changes to the order in which the data is returned. Cybersource uses semantic versioning practices, which enables you to retain backwards compatibility as new fields are introduced in minor version updates.

Endpoint

Production: `GET https://api.cybersource.com/flex/v2/payment-credentials/{ReferenceID}`

Test: `GET https://apitest.cybersource.com/flex/v2/payment-credentials/{ReferenceID}`

`{ReferenceID}` is the reference ID returned in the `id` field when you created the payment credentials.

Example: Sample Decrypted JWE Data Object

```
{ // header
  kid = "zu"
  cty = "json+pc"
}.
{
  // registered claims
  iss = "https://flex.visa.com"
  sub = "ps_hpa"                                 // Merchant ID
  aud = "https://online.MyBank.com"
  exp = 1683105553                               // expiry of payment credentials
  iat = 1683104035                               // timestamp when JWT was created
  jti = "ae798686-a849-4dfa-836d-43e09cb183a4"    // transaction id

  "paymentInformation": {
    "tokenizedCard": {
      "number": "4111111111111111",
      "expirationMonth": "12",
      "expirationYear": "2031",
      "type": "001",
      "cryptogram": "",
      "transactionType": "1"
    }
  },
```

```
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "102.21",
      "currency": "USD"
    },
    "billTo": {
      "firstName": "John",
      "lastName": "Doe",
      "address1": "1 Market St",
      "locality": "san francisco",
      "administrativeArea": "CA",
      "postalCode": "94105",
      "country": "US",
      "email": "test@cybs.com",
      "phoneNumber": "4158880000"
    }
  }
}
.SIGNATURE
```

Required Field for Retrieving Payment Credentials

Your payment credentials request must include this field:

ReferenceID

The reference ID that is returned in the `id` field when you created the payment credentials.

REST Example: Retrieving Payment Credentials

Endpoint:

- **Production:** `GET https://api.cybersource.com/flex/v2/payment-credentials/{ReferenceID}`
- **Test:** `GET https://apitest.cybersource.com/flex/v2/payment-credentials/{ReferenceID}`

`{ReferenceID}` is the reference ID returned in the `id` field when you created the payment credentials.

Request

```
https://api.cybersource.com/flex/v2/payment-credentials/E-firqlLk7GiziQwXxAsq
```

Encrypted Response to Successful Request


```
eyJhdWQiOiJwc3AiLCJzdWIiOiJwc19ocGEiLCJraWQiOiIyMDIzMDUxNC1kcmFmdC1wc3AtZW5jcnlwdCIsI
mN0eSI6IkpXVCIsImVuYyI6IkEyNTZHQ00iLCJleHAiOjE2ODQxNDk2NjQsImFsZyI6IlJTQS1PQUVQLTI1Ni
IsImp0aSI6IjA0NDUwNWNiLTM1ZDYtNDU2ZS05OTBlLWRkZjQwYzI5NzlhNCJ9.enhUfZJOjbMX-wZPIOb1zj
8sFZiix6JSJyNw2i9QJ4k_hd7Iy_UMYvOmS-X1FJwjH0IQxMIblSV8XqMegIOm5dYBYdqouUfC8zq4Zm_dsMo
Tp3m9T6z-A_eJ8MGaxqTHSf2vWiXB-EMrww2eCXPyVTBkI1OdmYIX-s85vsqYpW-s0ThlCKaGI7B4_rJKNa7m
ou9VMBtBnfzhHLtnHDW8vsX8rLmTT76Ct2jMdIoQnlQRgEOi-zYu0Jm0gHERavUtq_7lDw9Ta73_TFw3KA2fs
G13CURyR7ZXoZy9_nRifwHjwNVbaFRceAzXoVtvM8H8F-ZzIC8AdA1FRye7RqcK9Q.OlrMxOMDkVDU6goS.TP
fBhm1eBfRjCSSvuT6SxFeZ3SGwOC6qX2Z4rlAEY9lOor2Q2E1CMqB6o-q6DNkGtASFONBzKtoB0yAgXBpx3S7
2FltR8bd40qmRnPyTOAscXa3eWbP45EqZqHW58lwUtMwcBORcfSjxPnWUo-OGmKCtIgiUO4MTlBsl9HdCLx7R
Wpwslo0pKQAuFrURHJyhdE1JUArgjNQMdQwPvCjoZ2RxTzECEqE1l0KmBGM-w8suowrnTNZl8cwVUZKzHQEJV
-twAGykQIIRCI3ydHfCupyUuA-5-Wvlk6nhcL3qND4JF-E3EIRpzm7WH8pCV5nzByUue-grHejg774c7fi1eh
fTBUZ8v6X7rTZUBLL0V5343X3zQQy_G-vq5qcaJZ8AS2XWSi17r8UEHoU5emYu5QAuXy1AhL32nDRZuXzOzQ1
9JsrTN2CD8qxU7tDpkUCEmY2GEMp4sd-rfu_2qBZDdr74tjYNgMsTIXSpgGDiwjLMJu4r460YencO6-JweGCT
8woIySjBRYpX1_axxcO6I9RUTSopPbslZwq_zpy3UuDa9InlSexM--fatYfAehY857F7bFVXlnXeqr7X0_Lri
bJsx6CWJU1ihjMVtnF-SxeE3IdpJxyFYBb7D1iL3ywFooxcGqarXU-3_CBuDHvnJFDC_iQPaeH7csb-EMeNqF
TmFf8dWNQYG7IJDfEnrnRW_XtnczH-ZS67iVuGzGwJZDQfJZ-KLhnWr6FE1EnT1VLyXPM78WeocT7cnLXmr9B
gevNmU3q_SV5nxlDLPuCqF0PmFNxaTjqfF2Qw_zOCvazwFWuBdUDdHilPqhj3gfsOesAJVA7VoTDw2U3zte3V
09KcJLaHygwPomopWOODinKzcZeWfJ39984pQa5cOMSEToGegkRZyvSxpf5PTht30uB3F3qC4cVLOu4qukYsr
jXqOtxg3icde7lXywfAtEZgf54jAP2Cl8JFmGWL5YnIY44-zj-GVz2C8iCN1CCUP3U4eVxz2GtxNNSXuwY8OR
```

```
Udino4rF-OpqqdjX5F0Uw6J2D3uR9cWB4Ee3v8TIA3-tRkG4ScAcclEwjkwsILPgVLU57HOm0AnaEsznyHrd9
-Qfz_p-UjbsaD3e-_sr56-x2UZVVL6TAMmJqmS2C55CHgkkhtHBCu-vb0KOmssopIvaQA5jK6ZoCftewE8-98
816ZmoU8Sty05PSeK0yBlxFwTIeJxt-moszRawFuBrLAbOu72y_eeUtk1tHpHV2Db7T6XvaRD4NvOFZg8ianY
Y6uHidoTl1ApjCp8VG9oTJ-uKWAEp9TU6qEHUswZZUIBeGTKjzBkRAQ20cZs5POb-qtjteoWo9QdnczipZ8de
my-FSZwNRFPkeedl3oHLepeTgwVnmij9ovk0e5Wqq2GVUMe8sLa-4eEnjliIjAVUQ9YNJBeqLf6_wo3HF8o2k
4ZgSJTuPHAuP41-D6sYrOcM6WvkCfKRTXw7ue5unri3M0Rpd2TEnzyw.TaLt6G8QyRykbrxb0iV9Jg
```


