Digital Accept Secure Integration

Developer Guide

cybersource
A Visa Solution

Version: 26.05.01

Contents

Digital Accept Secure Integration Developer 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 Digital Accept digital payment methods 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.visaacceptance.com

Recent Revisions to This Document

26.05.01

Unified Checkout

Added information about expanded regional support for Tink Pay By Bank. See Tink Pay By Bank (on page 194).

Added support for PayPal and Venmo. See PayPal (on page 219) and Venmo (on page 222).

Updated the Visa Click to Pay test card numbers. See Visa and Mastercard Click to Pay Test Cards (on page 323).

Click to Pay Drop-In UI

Added global support for **clientVersion** `1.0`.

Updated Visa Click to Pay test cards. See Test Payment Details (on page 506).

Added information about handling errors. See Handle Errors (on page 509).

Added information about enabling Click to Pay Drop-In UI using the API. See Enabling Click to Pay Drop-In UI Using the API (on page 396) and Enable Click to Pay Customer Authentication Using the API (on page 481).

Added information about adding Click to Pay to a merchant account. See Add Click to Pay to a Merchant Account (on page 394).

Updated the Click to Pay workflow. See Click to Pay Customer Authentication (on page 476).

26.03.01

Unified Checkout

Updated the endpoint for retrieving payment details and dded endpoints for Saudi Arabia. See Payment Details API (on page 338).

Added information to the capture context section for Version 1. See Capture Context Components (on page 280).

Added support for client version `1.0`. See Client Version History (on page 235).

Added information about semantic versioning. See Versioning (on page 233).

Added endpoints for Saudi Arabia. See Sessions API (on page 280) and Validating the Capture Context (on page 309)

Added a quick-start section. See Unified Checkout Quick Start (on page 135).

Added support for the merchant experience in the Business Center. See Configure the Unified Checkout Merchant Experience (on page 249).

Added information about processing payments with Unified Checkout. See Process Payments with Unified Checkout (on page 276).

Updated the JavaScript examples and reference for Version 1. See Loading the JavaScript Library (on page 238) and JavaScript API Reference (on page 341).

Added information about handling errors. See Handle Errors (on page 329).

Added information about how to update your integration to Version 1. See Update to Unified Checkout Version 1 (on page 353).

Added information about security recommendations and PCI compliance. See Security Recommendations (on page 378).

Added endpoints for Saudi Arabia. See Authorizations with a Transient Token (on page 318).

25.12.01

Microform Integration

Added the `.flex-microform-incomplete` class. See Styling (on page 77).

Updated the default aria-label property value to `true`. See Class: Microform (on page 95).

Added `incomplete` as a CSS property. See Global (on page 103).

Added information about Web Content Accessibility Guidelines (WCAG) 2.2 compliance. See WCAG 2.2 Compliance (on page 129).

Updated the client-side setup code examples for accepting eCheck information. See Client-Side Setup (on page 69).

Unified Checkout

Added information about supported browsers. See Capture Context (on page 231).

Added support for **clientVersion** `0.34`. See Client Version History (on page 235).

Added information about Cartes Bancaires dual-branded cards. See Dual-Branded Cards (on page 317).

Added information about Konbini. See Konbini (on page 213).

Click to Pay Drop-In UI

Added support for **clientVersion** `0.34`. See Client Version History (on page 514).

Added information about supported browsers. See Introduction to the Click to Pay Drop-In UI (on page 385).

Removed reference to the complete mandate from JavaScript Examples. See JavaScript Example: Setting Up with Full Sidebar (on page 411) and JavaScript Example: Setting Up with the Embedded Component (on page 412).

Added support for the removal of the confirm and continue screen and mobile as identity for Click to Pay. See Features (on page 419).

Added the JavaScript API reference. See JavaScript API Reference (on page 462).

Updated the steps for configuration customer authentication for Visa Click to Pay. See Click to Pay Customer Authentication (on page 476).

Updated the test cards for testing authentication. See Test Payment Details (on page 506).

25.11.02

Unified Checkout

Added a complete capture context request example with all possible fields. See Example: Unified Checkout Complete Capture Context (on page 301).

Updated the capture context examples for **clientVersion** `0.31`. See these topics:

- Capture Context (on page 231)

Added information about testing your authentication method. See Test Authentication (on page 326).

Added information about customizing the Click to Pay UI. See Click to Pay UI Guidelines (on page 177).

25.11.01

Unified Checkout

Updated the list of allowed card networks. See Sessions API (on page 280).

Added support for **clientVersion** `0.31`. See Client Version History (on page 235).

Added information about Bancontact, Dragonpay and MyBank. See these topics:

- Bancontact (on page 192)
- DragonPay (on page 193)
- MyBank (on page 198)

25.10.02

Unified Checkout

Added support for **clientVersion** `0.30`. See Client Version History (on page 235).

Click to Pay Drop-In UI

Added support for **clientVersion** `0.30`. See Client Version History (on page 514).

25.10.01

Unified Checkout

Added missing reason codes. See Reason Codes (on page 335).

Updated the **clientVersion** field value to `0.30` in examples.

Added Pakistan locales. See Supported Locales (on page 376).

Added information about handling responses for Online Bank Transfers and Buy Now, Pay Later. See Handle Responses (on page 204) and Handle Responses (on page 209).

Click to Pay Drop-In UI

Updated the **clientVersion** field value to `0.30` in examples.

25.08.02

Flex API v2

Added more information about JSON Web Tokens. See JSON Web Tokens (on page 40).

Microform Integration

Added more information about JSON Web Tokens. See JSON Web Tokens (on page 127).

Unified Checkout

Added more information about JSON Web Tokens. See JSON Web Tokens (on page 362).

Click to Pay Drop-In UI

Added more information about JSON Web Tokens. See JSON Web Tokens (on page 362).

25.08.01

Unified Checkout

Added support for different payment methods. See Payment Methods (on page 149).

Added support for creating a trigger. See JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry (on page 241).

Added PayPak as a supported payment method. See Sessions API (on page 280).

Added support for client version 0.28. See Client Version History (on page 235).

Microform Integration

Added PayPak as a supported payment method. See Creating the Server-Side Capture Context for Cards (on page 48)and Creating the Server-Side Capture Context for Checks (on page 65).

25.06.01

Microform Integration

Added more information about the transient token time limit. See these topics:

- Transient Token Time Limit for Accepting Card Information (on page 56)
- Using the Transient Token to Process a Payment (on page 60)
- Transient Token Time Limit for Accepting eCheck Information (on page 72)

Flex API v2

Added more information about the transient token time limit. See these topics:

- Tokenizing Payment Information (on page 33)
- Validate the Transient Token (on page 37)
- Using the Transient Token to Process a Payment (on page 39)

25.05.01

This revision contains only editorial changes and no technical updates.

25.04.02

Unified Checkout

Added a client version history and the features included in each version. See Client Version History (on page 235).

Added information on the available features and the fields specific to each feature to the Capture Context API. See Sessions API (on page 280)

Microform Integration

Added more information on nested iframes in the capture context. See Capture Context (on page 47).

Click to Pay Drop-In UI

Added more information on nested iframes in the capture context. See Capture Context Using the Sessions API (on page 404).

25.04.01

Unified Checkout

Added more information on nested iframes in the capture context. See Capture Context (on page 231).

Microform Integration

Added more information on nested iframes in the capture context. See Capture Context (on page 47).

Click to Pay Drop-In UI

Added more information on nested iframes in the capture context. See Capture Context Using the Sessions API (on page 404).

24.08.01

This revision contains only editorial changes and no technical updates.

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.

Introducing Digital Accept Secure Integration Product Suite

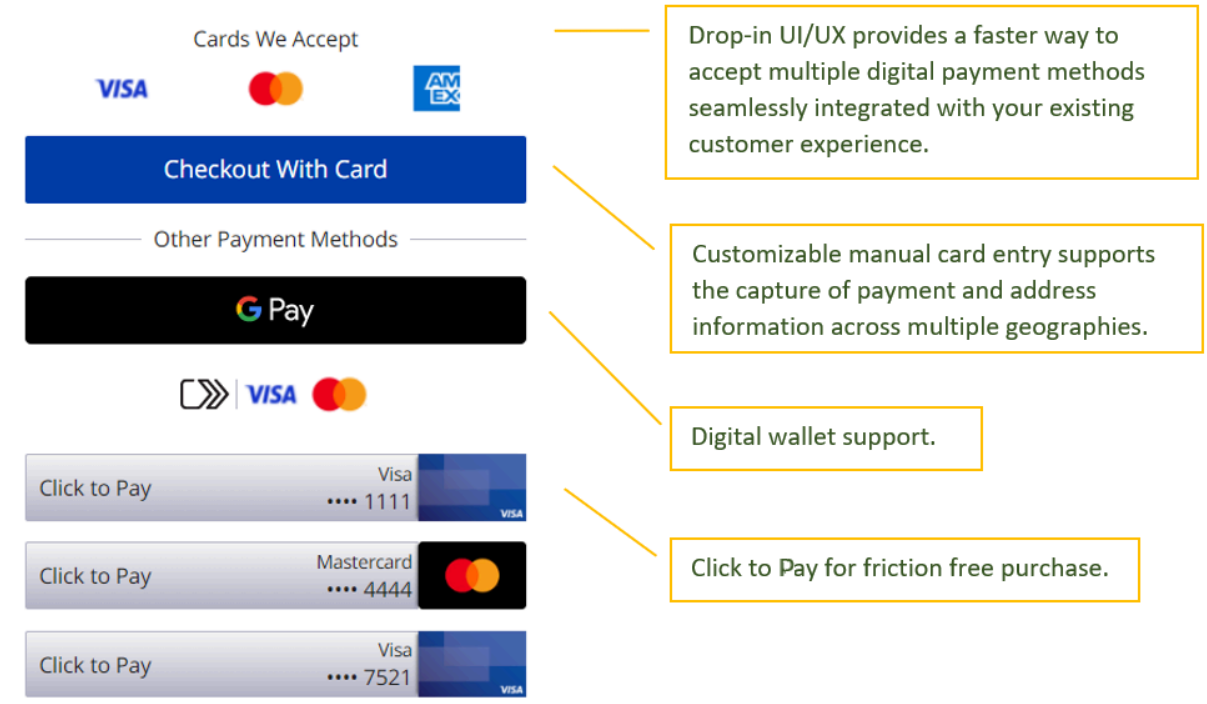
The Secure Integration Product Suite allows you to simplify the acceptance of sensitive customer payment information. When a customer enters their payment details on your webpage, app, or elsewhere, it is replaced with a transient token. Tokenization ensures that the card data can be transported securely, 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:

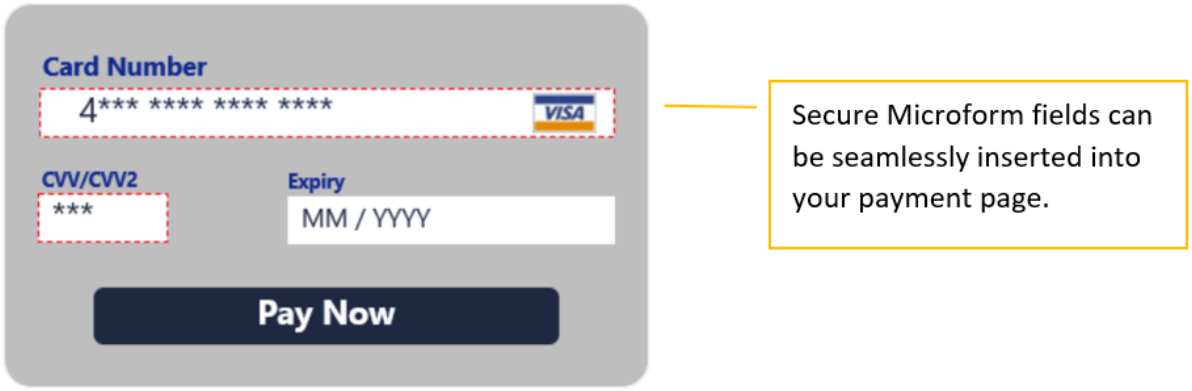
- Leading security technology
- Simple front-end integration
- Integrated with emerging digital standards
- Integrated with a range of payment methods
- Payment option presentation is optimized

For more information, see Introduction to Unified Checkout (on page 133).

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 42).

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 v2 (on page 19).

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 v2

The Flex API v2 suite enables a merchant to ensure secure transmission of payment information captured from client-side code. Integrate your system with Flex API v2 to enable Cybersource to protect your customer's primary account number (PAN), card verification number (CVN), and other payment information when payment processing activity crosses the Web.

Important: Flex API is not designed to be used from the browser. For securing payment information from the browser, please see the Microform Integration product.

Use the APIs in this suite to secure your customer's payment information, and exchange this sensitive data for a *transient token*. A transient token is a temporary reference to sensitive data that Cybersource has securely stored on your behalf. A transient token can be transported and stored safely without adding risk to your PCI DSS burden.

Important: The transient token response can be cryptographically validated to ensure that payload injection attacks can be mitigated.

Before you capture the payment data from the client application, generate the context in which the data is to be captured and tokenized. The *capture context* can help you to limit PCI exposure to the context in which it is captured.

After you capture the payment data from the client application, the Flex API v2 can secure and tokenize the data:

- Cybersource secures your customer's card data at the device using one-time public encryption keys.
- Cybersource then replaces the card data in the client application form with a transient token. A transient token can only be accessed by the merchant.

After you tokenize the payment information, you can initiate Cybersource services that use transient tokens in place of your customer's payment information.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Flex API v2 Integration Task List

The Flex API v2 is a suite of APIs that enables a merchant to safely and securely accept customer payment information from within a client application. The objective is to replace sensitive payment information with a transient token that can be transmitted without exposing the payment information.

Integrating your system with the Flex API v2 consists of these tasks:

1. **Use the `/sessions` API endpoint to generate the capture context.**

   See Generating the Capture Context (on page 20).

2. **Use the `/public-keys{}` API endpoint to validate the capture context.**

   See Validating the Capture Context (on page 27).

3. **Compile the payment information in the appropriate JWE format.**

   The data must match the data you specified in Generating the Capture Context (on page 20).

4. **Use the `/tokens` API endpoint to tokenize the payment information.**

   See Tokenizing Payment Information (on page 33).

5. Validate the Transient Token

6. Using the Transient Token to Process a Payment

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Generating the Capture Context

The first step to Flex API v2 is to generate the context of the customer payment information that is to be captured and tokenized.

Important: Declaring the capture context ensures that no data can be injected into the process by a malicious actor.

To generate the capture context, use the `/sessions` API endpoint to specify the payment data to be captured. The API returns a JSON Web Token (JWT) data object that contains the authentication component of the interactions and the one-time public encryption keys to which the payment information is to be secured.

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.

Resource

Send a fully authenticated POST request from your backend system to the `/sessions` API:

- Test: `https://apitest.cybersource.com/flex/v2/sessions`
- Production: `https://api.cybersource.com/flex/v2/sessions`

The resource returns a capture context, which is a JWT date element containing the keys necessary to encrypt the payment data.

Payment API Fields

This is the list of possible fields to capture and tokenize.

orderInformation.amountDetails.currency

orderInformation.amountDetails.totalAmount

orderInformation.billTo.address1

orderInformation.billTo.address2

orderInformation.billTo.administrativeArea

orderInformation.billTo.buildingNumber

orderInformation.billTo.company

orderInformation.billTo.country

orderInformation.billTo.district

orderInformation.billTo.email

orderInformation.billTo.firstName

orderInformation.billTo.lastName

orderInformation.billTo.locality

orderInformation.billTo.phoneNumber

orderInformation.billTo.postalCode

orderInformation.shipTo.address1

orderInformation.shipTo.address2

orderInformation.shipTo.administrativeArea

orderInformation.shipTo.buildingNumber

orderInformation.shipTo.company

orderInformation.shipTo.country

orderInformation.shipTo.district

orderInformation.shipTo.firstName

orderInformation.shipTo.lastName

orderInformation.shipTo.locality

orderInformation.shipTo.postalCode

paymentInformation.card.expirationMonth

paymentInformation.card.expirationYear

paymentInformation.card.number

paymentInformation.card.securityCode

paymentInformation.card.type

This example shows the JWT decoded, containing the JSON Web Key (JWK) encryption keys:

```
{
  "flx" : {
    "path" : "/flex/v2/tokens",
    "data" :
 "NTaTH27qZUlODxRUBEKIrhAAEFAAlrh8y17ghNZnyYVQb8vzBGNPWSmlznzPqC93XfuMJb+s7DTyk
Z5Q+yjPoF03Blczt5VviIGUcKh60cSgsHI=",
    "origin" : "https://sl73flxapq002.visa.com:8443",
```

```
    "jwk" : {
      "kty" : "RSA",
      "e" : "AQAB",
      "use" : "enc",
      "n" :
 "pFrA5Lsl22p3gNL5iPjBOYEuXs7z9P-dv7AICTGzlgNyNvyfF_tWGaLqS-lf2QgDvVW3cU0mqVxJXLE1
FcJZj71d1sgZB1n4irWsqPq54cfwEx425DDFZaiwQ_Fv1v1mAN3TRT2kaQK-_2dYMNLIWHqj93aw_bLTQT
_zo1jcaLTRje6xz7T4CqIQZ6KB_W21tcsMDGUbJ-v6yUpY2EmmcLp_vqIpsEBiCNocDGlnvMJdRyhBb8th
qiXrZjTLoOoWtiaHoAlLWL3cUoGRVGtWdEf-I-HfPDpO2HBFiFulwbv54Pjac_sVoGFzGglGrwIWB241c9
5u-bZUedpN_6ig0Q",
      "kid" : "00SvIaGIfyaw897rDeG9eFdODKaCKc1q"
    }
  },
  "ctx" : [ {
    "data" : {
      "requiredFields" : [ "paymentInformation.card.number" ],
      "optionalFields" : [ "paymentInformation.card.expirationYear",
 "paymentInformation.card.expirationMonth", "paymentInformation.card.type",
 "paymentInformation.card.securityCode" ]
    },
    "type" : "api-0.1.0"
  } ],
  "iss" : "Flex API",
  "exp" : 1614792268,
  "iat" : 1614791368,
  "jti" : "rOAksGcp8Bgg6WLj"
}
```


Related Information

- REST API Field Reference

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

REST Example: Generating the Capture Context

Minimum Request

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

Maximum Request

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

```
    },
    "orderInformation" : {
      "amountDetails" : {
        "totalAmount" : {
          "required" : false
        },
        "currency" : {
          "required" : false
        }
      },
      "billTo" : {
        "address1" : {
          "required" : false
        },
        "address2" : {
          "required" : false
        },
        "administrativeArea" : {
          "required" : false
        },
        "buildingNumber" : {
          "required" : false
        },
        "country" : {
          "required" : false
        },
        "district" : {
          "required" : false
        },
        "locality" : {
          "required" : false
        },
        "postalCode" : {
          "required" : false
        },
        "email" : {
          "required" : false
        },
        "firstName" : {
          "required" : false
        },
        "lastName" : {
          "required" : false
        },
        "phoneNumber" : {
          "required" : false
        },
        "company" : {
          "required" : false
```

```
        }
      },
      "shipTo" : {
        "address1" : {
          "required" : false
        },
        "address2" : {
          "required" : false
        },
        "administrativeArea" : {
          "required" : false
        },
        "buildingNumber" : {
          "required" : false
        },
        "country" : {
          "required" : false
        },
        "district" : {
          "required" : false
        },
        "locality" : {
          "required" : false
        },
        "postalCode" : {
          "required" : false
        },
        "firstName" : {
          "required" : false
        },
        "lastName" : {
          "required" : false
        },
        "company" : {
          "required" : false
        }
      }
    }
  }
}
```