Decrypted Response to Successful Request

```
{ // header
  kid = "zu"
  cty = "json+pc"
}.
{
  // registered claims
  iss = "https://flex.visa.com"
  sub = "ps_hpa"                                // Merchant ID
  aud = "https://online.MyBank.com"
  exp = 1683105553                              // expiry of payment credentials
  iat = 1683104035                              // timestamp when JWT was created
  jti = "ae798686-a849-4dfa-836d-43e09cb183a4"    // transaction id

  "paymentInformation": {
    "tokenizedCard": {
      "number": "4111111111111111",
      "expirationMonth": "12",
      "expirationYear": "2031",
      "type": "001",
      "cryptogram": "",
      "transactionType": "1"
    }
  },
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "102.21",
      "currency": "USD"
    },
    "billTo": {
      "firstName": "John",
      "lastName": "Doe",
      "address1": "1 Market St",
      "locality": "san francisco",
      "administrativeArea": "CA",
      "postalCode": "94105",
      "country": "US",
      "email": "test@cybs.com",
      "phoneNumber": "4158880000"
```

```
    }
  }
}
.SIGNATURE
```

Unified Checkout Configuration

This section contains information necessary to configure Unified Checkout in the Business Center:

- Upload Your Encryption Key (on page 172)
- Enable Click to Pay (on page 175)
- Manage Permissions (on page 103)

Upload Your Encryption Key

Payment information can be retrieved from the Unified Checkout platform by invoking the Payment Credentials API. This API retrieves all of the data captured by Unified Checkout. This information is transmitted in an encrypted format to ensure the security of the payment information while in transit.

You must generate an encryption key pair to retrieve this encrypted payment information, and the public encryption key must uploaded to the Unified Checkout system.

Generate a Public Private Key Pair

You must generate a public-private key pair to upload to the Unified Checkout system. The public key is uploaded to the Unified Checkout platform and is used to encrypt sensitive information in transit. The private key is used to decrypt the sensitive payment information on your server. Only the private key can properly decrypt the payment information.

Important: You must secure your private decryption key. This key must never be exposed to any external systems or it will risk the integrity of the secure channel.