Response Payload

```
eyJraWQiOiJzbiIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsIm
RhdGEiOiJOVGFUSDI3cVpVbE9EeFJVQkVLSXJoQUFFRkFBbHJoOHkxN2doTlpueVlWUWI4dnpCR05QV1Nt
bHpuelBxQzkzWGZ1TUpiK3M3RFR5a1o1USt5alBvRjAzQmxjenQ1VnZpSUdVY0toNjBjU2dzSElcdTAwM2
QiLCJvcmlnaW4iOiJodHRwczovL3NsNzNmbHhhcHEwMDIudmlzYS5jb206ODQ0MyIsImp3ayI6eyJrdHki
```

```
OiJSU0EiLCJlIjoiQVFBQiIsInVzZSI6ImVuYyIsIm4iOiJwRnJBNUxzbDIycDNnTkw1aVBqQk9ZRXVYcz
d6OVAtZHY3QUlDVEd6bGdOeU52eWZGX3RXR2FMcVMtbGYyUWdEdlZXM2NVMG1xVnhKWExFMUZjSlpqNzFk
MXNnWkIxbjRpcldzcVBxNTRjZndFeDQyNURERlphaXdRX0Z2MXYxbUFOM1RSVDJrYVFLLV8yZFlNTkxJV0
hxajkzYXdfYkxUUVRfem8xamNhTFRSamU2eHo3VDRDcUlRWjZLQl9XMjF0Y3NNREdVYkotdjZ5VXBZMkVt
bWNMcF92cUlwc0VCaUNOb2NER2xudk1KZFJ5aEJiOHRocWlYclpqVExvT29XdGlhSG9BbExXTDNjVW9HUl
ZHdFdkRWYtSS1IZlBEcE8ySEJGaUZ1bHdidjU0UGphY19zVm9HRnpHZ2xHcndJV0IyNDFjOTV1LWJaVWVk
cE5fNmlnMFEiLCJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSJ9fSwiY3R4Ijpbey
JkYXRhIjp7InJlcXVpcmVkRmllbGRzIjpbInBheW1lbnRJbmZvcm1hdGlvbi5jYXJkLm51bWJlciJdLCJv
cHRpb25hbEZpZWxkcyI6WyJwYXltZW50SW5mb3JtYXRpb24uY2FyZC5leHBpcmF0aW9uWVhciIsInBheW
1lbnRJbmZvcm1hdGlvbi5jYXJkLmV4cGlyYXRpb25Nb250aCIsInBheW1lbnRJbmZvcm1hdGlvbi5jYXJk
LnR5cGUiLCJwYXltZW50SW5mb3JtYXRpb24uY2FyZC5zZWN1cml0eUNvZGUiXX0sInR5cGUiOiJhcGktMC
4xLjAifV0sImlzcyI6IkZsZXggQVBJIiwiZXhwIjoxNjE0NzkyMjY4LCJpYXQiOjE2MTQ3OTEzNjgsImp0
aSI6InJPQWtzR2NwOEJnZzZXTGoifQ.uHMrYZFoqqDiiic-s-29GAI0V5Ex1361Izzhxiqt6eMZcTW-bAp
AxgfTe0eBK3vi9s6VZbm1fgE1dh8BdMeo2AkF-_Q4c3wch2YPOMhcuOpstZyLj22tnrmaLXmcHwTorDBMA
3fVH_8EIn8T4gonZ-ItTa05sxAk5rLVEWywlau5-Gi74tuxtDQOPIc7F9SzmqwGmLCuUZ6JuJf8bExAyL5
ChiqQ9MDsbP6Q2jtDXok4VAHVkJR3uRJvmblHfgRM1LRVH8XGv9GX69b30_rQ4Md5xOugvI6Hu7X30qo9f
FpfT3v9qQ6wocnJpowKe2v0u7rcid_GqqjZckbEVb47VQ
```


Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Validating the Capture Context

The capture context that you generate is a JSON Web Token (JWT) data object. The JWT is digitally signed using a public key and confirms the validity of the JWT and that it comes from Cybersource. When you do not have a key in the JWT header, Cybersource recommends that you follow cryptography best practices and validate the capture context signature.

To validate a JWT, you must obtain its public key. This public RSA key is in JSON Web Key (JWK) format. The 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, you must retrieve the key ID associated with the public key and then pass the key ID to the `/flex/v2/public-keys` endpoint:

1. From the header of the capture context, get the key ID (**kid**):

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

2. Send a GET request to the `/flex/v2/public-keys` endpoint and include the key ID. For example:

   - **Test:** `GET https://apitest.cybersource.com/flex/v2/public-keys/{3g}`
   - **Production:** `GET https://api.cybersource.com/flex/v2/public-keys/{3g}`
   - **Production in Saudi Arabia:** `GET https://api.sa.cybersource.com/flex/v2/public-keys/{3g}`
   - **Test in Saudi Arabia:** `GET https://apitest.sa.cybersource.com/flex/v2/public-keys/{3g}`

   Depending on the cryptographic method you use to validate the public key, you might need to convert the key to privacy-enhanced mail (PEM) format.

3. The resource returns the public key:

```
eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsImRhdGEiOiI2bUFLNTNPNVpGTUk5Y3Rob
WZmd2doQUFFRGNqNU5QYzcxelErbm8reDN6WStLOTVWQ2c5bThmQWs4czlTRXBtT21zMmVhbEx5Nk
hHZ29oQ0JEWjVlN3ZUSGQ5YTR5a2tNRDlNVHhqK3ZoWXVDUmRDaDhVY1dwVUNZWlZnbTE1UXVFMkE
iLCJvcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbSIsImp3ayI6eyJrdHki
OiJSU0EiLCJlIjoiQVFBQiIsInVzZSI6ImVuYyIsIm4iOiJyQmZwdDRjeGlkcVZwT0pmVTlJQXcwU
1JCNUZqN0xMZjA4U0R0VmNyUjlaajA2bEYwTVc1aUpZb3F6R3ROdnBIMnFZbFN6LVRsSDdybVNTUE
ZIeTFJQ3BfZ0I3eURjQnJ0RWNEanpLeVNZSTVCVjNsNHh6Qk5CNzRJdnB2Smtqcnd3QVZvVU4wM1R
aT3FVc0pfSy1jT0xpYzVXV0ZhQTEyOUthWFZrZFd3N3c3LVBLdnMwNmpjeGwyV05STUIzTS1ZQ0xO
b3FCdkdCSk5oYy1uM1lBNU5hazB2NDdiYUswYWdHQXRfWEZ0ZGItZkphVUVUTW5WdW9fQmRhVm90d
1NqUFNaOHFMOGkzWUdmemp2MURDTUM2WURZRzlmX0tqNzJjTi1OaG9BRURWUlZyTUtiZ3QyRDlwWk
J1d2gzZlNfS3VRclFWTVdPelRnT3AzT2s3UVFGZ1EiLCJraWQiOiIwOEJhWXMxbjdKTUhjSDh1bkc
xc1NDUVdxN2VveWQ1ZyJ9fSwiY3R4IjpbeyJkYXRhIjp7InRhcmdldE9yaWdpbnMiOlsiaHR0cHM6
Ly93d3cudGVzdC5jb20iXSwibWZPcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlL
mNvbSJ9LCJ0eXBlIjoibWYtMC4xMS4wIn1dLCJpc3MiOiJGbGV4IEFQSSIsImV4cCI6MTYxNjc3OT
A5MSwiaWF0IjoxNjE2Nzc4MTkxLCJqdGkiOiJ6SG1tZ25uaTVoN3ptdGY0In0.GvBzyw6JKl3b2Pz
tHb9rZXawx2T817nYqu6goxpe4PsjqBY1qeTo19R-CP_DkJXov9hdJZgdlzlNmRY6yoiziSZnGJdp
nZ-pCqIlC06qrpJVEDob3O_efR9L03Gz7F5JlLOiTXSj6nVwC5mRlcP032ytPDEx5TMI9Y0hmBadJ
YnhEMwQnn_paMm3wLh2v6rfTkaBqd8n6rPvCNrWMOwoMdoTeFxku-
```

   Use this public RSA key to validate the capture context.

4. Parse the JWT capture context to get the **kid** from its header:

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

5. Send a GET request to retrieve the 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"
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Securing Payment Information

After you compile and encrypt the customer payment information, secure the payment information.

Important: Payment information must be secured before it is tokenized so that data in transit cannot be compromised and exposed to malicious actors.

To secure the payment information for transport, you must encrypt the data using the one-time public encryption keys provided in the capture context.

Process Overview

The process for securing the payment information consists of these steps:

1. Construct the payment JSON payload.
2. Extract the one-time public encryption keys from the capture context.
3. Use the keys to generate a JWE (JSON Web Encryption) data object.

Internal Structure

The encrypted payment data consists of these parts:

- **data**—Structure that specifies the data to capture and store.
- **context**—Structure that specifies the capture context that you obtained from the `/sessions` API request.
- **index**—Component that must be set to `0`.

This example shows basic structure of the payload:

```
{
"data": {
},
"context": "",
"index": 0
}
```

Example

This example shows a fully populated request payload prior to encryption:

```
{
"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@email.com",
      "phoneNumber": "4158880000"
    }
  }
},
"context":
 "eyJraWQiOiIzZyIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIs
ImRhdGEiOiJyMlh5b2QxUk9SdUEyajFwUnA0cUpoQUFFSkFvUVN1QzZzZXFkVHpMaUJuTmZrMzljOXJQSH
JnQTRsSEZ1QXRrS0JiRmpqa0tHV2tmNUVjNHhBRVBMTzc0b0NsdjhneUhueFJOb1E1dHYwVnpNYU5pOWNx
d21EWmJReExENW5pVk1SWGMiLCJvcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbS
IsImp3ayI6eyJrdHkiOiJSU0EiLCJlIjoiQVFBQiIsInVzZSI6ImVuYyIsIm4iOiJqYlA4dHpIX21FQUlo
YUdmcXJ3TEQtZHZsbTZSLXgySWVaVDNweUU2YXF2SkxkY0h4bzRQZktOSXpMZ0hfZEJVTjZENGxFc2dTY3
NoT1RVOVVGVVQyVERpZUlaMVJjNW5rclNub2lYcmR5MFJscUlrS3BCa2h1WXRsSWM4OTZQb3JYVENmUk45
MmpXOXgzN2dUUnRBc2l2QXJQR2p0WGV4QnhaN29SWkFXRVY5Yy1FYVFybU55N2ZzTnJxdEZMR2xVbXdEQ0
5ONEVERXdjaWd3ck5JUlJQaHpPQkJ5UWFvenB6VlhXSVctS3RRb2otSHFfTmk2YUN0MXkwdWVLZjFkZ0dy
UHpibDV6WVNFYUJtM3gzdGZzTmM3MXVQbGJXZzY0LU83SnlMcFJWVU5UYnR1NC1ONWNic0ZaMnZBeGYwWT
dWRnRaclZiR0ZTRmFLQjZPWVdWVnciLCJraWQiOiIwOGlHZEN2Z2lCWEM4YXd6U0szWjRoUm9hbElkTzVv
MSJ9fSwiY3R4IjpbeyJkYXRhIjp7InRhcmdldE9yaWdpbnMiOlsiaHR0cDovL2xvY2FsaG9zdDozMDAwIi
wiaHR0cDovL2xvY2FsaG9zdDo1MDAwIl0sIm1mT3JpZ2luIjoiaHR0cHM6Ly90ZXN0ZmxleC5jeWJlcnNv
dXJjZS5jb20ifSwidHlwZSI6Im1mLTAuMTEuMCJ9XSwiaXNzIjoiRmxleCBBUEkiLCJleHAiOjE2MDQ2MT
c4MjgsImlhdCI6MTYwNDYxNjkyOCwianRpIjoiR1oxb1dCbTVBbHkzendwOCJ9.ZF9-CG_FvIQTMocIMwc
BH6IMWBiFfl-ufPj0TdXFuTSpusL6fAsxnyxdlf6V6i6wO0PDgv6SY-2MWP-Q600WAjFZfmR1y3r13Tig9
Ldql4WOp8zhIb6klLD01PYWeyXYZ0xqRQL0_eYTliDrV66P72PVX6DqCeoJFYnh_csEcAChmyBVRqI2Gxd
9zelALqBNU6WeHiN8FT36xRHHruxRJ2hBCI_OE0p9haQjuD4qtfk9grfhnt2mFpiC4s0j0yHaHCgiVm5NP
uPecpS7t47cjsSG6PfIHNbBAjdIVcNpmFFyH6sCLRplOgW0vPYw4nUOgtq7y_voHe_nOal6eHFr4A",
"index": 0
}
```


JSON Web Encryption Payload

The `/tokens` API endpoint accepts a JSON Web Encryption (JWE) data object that has been encrypted using the RsaOaep encryption scheme and the JSON Web Key (JWK) provided in the capture context. JWE is a standard that defines how data can be encrypted using JSON-based data structures. For details, see IETF RFC 7516 at https://tools.ietf.org/html/rfc7516.

The JWE format consists of five sections separated by a period (.) delimiters:

header . encrypted_key . cypher_text . iv . auth_tag

The payload for the `/tokens` request is a JWE data object.

This shows an encrypted payload in a fully formed JWE data object:

```
eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImVuYyI6IkEyNTZHQ00iLCJhbG
ciOiJSU0EtT0FFUCJ9.juQDhF5XcZ1rDbupn1nZ1qHhephzWpa8FumH4KrsD0yF1tCOD0L8WfpSyd5VGIe
wb4I1IipmSB5vV0O3Cb6FrNLipjFq-oexFRwSK92NbB88ySFO-7FyvPddiqaQFkA81xn8nwdoHMwUsQuqe
8Ts_krLsvYghmscxXKkwcEKqxoWbmD-yEfvKxGyHACLprAKLm-xusexaJLF42OTxYuEhzzrSe6MRll0zXu
k2DAhtUL2oHCgu8P3shgJBJqsOPcAFtwtLBRoDwlDt0ybOHjd34Svbpgf_3ncFnDkEQYe5QeElEHaB2a0N
bwo61I1UETfhedHQc8IMtDmVuKk9pgCTg.uWrwGp2jZxZd5wF0.oFzZ3I2ry77jf-3wB_2q8G-0tbYJWQj
88NdzRmVNO34JbreX5WOCju7ntvN8h83NJXEA_cQech2PEGIZV_tADBaLbSxJeitYKwaQhs_tRVrzrcd8Q
hgs4OADfky2m310eV8bUG8D4GZBKRHL6ScLf5p30b6Hoa5fDYsU7IHNyCReiaiGPExlY4luwL9QQxrfY2L
Tv74Pcqyh-B4byNxR5hTw3SJm7DT7YQLl6_-2ROqJhJoweTdDJtmJoM-LxKEij2TLgHBdqso9f036dfn0S
HLl1vG86C1-6DA9yFIZB3gLYnyom1jZuGxUOPXDojUfXo0OpUj8OI6CnQWdhKpC9X19s8xAhIAUYYdvWrE
qFfBzd9S-4E-ZdyUGfxG7fLQuLZKQJeYBbGCssLGSIXLOb15sKOopIgqCTU7M5EN_F7zW0IwJ4-b8OVf_J
80-hW1e043RlzBoMr3aGdXFIaLmVbEIzTNeZrulYTTWWLbQlcLTXqAM0yFlKmIrpq55VruvVR8i_iju5MF
zzTYuLut9ecvYbFFeUkUaUBihNXg4Np57Ix23gaJuMcPBgUqkH3nCTZQE7yQOynzO-lho_jAHy1xcwV_DJ
hhAJnACO5HUDAjVKmr-GKqxvDZWVzrqjFkPArX81eRSnn9Dr2Ahozehn9FTB37AJV3BEC2i7WMvAbQE1Ep
PVGTdvVDhH2xlLAHqHTBeQakzY4e81h2L3EDCmdjx_yZdZOUUSG3mLQSp864OV5pHc2X22ZRadGbrLwnA-
m2W1oDZIzh2t5nZdJhePnNzHbNXTf0xWSklxdgJdfG52FVSH-cKiJQnDhmCH6nPVK7NKnL0vRuZ-uuOa4P
JQDoT2H8eSjpvo8fo9rwfLYmQJa042t7OSE95bER9k1oJTUm83LNA3bxhWk5en2UFgcip3z3KlOmFwPLVN
CpzitULzAEHwBJlrB0aGXkQi1bJMxo9XZNREnFyYAlX3-aruXIe47pwAyOEX-hd-3Y7UsxBVYB86se51q2
-VUldR0zj6cwZvrTxhFM_gAsD0HisAGa6E3n3n3w1JAvjuZdHRoQqaT00YFmTdSbocmTOEUammYmBjagKK
ycOzgmoZSaYpffQl_R06tEZke6uhJrPQuTwLwivZMtnWE8O16VIRX4cG3OfzaRYs0GvPWumDlrSbM8FugM
IEaUTng5T9CdkixegRmszDELzNjNTJLe2WwxJG4Kb_1-yGMRlhFys4FEwVMk8AWJJRDpwG0jdmHkBz9l7z
1PFdIcidbIpmgH7m5RD6kwRSxaG_BJWDc2IkIFyNa2G_-gHjQh_utablUOL9CXxxFCKD9UHojtsHneFt1b
hV2P_sfYYhtZo5XloKAAEXqmOSY2boYyj0hMlKNuVqukrnWG6-bV-LBf9DvpYNKO9YeU6rYD_WOxSQlliq
VvEK8n9xLCmQQKsK2Xj2WGh7wWTQTMh18hcsNENN3Loq9DofAbOrCXqdREAshxg_MOI5vGe0JvIR9Gj6kA
hKGFf2DYBqMynbb9jWJnjCzFXBCqXXjTOuCoZdzlV9RbLxIBOOojIfLfdtVLGKPLKizXaSQ8YrLiBATark
pO7WFSSF66lvezwDZlfDErA-0kij1n2poKqDLYL3vNfX8vU33ef96VQc9I3auTpiWd0NLa5yw0RWREAjqa
4pHYTEZDiLcD0vETt84_aon3U7co_8fAYrztokTIJ2ORuhN_xA0rV1MbOZIwW6m-duqYLFLQlcwjxNwTda
berNy6bCg9otljd5l7nSbzZ6UpHrHDF02LrM41NmQUx9tZFHypYjFdgiKKgqk-kTe3pq6ithsTPvcDvDkN
gCSb9H_X30qm2-0VXaGIcYBcmJdsbBt7VJuYVZ1I_2l4-_6glgvgQz9d5KaHyZeJimSXqOsbqUQzNKWC7_
K81Z5XmqCPJByrOiROkO6iEe_poqRgVzHETHYmstAzUlgUvPD3XocZdlHuPHArQe6GddVmxnhTDV1M0TmX
wK03f0jGg7LMjWjU1k15X8xYZTk_HMo76IetUOdf9BIoaMBqMHJkk936uzjIeiW1DbEb4ExLtpIeSoq_fn
elAWoVEDMa_XoVkWCR5R7wTJjGyZKjJJkJ6UqYQguS9oO95MZp8N0Qa41wKCvztLbFKtEU7sPz3pU5oUVb
n9cZS7WCzCUNWGxb3PO0nTzPsP_MhD71JcuAEFSLS05m1hkoNiYe_6pmLv8Rrgp71kFsTOIOUrcUvwdJRi
kDOLdNbO5b-_6HjczDPzx9PaM_Zn-34mfOQPthWAfum3YvpmthuKxAWfdBChZXe9oCMeBGewGl7mKMh9H5
SP6su5yw-IFe7iBd338LVVPjRXif1rNsU631YXBu9Lz-l6o4cuGuYPVHPhHf4lifFXvlvi702wD7fbYn3c
Z55_yGVJvcFPq6OMUGJUSy5ncj-n7a8-IcGmSFpMtgnMc1ycJa_0N1vtwyjm0WvdzkUrBNC_OoCmHlLaG3
XTRenL_WYhzxDUdQQBuSC3acFu28x3NL8cmR5iqy7sBGUKcwt_ogX9ZoQyFzUTFOw.QqKIuF8EnuhOTM8P
vGEs8A
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Tokenizing Payment Information

After the payment information is secured in a JWE data object, it can be tokenized. Use the resulting transient token within Cybersource APIs in place of the payment information associated with it.

To tokenize the payment information, send the encrypted payment data to the `/tokens` API endpoint. The API responds to the request with a transient token in the form of a JSON Web Token (JWT).

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.

Resource

Important: Flex API is not designed to be used from the browser. For securing payment information from the browser, please see the Microform Integration product.

Send an unauthenticated POST request from your customer's device or backend system to the `/tokens` API endpoint:

- Test: `https://apitest.cybersource.com/flex/v2/tokens`
- Production: `https://api.cybersource.com/flex/v2/tokens`

The resource returns a transient token that represents the supplied customer card data. The token can replace the payment information in any follow-on Cybersource services.

Tokens API Request Payload

The payload for the `/tokens` request is a JWE data object.

Example


```
eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImVuYyI6IkEyNTZHQ00iLCJhbG
ciOiJSU0EtT0FFUCJ9.juQDhF5XcZ1rDbupn1nZ1qHhephzWpa8FumH4KrsD0yF1tCOD0L8WfpSyd5VGIe
wb4I1IipmSB5vV0O3Cb6FrNLipjFq-oexFRwSK92NbB88ySFO-7FyvPddiqaQFkA81xn8nwdoHMwUsQuqe
8Ts_krLsvYghmscxXKkwcEKqxoWbmD-yEfvKxGyHACLprAKLm-xusexaJLF42OTxYuEhzzrSe6MRll0zXu
k2DAhtUL2oHCgu8P3shgJBJqsOPcAFtwtLBRoDwlDt0ybOHjd34Svbpgf_3ncFnDkEQYe5QeElEHaB2a0N
bwo61I1UETfhedHQc8IMtDmVuKk9pgCTg.uWrwGp2jZxZd5wF0.oFzZ3I2ry77jf-3wB_2q8G-0tbYJWQj
88NdzRmVNO34JbreX5WOCju7ntvN8h83NJXEA_cQech2PEGIZV_tADBaLbSxJeitYKwaQhs_tRVrzrcd8Q
hgs4OADfky2m310eV8bUG8D4GZBKRHL6ScLf5p30b6Hoa5fDYsU7IHNyCReiaiGPExlY4luwL9QQxrfY2L
Tv74Pcqyh-B4byNxR5hTw3SJm7DT7YQLl6_-2ROqJhJoweTdDJtmJoM-LxKEij2TLgHBdqso9f036dfn0S
HLl1vG86C1-6DA9yFIZB3gLYnyom1jZuGxUOPXDojUfXo0OpUj8OI6CnQWdhKpC9X19s8xAhIAUYYdvWrE
qFfBzd9S-4E-ZdyUGfxG7fLQuLZKQJeYBbGCssLGSIXLOb15sKOopIgqCTU7M5EN_F7zW0IwJ4-b8OVf_J
80-hW1e043RlzBoMr3aGdXFIaLmVbEIzTNeZrulYTTWWLbQlcLTXqAM0yFlKmIrpq55VruvVR8i_iju5MF
zzTYuLut9ecvYbFFeUkUaUBihNXg4Np57Ix23gaJuMcPBgUqkH3nCTZQE7yQOynzO-lho_jAHy1xcwV_DJ
hhAJnACO5HUDAjVKmr-GKqxvDZWVzrqjFkPArX81eRSnn9Dr2Ahozehn9FTB37AJV3BEC2i7WMvAbQE1Ep
PVGTdvVDhH2xlLAHqHTBeQakzY4e81h2L3EDCmdjx_yZdZOUUSG3mLQSp864OV5pHc2X22ZRadGbrLwnA-
m2W1oDZIzh2t5nZdJhePnNzHbNXTf0xWSklxdgJdfG52FVSH-cKiJQnDhmCH6nPVK7NKnL0vRuZ-uuOa4P
JQDoT2H8eSjpvo8fo9rwfLYmQJa042t7OSE95bER9k1oJTUm83LNA3bxhWk5en2UFgcip3z3KlOmFwPLVN
CpzitULzAEHwBJlrB0aGXkQi1bJMxo9XZNREnFyYAlX3-aruXIe47pwAyOEX-hd-3Y7UsxBVYB86se51q2
-VUldR0zj6cwZvrTxhFM_gAsD0HisAGa6E3n3n3w1JAvjuZdHRoQqaT00YFmTdSbocmTOEUammYmBjagKK
ycOzgmoZSaYpffQl_R06tEZke6uhJrPQuTwLwivZMtnWE8O16VIRX4cG3OfzaRYs0GvPWumDlrSbM8FugM
IEaUTng5T9CdkixegRmszDELzNjNTJLe2WwxJG4Kb_1-yGMRlhFys4FEwVMk8AWJJRDpwG0jdmHkBz9l7z
1PFdIcidbIpmgH7m5RD6kwRSxaG_BJWDc2IkIFyNa2G_-gHjQh_utablUOL9CXxxFCKD9UHojtsHneFt1b
hV2P_sfYYhtZo5XloKAAEXqmOSY2boYyj0hMlKNuVqukrnWG6-bV-LBf9DvpYNKO9YeU6rYD_WOxSQlliq
VvEK8n9xLCmQQKsK2Xj2WGh7wWTQTMh18hcsNENN3Loq9DofAbOrCXqdREAshxg_MOI5vGe0JvIR9Gj6kA
hKGFf2DYBqMynbb9jWJnjCzFXBCqXXjTOuCoZdzlV9RbLxIBOOojIfLfdtVLGKPLKizXaSQ8YrLiBATark
pO7WFSSF66lvezwDZlfDErA-0kij1n2poKqDLYL3vNfX8vU33ef96VQc9I3auTpiWd0NLa5yw0RWREAjqa
4pHYTEZDiLcD0vETt84_aon3U7co_8fAYrztokTIJ2ORuhN_xA0rV1MbOZIwW6m-duqYLFLQlcwjxNwTda
berNy6bCg9otljd5l7nSbzZ6UpHrHDF02LrM41NmQUx9tZFHypYjFdgiKKgqk-kTe3pq6ithsTPvcDvDkN
gCSb9H_X30qm2-0VXaGIcYBcmJdsbBt7VJuYVZ1I_2l4-_6glgvgQz9d5KaHyZeJimSXqOsbqUQzNKWC7_
K81Z5XmqCPJByrOiROkO6iEe_poqRgVzHETHYmstAzUlgUvPD3XocZdlHuPHArQe6GddVmxnhTDV1M0TmX
wK03f0jGg7LMjWjU1k15X8xYZTk_HMo76IetUOdf9BIoaMBqMHJkk936uzjIeiW1DbEb4ExLtpIeSoq_fn
elAWoVEDMa_XoVkWCR5R7wTJjGyZKjJJkJ6UqYQguS9oO95MZp8N0Qa41wKCvztLbFKtEU7sPz3pU5oUVb
n9cZS7WCzCUNWGxb3PO0nTzPsP_MhD71JcuAEFSLS05m1hkoNiYe_6pmLv8Rrgp71kFsTOIOUrcUvwdJRi
kDOLdNbO5b-_6HjczDPzx9PaM_Zn-34mfOQPthWAfum3YvpmthuKxAWfdBChZXe9oCMeBGewGl7mKMh9H5
SP6su5yw-IFe7iBd338LVVPjRXif1rNsU631YXBu9Lz-l6o4cuGuYPVHPhHf4lifFXvlvi702wD7fbYn3c
Z55_yGVJvcFPq6OMUGJUSy5ncj-n7a8-IcGmSFpMtgnMc1ycJa_0N1vtwyjm0WvdzkUrBNC_OoCmHlLaG3
XTRenL_WYhzxDUdQQBuSC3acFu28x3NL8cmR5iqy7sBGUKcwt_ogX9ZoQyFzUTFOw.QqKIuF8EnuhOTM8P
vGEs8A
```

Tokens API Response Payload

Example

```
eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFsZyI6IlJTMjU2In0.eyJpc3
MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTAuMS4wIiwiaWF0IjoxNjE0Nzkx
NjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPRUhJVldBSURPSkhLNjJJSFQxUV
E1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4TRIVZN-jpNbamcluAKVZ1zfdh
bkrB6aNVWECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFfzTuq41sDhlIqB-UatveU_2ukP
xLYl87EX9ytpx4zCJVmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85tl2OHaajEc25tS1FwH3jDOfjA
C8mu2MEk-Ew0-ukZ70Ce7Zaq4cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bpvcKLr9k_8b9YnddIzp0p0JOCj
XC_nuofQT7_x_-CQayx2czE0kD53HeNYC5hQ
```

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.

Example: Capture Context Public Key

```
"jwk": {
    "kty": "RSA",
    "e": "AQAB",
    "use": "enc",
    "n":
 "3DhDtIHLxsbsSygEAG1hcFqnw64khTIZ6w9W9mZNl83gIyj1FVk-H5GDMa85e8RZFxUwgU_zQ0kHLtON

 o8SB52Z0hsJVE9wqHNIRoloiNPGPQYVXQZw2S1BSPxBtCEjA5x_-bcG6aeJdsz_cAE7OrIYkJa5Fphg9

 _pxgYRod6JCFjgdHj0iDSQxtBsmtxagAGHjDhW7UoiIig71SN-f-gggaCpITem4zlb5kkRVvmKMUANe4B

 36v4XSSSpwdP_H5kv4JDz_cVlp_Vy8T3AfAbCtROyRyH9iH1Z-4Yy6T5hb-9y3IPD8vlc8E3JQ4qt6U46
         EeiKPH4KtcdokMPjqiuQ",
    "kid": "00UaBe20jy9VkwZUQPZwNNoKFPJA4Qhc"
}
```

Use the capture context public key to cryptographically validate the JWT provided from a successful `/tokens` 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.

Using the Transient Token to Process a Payment

After you validate the transient token, you can use it in place of the PAN with payment services for 15 minutes. The transient token can be used multiple times within the 15-minute period.

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.

REST API with Transient Token JSON Web Token

```
"tokenInformation": {
     "transientTokenJwt":
 "eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFs

 ZyI6IlJTMjU2In0.eyJpc3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTA
uMS

 4wIiwiaWF0IjoxNjE0NzkxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPR
UhJ

 VldBSURPSkhLNjJJSFQxUVE1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4
TRI

 VZN-jpNbamcluAKVZ1zfdhbkrB6aNVWECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFf
zTu

 q41sDhlIqB-UatveU_2ukPxLYl87EX9ytpx4zCJVmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85
tl2

 OHaajEc25tS1FwH3jDOfjAC8mu2MEk-Ew0-ukZ70Ce7Zaq4cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bp
vcK
     Lr9k_8b9YnddIzp0p0JOCjXC_nuofQT7_x_-CQayx2czE0kD53HeNYC5hQ"
}
```

REST API withJSON Web Token ID

```
"tokenInformation": {
    "jti": "1E3GQY1RNKBG6IBD2EP93C43PIZ2NQ6SQLUIM3S16BGLHTY4IIEK5EB 1AE5D73A4",
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Validate 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.

Example: Capture Context Public Key

```
"jwk": {
    "kty": "RSA",
    "e": "AQAB",
    "use": "enc",
    "n":
 "3DhDtIHLxsbsSygEAG1hcFqnw64khTIZ6w9W9mZNl83gIyj1FVk-H5GDMa85e8RZFxUwgU_zQ0kHLtON

 o8SB52Z0hsJVE9wqHNIRoloiNPGPQYVXQZw2S1BSPxBtCEjA5x_-bcG6aeJdsz_cAE7OrIYkJa5Fphg9

 _pxgYRod6JCFjgdHj0iDSQxtBsmtxagAGHjDhW7UoiIig71SN-f-gggaCpITem4zlb5kkRVvmKMUANe4B

 36v4XSSSpwdP_H5kv4JDz_cVlp_Vy8T3AfAbCtROyRyH9iH1Z-4Yy6T5hb-9y3IPD8vlc8E3JQ4qt6U46
        EeiKPH4KtcdokMPjqiuQ",
    "kid": "00UaBe20jy9VkwZUQPZwNNoKFPJA4Qhc"
}
```

Use the capture context public key to cryptographically validate the JWT provided from a successful `/tokens` 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.

Using the Transient Token to Process a Payment

After you validate the transient token, you can use it in place of the PAN with payment services for 15 minutes. The transient token can be used multiple times within the 15-minute period.

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.

REST API with Transient Token JSON Web Token

```
"tokenInformation": {
     "transientTokenJwt":
 "eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFs

 ZyI6IlJTMjU2In0.eyJpc3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTA
uMS

 4wIiwiaWF0IjoxNjE0NzkxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPR
UhJ

 VldBSURPSkhLNjJJSFQxUVE1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4
TRI

 VZN-jpNbamcluAKVZ1zfdhbkrB6aNVWECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFf
zTu

 q41sDhlIqB-UatveU_2ukPxLYl87EX9ytpx4zCJVmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85
tl2

 OHaajEc25tS1FwH3jDOfjAC8mu2MEk-Ew0-ukZ70Ce7Zaq4cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bp
vcK
     Lr9k_8b9YnddIzp0p0JOCjXC_nuofQT7_x_-CQayx2czE0kD53HeNYC5hQ"
}
```

REST API withJSON Web Token ID

```
"tokenInformation": {
    "jti": "1E3GQY1RNKBG6IBD2EP93C43PIZ2NQ6SQLUIM3S16BGLHTY4IIEK5EB 1AE5D73A4",
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Using the Transient Token to Process a Payment

After you validate the transient token, you can use it in place of the PAN with payment services for 15 minutes. The transient token can be used multiple times within the 15-minute period.

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.

REST API with Transient Token JSON Web Token

```
"tokenInformation": {
     "transientTokenJwt":
 "eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFs

 ZyI6IlJTMjU2In0.eyJpc3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTA
uMS

 4wIiwiaWF0IjoxNjE0NzkxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPR
UhJ

 VldBSURPSkhLNjJJSFQxUVE1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4
TRI

 VZN-jpNbamcluAKVZ1zfdhbkrB6aNVWECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFf
zTu

 q41sDhlIqB-UatveU_2ukPxLYl87EX9ytpx4zCJVmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85
tl2
```

```
 OHaajEc25tS1FwH3jDOfjAC8mu2MEk-Ew0-ukZ70Ce7Zaq4cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bp
vcK
     Lr9k_8b9YnddIzp0p0JOCjXC_nuofQT7_x_-CQayx2czE0kD53HeNYC5hQ"
}
```

REST API with JSON Web Token ID

```
"tokenInformation": {
    "jti": "1E3GQY1RNKBG6IBD2EP93C43PIZ2NQ6SQLUIM3S16BGLHTY4IIEK5EB 1AE5D73A4",
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

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

```
<Header>.<Payload>.<Signature>
```

The header and payload is **Base64-encoded JSON** and contains these claims:

- **Header**: The algorithm and token type. For example:

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

- **Payload**: The claims of what the token represents. For example:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

- **Signature**: The signature is computed from the header and payload using a secret or private key.

Important: When working with JWTs, Cybersource recommends that you use a well-maintained JWT library to ensure proper decoding and parsing of the JWT.

Important: When parsing the JWT's JSON payload, you must ensure that you implement a robust solution for transversing JSON. Additional elements can be added to the JSON in future releases. Follow JSON parsing best practices to ensure that you can handle the addition of new data elements in the future.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Microform Integration v2

Microform Integration replaces the sensitive payment input fields of a client application with secure Cybersource-hosted fields. These fields securely accept payment information, including card and check data, and replaces it with a non-sensitive tokens.

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

Microform Integration provides the most secure method for tokenizing card and check 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.

Important: Each request that you send to Cybersource requires header information. For information about constructing the headers for your request, see the *Getting Started with REST Developer Guide*.

How It Works

You can use the Microform Integration JavaScript library to securely replace sensitive input fields with Cybersource-hosted secure iframes. These iframes capture payment information, including card numbers and eCheck data. This ensures that sensitive data is handled securely within your checkout process.

Accepting Card Information

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

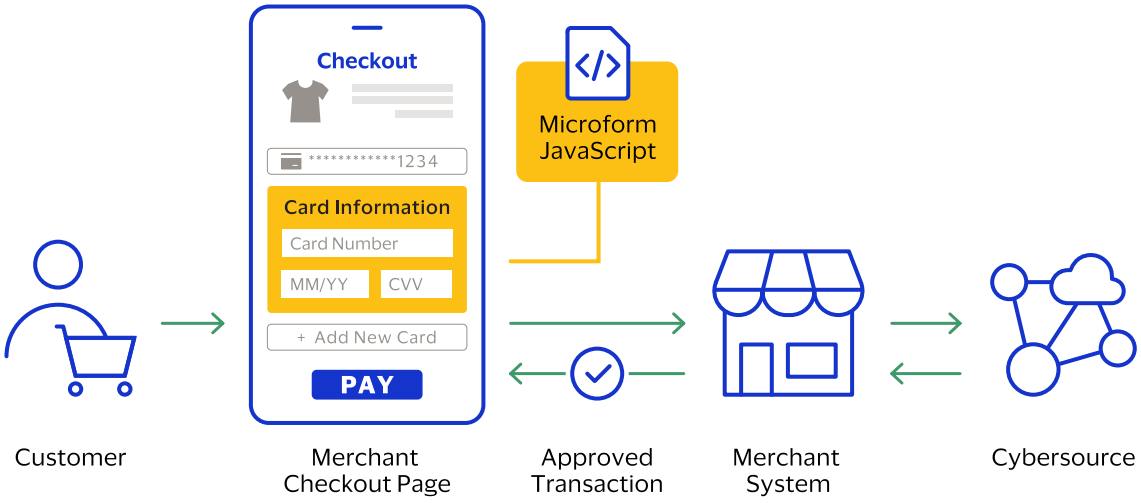
Microform Integration replaces the following card payment fields in your input form:

- Payment Card (PAN)
- CVN
- Month (non-sensitive)
- Year (non-sensitive)

With this option you can pass imonth and year in the request (if required), but these fields are non-sensitive.

This figure shows the Microform Integration process for accepting card information.

Microform Integration Process with Card Information



Accepting eCheck Information

Microform Integration also supports the acceptance of eCheck information. As with card transactions, the sensitive eCheck data is securely captured and replaced with a token.

Accepting eCheck information enables merchants to collect funds from a customer's bank account through both the ACH service and eCheck service (US only) for either of these flows:

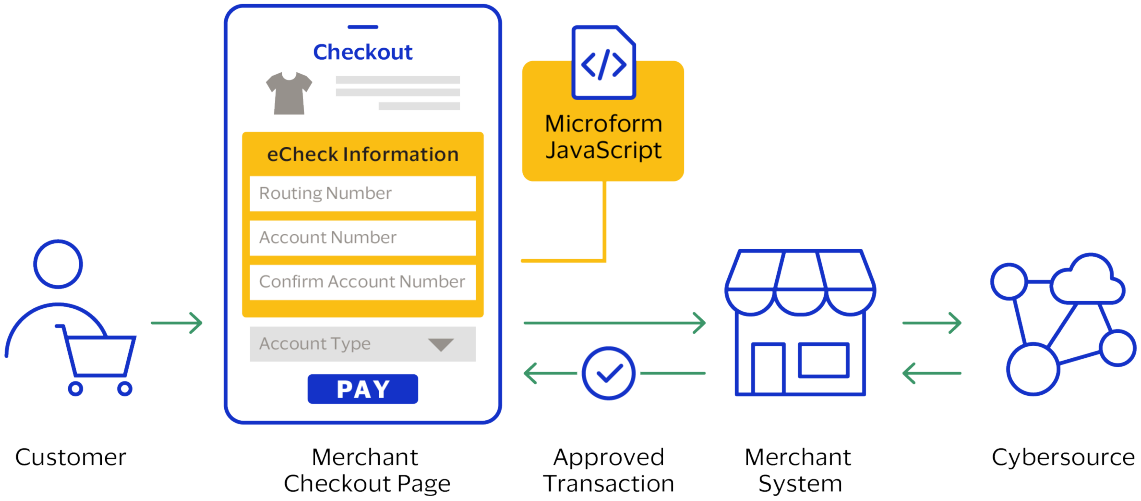
- **ACH services** are a set of connections composed of the legacy gateway solutions where Cybersource serves as the gateway.
- **eCheck**, the new service on Payments 2.0, is the acquirer solution where Cybersource is the acquirer.

Microform Integration replaces these eCheck information fields in your payment input form:

- Routing Number
- Account Number
- Account type (non-sensitive)

This figure shows the Microform Integration process for accepting eCheck information.

Microform Integration Process with eCheck Information



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 information as follows:

- Card information input fields:
  - Payment card or PAN
  - CVN
- eCheck information input fields:
  - Routing number
  - Account number

These iframes are hosted by Microform Integration, and the payment data is submitted directly to Cybersource through the secure Flex API v2 suite. This means that this data never passes through your systems.