Unified Checkout accepts only keys that meet these requirements:

- Only RSA keys are supported. Elliptical curves are not supported.
- The minimum accepted RSA key size is 2048 bits.
- RSA keys must be in JWK format. More information on JWK format is available here:

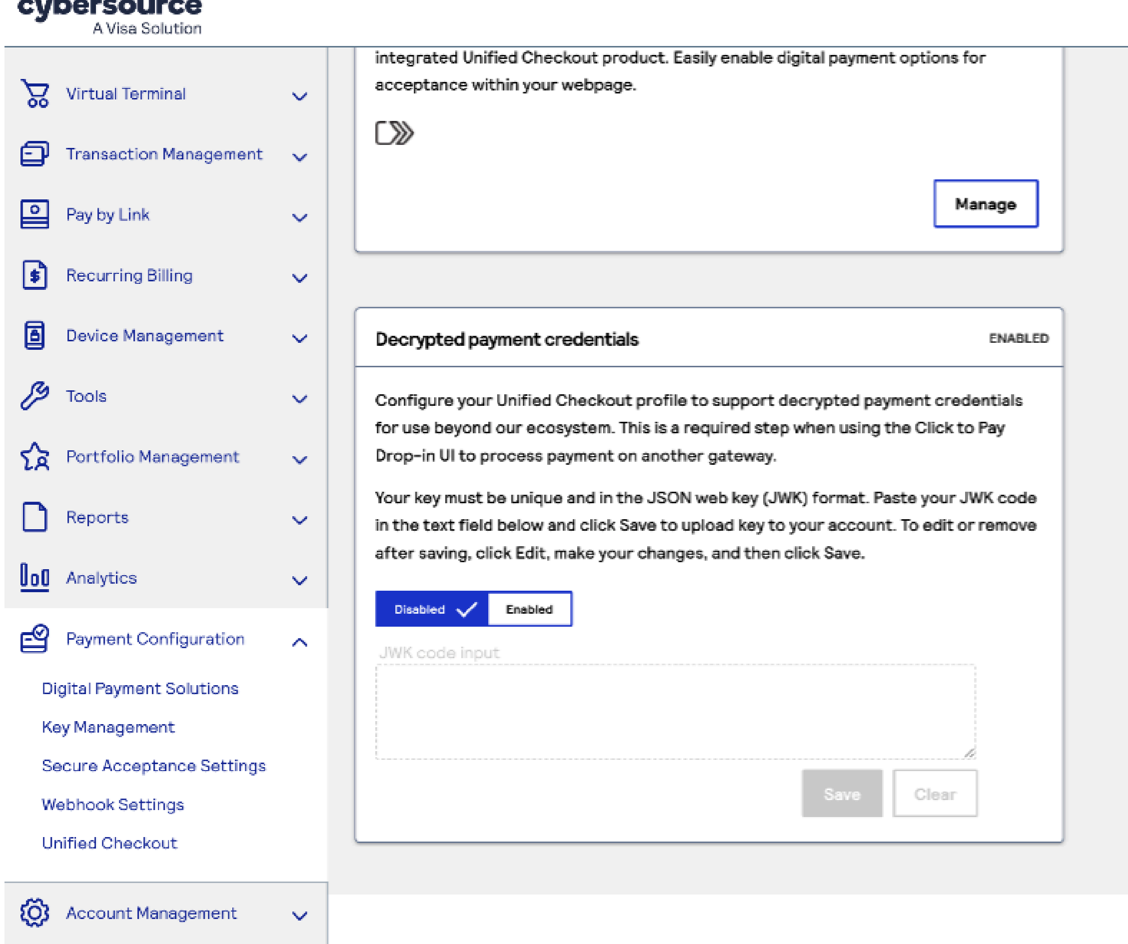
  https://datatracker.ietf.org/doc/html/rfc7517.
- The key ID must be a valid UUID.

Uploading Your Key Pair

When you have generated your encryption key pairs, you can upload your key to the Unified Checkout platform. Keys can be loaded at any hierarchy that is enabled for them and are used for all child entities that do not have keys loaded. You can upload a key at parent and child levels, but child keys override parent keys.