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:
    - Card information input fields:
        - Payment card or PAN
        - CVN
    - eCheck information input fields:
        - Routing number
        - Account number

Accept Card Information

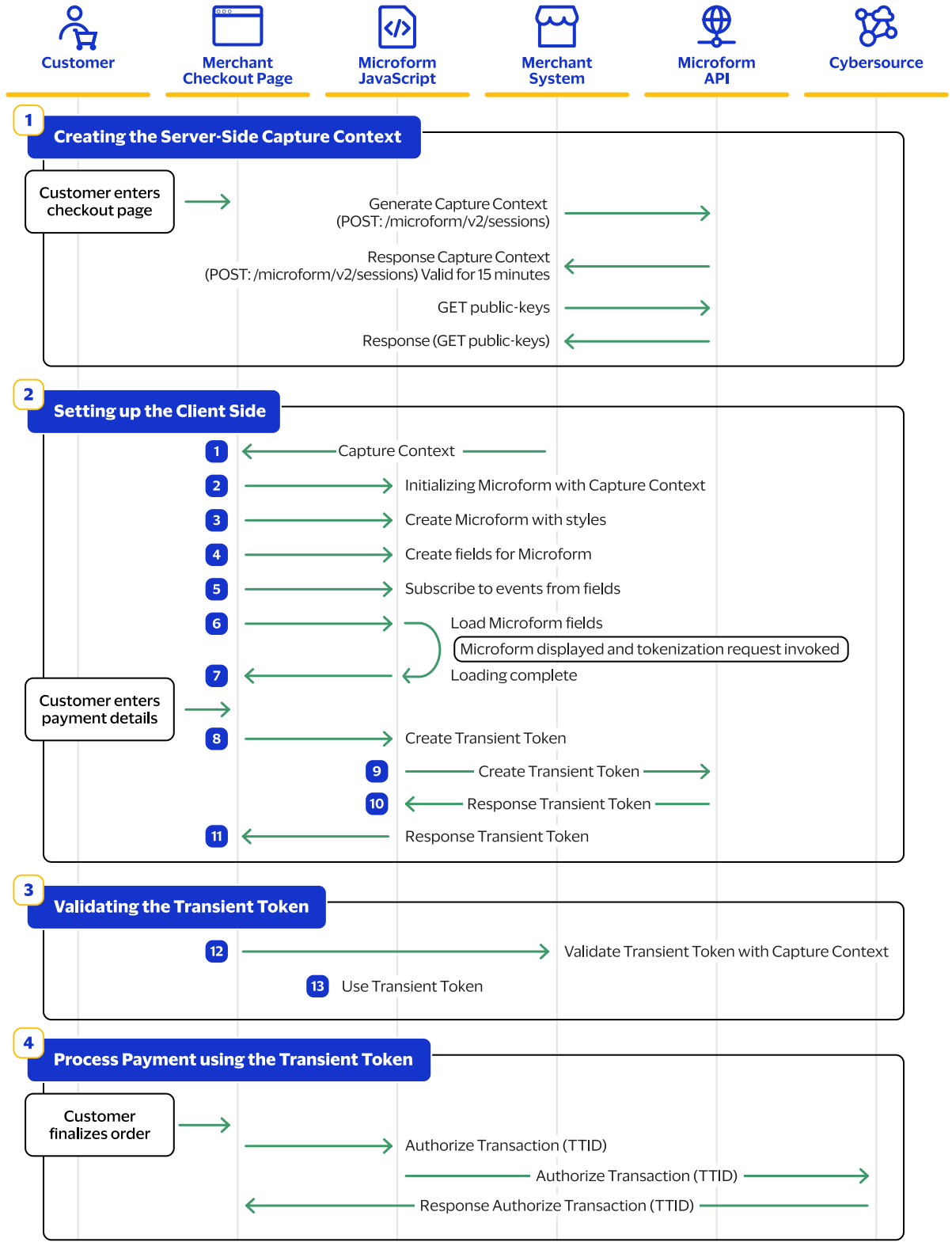
This section covers the implementation steps needed to complete the server-side and client-side setup for accepting card information with Microform Integration.

Implementing Microform Integration is a three-step process:

1. Creating the Server-Side Capture Context (on page 48)
2. Client-Side Setup (on page 52)
3. Validating the Transient Token (on page 59)

This figure shows the flow for implementing Microform Integration:

Microform Integration Implementation Workflow



Server-Side Setup

This section contains the information you need to set up your server. Initializing Microform Integration within your webpage begins with a server-to-server call to the sessions API. This step authenticates your merchant credentials and establishes how the Microform Integration front-end components will function. The sessions API request contains parameters that define how Microform Integration 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 about JWTs, see JSON Web Tokens (on page 127).

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 front-end JavaScript library how to behave within your payment experience. The request provides authentication, one-time keys, the target origin to the Microform Integration, in addition to allowed card networks and payment types (card or check).

These fields are available for requesting the capture context for accepting card information:

Required fields:

allowedCardNetworks

clientVersion

targetOrigins

Optional fields:

allowedPaymentTypes

transientTokenResponseOptions

For information about JWTs, see JSON Web Tokens (on page 127).

For more information on requesting the capture context, see Capture Context API (on page 106).

Creating the Server-Side Capture Context

The first step in integrating with Microform Integration is to develop the server-side code that generates the capture context. The capture context is also known as a *session*.

You can use the SDK or call the API directly to generate the capture context.

To use the SDK to generate the capture context, use the sample code here: Flex Samples on Github.

Follow these steps to call the API directly to generate the capture context:

1. Send an authenticated POST request to the `/sessions` endpoint to create your capture context session:

   - **Production**: `https://api.cybersource.com/microform/v2/sessions`
   - **Test**: `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. You must also include the type of Microform Integration you want to include in the capture context for accepting card information. If you do not include the **allowedPaymentTypes** field in your capture request, the value defaults to `CARD`.

   For example:

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

   To embed the target origin URL within multiple nested iframes, you must specify the origins of all the browser contexts used. For example:

   ```
   {
    "clientVersion": "v2",
    "targetOrigins": ["https://www.example.com",
    "https://www.basket.example.com", "https://ecom.example.com"],
    "allowedCardNetworks": ["VISA",
     "MASTERCARD",
     "AMEX",
     "CARTESBANCAIRES",
     "CARNET",
     "CUP",
     "DINERSCLUB",
     "DISCOVER",
     "EFTPOS",
   ```

```
  "ELO",
              "JAYWAN",
  "JCB",
  "JCREW",
              "KSCP",
  "MADA",
  "MAESTRO",
  "MEEZA",
              "PAYPAK",
              "UATP"
 ],
 "allowedPaymentTypes": ["CARD"]
}
```

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

```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2Vuc
yIsImRhdGEiOiJ1Q0RERW94M2dDQk1VaHI2T1ZDVGt4QUFFS1pHSTRHcDFvQ2pyYXlVb1MxQzdGOX
E2WFpyYXhGbGxMVDMvenE2cjFnNXoxS1U2UDZseldqRVFTVVJoZUtxUThoVWJkZVNNdmt5SERTTXU
wV01tMzhcdTAwM2QiLCJvcmlnaW4iOiJodHRwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20i
LCJqd2siOnsia3R5IjoiUlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJuIjoiMXNDY3NZNC1WZ
TNWU0VKekhnelJ5WjVDOURrM0VHZ2ZPOGd5SDc5bVJfSlN6NzdmWTdfV1loM3psdTkyTFVfeU5KVT
BUMzdOQmVzd0szU2c0YnRNaU41Q0FCbWNXLWNSckhta2k0MVZoNUZRMmtjcWZSSlgxNVhZN1A3R25
GTnd4QzVkUG9UM29NM1czRFVHaUMyYW56enhIN3pNNlA3N2hFbnc2TkZHSXlBdXhJRWFwRG9DaXlE
VW5NdFRwV2lBV3YzTF9OVHZOaHRkVE4tNm1GRWU1RmdVYmlzeWtrTzlWMHZaS0d6SWRWWmdTdE42c
HlnUGhVbnlNXzJIVmIxQmkyWjNKaElhZDFLUW02SGl0NklwYjNyUTBHRWZsN0ZWOUV3NGZyNzJpek
Q0WVg2WHo0V3ZuMzlLN3J3WkhCRXdNM3l5Wl9ELTBUbjM1MFhvUlBUVjB3Iiwia2lkIjoiMDB4V1A
1eUh1UE1kNkFkNHdwVzNzQkt1bWFaQ01zYWMifX0sImN0eCI6W3siZGF0YSI6eyJjbGllbnRMaWJy
YXJ5SW50ZWdyaXR5Ijoic2hhMjU2LXZkWWkxaDV1ZTNwcm5iVC8xYThJSkxlUkNrSGVqSHBkRGR3M
y95RkxaREFcdTAwM2QiLCJjbGllbnRMaWJyYXJ5IjoiaHR0cHM6Ly9zdGFnZWZsZXguY3liZXJzb3
VyY2UuY29tL21pY3JvZm9ybS9idW5kbGUvdjIuNS4xL2ZsZXgtbWljcm9mb3JtLm1pbi5qcyIsImF
sbG93ZWRDYXJkTmV0d29ya3MiOlsiVklTQSIsIk1BU1RFUkNBUkQiLCJBTUVYIiwiTUFFU1RSTyIs
IkRJU0NPVkVSIiwiRElORVJTQ0xVQiIsIkpDQiIsIkNVUCIsIkNBUlRFU0JBTkNBSVJFUyJdLCJ0Y
XJnZXRPcmlnaW5zIjpbImh0dHBzOi8vdGhlLXVwLWRlbW8uYXBwc3BvdC5jb20iXSwibWZPcmlnaW
4iOiJodHRwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJhbGxvd2VkUGF5bWVudFR5cGV
zIjpbIkNBUkQiXX0sInR5cGUiOiJtZi0yLjEuMCJ9XSwiaXNzIjoiRmxleCBBUEkiLCJleHAiOjE3
MzY0MzA0MTQsImlhdCI6MTczNjQyOTUxNCwianRpIjoiZDVZbzVhNU0wWFBPQ1BxZiJ9.G4Ea-gIk
6SG5ULE4NE5OsdPI41YaAuTEMHDstBgkFzczIWwzJScvXs4hgWiyA-1ZLGITedlumGj-0x8jxmYTW
eTm7D0fP8RL0w148EpDLMD8xMHpAJMdMqZTmYHyichsy8uOZKVOn9NbnuQqfDeQS_rLpJV3tMe2Nw
JL3RdBXdJ894ihKpFP2yXE1wQeLekNiYJ6s-Uuxwf0jf2CSN_TJAjnfVR6bqlpWbUpiUaBLcqDsHH
e_pcrd5g2r-1LEfCiOV9RIw7844XKFNLQZvt_alQjItuMy8M9LVhnlRWCSnTKB1iV1RUxuTWtMzTv
HmQWPx4nShqzE3j0Hp61c0PmBw
```

Important:

- 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` field in any external requests. Hard code it on the server side.

For more information on requesting the capture context, see Capture Context (on page 47).

Validating the Server-Side 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: Depending on the cryptographic method you use to validate the public key, you might need to convert the key to privacy-enhanced mail (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

```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsIm
RhdGEiOiJ1Q0RERW94M2dDQk1VaHI2T1ZDVGt4QUFFS1pHSTRHcDFvQ2pyYXlVb1MxQzdGOXE2WFpyYXhG
bGxMVDMvenE2cjFnNXoxS1U2UDZseldqRVFTVVJoZUtxUThoVWJkZVNNdmt5SERTTXUwV01tMzhcdTAwM2
QiLCJvcmlnaW4iOiJodHRwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJqd2siOnsia3R5Ijoi
UlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJuIjoiMXNDY3NZNC1WZTNWU0VKekhnelJ5WjVDOURrM0
VHZ2ZPOGd5SDc5bVJfSlN6NzdmWTdfV1loM3psdTkyTFVfeU5KVTBUMzdOQmVzd0szU2c0YnRNaU41Q0FC
bWNXLWNSckhta2k0MVZoNUZRMmtjcWZSSlgxNVhZN1A3R25GTnd4QzVkUG9UM29NM1czRFVHaUMyYW56en
hIN3pNNlA3N2hFbnc2TkZHSXlBdXhJRWFwRG9DaXlEVW5NdFRwV2lBV3YzTF9OVHZOaHRkVE4tNm1GRWU1
RmdVYmlzeWtrTzlWMHZaS0d6SWRWWmdTdE42cHlnUGhVbnlNXzJIVmIxQmkyWjNKaElhZDFLUW02SGl0Nk
lwYjNyUTBHRWZsN0ZWOUV3NGZyNzJpekQ0WVg2WHo0V3ZuMzlLN3J3WkhCRXdNM3l5Wl9ELTBUbjM1MFhv
UlBUVjB3Iiwia2lkIjoiMDB4V1A1eUh1UE1kNkFkNHdwVzNzQkt1bWFaQ01zYWMifX0sImN0eCI6W3siZG
F0YSI6eyJjbGllbnRMaWJyYXJ5SW50ZWdyaXR5Ijoic2hhMjU2LXZkWWkxaDV1ZTNwcm5iVC8xYThJSkxl
UkNrSGVqSHBkRGR3My95RkxaREFcdTAwM2QiLCJjbGllbnRMaWJyYXJ5IjoiaHR0cHM6Ly9zdGFnZWZsZX
guY3liZXJzb3VyY2UuY29tL21pY3JvZm9ybS9idW5kbGUvdjIuNS4xL2ZsZXgtbWljcm9mb3JtLm1pbi5q
cyIsImFsbG93ZWRDYXJkTmV0d29ya3MiOlsiVklTQSIsIk1BU1RFUkNBUkQiLCJBTUVYIiwiTUFFU1RSTy
IsIkRJU0NPVkVSIiwiRElORVJTQ0xVQiIsIkpDQiIsIkNVUCIsIkNBUlRFU0JBTkNBSVJFUyJdLCJ0YXJn
ZXRPcmlnaW5zIjpbImh0dHBzOi8vdGhlLXVwLWRlbW8uYXBwc3BvdC5jb20iXSwibWZPcmlnaW4iOiJodH
RwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJhbGxvd2VkUGF5bWVudFR5cGVzIjpbIkNBUkQi
XX0sInR5cGUiOiJtZi0yLjEuMCJ9XSwiaXNzIjoiRmxleCBBUEkiLCJleHAiOjE3MzY0MzA0MTQsImlhdC
I6MTczNjQyOTUxNCwianRpIjoiZDVZbzVhNU0wWFBPQ1BxZiJ9.G4Ea-gIk6SG5ULE4NE5OsdPI41YaAuT
EMHDstBgkFzczIWwzJScvXs4hgWiyA-1ZLGITedlumGj-0x8jxmYTWeTm7D0fP8RL0w148EpDLMD8xMHpA
JMdMqZTmYHyichsy8uOZKVOn9NbnuQqfDeQS_rLpJV3tMe2NwJL3RdBXdJ894ihKpFP2yXE1wQeLekNiYJ
6s-Uuxwf0jf2CSN_TJAjnfVR6bqlpWbUpiUaBLcqDsHHe_pcrd5g2r-1LEfCiOV9RIw7844XKFNLQZvt_a
lQjItuMy8M9LVhnlRWCSnTKB1iV1RUxuTWtMzTvHmQWPx4nShqzE3j0Hp61c0PmBw
```

Parse the JWT 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"
}
```


Related Information

- *Introduction to JSON Web Tokens*
- IETF RFC 7515: JSON Web Signature (JWS) Signature
- IETF RFC 7517: JSON Web Key (JWK)

Client-Side Setup

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. Decode the JWT from the `/microform/v2/sessions` response to get the capture context.
2. Use the **clientLibrary** and **clientLibraryIntegrity** values that are returned in the JWT from `/microform/v2/sessions` response to obtain the Microform Integration JavaScript library URL and integrity value that you use to create your script tags.

> **Important:** You must use these values for every transaction. These values can be unique for each transaction.

Important: Do not hard code these values, because doing so can result in client-side Microform Integration errors. If you do not hard code these values for all transactions, you can load a JavaScript library that is incompatible with the version you requested in the `/sessions` request.

Example `/sessions` Response:

```
"data":{ "clientLibrary":"[EXTRACT clientLibrary VALUE from here]",
 "clientLibraryIntegrity": "[EXTRACT clientLibraryIntegrity VALUE from
 here]" }
```

Example Script Tags

```
<script src="[INSERT clientLibrary VALUE HERE]" integrity=”[INSERT
 clientLibraryIntegrity VALUE HERE]” crossorigin=”anonymous”> </script>
```

Example Code for Loading the Script Dynamically Following JWT Extraction Tags

```
// values read from capture context
const clientLibrary = " ";
const clientLibraryIntegrity = " ";
// create script tag
const script = document.createElement("script");
script.src = clientLibrary;
script.type = "text/javascript";
script.async = true;
script.integrity = clientLibraryIntegrity;
script.crossOrigin = "anonymous";
// run setup code after script has loaded successfully
script.onload = function () {
  // SETUP
};
// insert to document
document.head.appendChild(script);
```

3. 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. This 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>`.

Example: Accept Card Information Checkout Form

```
<h1>Checkout Page</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>
```

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

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

5. 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:

   ```
   const microform = flex.microform("card", { styles: myStyles });
   ```

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

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

7. 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 a native payment acceptance mobile application, follow the steps for web page set up, and ensure that these additional requirements are met:

- The card information acceptance fields of PAN and CVV must be hosted on a web page.
- The eCheck information acceptance fields of routing number, account number, and confirmed account number must be hosted on a web page.
- The native application must load the hosted card entry form web page in a web view.

Transient Tokens for Accepting Card Information

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

Transient Token Time Limit

The sensitive data associated with the transient token is available for use in API requests for a 15-minute duration. The transient token can be used multiple times within the 15-minute period. After 15 minutes, you must prompt the customer to restart the checkout flow.

Example: Creating the Pay Button with Event Listener for Accepting Card Information

```
const button = document.querySelector("#myButton");

button.addEventListener("click", function () {
  // Compiling MM & YY into optional parameters
  const options = {
    expirationMonth: document.querySelector("#expMonth").value,
    expirationYear: document.querySelector("#expYear").value,
  };
  //
  microform.createToken(options, function (err, token) {
    // handle err
    if (err) {
      console.error(err);
      errorsOutput.textContent = err.message;
      return;
    }

    // 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();
  });
});
```

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 Account Type is 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 card information when you use supported payment services.

Example: Customer-Submitted Form for Accepting Card Information

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

```
   const expYear = document.querySelector('#expYear');
   const errorsOutput = document.querySelector('#errors-output');

   // the capture context that was requested server-side for this transaction
   const captureContext = <% -keyInfo %> ;
   // custom styles that will be applied to each field we create using Microform
   const 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
   const flex = new Flex(captureContext);
   const microform = flex.microform({ styles: myStyles });
   const number = microform.createField('number', { placeholder: 'Enter card
number' });
   const 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 parameters
       const 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>
```

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. For more information, see JSON Web Tokens (on page 127).

Important: When you integrate 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.

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.

Example: Token Payload for Accepting Card Information

```
{
  "iss": "Flex/00",
  "exp": 1728911080,
  "type": "mf-2.0.0",
  "iat": 1728910180,
  "jti": "1D1S6JK9RL6EK667H1I370689A63I2I8YLFJSPJ1EUSKIPMJJWEL670D16E89AF8",
  "content": {
    "paymentInformation": {
```

```
      "card": {
        "expirationYear": {
          "value": "2025"
        },
        "number": {
          "detectedCardTypes": [
            "001"
          ],
          "maskedValue": "XXXXXXXXXXXX1111",
          "bin": "411111"
        },
        "securityCode": {},
        "expirationMonth": {
          "value": "01"
        }
      }
    }
  }
}
```

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.

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"    }
```

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.

Using the Transient Token to Process a Payment

After you validate the transient token, you can use it in place of the PAN with payment services for 15 minutes. The transient token can be used multiple times within the 15-minute period.