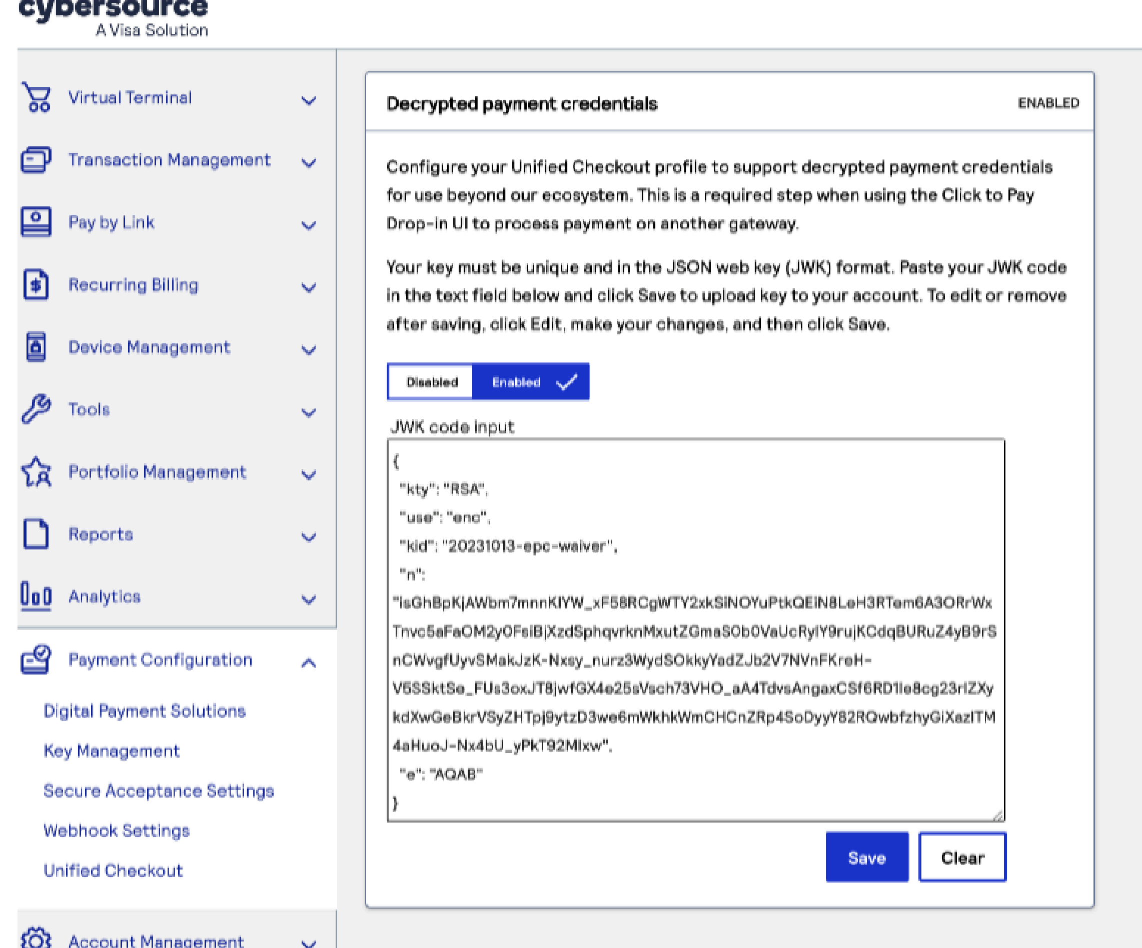
Follow these steps to upload your key pair:

1. Navigate to **Payment Configuration > Unified Checkout**. The Unified Checkout configuration page opens.



2. Click **Enabled**. You can upload your key in the appropriate section.

3. Upload the public encryption key in JWK format, and click **Save**.



Enable Click to Pay

To enable Click to Pay on Unified Checkout, you must first register Click to Pay. This process sends the appropriate information to the digital payment systems and registers your page with each system.

Enable Click to Pay for Unified Checkout in the Business Center. Click to Pay is listed as an available digital payment method offered by Unified Checkout.

Enabling Click to Pay

Click to Pay is a digital payment solution that allows customers to pay with their preferred card network and issuer without entering their card details on every website. Customers can use Visa, Mastercard, and American Express cards to streamline their purchase experience. Click to Pay provides a fast, secure, and consistent checkout experience across devices and browsers.

Follow these steps to enable in Click to Pay on Unified Checkout:

1. Navigate to **Payment Configuration > Unified Checkout**.
2. In the Click to Pay section, click **Set Up**.
3. Enter your business name and website URL.
4. Click **Submit**.
   You can now accept digital payments with Click to Pay.

Manage Permissions

Portfolio administrators can set permissions for new or existing Business Center user roles for Unified Checkout. Administrators retain full read and write permissions. They enable you to regulate access to specific pages and specify who can access, view, or amend digital products within Unified Checkout.

Portfolio administrators must apply the appropriate user role permission for any existing or newly created Business Center user roles for Unified Checkout. For information on managing permissions as a portfolio administrator, see Managing Permissions as a Portfolio Administrator (on page 105).

If you are a transacting merchant, you might find that your permissions are restricted. If your permissions are restricted, a message appears indicating that you do not have access, or buttons might appear gray. To make changes to your digital products within Unified Checkout that have restricted permissions, contact your portfolio administrator's customer support representative. For more information, see Managing Permissions as a Direct Merchant (on page 104).

Managing Permissions as a Direct Merchant

Follow these steps to configure and manage user permissions in the Business Center for Unified Checkout as a direct merchant:

1. On the left navigation panel, navigate to **Account Management**.
2. Click **Roles** to display a list of your user roles.
3. Click the pencil icon next to the user role that you want to update.
4. Click **Payment Configuration Permission**.
5. Select the relevant permission for the specific user role you are editing. You can select from these Unified Checkout permissions:
    - Unified Checkout View
    - Unified Checkout Manage

Important: If you are a transacting merchant without view permissions, Unified Checkout will still appear on the navigation bar, however, a *no access* message appears when you access Unified Checkout.

If you are a transacting merchant with view permissions but not management permissions, you can access the Unified Checkout screens and view the different payment methods configurations, however, you cannot edit or enroll new products.