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",
	
	}

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":
 "eyJraWQiOiIwN0JwSE9abkhJM3c3UVAycmhNZkhuWE9XQlhwa1ZHTiIsImFsZyI6IlJTMjU2In0.eyJk
YXRhIjp7ImV4cGlyYXRpb25ZZWFyIjoiMjAyMCIsIm51bWJlciI6IjQxMTExMVhYWFhYWDExMTEiLCJleH
BpcmF0aW9uTW9udGgiOiIxMCIsInR5cGUiOiIwMDEifSwiaXNzIjoiRmxleC8wNyIsImV4cCI6MTU5MTc0
NjAyNCwidHlwZSI6Im1mLTAuMTEuMCIsImlhdCI6MTU5MTc0NTEyNCwianRpIjoiMUMzWjdUTkpaVjI4OV
M5MTdQM0JHSFM1T0ZQNFNBRERCUUtKMFFKMzMzOEhRR0MwWTg0QjVFRTAxREU4NEZDQiJ9.cfwzUMJf115
K2T9-wE_A_k2jZptXlovls8-fKY0muO8YzGatE5fu9r6aC4q7n0YOvEU6G7XdH4ASG32mWnYu-kKlqN4IY
_cquRJeUvV89ZPZ5WTttyrgVH17LSTE2EvwMawKNYnjh0lJwqYJ51cLnJiVlyqTdEAv3DJ3vInXP1YeQjL
X5_vF-OWEuZfJxahHfUdsjeGhGaaOGVMUZJSkzpTu9zDLTvpb1px3WGGPu8FcHoxrcCGGpcKk456AZgYMB
SHNjr-pPkRr3Dnd7XgNF6shfzIPbcXeWDYPTpS4PNY8ZsWKx8nFQIeROMWCSxIZOmu3Wt71KN9iK6DfOPr
o7w"
    }
}
```

Accept eCheck Information

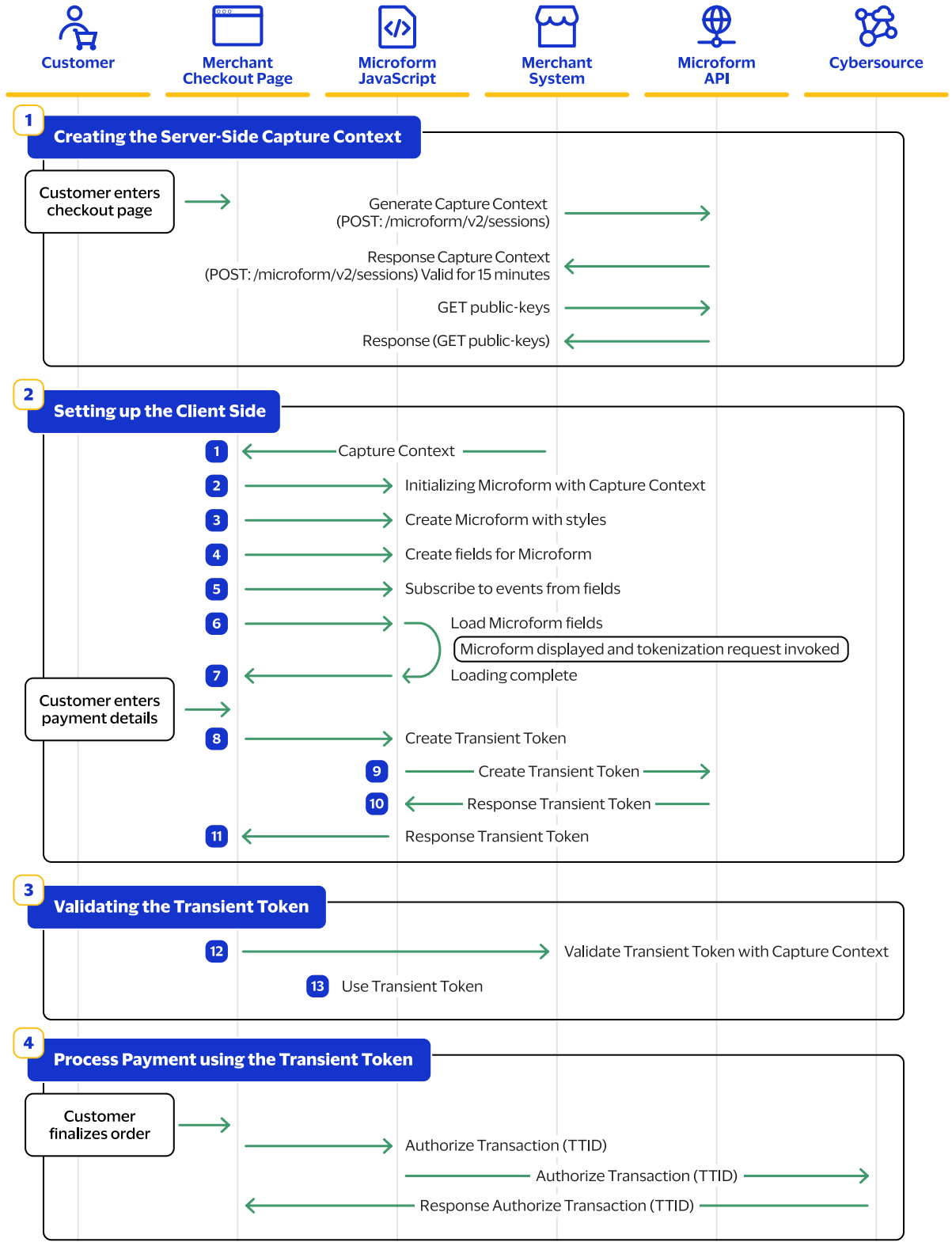
This section covers the implementation steps needed to complete the server-side and client-side setup for accepting eCheck information with Microform Integration.

Implementing Microform Integration is a three-step process:

1. Creating the Server-Side Capture Context (on page 65)
2. Client-Side Setup (on page 69)
3. Validating the Transient Token (on page 76)

This figure shows the flow for implementing Microform Integration:

Microform Integration Implementation Workflow



Server-Side Setup

This section contains the information you need to set up your server. Initializing Microform Integration within your webpage begins with a server-to-server call to the sessions API. This step authenticates your merchant credentials and establishes how the Microform Integration front-end components will function. The sessions API request contains parameters that define how Microform Integration 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 about JWTs, see JSON Web Tokens (on page 127).

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 front-end JavaScript library how to behave within your payment experience. The request provides authentication, one-time keys, the target origin to the Microform Integration, in addition to allowed card networks and payment types.

These fields are available for requesting the capture context for accepting eCheck information:

Required fields:

allowedPaymentTypes

clientVersion

targetOrigins

Optional fields:

allowedCardNetworks

transientTokenResponseOptions

For information about JWTs, see JSON Web Tokens (on page 127).

For more information on requesting the capture context, see Capture Context API (on page 106).

Creating the Server-Side Capture Context

The first step in integrating with Microform Integration is to develop the server-side code that generates the capture context. The capture context is also known as a session.

You can use the SDK or call the API directly to generate the capture context.

To use the SDK to generate the capture context, use the sample code here: Flex Samples on Github.

Follow these steps to call the API directly to generate the capture context:

1. Send an authenticated POST request to the `/sessions` endpoint to create your capture context session:

   - **Production**: `https://api.cybersource.com/microform/v2/sessions`
   - **Test**: `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. You must also include the type of Microform Integration you want to include in the capture context for accepting eCheck information. If you do not include the **allowedPaymentTypes** field in your capture request, the value defaults to `CARD`.

   For example:

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

   To embed within multiple nested iframes, you must specify the origins of all the browser contexts used. For example:

```
{
 "clientVersion": "v2",
 "targetOrigins": [
  "https://www.example.com",
  "https://www.basket.example.com",
  "https://ecom.example.com"
 ],
```

```
 "allowedCardNetworks": [
  "VISA",
  "MASTERCARD",
  "AMEX",
  "CARTESBANCAIRES",
  "CARNET",
  "CUP",
  "DINERSCLUB",
  "DISCOVER",
  "EFTPOS",
  "ELO",
  "JCB",
  "JCREW",
  "MADA",
  "MAESTRO",
  "MEEZA",
              "PAYPAK"
 ],
 "allowedPaymentTypes": [
  "CHECK"
 ]
}
```

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

```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2Vuc
yIsImRhdGEiOiJtdnkwVk9OVk40bzA4bTRGQjhmU3FCQUFFS0JWOTdlNnR2VDd4cHdqaFkwMDRydF
J1dGI0R2YwWlNwNGdNeEkvanBVSWxFblZKa2JtUVNHaWFnUEdGc0NPazdMbHJGTHFKcXN2eitoTHh
rY08xRkFcdTAwM2QiLCJvcmlnaW4iOiJodHRwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20i
LCJqd2siOnsia3R5IjoiUlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJuIjoidmRpN0gtM1MzM
TkyZlc5WC1BTmpvdjlFdXU4ZGxPOTBtU2gyUGVyMF9PdHZ4YlJITTBrakZpTHlKaGQwUUR3VlNWbU
lhRFc2aGtCa1k2Ui1lcWRnaTdUVUNGZEQ3UUU1ckNkZGhZZTIycTh0RUNQZkpOWWJ6STZZTVBxTkF
yYWc5LUhhWVo1X2tOX0JvMm5EclN4RFJ0MHBDbGxyd2d2Q1ZLb2M0RWF6ZE93QUE4dnI2VVh4Ty1S
WVI2Z1R5VEZia244Q2hDVHNvWDByam5VWVI1VjdRaE95YzMzWEJUTVNDYTVBOHFQNDZnZXpvQjZ0d
DA0SlQtRVVMWE9vYndVcVdvd0E3TTJzWUYydkFoQkVuMmt0REJFWVJSN3E0aWEyVHRIS1JPUW9FTj
hZNjNiNFNaTGZDQk82cEc2QXpnSWpya3RkQXhIOXR0WURYdFJYS1YxeTN3Iiwia2lkIjoiMDBiQlN
1d3VpdGtYeExxOGFISWloMm5qMFhQNFpXYUsifX0sImN0eCI6W3siZGF0YSI6eyJjbGllbnRMaWJy
YXJ5SW50ZWdyaXR5Ijoic2hhMjU2LXZkWWkxaDV1ZTNwcm5iVC8xYThJSkxlUkNrSGVqSHBkRGR3M
y95RkxaREFcdTAwM2QiLCJjbGllbnRMaWJyYXJ5IjoiaHR0cHM6Ly9zdGFnZWZsZXguY3liZXJzb3
VyY2UuY29tL21pY3JvZm9ybS9idW5kbGUvdjIuNS4xL2ZsZXgtbWljcm9mb3JtLm1pbi5qcyIsInR
hcmdldE9yaWdpbnMiOlsiaHR0cHM6Ly90aGUtdXAtZGVtby5hcHBzcG90LmNvbSJdLCJtZk9yaWdp
biI6Imh0dHBzOi8vc3RhZ2VmbGV4LmN5YmVyc291cmNlLmNvbSIsImFsbG93ZWRQYXltZW50VHlwZ
XMiOlsiQ0hFQ0siXX0sInR5cGUiOiJtZi0yLjEuMCJ9XSwiaXNzIjoiRmxleCBBUEkiLCJleHAiOj
```

```
E3MzM0OTAxODEsImlhdCI6MTczMzQ4OTI4MSwianRpIjoiSXdEdHAxZkVZM2QwYUh6OSJ9.arokac
vdTSUIehBY0ICi-QYynhFj7_0k-G39qbkNJydB3UyF2qJSaqwZiopO27kuqk8u9Z0cY-V9Nu04Jga
V4s18doxnzx6vdTCC3krrIcxeINi23Qu-Szcpg7aaGvPVXMC0DVC14WUQiGJkOakJ54jWtl2VoFAg
YziUMcYYpk4hxLVxurBtT7lvrfCXKoyWtxiUxoEpOc_Td_qi5nA8ByWUaieQmp1Zej61khQJ_hmXt
lsAt4BqxeJWoJeR_5Sjz0vD5y4-oAeNNrAulDem7CKiRJQbI9fyqT-
```

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.

For more information on requesting the capture context, see Capture Context (on page 64).

Validating the Server-Side 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: Depending on the cryptographic method you use to validate the public key, you might need to convert the key to privacy-enhanced mail (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

```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsIm
RhdGEiOiJtdnkwVk9OVk40bzA4bTRGQjhmU3FCQUFFS0JWOTdlNnR2VDd4cHdqaFkwMDRydFJ1dGI0R2Yw
WlNwNGdNeEkvanBVSWxFblZKa2JtUVNHaWFnUEdGc0NPazdMbHJGTHFKcXN2eitoTHhrY08xRkFcdTAwM2
QiLCJvcmlnaW4iOiJodHRwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJqd2siOnsia3R5Ijoi
UlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJuIjoidmRpN0gtM1MzMTkyZlc5WC1BTmpvdjlFdXU4ZG
xPOTBtU2gyUGVyMF9PdHZ4YlJITTBrakZpTHlKaGQwUUR3VlNWbUlhRFc2aGtCa1k2Ui1lcWRnaTdUVUNG
ZEQ3UUU1ckNkZGhZZTIycTh0RUNQZkpOWWJ6STZZTVBxTkFyYWc5LUhhWVo1X2tOX0JvMm5EclN4RFJ0MH
BDbGxyd2d2Q1ZLb2M0RWF6ZE93QUE4dnI2VVh4Ty1SWVI2Z1R5VEZia244Q2hDVHNvWDByam5VWVI1VjdR
aE95YzMzWEJUTVNDYTVBOHFQNDZnZXpvQjZ0dDA0SlQtRVVMWE9vYndVcVdvd0E3TTJzWUYydkFoQkVuMm
t0REJFWVJSN3E0aWEyVHRIS1JPUW9FTjhZNjNiNFNaTGZDQk82cEc2QXpnSWpya3RkQXhIOXR0WURYdFJY
S1YxeTN3Iiwia2lkIjoiMDBiQlN1d3VpdGtYeExROGFISWloMm5qMFhQNFpXYUsifX0sImN0eCI6W3siZG
F0YSI6eyJjbGllbnRMaWJyYXJ5SW50ZWdyaXR5Ijoic2hhMjU2LXZkWWkxaDV1ZTNwcm5iVC8xYThJSkxl
UkNrSGVqSHBkRGR3My95RkxaREFcdTAwM2QiLCJjbGllbnRMaWJyYXJ5IjoiaHR0cHM6Ly9zdGFnZWZsZX
guY3liZXJzb3VyY2UuY29tL21pY3JvZm9ybS9idW5kbGUvdjIuNS4xL2ZsZXgtbWljcm9mb3JtLm1pbi5q
cyIsInRhcmdldE9yaWdpbnMiOlsiaHR0cHM6Ly90aGUtdXAtZGVtby5hcHBzcG90LmNvbSJdLCJtZk9yaW
dpbiI6Imh0dHBzOi8vc3RhZ2VmbGV4LmN5YmVyc291cmNlLmNvbSIsImFsbG93ZWRQYXltZW50VHlwZXMi
OlsiQ0hFQ0siXX0sInR5cGUiOiJtZi0yLjEuMCJ9XSwiaXNzIjoiRmxleCBBUEkiLCJleHAiOjE3MzM0OT
AxODEsImlhdCI6MTczMzQ4OTI4MSwianRpIjoiSXdEdHAxZkVZM2QwYUh6OSJ9.arokacvdTSUIehBY0IC
i-QYynhFj7_0k-G39qbkNJydB3UyF2qJSaqwZiopO27kuqk8u9Z0cY-V9Nu04JgaV4s18doxnzx6vdTCC3
krrIcxeINi23Qu-Szcpg7aaGvPVXMC0DVC14WUQiGJkOakJ54jWtl2VoFAgYziUMcYYpk4hxLVxurBtT7l
vrfCXKoyWtxiUxoEpOc_Td_qi5nA8ByWUaieQmp1Zej61khQJ_hmXtlsAt4BqxeJWoJeR_5Sjz0vD5y4-o
AeNNrAulDem7CKiRJQbI9fyqT-
```

Parse the JWT 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"
}
```


Related Information

- *Introduction to JSON Web Tokens*
- IETF RFC 7515: JSON Web Signature (JWS) Signature
- IETF RFC 7517: JSON Web Key (JWK)

Client-Side Setup

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. Decode the JWT from the `/microform/v2/sessions` response to get the capture context.
2. Use the **clientLibrary** and **clientLibraryIntegrity** values that are returned in the JWT from `/microform/v2/sessions` response to obtain the Microform Integration JavaScript library URL and integrity value that you use to create your script tags.

   > **Important:** You must do this for every transaction as these values can be unique for each transaction. Do not hard code these values, as this can result in client-side Microform Integration errors. If you do not do this for every transaction, you may load a JavaScript library that is incompatible with the version you requested in the `/sessions` request.

   Example `/sessions` Response:

```
"data":{ "clientLibrary":"[EXTRACT clientLibrary VALUE from here]",
 "clientLibraryIntegrity": "[EXTRACT clientLibraryIntegrity VALUE from
 here]" }
```

Example Script Tags

```
<script src="[INSERT clientLibrary VALUE HERE]" integrity=”[INSERT
 clientLibraryIntegrity VALUE HERE]” crossorigin=”anonymous”> </script>
```

Example Code for Loading the Script Dynamically Following JWT Extraction Tags

```
// values read from capture context
const clientLibrary = " ";
const clientLibraryIntegrity = " ";
// create script tag
const script = document.createElement("script");
script.src = clientLibrary;
script.type = "text/javascript";
script.async = true;
script.integrity = clientLibraryIntegrity;
script.crossOrigin = "anonymous";
// run setup code after script has loaded successfully
script.onload = function () {
  // SETUP
};
// insert to document
document.head.appendChild(script);
```

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

   Within your HTML checkout, replace the routing number, account number, and confirm account number tag with a simple container. Microform Integration uses the container to render an iframe for secured credit card input. This example contains simple `div` tags to define where to place the routing number, account number, and confirm account number fields within the payment acceptance page: `<div id="number-container" class="form-control"></div>`.

   Example: Accept eCheck Information Checkout Form

```
<h1>Checkout Page</h1>
<div id="errors-output" role="alert"></div>
<form action="/token" id="my-sample-form" method="post">
    <div class="form-group">
        <label id="routingNumber-label">Routing Number</label>
        <div id="routingNumber-container" class="form-control"></div>
```

```
        <label for="accountNumber-label">Account Number</label>
        <div id="accountNumber-container" class="form-control"></div>
        <label for="accountNumberConfirm-label">Account Number
 Confirm</label>
        <div id="accountNumberConfirm-container" class="form-control"></div>
    </div>

    <div class="form-row">
        <div class="form-group col-md-6">
            <label for="accountType">Account Type</label>
            <select id="accountType" name="accountType" class="form-control">
                <option value="C">Checking</option>
                <option value="S">Savings</option>
                <option value="X">Corporate checking</option>
            </select>
        </div>
    </div>

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

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

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

5. 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:

   ```
   const microform = flex.microform("check", { styles: myStyles });
   ```

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

   ```
   const routingNumber = microform.createField("routingNumber", { placeholder:
    "Enter routing number" });
   const accountNumber = microform.createField("accountNumber", { placeholder:
    "Enter account number" });
   const accountNumberConfirm = microform.createField("accountNumberConfirm",
    { placeholder: "accountNumberConfirm" });

   routingNumber.load('#routingNumber-container');
   ```

```
accountNumber.load('#accountNumber-container');
accountNumberConfirm.load('#accountNumberConfirm-container');
```

7. 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 a native payment acceptance mobile application, follow the steps for web page setup, and ensure that these additional requirements are met:

- The card information acceptance fields of routing number, account number, and confirm account number must be hosted on a web page.
- The eCheck information acceptance fields of routing number, account number, and confirmed account number must be hosted on a web page.
- The native application must load the hosted eCheck entry form web page in a web view.

Transient Tokens for Accepting eCheck Information

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

Transient Token Time Limit

The sensitive data associated with the transient token is available for use in API requests for a 15-minute duration. The transient token can be used multiple times within the 15-minute period. After 15 minutes, you must prompt the customer to restart the checkout flow.

Example: Creating the Pay Button with Event Listener for Accepting eCheck Information

```
const button = document.querySelector("#myButton");

button.addEventListener("click", function () {
  // Compiling accounttype into optional parameters
  const options = {
    accountType: document.querySelector(“#accountType”).value,
  };
```

```
  //
  microform.createToken(options, function (err, token) {
    // handle err
    if (err) {
      console.error(err);
      errorsOutput.textContent = err.message;
      return;
    }

    // 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();
  });
});
```

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. 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 eCheck information when you use supported payment services.

Example: Customer-Submitted Form for Accepting eCheck Information

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

    // the capture context that was requested server-side for this transaction
    const captureContext = <% -keyInfo %> ;
    // custom styles that will be applied to each field we create using Microform
    const 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
    const flex = new Flex(captureContext);
```

```
    const microform = flex.microform("check", { styles: myStyles });
    const routingNumber = microform.createField("routingNumber", { placeholder:
 "Enter routing number" });
    const accountNumber = microform.createField("accountNumber", { placeholder:
 "Enter account number" });
    const accountNumberConfirm = microform.createField("accountNumberConfirm",
 { placeholder: "accountNumberConfirm" });
    routingNumber.load('#routingNumber-container')
    accountNumber.load('#accountNumber-container')
    accountNumberConfirm.load('#accountNumberConfirm-container')


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

        // Compiling MM & YY into optional parameters
        const options = {
            accountType: document.querySelector('#accountType').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>
```

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. For more information, see JSON Web Tokens (on page 127).

Important: When you integrate 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.

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.

Example: Token Payload for Accepting eCheck Information

```
{
  "iss" : "Flex/00",
  "exp" : 1732527524,
  "type" : "mf-2.1.0",
  "iat" : 1732526624,
  "jti" : "1D3HRVI3KM4HFWQAZ2JFI993NEVBAH5NYJFIH82RAMYWDUJ444KT674445A4EAC0",
  "content" : {
    "paymentInformation" : {
      "bank" : {
        "routingNumber" : { },
        "account" : {
          "number" : { },
          "type" : { }
        }
      },
      "paymentType" : {
        "name" : {
          "value" : "CHECK"
        }
      }
```

```
    }
  }
}
```

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.

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"    }
```

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.

Next Steps

After you complete the server-side and client-side setup for accepting payment information with Microform Integration, you can customize your configuration:

- **Styling**: Customize the style and behavior of Microform Integration on your site. For information, see Styling (on page 77).
- **Events**: Subscribe to Microform Integration events and obtain them through event listeners. For information, see Events (on page 80).

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-autocomplete`	The field has been filled using an `autocomplete/autofill` event.
`.flex-microform-disabled`	The field has been disabled.
`.flex-microform-focused`	The field has user focus.
`.flex-microform-incomplete`	The field is incomplete and could be valid.
`.flex-microform-invalid`	The input card number invalid.
`.flex-microform-valid`	The input card number is valid.

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 specific 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 set-up options accepts a CSS-like object that allows customization of the text. Only a subset of the CSS properties is supported.

```
const 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'
  }
};

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

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

Supported Properties

These 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.

Events (continued)

Event Name	Emitted When
`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.

```
const secCodeLbl = document.querySelector('#mySecurityCodeLabel');
const numberField = microform.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 Integration attempts to detect the card type as it is entered. As card numbers are entered, detection information is sent back in the `change` event. You can use this information to build a dynamic user experience by providing feedback to the user as they type their card number.

```
{
   "card": [
       {
           "name": "visa",
           "brandedName": "Visa",
           "cybsCardType": "001",
           "spaces": [ 4, 8, 12 ],
           "blocks": [ 4, 4, 4, 7 ],
           "lengths": [ 13, 14, 15, 16, 17, 18, 19 ],
           "securityCode": {
               "name": "CVV",
               "length": 3
           },
           "luhn": true,
           "valid": true,
           "couldBeValid": true
       },
       {
           "name": "cartesbancaires",
           "brandedName": "Cartes Bancaires",
           "cybsCardType": "036",
           "spaces": [ 4, 8, 12 ],
           "blocks": [ 4, 4, 4, 7 ],
           "lengths": [ 13, 14, 15, 16, 17, 18, 19 ],
           "securityCode": {
               "name": "CVV",
               "length": 3
           },
           "luhn": true,
           "valid": true,
           "couldBeValid": true
       }
   ],
   "empty": false,
   "couldBeValid": true,
   "valid": true
}
```

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.

It is up to the merchant to then take the results from **cardDetection** and pass that into the **type** parameter within the `microformInstance.createToken(options,...)` method. Microform Integration no longer attempts to determine a single card type by default. Instead, it returns **detectedCardTypes**, in the transient token response and the merchant can decide how to handle this information.

Support for Dual-Branded Cards

Microform Integration supports dual-branded cards. To utilize this feature, you must include the card networks that have overlapping BIN ranges in the capture context request. For example:

```
"allowedCardNetworks": ["VISA", "MASTERCARD", "AMEX", "CARTESBANCAIRES"]
```

When a card number within an overlapping BIN range is entered, Microform Integration returns the detected card types based on the order specified in the **allowedCardNetworks** array. You must then decide which card type to pass.

Support for Card Types without Security Code

Some card types, such as KCP and UATP, do not have security codes (CVV or CVN). Microform Integration supports these card types and provides automatic card detection. This enables you to dynamically display the fields that are relevant for the detected card type.

If you support only card types that do not have security codes, you must render only the card number field and the required fields. Do not render the security code field.

If you support card types that do not have security codes as well as cards types that do have security codes, you can use card detection to determine if the security code field should be enabled or disabled based on the card details that are entered by the customer.

This example shows an implementation that accounts for both cards that do and do not have security codes:

```
const numberField = microform.createField('number');
const securityCodeField = microform.createField('securityCode');

numberField.on('change', function (data) {
  // Check if detected cards don't have a security code
  const allCardsWithoutSecurityCode =
```

```
    data.card &&
    data.card.length > 0 &&
    data.card.every(function (card) {
      return card.securityCode === false;
    });

  if (allCardsWithoutSecurityCode) {
    // Disable the security code field only when detected cards don't require a
 security code
    securityCodeField.update({ disabled: true });
  } else {
    // Enable the security code field when any card requires CVV
    securityCodeField.update({ disabled: false });
  }
});
```

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. Verify the properties before using the event. This example displays how to use this event to update other fields in your checkout process:

```
const 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/

Reference

This reference provides additional information for creating Microform Integration web pages.

JavaScript 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 95). 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

```
const 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.
const 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
- 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
const 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 verify that it exists 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.

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

// create an object to map card names to the URL of your custom images
const 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 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.

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

field.on('change', function(data) {
  // extract the identified card types
  const 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.

```
const 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()`.
2. Call Microform.createToken() (on page 95). For more information, see these topics:
    - Module: FLEX (on page 93)
    - Class: Microform (on page 95)
3. Take the result and add it to a hidden input on your checkout.
4. Trigger submission of the form containing the newly created token for you to use server-side.

Example

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

field.on('inputSubmitRequest', function() {
 const 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 45).

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="[INSERT clientLibrary VALUE HERE]" integrity=”[INSERT
 clientLibraryIntegrity VALUE HERE]” crossorigin=”anonymous”>
</script>//Note: Script location and integrity value should be sourced from the
 capture context response clientLibrary and clientLibraryIntegrity values.
<script> const flex = new Flex('captureContext');</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 95).

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

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

Custom Styling

```
const flex = new Flex('header.payload.signature');
const 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 93).

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
aria-label	string	<optional>	`true`	Set the input's label for use by assistive technologies using the aria-label attribute.
aria-required	boolean	<optional>	`true`	Used to indicate through assistive technologies that this input is required for submission using the aria-required attribute.
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.
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.

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`.
placeholder	string	<optional>		Sets the `placeholder` attribute on the input.
styles	stylingOptions	<optional>		Apply custom styling to this field.
title	string	<optional>		Sets the title attribute on the input. Typically used to display tooltip text on hover.

Returns

Type: Field

Examples

Minimal Setup

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

Providing Custom Styles

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

Providing Custom Styles to All Fields within the Microform Integration

```
const= new Flex('.........');

// apply styles to all fields
const microform = flex.microform('card', { styles: customStyles });
```

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

Providing Custom Styles to A Specific Field within the Microform Integration

```
const= new Flex('.........');
const microform = flex.microform('card');

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

Setting the Length of a Security Code Field

```
const= new Flex('.........');
const microform = flex.microform('card');

const 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.

Name	Type	Attributes	Description
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  card
 type codes:
// <select id="cardTypeOverride">
//   <option value="001">Visa</option>
//   <option value="002">Mastercard</option>
//   <option value="003">American Express</option>
//    etc...
// </select>

const 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 103).

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.

Name	Type	Description
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 95).

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.

Name	Type	Description
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 **Within Capture Context Validity (15 mins):** Do NOT resend the capture context or reload Microform. Instruct the customer to retry until successful. **After Capture Context Validity (>15 mins):** Resend the capture context request and reload Microform.
CREATE_TOKEN_XHR_ERROR	string	Occurs when there is a network error when attempting to create a token. Resend the capture context request and reload Microform.
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.

Name	Type	Description
		Instruct the customer to enter valid data.
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

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 99).	<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

This 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
- incomplete

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`
- `incomplete`
- `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 will raise a `console.warn()` alert.

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.

Name	Type	Attributes	Description
: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'
 }
};
```

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 front-end JavaScript library how to behave within your payment experience. The request provides authentication, one-time keys, the target origin to the Microform Integration, in addition to allowed card networks and payment types (card or check). The capture context request includes these elements:

- **allowedCardNetworks**
- **allowedPaymentTypes**
- **clientVersion**
- **targetOrigins**
- **transientTokenResponseOptions.includeCardPrefix**

For information on JSON Web Tokens, see JSON Web Tokens (on page 127).

Target Origin

The target origin is defined by the scheme (protocol), hostname (domain) and port number (if used).

You must use the https:// protocol. Sub domains must also be included in the target origin.

Any valid top-level domains, such as .com, .co.uk, and .gov.br, are supported. Wildcards are not supported.

For example, if you are launching Microform Integration on example.com, the target origin could be any of the following:

- https://example.com
- https://subdomain.example.com
- https://example.com:8080

You can define the payment cards and digital payments that you want to accept in the capture context.

You can provide up to nine origins in the **targetOrigins** field for nested iframes. If your list of origins in the **targetOrigins** field contains more than five entries, you must do the following:

- Compare the list of origins in the `v2/sessions` **targetOrigins** field with the location.ancestorOrigins of the browser and ensure that they match. For more information, see this description about the location.ancestorOrigins property on the Mozilla Developer website.
- Confirm that the count of origins in the **targetOrigins** field matches that in the content. If any origins are missing or are mismatched, the system will not allow Microform to load and a client-side error message appears.

If your application does not require up to nine nested iframes, Cybersource recommends that you minimize the number of nested iframes to maintain optimal performance.

Allowed Card Networks

Use the **allowedCardNetworks** field to define the card types.

These card networks are available for card entry:

- American Express
- Carnet
- Cartes Bancaires
- China UnionPay
- Diners Club
- Discover
- EFTPOS
- ELO
- Jaywan
- JCB
- JCrew
- KCP
- Mada
- Maestro
- Mastercard
- Meeza
- PayPak
- UATP
- Visa

When you integrate with Microform Integration to accept card or eCheck information, you must include at least one card network in the **allowedCardNetworks** field in the capture context request.

Allowed Payment Types

You can specify the type of Microform Integration you want to accept in the capture context. You can accept card and eCheck information.

Use the **allowedPaymentTypes** field to define the payment type:

- `CARD`
- `CHECK`

The **allowedPaymentTypes** field is optional. When this field is provided in the capture context, the Microform Integration defaults the field value to `CARD` and is returned in the response.

Include Card Prefix

You can control the length of the card number prefix to be received in the response to the capture context `/sessions` request:

- 6 digits
- 8 digits
- no prefix at all

To specify your preferred card number prefix length, include or exclude the **transientTokenResponseOptions.includeCardPrefix** field in the capture context `/sessions` request.

If you want to receive a 6-digit card number prefix in the response:

- Do not include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context `/sessions` request.
- This example shows how a 6-digit card number prefix `411111` is returned in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111”,
"prefix" : "411111"
```

If you want to receive an 8-digit card number prefix in the response:

- Include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request, and set the value to `true`.

> **Important:** Per PCI DSS requirements, this requirement applies only to card numbers longer than 15 digits and for Discover, JCB, Mastercard, UnionPay, and Visa brands.
>
> - If the card type entered is not part of these brands, a 6-digit card number prefix is returned instead.
> - If the card type entered is not part of these brands but is *co-branded* with these brands, an 8-digit card number prefix is returned.

- This example shows how an 8-digit card prefix `41111102` is returned in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111”,
"prefix" : "41111102"
```

If you do not want to receive a card number prefix in the response

- Include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request, and set the value to `false`.
- This example shows how a card number is returned without a card number prefix in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111"
```

Best practice: If your application does not require card number prefix information for routing or identification purposes, Cybersource recommends that you include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request and set its value to `false`. Doing so limits the exposure of payment data to only what is necessary for your processing needs.

For more information about PCI DSS, see *Frequently Asked Questions* on the PCI Security Standards Council site.

Endpoint

Production: `POST https://api.cybersource.com/microform/v2/sessions`

Test: `POST https://apitest.cybersource.com/microform/v2/sessions`

Required Fields for Requesting the Capture Context

Your capture context request for accepting card and eCheck information must include these fields:

allowedCardNetworks

This field is required only for accepting card information.

allowedPaymentTypes

This field is required only for accepting eCheck information.

clientVersion

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

Capture Context Request for Accepting Card and eCheck Information

```
{
  "clientVersion": "v2",
  "targetOrigins": [
    "https://www.example.com",
    "https://www.basket.example.com",
    "https://ecom.example.com"
  ],
  "allowedCardNetworks": [
    "VISA",
    "MASTERCARD",
    "AMEX",
    "CARTESBANCAIRES",
    "CARNET",
    "CUP",
    "DINERSCLUB",
    "DISCOVER",
    "EFTPOS",
    "ELO",
    "JCB",
    "JCREW",
    "MADA",
    "MAESTRO",
    "MEEZA"
```

```
  ],
  "allowedPaymentTypes": [
    "CARD",
    "CHECK"
  ]
}
```

Successful Encrypted JWT Response for Accepting Card and eCheck Information

```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsIm
RhdGEiOiJ4Sy9RZzJzcWNiajNaODdIS0VwZGpSQUFFUEJLenFVaUlrdXdrelRadVNQeG9GWDNOK0VtT3k2
ZGlBSXhwMHhvQStJc1Y5czlZZmRZVUJmbDFpV01kRVVmekhMTmhLQ1ZiOFAzSHg0bTQxejl6aEVcdTAwM2
QiLCJvcmlnaW4iOiJodHRwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJqd2siOnsia3R5Ijoi
UlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJuIjoiaTI5NmZmbUdiVkVRbG5zanNrQnBrajRCU1pRbE
9Vd2Z2SUpsdUNIaHVkTEsxQmk4MEtpOVVfb0h2Mmxsd0NJSXFUMHdJWTBvaS1HM0xVc09BUUpjNUpwX08t
Y1VIdzFjeXV3aDVtWXZkUWZFM0JlcnRUcDZHQ1JLcFVyUjlDOGZoaTNfV1lQTDMwNjMxMGJXajh4OXY0Un
lpR1BYUHdnUlJhVDlmbXJRV1diTHZaeFNZQ0Zpal9lSEZaYXQxa1JNdG5pTk5IUTNlV1ViWUV5QWZPSExO
WDhUUmN6cjYybkdpeGh4NEFNYVB5YlEtTy0wM1pjRER4UldTLTI0UGhheVVYRjZnVlAydzNHc2hhOFQxSk
lnYlZSc253bHVuZ01jREXtZFI1cF9ITFlRdnNyX3BFS3pZd2tDQmFOUXA0ZjRkbXhyaVIyT2IyUU5fMXRk
U2FUVzN3Iiwia2lkIjoiMDBXSXlhTndYcUJhdHNuYkVmMVNFTjFncHREbDExOUYifX0sImN0eCI6W3siZG
F0YSI6eyJjbGllbnRMaWJyYXJ5SW50ZWdyaXR5Ijoic2hhMjU2LXZkWWkxaDV1ZTNwcm5iVC8xYThJSkxl
UkNrSGVqSHBkRGR3My95RkxaREFcdTAwM2QiLCJjbGllbnRMaWJyYXJ5IjoiaHR0cHM6Ly9zdGFnZWZsZX
guY3liZXJzb3VyY2UuY29tL21pY3JvZm9ybS9idW5kbGUvdjIuNS4xL2ZsZXgtbWljcm9mb3JtLm1pbi5q
cyIsImFsbG93ZWRDYXJkTmV0d29ya3MiOlsiVklTQSIsIk1BU1RFUkNBUkQiLCJBTUVYIiwiTUFFU1RSTy
IsIkRJU0NPVkVSIiwiRElORVJTQ0xVQiIsIkpDQiIsIkNVUCIsIkNBUlRFU0JBTkNBSVJFUyJdLCJ0YXJn
ZXRPcmlnaW5zIjpbImh0dHBzOi8vdGhlLXVwLWRlbW8uYXBwc3BvdC5jb20iXSwibWZPcmlnaW4iOiJodH
RwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJhbGxvd2VkUGF5bWVudFR5cGVzIjpbIkNBUkQi
LCJDSEVDSyJdfSwidHlwZSI6Im1mLTIuMS4wIn1dLCJpc3MiOiJGbGV4IEFQSSIsImV4cCI6MTczMzQ5MD
QwNywiaWF0IjoxNzMzNDg5NTA3LCJqdGkiOiJZcHdxWlNaTTZ1T1FBV1RoIn0.f24avtv-oUGfSaGqlQZf
OZ4B6A-6E6yWymdgUtZmDDOVNanx5uLt5fxSBzAdmtC4em0kORatiS5pMhE66bCJT-ujIMHtdPITq9JJuE
4Tm-NdfzznXlhz-qMM_setgmDXYLIIAeaUmSVebMwWqxBVmpQHRIBq2plwfB5dAH411aO-U1_DDJi14sIO
LzUr_xhgfLJWsJ_B3gZkSaHrRaHWpnO-okCanTrVKCaFP1X5rKUilGII4wcJLdUyU3f9zFwteQ7wFfG81m
RRWz4Gb4YgLt43TCD-jSigCAtgX_mqRyMqzCJtZXkf0Nf-o0bJLAc8-ce8MmeZD8H4uG42Eu-0UA
```

Decrypted Capture Context Header for Accepting Card and eCheck Information

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

Decrypted Capture Context Body with Selected Fields for Accepting Card and eCheck Information


```
{
  "flx": {
    "path": "/flex/v2/tokens",
    "data":
 "xK/Qg2sqcbj3Z87HKEpdjRAAEPBKzqUiIkuwkzTZuSPxoFX3N+EmOy6diAIxp0xoA+IsV9s9YfdYUBfl
1iWMdEUfzHLNhKCVb8P3Hx4m41z9zhE=",
    "origin": "https://example.com",
    "jwk": {
      "kty": "RSA",
      "e": "AQAB",
      "use": "enc",
      "n":
 "i296ffmGbVEQlnsjskBpkj4BSZQlOUwfvIJluCHhudLK1Bi80Ki9U_oHv2llwCIIqT0wIY0oi-G3LUsO
AQJc5Jp_O-cUHw1cyuwh5mYvdQfE3BertTp6GCRKpUrR9C8fhi3_WYPL306310bWj8x9v4RyiGPXPwgRRa
T9fmrQWWbLvZxSYCFij_eHFZat1kRMtniNNHQ3eWUbYEyAfOHLNX8TRczr62nGixhx4AMaPybQ-O-03ZcD
DxRWS-24PhayUXF6gVP2w3Gsha8T1JIgbVRsnwlungMcDLmdR5p_HLYQvsr_pEKzYwkCBaNQp4f4dmxriR
2Ob2QN_1tdSaTW3w",
      "kid": "00WIyaNwXqBatsnbEf1SEN1gptDl119F"
    }
  },
  "ctx": [
    {
      "data": {
        "clientLibraryIntegrity": "CLIENT LIBRARY INTEGRITY VALUE GOES HERE",
        "clientLibrary": "CLIENT LIBRARY VALUE GOES HERE",
        "allowedCardNetworks": [
          "VISA",
          "MASTERCARD",
          "AMEX",
          "MAESTRO",
          "DISCOVER",
          "DINERSCLUB",
          "JCB",
          "CUP",
          "CARTESBANCAIRES"
        ],
        "targetOrigins": [
          "https://the-up-demo.appspot.com"
        ],
        "mfOrigin": "https://example.com",
        "allowedPaymentTypes": [
          "CARD",
          "CHECK"
        ]
      },
```

```
      "type": "mf-2.1.0"
    }
  ],
  "iss": "Flex API",
  "exp": 1733490407,
  "iat": 1733489507,
  "jti": "YpwqZSZM6uOQAWTh"
}
```

Capture Context Request for Accepting Card Information

```
{
  "clientVersion": "v2",
  "targetOrigins": [
    "https://www.example.com",
    "https://www.basket.example.com",
    "https://ecom.example.com"
  ],
  "allowedCardNetworks": [
    "VISA",
    "MASTERCARD",
    "AMEX",
    "CARTESBANCAIRES",
    "CARNET",
    "CUP",
    "DINERSCLUB",
    "DISCOVER",
    "EFTPOS",
    "ELO",
    "JCB",
    "JCREW",
    "MADA",
    "MAESTRO",
    "MEEZA"
  ],
  "allowedPaymentTypes": [
    "CARD"
  ]
}
```

Successful Encrypted JWT Response for Accepting Card Information