Managing Permissions as a Portfolio Administrator

Follow these steps to configure and manage user permissions in the Business Center for Unified Checkout as a portfolio administrator:

1. On the left navigation panel, navigate to **Account Management**.
2. Click **Roles** to see a list of your user roles.
3. Click the pencil icon next to the user role that you want to update.
4. Click **Payment Configuration Permission**.
5. Select the relevant permission for the specific user role you are editing. You can choose from these Unified Checkout permissions:
    - Unified Checkout View
    - Unified Checkout Manage
    - Unified Checkout Portfolio View (available for portfolio users only)
    - Unified Checkout Portfolio Manage (available for portfolio users only)

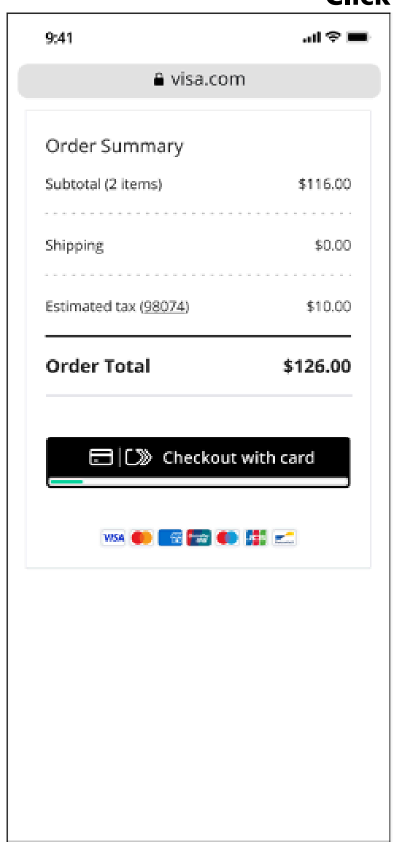
Important: If all permissions are left unselected, the user has restricted permission. A *no access* message appears when the user tries to access the Unified Checkout digital product enablement pages. The user is advised to contact a customer representative.

If a portfolio user has view permissions and does not have a management role, they can access the Unified Checkout pages, but they cannot modify toggles for different digital payments.

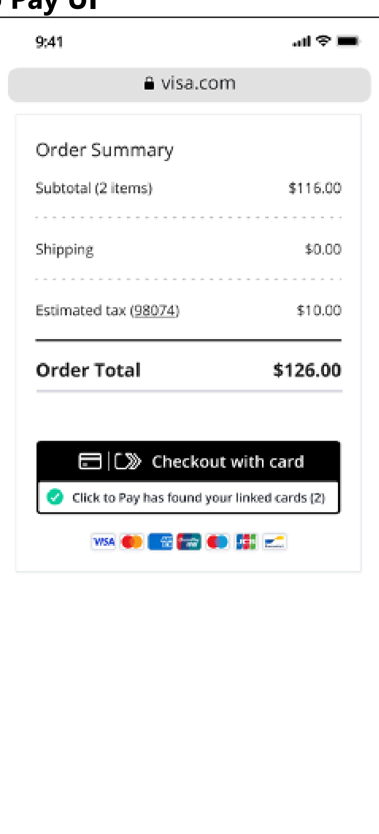
Unified Checkout UI

Completing a payment with Unified Checkout requires the customer to navigate through a sequence of interfaces. This section includes examples of the interfaces that your customers can expect when completing a payment with Click to Pay.

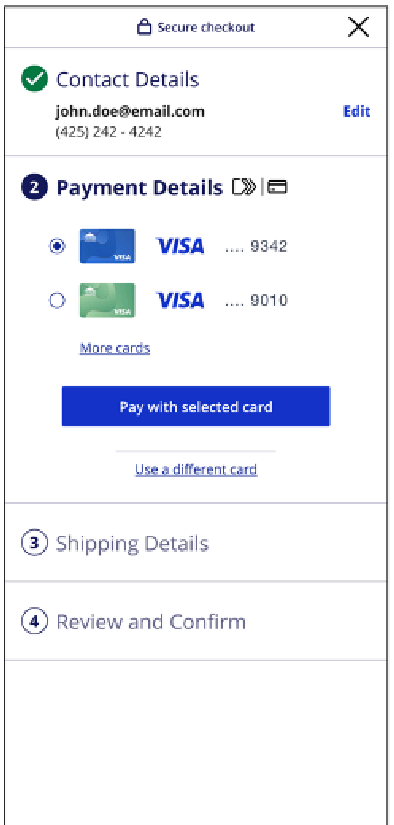
Click to Pay UI



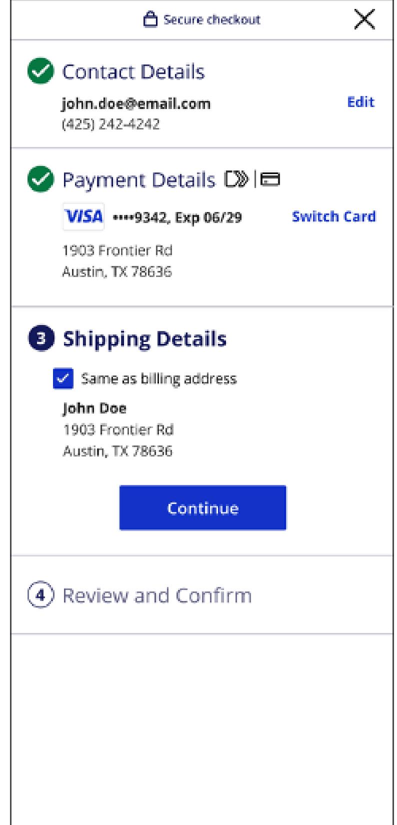
Click to Pay loader animation



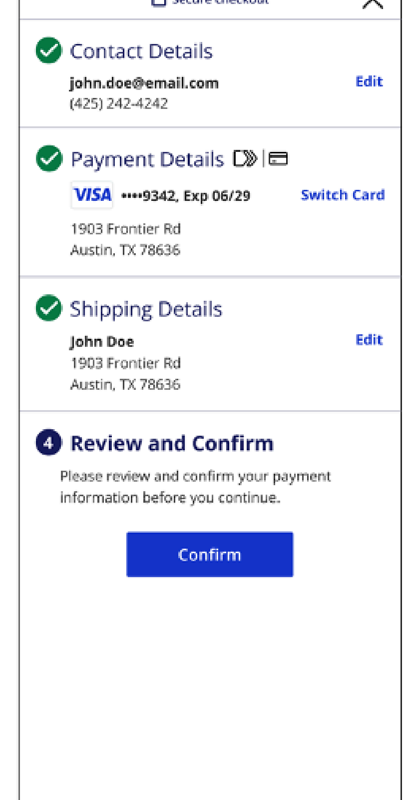
Click to Pay recognized user



Click to Pay saved cards



Click to Pay saved cards



Review screen

JSON Web Tokens

JSON Web Tokens (JWTs) are digitally signed JSON objects based on the open standard RFC 7519. These tokens provide a compact, self-contained method for securely transmitting information between parties. These tokens are signed with an RSA-encoded public/private key pair. The signature is calculated using the header and body, which enables the receiver to validate that the content has not been tampered with. Token-based applications are best for applications that use browser and mobile clients.

A JWT takes the form of a string, consisting of three parts separated by dots:

- Header
- Payload
- Signature

This example shows a JWT:

```
xxxxx.yyyyy.zzzzz
```

Supported Countries for Click to Pay

Click to Pay is supported in these countries:

- Argentina
- Australia
- Austria
- Brazil
- Canada
- China
- Colombia
- Costa Rica
- Czech Republic
- Denmark
- Dominican Republic

- Ecuador
- El Salvador
- Finland
- France
- Germany
- Honduras
- Hong Kong
- Hungary
- India
- Indonesia
- Ireland
- Italy
- Jordan
- Kuwait
- Malaysia
- Mexico
- Netherlands
- New Zealand
- Nicaragua
- Norway
- Panama
- Paraguay
- Peru
- Poland
- Qatar
- Saudi Arabia
- Singapore

- Slovakia
- South Africa
- Spain
- Sweden
- Switzerland
- Ukraine
- United Arab Emirates
- United Kingdom
- United States
- Uruguay

Supported Locales

The locale field within the capture context request consists of an ISO 639 language code, an underscore (_), and an ISO 3166 region code. The locale controls the language in which the application is rendered. The following locales are supported:

- ar_AE
- ca_ES
- cs_CZ
- da_DK
- de_AT
- de_DE
- el_GR
- en_AU
- en_CA
- en_GB
- en_IE
- en_NZ
- en_US
- es_AR
- es_CL
- es_CO
- es_ES
- es_MX
- es_PE
- es_US
- fi_FI
- fr_CA
- fr_FR

- he_IL
- hr_HR
- hu_HU
- id_ID
- it_IT
- ja_JP
- km_KH
- ko_KR
- lo_LA
- ms_MY
- nb_NO
- nl_NL
- pl_PL
- pt_BR
- ru_RU
- sk_SK
- sv_SE
- th_TH
- tl_PH
- tr_TR
- vi_VN
- zh_CN
- zh_HK
- zh_MO
- zh_SG
- zh_TW

Processing Authorizations with a Transient Token

After you validate the transient token, you can use it in place of the PAN with payment services for 15 minutes.

Authorization with a Transient Token

This section provides the minimal set of information required to perform a successful authorization with a transient token that is generated by the Flex API .

Endpoint

Production: `POST https://api.cybersource.com/pts/v2/payments`

Test: `POST https://apitest.cybersource.com/pts/v2/payments`

Required Field for an Authorization with a Transient Token

tokenInformation.transientTokenJwt

REST Interactive Example: Authorization with a Transient Token

Live Console URL: https://developer.cybersource.com/api-reference-assets/index.html#payments_payments_process-a-payment_samplerequests-dropdown_payment-with-flex-token_liveconsole-tab-request-body

REST Example: Authorization with a Transient Token

Endpoint:

- **Production:** `POST https://api.cybersource.com/pts/v2/payments`
- **Test:** `POST https://apitest.cybersource.com/pts/v2/payments`

Request

Important: The transient token may already contain information such as billing address and total amount. Any fields included in the request will supersede the information contained in the transient token.


```
{
  "tokenInformation": {

  "transientTokenJwt": "eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFsZyI6Il
JTMjU2In0.eyJpc3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTAuMS4wIiwiaWF0Ijox
NjE0NzkxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPRUhJVldBSURPkhLNjJJSFQxUVE
1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4TRIVZN-jpNbamcluAKVZ1zfdhbkrB6aNVW
ECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFfzTuq41sDhlIqB-UatveU_2ukPxLYl87EX9ytpx4zCJ
Vmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85tl2OHaajEc25tS1FwH3jDOfjAC8mu2MEk-Ew0-ukZ70Ce7Zaq4
cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bpvcKLr9k_8b9YnddIzp0p0JOCjXC_nuofQT7_x_-CQayx2czE0kD53HeNYC
5hQ"
  }
}
```


Response to Successful Request


```
{
    "_links": {
        "authReversal": {
            "method": "POST",
            "href": "/pts/v2/payments/6826225725096718703955/reversals"
```

```
        },
        "self": {
            "method": "GET",
            "href": "/pts/v2/payments/6826225725096718703955"
        },
        "capture": {
            "method": "POST",
            "href": "/pts/v2/payments/6826225725096718703955/captures"
        }
    },
    "clientReferenceInformation": {
        "code": "TC50171_3"
    },
    "id": "6826225725096718703955",
    "orderInformation": {
        "amountDetails": {
            "authorizedAmount": "102.21",
            "currency": "USD"
        }
    },
    "paymentAccountInformation": {
        "card": {
            "type": "001"
        }
    },
    "paymentInformation": {
        "tokenizedCard": {
            "type": "001"
        },
        "card": {
            "type": "001"
        },
        "customer": {
            "id": "AAE3DD3DED844001E05341588E0AD0D6"
        }
    },
    "pointOfSaleInformation": {
        "terminalId": "111111"
    },
    "processorInformation": {
        "approvalCode": "888888",
        "networkTransactionId": "123456789619999",
        "transactionId": "123456789619999",
        "responseCode": "100",
        "avs": {
            "code": "X",
            "codeRaw": "I1"
        }
    },
```

```
    "reconciliationId": "68450467YGMSJY18",
    "status": "AUTHORIZED",
    "submitTimeUtc": "2023-04-27T19:09:32Z"
    }
}
```

Authorization and Creating TMS Tokens with a Transient Token

This section provides the minimal information required in order to perform a successful authorization and create TMS tokens (customer, payment instrument, and shipping address) with a transient token.

Endpoint

Production: `POST https://api.cybersource.com/pts/v2/payments`

Test: `POST https://apitest.cybersource.com/pts/v2/payments`

Required Fields for an Authorization and Creating TMS Tokens with a Transient Token

orderInformation.amountDetails.currency

orderInformation.amountDetails.totalAmount

orderInformation.billTo.address1

orderInformation.billTo.administrativeArea

orderInformation.billTo.country

orderInformation.billTo.email

orderInformation.billTo.firstName

orderInformation.billTo.lastName

orderInformation.billTo.locality

orderInformation.billTo.postalCode

orderInformation.shipTo.address1

orderInformation.shipTo.administrativeArea

orderInformation.shipTo.country

orderInformation.shipTo.firstName

orderInformation.shipTo.lastName

orderInformation.shipTo.localityorderInformation.shipTo.locality

orderInformation.shipTo.postalCode

processingInformation.actionList

Set this field to `TOKEN_CREATE`.

processingInformation.actionTokenTypes

Set to one of the following values:

- `customer`
- `paymentInstrument`
- `shippingAddress`

tokenInformation.transientTokenJwttokenInformation.transientTokenJwt

REST Interactive Example: Authorization and Creating TMS Tokens with a Transient Token

Live Console URL: https://developer.cybersource.com/api-reference-assets/index.html#payments_payments_process-a-payment_samplerequests-dropdown_payment-with-flex-token-create-permanent-tms-token_liveconsole-tab-request-body

REST Example: Authorization and Creating TMS Tokens with a Transient Token

Endpoint: `POST https://api.cybersource.com/pts/v2/payments`

```
{
  "clientReferenceInformation": {
    "code": "TC50171_3"
  },
  "processingInformation": {
    "actionList": [
      "TOKEN_CREATE"
    ],
    "actionTokenTypes": [
      "customer",
      "paymentInstrument",
      "shippingAddress"
    ]
  },
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "102.21",
      "currency": "USD"
    },
    "billTo": {
      "firstName": "John",
      "lastName": "Doe",
      "address1": "1 Market St",
      "locality": "san francisco",
      "administrativeArea": "CA",
      "postalCode": "94105",
      "country": "US",
      "email": "test@cybs.com",
      "phoneNumber": "4158880000"
    },
    "shipTo": {
      "firstName": "John",
      "lastName": "Doe",
      "address1": "1 Market St",
```

```
      "locality": "san francisco",
      "administrativeArea": "CA",
      "postalCode": "94105",
      "country": "US"
    }
  },
  "tokenInformation": {

  "transientTokenJwt": "eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFsZyI6Il
JTMjU2In0.eyJpc3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTAuMS4wIiwiaWF0Ijox
NjE0NzkxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPRUhJVldBSURPkhLNjJJSFQxUVE
1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4TRIVZN-jpNbamcluAKVZ1zfdhbkrB6aNVW
ECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFfzTuq41sDhlIqB-UatveU_2ukPxLYl87EX9ytpx4zCJ
Vmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85tl2OHaajEc25tS1FwH3jDOfjAC8mu2MEk-Ew0-ukZ70Ce7Zaq4
cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bpvcKLr9k_8b9YnddIzp0p0JOCjXC_nuofQT7_x_-CQayx2czE0kD53HeNYC
5hQ"
  }
}
```

Successful Response


```
{
    "_links": {
        "authReversal": {
            "method": "POST",
            "href": "/pts/v2/payments/6826220442936119603954/reversals"
        },
        "self": {
            "method": "GET",
            "href": "/pts/v2/payments/6826220442936119603954"
        },
        "capture": {
            "method": "POST",
            "href": "/pts/v2/payments/6826220442936119603954/captures"
        }
    },
    "clientReferenceInformation": {
        "code": "TC50171_3"
    },
    "id": "6826220442936119603954",
    "orderInformation": {
        "amountDetails": {
            "authorizedAmount": "102.21",
            "currency": "USD"
        }
    },
    "paymentAccountInformation": {
        "card": {
            "type": "001"
        }
    },
    "paymentInformation": {
        "tokenizedCard": {
            "type": "001"
        },
        "card": {
            "type": "001"
        }
    },
    "pointOfSaleInformation": {
        "terminalId": "111111"
    },
    "processorInformation": {
        "approvalCode": "888888",
        "networkTransactionId": "123456789619999",
        "transactionId": "123456789619999",
        "responseCode": "100",
        "avs": {
```

```
            "code": "X",
            "codeRaw": "I1"
        }
    },
    "reconciliationId": "68449782YGMSJXND",
    "status": "AUTHORIZED",
    "submitTimeUtc": "2023-04-27T19:00:44Z",
    "tokenInformation": {
        "instrumentidentifierNew": false,
        "instrumentIdentifier": {
            "state": "ACTIVE",
            "id": "7010000000016241111"
        },
        "shippingAddress": {
            "id": "FA56F3248492C901E053A2598D0A99E3"
        },
        "paymentInstrument": {
            "id": "FA56E8725B06A553E053A2598D0A2105"
        },
        "customer": {
            "id": "FA56DA959B6AC8FBE053A2598D0AD183"
        }
    }
}
```

VISA Platform Connect: Specifications and Conditions for Resellers/Partners

The following are specifications and conditions that apply to a Reseller/Partner enabling its merchants through Cybersource for Visa Platform Connect ("VPC") processing. Failure to meet any of the specifications and conditions below is subject to the liability provisions and indemnification obligations under Reseller/Partner's contract with Visa/Cybersource.

1. Before boarding merchants for payment processing on a VPC acquirer's connection, Reseller/Partner and the VPC acquirer must have a contract or other legal agreement that permits Reseller/Partner to enable its merchants to process payments with the acquirer through the dedicated VPC connection and/or traditional connection with such VPC acquirer.

2. Reseller/Partner is responsible for boarding and enabling its merchants in accordance with the terms of the contract or other legal agreement with the relevant VPC acquirer.

3. Reseller/Partner acknowledges and agrees that all considerations and fees associated with chargebacks, interchange downgrades, settlement issues, funding delays, and other processing related activities are strictly between Reseller and the relevant VPC acquirer.

4. Reseller/Partner acknowledges and agrees that the relevant VPC acquirer is responsible for payment processing issues, including but not limited to, transaction declines by network/issuer, decline rates, and interchange qualification, as may be agreed to or outlined in the contract or other legal agreement between Reseller/Partner and such VPC acquirer.

DISCLAIMER: NEITHER VISA NOR CYBERSOURCE WILL BE RESPONSIBLE OR LIABLE FOR ANY ERRORS OR OMISSIONS BY THE VISA PLATFORM CONNECT ACQUIRER IN PROCESSING TRANSACTIONS. NEITHER VISA NOR CYBERSOURCE WILL BE RESPONSIBLE OR LIABLE FOR RESELLER/PARTNER BOARDING MERCHANTS OR ENABLING MERCHANT PROCESSING IN VIOLATION OF THE TERMS AND CONDITIONS IMPOSED BY THE RELEVANT VISA PLATFORM CONNECT ACQUIRER.