```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsIm
RhdGEiOiJ1Q0RERW94M2dDQk1VaHI2T1ZDVGt4QUFFS1pHSTRHcDFvQ2pyYXlVb1MxQzdGOXE2WFpyYXhG
bGxMVDMvenE2cjFnNXoxS1U2UDZseldqRVFTVVJoZUtxUThoVWJkZVNNdmt5SERTTXUwV01tMzhcdTAwM2
```

QiLCJvcmlnaW4iOiJodHRwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJqd2siOnsia3R5Ijoi
UlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJuIjoiMXNDY3NZNC1WZTNWU0VKekhnelJ5WjVDOURrM0
VHZ2ZPOGd5SDc5bVJfSlN6NzdmWTdfV1loM3psdTkyTFVfeU5KVTBUMzdOQmVzd0szU2c0YnRNaU41Q0FC
bWNXLWNSckhta2k0MVZoNUZRMmtjcWZSSlgxNVhZN1A3R25GTnd4QzVkUG9UM29NM1czRFVHaUMyYW56en
hIN3pNNlA3N2hFbnc2TkZHSXlBdXhJRWFwRG9DaXlEVW5NdFRwV2lBV3YzTF9OVHZOaHRkVE4tNm1GRWU1
RmdVYmlzeWtrTzlWMHZaS0d6SWRWWmdTdE42cHlnUGhVbnlNXzJIVmIxQmkyWjNKaElhZDFLUW02SGl0Nk
lwYjNyUTBHRWZsN0ZWOUV3NGZyNzJpekQ0WVg2WHo0V3ZuMzlLN3J3WkhCRXdNM3l5Wl9ELTBUbjM1MFhv
UlBUVjB3Iiwia2lkIjoiMDB4V1A1eUh1UE1kNkFkNHdwVzNzQkt1bWFaQ01zYWMifX0sImN0eCI6W3siZG
F0YSI6eyJjbGllbnRMaWJyYXJ5SW50ZWdyaXR5Ijoic2hhMjU2LXZkWWkxaDV1ZTNwcm5iVC8xYThJSkxl
UkNrSGVqSHBkRGR3My95RkxaREFcdTAwM2QiLCJjbGllbnRMaWJyYXJ5IjoiaHR0cHM6Ly9zdGFnZWZsZX
guY3liZXJzb3VyY2UuY29tL21pY3JvZm9ybS9idW5kbGUvdjIuNS4xL2ZsZXgtbWljcm9mb3JtLm1pbi5q
cyIsImFsbG93ZWRDYXJkTmV0d29ya3MiOlsiVklTQSIsIk1BU1RFUkNBUkQiLCJBTUVYIiwiTUFFU1RSTy
IsIkRJU0NPVkVSIiwiRElORVJTQ0xVQiIsIkpDQiIsIkNVUCIsIkNBUlRFU0JBTkNBSVJFUyJdLCJ0YXJn
ZXRPcmlnaW5zIjpbImh0dHBzOi8vdGhlLXVwLWRlbW8uYXBwc3BvdC5jb20iXSwibWZPcmlnaW4iOiJodH
RwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJhbGxvd2VkUGF5bWVudFR5cGVzIjpbIkNBUkQi
XX0sInR5cGUiOiJtZi0yLjEuMCJ9XSwiaXNzIjoiRmxleCBBUEkiLCJleHAiOjE3MzY0MzA0MTQsImlhdC
I6MTczNjQyOTUxNCwianRpIjoiZDVZbzVhNU0wWFBPQ1BxZiJ9.G4Ea-gIk6SG5ULE4NE5OsdPI41YaAuT
EMHDstBgkFzczIWwzJScvXs4hgWiyA-1ZLGITedlumGj-0x8jxmYTWeTm7D0fP8RL0w148EpDLMD8xMHpA
JMdMqZTmYHyichsy8uOZKVOn9NbnuQqfDeQS_rLpJV3tMe2NwJL3RdBXdJ894ihKpFP2yXE1wQeLekNiYJ
6s-Uuxwf0jf2CSN_TJAjnfVR6bqlpWbUpiUaBLcqDsHHe_pcrd5g2r-1LEfCiOV9RIw7844XKFNLQZvt_a
lQjItuMy8M9LVhnlRWCSnTKB1iV1RUxuTWtMzTvHmQWPx4nShqzE3j0Hp61c0PmBw

Decrypted Capture Context Body with Selected Fields for Accepting Card Information


```
{
  "flx": {
    "path": "/flex/v2/tokens",
    "data":
 "gEQUL6QggbM2m8R3/KhDLxAAEOmhvNFhDd+amn9NHURTfjqqN8+7dy/YKQXz0Ik2yhRVQ8omdKZ8VojI
YkwOB9yUo/8LdddXruIQ3O5dSfmbW6A=",
    "origin": "https://example.com",
    "jwk": {
      "kty": "RSA",
      "e": "AQAB",
      "use": "enc",
      "n":
 "vHtaFM0e1ljAQ1Lnza95LdsBh10p78lz13rUdMe29vBESIeI912Fix514WXa97Ijh-pBuonFRcsyL0_-
CF98rdhow6sZMhPdyOA9ud-PcAOwSHm-HrUUU_XkHGUVslBINAFOpOCYZh9qZ7jk6-5-Gk6MeyD4ok0BNz
4XIKht_hj8yJQhphPz17hjguL9KPqK45HTl_D3SEStkbSaPK4fe-glMv2YJRh3nvdQYXkm-0cDmx4nets_
4SH8U5DUwoFAB-Zh30-KHHe2nGbCYNrh7oUOEoC7mmF90HG0jwsbI5KSNDStckQ8pEZhpppVWyPh0CjzOm
DidlkjUZ8hMJARWw",
      "kid": "008vKoQ3ycDeaUxLN6bPlfk9cjIjqSrb"
    }
  },
  "ctx": [
    {
```

```
      "data": {
        "clientLibraryIntegrity": " CLIENT LIBRARY INTEGRITY VALUE GOES HERE",
        "clientLibrary": "CLIENT LIBRARY VALUE GOES HERE",
        "allowedCardNetworks": [
          "VISA",
          "MASTERCARD",
          "AMEX",
          "MAESTRO",
          "DISCOVER",
          "DINERSCLUB",
          "JCB",
          "CUP",
          "CARTESBANCAIRES"
        ],
        "targetOrigins": [
          "https://example.com"
        ],
        "mfOrigin": "https://example.com",
        "allowedPaymentTypes": [
          "CARD"
        ]
      },
      "type": "mf-2.1.0"
    }
  ]
}
```

Capture Context Request for Accepting eCheck Information

```
{
  "clientVersion": "v2",
  "targetOrigins": [
    "https://www.example.com",
    "https://www.basket.example.com",
    "https://ecom.example.com"
  ],
  "allowedPaymentTypes": ["CHECK"]
}
```

Successful Encrypted JWT Response for Accepting eCheck Information

```
eyJraWQiOiJqNCIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsIm
RhdGEiOiJtdnkwVk9OVk40bzA4bTRGQjhmU3FCQUFFS0JWOTdlNnR2VDd4cHdqaFkwMDRydFJ1dGI0R2Yw
WlNwNGdNeEkvanBVSWxFblZKa2JtUVNHaWFnUEdGc0NPazdMbHJGTHFKcXN2eitoTHhrY08xRkFcdTAwM2
QiLCJvcmlnaW4iOiJodHRwczovL3N0YWdlZmxleC5jeWJlcnNvdXJjZS5jb20iLCJqd2siOnsia3R5Ijoi
```

```
UlNBIiwiZSI6IkFRQUIiLCJ1c2UiOiJlbmMiLCJuIjoidmRpN0gtM1MzMTkyZlc5WC1BTmpvdjlFdXU4ZG
xPOTBtU2gyUGVyMF9PdHZ4YlJITTBrakZpTHlKaGQwUUR3VlNWbUlhRFc2aGtCa1k2Ui1lcWRnaTdUVUNG
ZEQ3UUU1ckNkZGhZZTIycTh0RUNQZkpOWWJ6STZZTVBxTkFyYWc5LUhhWVo1X2tOX0JvMm5EclN4RFJ0MH
BDbGxyd2d2Q1ZLb2M0RWF6ZE93QUE4dnI2VVh4Ty1SWVI2Z1R5VEZia244Q2hDVHNvWDByam5VWVI1VjdR
aE95YzMzWEJUTVNDYTVBOHFQNDZnZXpvQjZ0dDA0SlQtRVVMWE9vYndVcVdvd0E3TTJzWUYydkFoQkVuMm
t0REJFWVJSN3E0aWEyVHRIS1JPUW9FTjhZNjNiNFNaTGZDQk82cEc2QXpnSWpya3RkQXhIOXR0WURYdFJY
S1YxeTN3Iiwia2lkIjoiMDBiQlN1d3VpdGtYeExROGFISWloMm5qMFhQNFpXYUsifX0sImN0eCI6W3siZG
F0YSI6eyJjbGllbnRMaWJyYXJ5SW50ZWdyaXR5Ijoic2hhMjU2LXZkWWkxaDV1ZTNwcm5iVC8xYThJSkxl
UkNrSGVqSHBkRGR3My95RkxaREFcdTAwM2QiLCJjbGllbnRMaWJyYXJ5IjoiaHR0cHM6Ly9zdGFnZWZsZX
guY3liZXJzb3VyY2UuY29tL21pY3JvZm9ybS9idW5kbGUvdjIuNS4xL2ZsZXgtbWljcm9mb3JtLm1pbi5q
cyIsInRhcmdldE9yaWdpbnMiOlsiaHR0cHM6Ly90aGUtdXAtZGVtby5hcHBzcG90LmNvbSJdLCJtZk9yaW
dpbiI6Imh0dHBzOi8vc3RhZ2VmbGV4LmN5YmVyc291cmNlLmNvbSIsImFsbG93ZWRQYXltZW50VHlwZXMi
OlsiQ0hFQ0siXX0sInR5cGUiOiJtZi0yLjEuMCJ9XSwiaXNzIjoiRmxleCBBUEkiLCJleHAiOjE3MzM0OT
AxODEsImlhdCI6MTczMzQ4OTI4MSwianRpIjoiSXdEdHAxZkVZM2QwYUh6OSJ9.arokacvdTSUIehBY0IC
i__2szuCdrvykJZq4T69n4OcWGym5PErJO0moJD-QYynhFj7_0k-G39qbkNJydB3UyF2qJSaqwZiopO27k
uqk8u9Z0cY-V9Nu04JgaV4s18doxnzx6vdTCC3krrIcxeINi23Qu-Szcpg7aaGvPVXMC0DVC14WUQiGJkO
akJ54jWtl2VoFAgYziUMcYYpk4hxLVxurBtT7lvrfCXKoyWtxiUxoEpOc_Td_qi5nA8ByWUaieQmp1Zej6
1khQJ_hmXtlsAt4BqxeJWoJeR_5Sjz0vD5y4-oAeNNrAulDem7CKiRJQbI9fyqT-H5Cmjd6YQchxQ
```


Decrypted Capture Context Body with Selected Fields for Accepting eCheck Information


```
{
  "flx": {
    "path": "/flex/v2/tokens",
    "data":
 "mvy0VONVN4o08m4FB8fSqBAAEKBV97e6tvT7xpwjhY004rtRutb4Gf0ZSp4gMxI/jpUIlEnVJkbmQSGi
agPGFsCOk7LlrFLqJqsvz+hLxkcO1FA=",
    "origin": "https://example.com",
    "jwk": {
      "kty": "RSA",
      "e": "AQAB",
      "use": "enc",
      "n":
 "vdi7H-3S3192fW9X-ANjov9Euu8dlO90mSh2Per0_OtvxbRHM0kjFiLyJhd0QDwVSVmIaDW6hkBkY6R-
eqdgi7TUCFdD7QE5rCddhYe22q8tECPfJNYbzI6YMPqNArag9-HaYZ5_kN_Bo2nDrSxDRt0pCllrwgvCVK
oc4EazdOwAA8vr6UXxO-RYR6gTyTFbkn8ChCTsoX0rjnUYR5V7QhOyc33XBTMSCa5A8qP46gezoB6tt04J
T-EULXOobwUqWowA7M2sYF2vAhBEn2ktDBEYRR7q4ia2TtHKROQoEN8Y63b4SZLfCBO6pG6AzgIjrktdAx
H9ttYDXtRXKV1y3w",
      "kid": "00bBSuwuitkXxLQ8aHIih2nj0XP4ZWaK"
    }
  },
  "ctx": [
    {
      "data": {
        "clientLibraryIntegrity": "CLIENT LIBRARY INTEGRITY VALUE GOES HERE",
        "clientLibrary": "CLIENT LIBRARY VALUE GOES HERE ",
```

```
        "targetOrigins": [
          "https://the-up-demo.appspot.com"
        ],
        "mfOrigin": "https://example.com",
        "allowedPaymentTypes": [
          "CHECK"
        ]
      },
      "type": "mf-2.1.0"
    }
  ],
  "iss": "Flex API",
  "exp": 1733490181,
  "iat": 1733489281,
  "jti": "IwDtp1fEY3d0aHz9"
}
```

Getting Started Examples

Example: Checkout Payment Form for Accepting Card Information

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

Back to Client-Side Setup (on page 52).

```
<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>
```

Example: Checkout Payment Form for Accepting eCheck Information

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

Back to Client-Side Setup (on page 69).

```
<h1>Checkout</h1>
<div id="errors-output" role="alert"></div>
<form action="/token" id="my-sample-form" method="post">
    <div class="form-group">
        <label id="routingNumber-label">Routing Number</label>
        <div id="routingNumber-container" class="form-control"></div>
        <label for="accountNumber-label">Account Number</label>
        <div id="accountNumber-container" class="form-control"></div>
        <label for="accountNumberConfirm-label">Account Number Confirm</label>
        <div id="accountNumberConfirm-container" class="form-control"></div>
    </div>

    <div class="form-row">
        <div class="form-group col-md-6">
```

```
            <label for="accountType">Account Type</label>
            <select id="accountType" name="accountType" class="form-control">
                <option value="C">Checking</option>
                <option value="S">Savings</option>
                <option value="X">Corporate checking</option>
            </select>
        </div>
    </div>

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

Example: Creating the Pay Button with Event Listener for Cards

Back to Transient Token Time Limit (on page 56).

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

    // Compiling MM & YY into optional parameters
    const 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();
        }
    });
});
```

Example: Creating the Pay Button with Event Listener for Checks

Back to Transient Token Time Limit (on page 72).

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

  // Compiling account type into optional parameters
  var options = {
    accountType: document.querySelector('#accountType').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();
    }
  });
});
```

Example: Customer-Submitted Form with Card Information

Back to Transient Token Time Limit (on page 56).

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

    // the capture context that was requested server-side for this transaction
    const captureContext = <% -keyInfo %> ;
    // custom styles that will be applied to each field we create using Microform
    const 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
    const flex = new Flex(captureContext);
    const microform = flex.microform({ styles: myStyles });
    const number = microform.createField('number', { placeholder: 'Enter card
 number' });
    const 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 parameters
        const 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>
```

Example: Customer-Submitted Form with eCheck Information

Back to Transient Token Time Limit (on page 72).

```
<script>
    // Variables from the HTML form
    const form = document.querySelector('#my-sample-form');
```

```
   const payButton = document.querySelector('#pay-button');
   const flexResponse = document.querySelector('#flexresponse');
   const accountType = document.querySelector('#accountType')
   const errorsOutput = document.querySelector('#errors-output');

   // the capture context that was requested server-side for this transaction
   const captureContext = <% -keyInfo %> ;
   // custom styles that will be applied to each field we create using Microform
   const 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
   const flex = new Flex(captureContext);
   const microform = flex.microform("check", { styles: myStyles });
   const routingNumber = microform.createField("routingNumber", { placeholder:
"Enter routing number" });
   const accountNumber = microform.createField("accountNumber", { placeholder:
"Enter account number" });
   const accountNumberConfirm = microform.createField("accountNumberConfirm",
{ placeholder: "accountNumberConfirm" });
   routingNumber.load('#routingNumber-container')
   accountNumber.load('#accountNumber-container')
   accountNumberConfirm.load('#accountNumberConfirm-container')


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

       // Compiling MM & YY into optional parameters
       const options = {
           accountType: document.querySelector('#accountType').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>
```

Example: Token Payload with Card Information

Back to Transient Token Response Format (on page 58).

```
{
  "iss": "Flex/00",
  "exp": 1728911080,
  "type": "mf-2.0.0",
  "iat": 1728910180,
  "jti": "1D1S6JK9RL6EK667H1I370689A63I2I8YLFJSPJ1EUSKIPMJJWEL670D16E89AF8",
  "content": {
    "paymentInformation": {
      "card": {
        "expirationYear": {
          "value": "2025"
        },
        "number": {
          "detectedCardTypes": [
            "001"
          ],
          "maskedValue": "XXXXXXXXXXXX1111",
          "bin": "411111"
        },
        "securityCode": {},
        "expirationMonth": {
          "value": "01"
        }
      }
    }
  }
}
```

Example: Token Payload with eCheck Information

Back to Transient Token Response Format (on page 74).

```
{
  "iss" : "Flex/00",
  "exp" : 1732527524,
  "type" : "mf-2.1.0",
  "iat" : 1732526624,
  "jti" : "1D3HRVI3KM4HFWQAZ2JFI993NEVBAH5NYJFIH82RAMYWDUJ444KT674445A4EAC0",
  "content" : {
    "paymentInformation" : {
      "bank" : {
        "routingNumber" : { },
        "account" : {
          "number" : { },
          "type" : { }
        }
      },
      "paymentType" : {
        "name" : {
          "value" : "CHECK"
        }
      }
    }
  }
}
```

Example: Token Payload with Multiple Card Types

Back to Transient Token Response Format (on page 58).

```
{
  "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"
        }
      }
    }
  }
}
```

Example: Capture Context Public Key

Back to Validating the Transient Token for Accepting Card Information (on page 59) and Validating the Transient Token for Accepting eCheck Information (on page 76).

```
"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"    }
```

Example: Authorization with a Transient Token Using the REST API

Back to Using the Transient Token to Process a Payment (on page 60).

```
{
    "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":
 "eyJraWQiOiIwN0JwSE9abkhJM3c3UVAycmhNZkhuWE9XQlhwa1ZHTiIsImFsZyI6IlJTMjU2In0.eyJk
YXRhIjp7ImV4cGlyYXRpb25ZZWFyIjoiMjAyMCIsIm51bWJlciI6IjQxMTExMVhYWFhYWDExMTEiLCJleH
BpcmF0aW9uTW9udGgiOiIxMCIsInR5cGUiOiIwMDEifSwiaXNzIjoiRmxleC8wNyIsImV4cCI6MTU5MTc0
NjAyNCwidHlwZSI6Im1mLTAuMTEuMCIsImlhdCI6MTU5MTc0NTEyNCwianRpIjoiMUMzWjdUTkpaVjI4OV
M5MTdQM0JHSFM1T0ZQNFNBRERCUUtKMFFKMzMzOEhRR0MwWTg0QjVFRTAxREU4NEZDQiJ9.cfwzUMJf115
K2T9-wE_A_k2jZptXlovls8-fKY0muO8YzGatE5fu9r6aC4q7n0YOvEU6G7XdH4ASG32mWnYu-kKlqN4IY
_cquRJeUvV89ZPZ5WTttyrgVH17LSTE2EvwMawKNYnjh0lJwqYJ51cLnJiVlyqTdEAv3DJ3vInXP1YeQjL
X5_vF-OWEuZfJxahHfUdsjeGhGaaOGVMUZJSkzpTu9zDLTvpb1px3WGGPu8FcHoxrcCGGpcKk456AZgYMB
SHNjr-pPkRr3Dnd7XgNF6shfzIPbcXeWDYPTpS4PNY8ZsWKx8nFQIeROMWCSxIZOmu3Wt71KN9iK6DfOPr
o7w"
    }
}
```


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.

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

```
<Header>.<Payload>.<Signature>
```

The header and payload is **Base64-encoded JSON** and contains these claims:

- **Header**: The algorithm and token type. For example:

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

- **Payload**: The claims of what the token represents. For example:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

- **Signature**: The signature is computed from the header and payload using a secret or private key.

Important: When working with JWTs, Cybersource recommends that you use a well-maintained JWT library to ensure proper decoding and parsing of the JWT.

Important: When parsing the JWT's JSON payload, you must ensure that you implement a robust solution for transversing JSON. Additional elements can be added to the JSON in future releases. Follow JSON parsing best practices to ensure that you can handle the addition of new data elements in the future.

Browser Support

Microform Integration is supported on these browsers and versions:

- Chrome 80 or later
- Edge 109 or later
- Firefox 115 or later
- Opera 106 or later
- Safari 13 or later

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.

WCAG 2.2 Compliance

Your integration must be compliant with the Web Content Accessibility Guidelines (WCAG 2.2) in order to meet accessibility standards and regulations. Microform Integration is designed with these guidelines in mind but some accessibility compliance is dependent on your integration, particularly the custom styling of fields. This section contains the minimum required information to ensure that Microform Integration is compliant with WCAG 2.2.

Microform Integration automatically handles many accessibility requirements, but the hosting page and custom styling elements are your responsibility. The parent page that contains Microform Integration must meet WCAG 2.2 standards. This includes proper heading structure, page titles, and other accessibility requirements. All fields that are rendered in Microform Integration are automatically assigned as **aria-required** set to `true`. The parent page must implement clear visual indicators to indicate that these fields are required for users. For information about WCAG 2.2 guidelines, see the Web Content Accessibility Guidelines (WCAG) 2.2 on the W3C website.

Font Configuration

You must follow these guidelines when you apply custom styles to Microform Integration:

- Set the font size to at least 16px (1rem) for input fields.
- Select font families that are compatible with screen readers and assistive technologies.
- Set a line height that is at least 1.5 times the font size.

This is an example Microform Integration font configuration:

```
// define accessible custom styles
var customStyles = {
    'input': {
        'font-size': '16px',
        'family': 'Arial, sans-serif',
        'lineHeight': '1.5'
    }
}
// apply styles to all fields
var microform = flex.microform({ styles: customStyles });
```

Color and Contrast

Follow these guidelines to ensure that visibility is compliant for all users:

- Maintain a minimum contrast ratio of 4.5:1 between text and background colors.
- Do not rely solely on color to convey information.
- Implement distinct focus states using both color changes and other visual indicators.
- Use the `flex-microform-focused` class for consistent focus indication.

This is an example of color and contrast configuration for Microform Integration:

```
/* add a visual indicator for focus */
.flex-microform-focused {
  background: lightyellow;
}
```

Handle Errors

Microform Integration provides managed classes to indicate field validation states. You must handle any errors that are returned by the **createToken** method. Follow these guidelines to handle errors:

- Implement error handling that programmatically associates error messages with their respective form fields.
- Associate all helper text with its corresponding form control by setting **aria-describedby** to `helperTextId`. `helperTextId` is the unique ID of the helper text container element.
- Use the **MicroformError** object to determine to which fields the error applies.
- Ensure that all error messages are clear, descriptive, and concise.
- Make error messages visible and properly announced by screen readers.

This is an example **MicroformError** object:

```
{
  "name": "MicroformError",
  "reason": "CREATE_TOKEN_VALIDATION_FIELDS",
  "message": "One or more fields have a validation error.",
  "informationLink":
 "https://www.cybersource.com/products/payment_security/secure_acceptance",
  "details": [
    {
      "message": "Validation error",
      "location": "number"
    }
  ]
}
```

Testing

Follow these guidelines to verify that your integration is compliant before you deploy:

- Test keyboard navigation through the entire payment form.
- Validate that your implementation works with screen readers (NVDA, JAWS, VoiceOver).
- Run automated accessibility checks with tools such as Axe or Lighthouse.
- Conduct manual testing with common assistive technologies.

When you follow these guidelines, your Microform Integration implementation maintains compliance with WCAG 2.2 while you provide an accessible payment experience for all users.

Test Card Numbers

Use these test card numbers to test your Microform Integration configuration.

Combine the BIN with the card number when sending to Microform Integration.

Test Card Numbers

Card Brand	BIN	Card Number	Expiration Date	CVV
Visa	424242	4242424242	12/2026	123
Mastercard	555555	5555554444	02/2026	265
American Express	378282	246310005	03/2026	7890
Cartes Bancaires	436000	0001000005	04/2040	123
Carnet	506221	0000000009	04/2024	123
China UnionPay	627988	6248094966	04/2040	123
Diners Club	305693	09025904	04/2040	123
Discover	644564	4564456445	04/2040	123
JCB	353011	13333 0000	04/2040	123
Maestro	675964	9826438453	04/2040	123
Mada	446404	0000000007	04/2040	123
ELO	451416	0000000003	04/2040	123
JCrew	515997	1500000005	04/2040	123
EFTPOS	401795	000000000009	04/2040	123
Meeza	507808	3000000002	04/2040	123
UATP	148512	345678905	04/2040	
KCP	949022	0011669217	04/2040	
Jaywan	669000	0000000000	04/2040	123

Introduction to Unified Checkout

Unified Checkout provides a single interface with which you can accept numerous types of card, digital, and alternative payments. Unified Checkout calls other follow-on services such as Payments, Decision Manager, Payer Authentication, and Token Management Service (TMS).

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

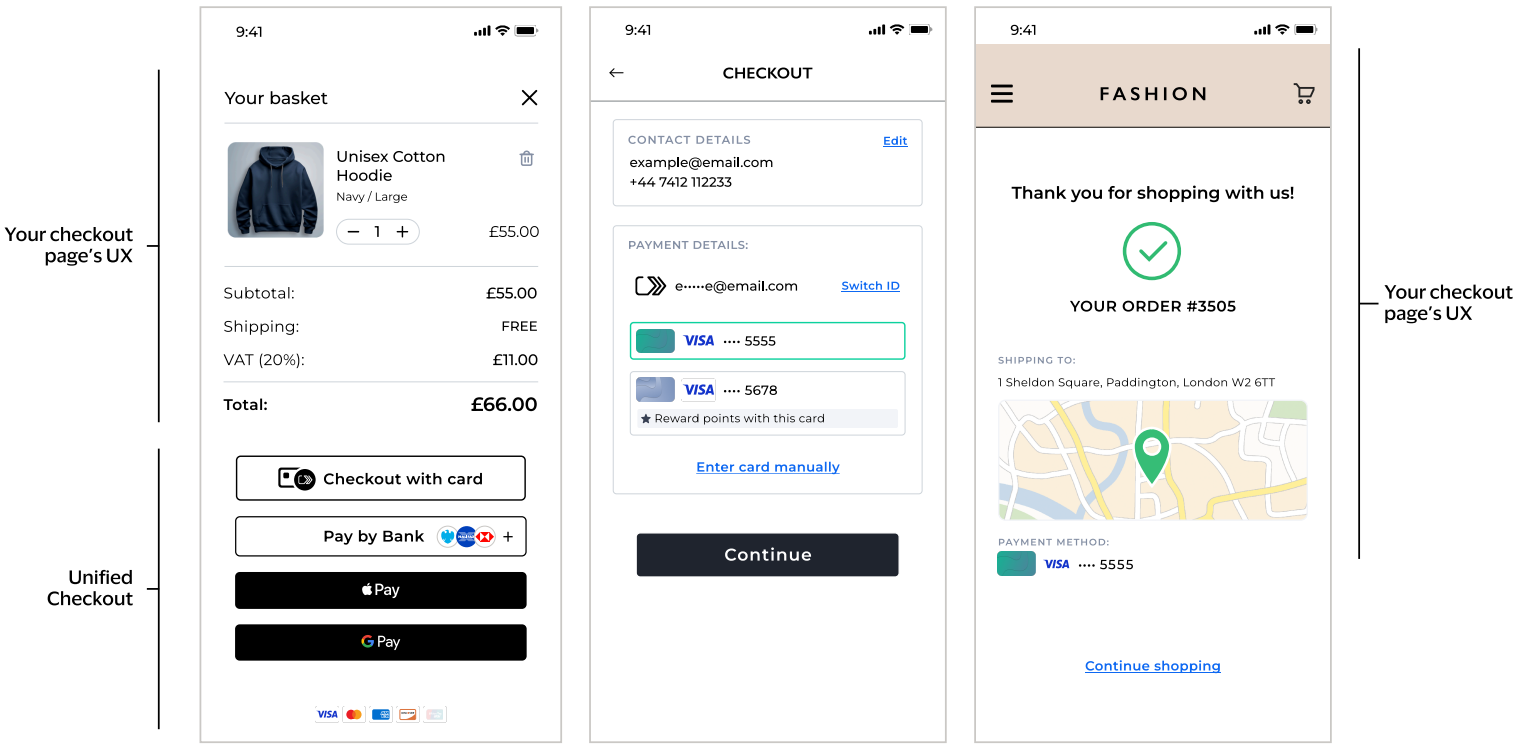
The server-side component authenticates your merchant identity and instructs the system to act within your payment environment. The response contains limited-use public keys. The keys are for end-to-end encryption and contain merchant-specific payment information that drives the interaction of the application. 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 selects a payment method from the button widget, Unified Checkout handles all interactions with the payment method that was chosen. Unified Checkout is also able to orchestrate requests for to follow-on services such as Payments, Decision Manager, Payer Authentication, and TMS before it provides a response to your e-commerce system.

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

Button Widget



For examples of different payment method UIs through Unified Checkout, see these topics:

- Click to Pay UI (on page 177)
- Google Pay UI (on page 168)
- Pay with eCheck/ACH Service UI (on page 158)
- Paze UI (on page 187)
- Apple Pay UI (on page 164)

.

Important: Each request that you send to Cybersource requires header information. For information about constructing the headers for your request, see the *Getting Started with REST Developer Guide*.

Key Features

- **Low-code integration**: You can use as few as three lines of JavaScript to accept payments, as well as add or remove payment methods through portal configuration without changing your integration code.
- **PCI SAQ-A compliant**: Payment data never touches your systems.
- **Fully customizable**: You can match the payment experience to your brand with theming, fonts, and layout options. Embed inline or display as a sidebar overlay.
- **Service orchestration**: You can use Decision Manager, Payer Authentication (3-D Secure), and Token Management Service (TMS) throughout the session.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Unified Checkout Quick Start

Unified Checkout is a powerful and flexible payment solution that simplifies the integration process and enhances the customer checkout experience. This guide will help you get up and running with Unified Checkout.

Key Features

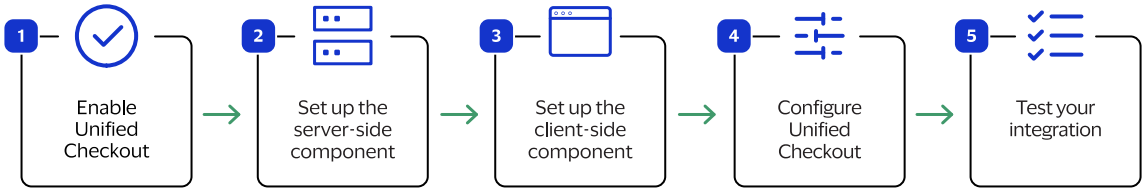
- Seamless integration with your existing e-commerce platform.
- Support for multiple payment methods.
- Customizable checkout flow.
- Enhanced security features.
- Responsive design for mobile and desktop.

Benefits

- Simplified integration process.
- Improved conversion rates.
- Reduced cart abandonment.
- Enhanced customer experience.
- Compliance with industry security standards.

This graphic provides an overview of the steps you must follow to get set up with Unified Checkout:

Unified CheckoutIntegration Overview



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

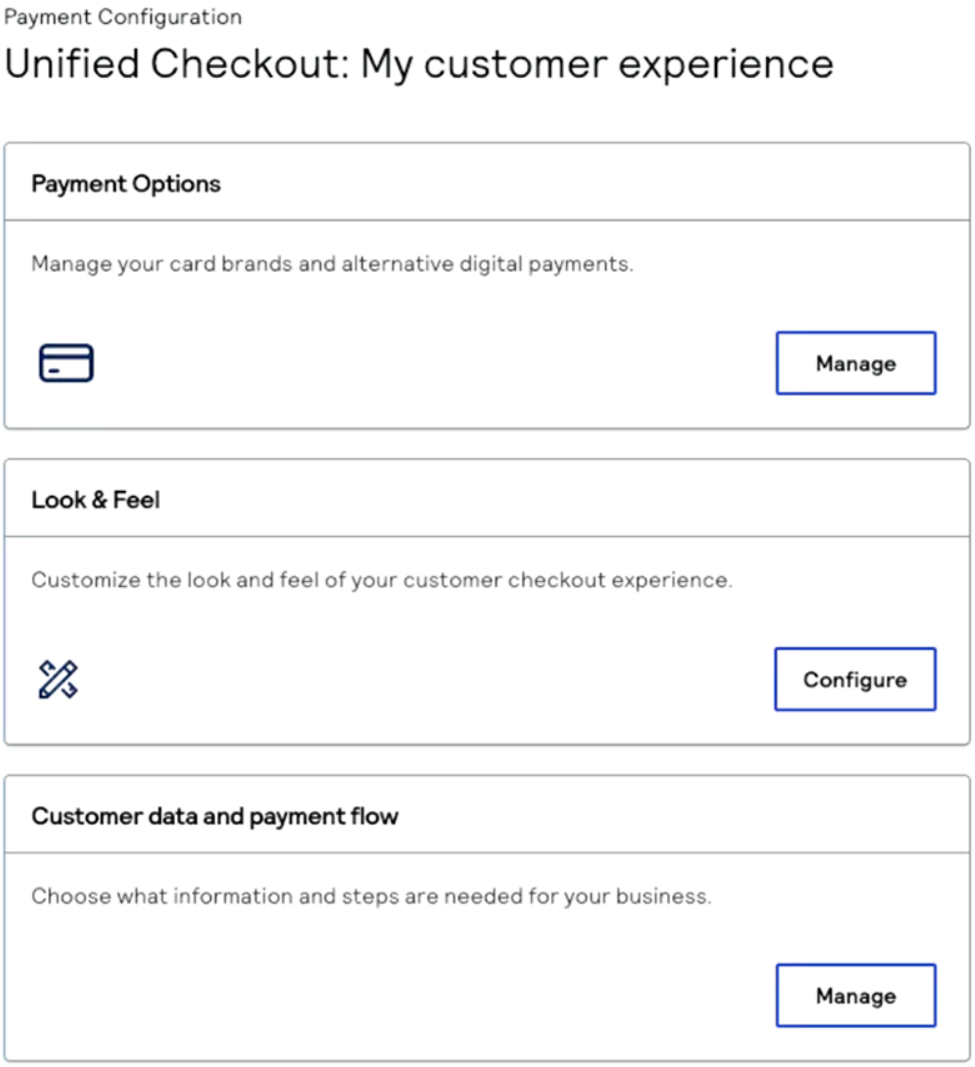
Step 1: Enable Unified Checkout

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**. The Unified Checkout customer experience page appears:

Unified Checkout Customer Experience



3. The Unified Checkout configuration interface provides complete low-code control over your checkout experience by using dedicated configuration screens accessible through the Business Center. The configuration interface is organized into separate screens, each accessible from the *My customer experience* page. Configure each of these components of the checkout experience:

- **Payment Options**: Add, remove, and arrange payment methods. For information about configuring payment methods, see Configure Payment Options (on page 253).
- **Look & feel**: Configure visual appearance and branding. For information about configuring the look and feel, see Customize the Unified Checkout Look and Feel (on page 255).
- **Customer information and payment flow**: Control data collection and manage payment processing service. For information about configuring customer data, see Configure Customer Data and Payment Flow (on page 266).

Proceed to Step 2: Set Up the Server-Side Component (on page 139).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Step 2: Set Up the Server-Side Component

To initialize Unified Checkout within your webpage, you need to set up the server-side component. This task involves generating a capture context. A capture context is a signed JSON Web Token (JWT) that contains your merchant configuration, one-time encryption keys, and payment parameters.

Follow these steps to make a server-to-server call to the Sessions API to authenticate your merchant credentials and establish how the Unified Checkout front-end components will function:

1. Implement a server-to-server call to the Sessions API.
   This call should include parameters that define how Unified Checkout performs.

2. Handle the response from the Sessions API.
   The response will contain:

   - A transaction-specific public key for securing the transaction in the customer's browser.
   - An authenticated context description package that manages the payment experience on the client side, including available payment options, interface styling, and payment methods.

3. Store and manage the JSON Web Token (JWT) object, referred to as the *capture context*.
   This JWT contains all the functions compiled from the Sessions API response:

```
{
  "targetOrigins": ["https://merchant.com", "https://reseller.com:8443"],
  "locale": "en_US",
  "country": "US",
    "data": {
        "orderInformation": {
          "amountDetails": {
            "totalAmount": "21.00",
            "currency": "USD"
      }
    }
  }
}
```

   The **targetOrigins** array must include every origin that will host the SDK. The response JWT is passed to the client-side library.

   This capture context contains only the minimum required fields. For information about the components of the capture context and how to create one using the Sessions API, see Sessions API (on page 280). For a complete capture context with all available fields, see Example: Unified Checkout Complete Capture Context (on page 301).

Proceed to Step 3: Set Up the Client-Side Component (on page 141).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Step 3: Set Up the Client-Side Component

To add the payment interface to your e-commerce site, you need to set up the client-side component using the Unified Checkout JavaScript library. This setup involves two primary components:

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

This example shows a complete client-side integration:

```
async function launchCheckout() {
  try {
    const client = await VAS.UnifiedCheckout(sessionJWT);
    const checkout = await client.createCheckout();
    const result = await checkout.mount('#payment-buttons');

    // result contains the completed payment result JWT
    // Send result to your server for verification
    sendToServer(result);
  } catch (error) {
    if (error.name === 'UnifiedCheckoutError') {
      handleError(error.reason, error.message);
    }
  } finally {
    checkout.destroy();
    client.destroy();
  }
}

launchCheckout();
```

Follow these steps to create the client-side integration:

1. Load the Unified Checkout JavaScript library:

   ```
   <script
    src="https://
   apitest.cybersource.com/uc/v1/assets/1.0.0/UnifiedCheckout.js"></script>
   ```

   Replace the domain with the production URL for live environments:

   - **Test**: `https://apitest.cybersource.com`
   - **Production**: `https://api.cybersource.com`

You must include the library in your webpage's HTML.

2. Initialize the SDK by calling `VAS.UnifiedCheckout()`. This returns a client instance:

```
const client = await VAS.UnifiedCheckout(sessionJWT);
```

Use the JWT obtained from the server-side setup in Step 2: Set Up the Server-Side Component (on page 139).

Initialization validates the JWT signature, checks that the current page origin matches `targetOrigins`, and prepares the SDK for use. If the JWT is invalid or expired, a `UnifiedCheckoutError` is returned.

3. Create a checkout to render a list of available payment methods and handles the payment flow:

```
const checkout = await client.createCheckout();
```

When you include `autoProcessing` and set it to `true`, `mount()` returns the completed payment result:

```
// Explicit auto-processing
const checkout = await client.createCheckout({ autoProcessing: true });
```

When you include it and set it to `false`, `mount()` returns a transient token that you pass to `checkout.complete()`:

```
// Explicit manual processing
const checkout = await client.createCheckout({ autoProcessing: false });
```

The default value of `autoProcessing` us `true` when a **completeMandate** is included in the session.

4. Mount the checkout by calling `mount()`. This attaches the payment UI to your page. The argument determines the display mode:

Sidebar Mode

The payment screen appears as an overlay sidebar. Pass a CSS selector for the payment button list, or omit it entirely:

```
// Full sidebar — buttons and payment screen both in sidebar
const result = await checkout.mount();

// Buttons embedded, payment screen in sidebar
const result = await checkout.mount('#payment-buttons');
```

Embedded Mode

Both the button list and payment screen render inline within your page layout:

```
const result = await checkout.mount({
  paymentSelection: '#payment-buttons',
  paymentScreen: '#payment-form'
});
```

Mount Result

When `autoProcessing` is enabled, `mount()` resolves with the completed payment result JWT once the customer finishes the payment flow.

When `autoProcessing` is disabled, `mount()` resolves with a transient token JWT. You then call **checkout.complete()** to finish the payment:

```
const checkout = await client.createCheckout({ autoProcessing:
 false });
const transientToken = await checkout.mount('#payment-buttons');

// Later, complete the payment
const result = await checkout.complete(transientToken);
```

Unmount

You can remove the payment UI from the page without destroying the checkout:

```
checkout.unmount();

// Later, mount again
const result = await checkout.mount('#payment-buttons');
```

For more information about setting up the client side, see Client-Side Set Up (on page 238). For information about handling errors on the client side, see Handle Errors (on page 329). Proceed to Step 4: Configure Unified Checkout (on page 145).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Step 4: Configure Unified Checkout

Proper configuration ensures that your checkout process aligns with your business needs and provides a smooth experience for your customers. You can configure the checkout process in the Business Center:

Select and configure the payment methods you support:

a. Log in to the Business Center and navigate to the Unified Checkout configuration section.

b. Select the payment methods you want to use:

- Credit and debit cards. See Cards (on page 151).
- eCheck/ACH Services. See eCheck/ACH Service (on page 156).
- Digital wallets such as Apple Pay, Click to Pay, and Google Pay. See Digital Wallets (on page 160).
- Alternative payment methods. See Alternative Payment Methods (on page 189).

> **Important:**
>
> You must configure the payment methods you want to use for each transacting MID.
>
> For information about which payment methods are supported on Unified Checkout, see Payment Methods (on page 149).

c. Configure the settings for each selected payment method.

For information about configuring Unified Checkout in the Business Center, see Configure the Unified Checkout Merchant Experience (on page 249).

Proceed to Step 5: Test Your Unified Checkout Integration (on page 146).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Step 5: Test Your Unified Checkout Integration

After configuring Unified Checkout, it's crucial to thoroughly test your integration to ensure it works correctly and provides a smooth checkout experience for your customers. This section outlines the steps to test your Unified Checkout integration.

1. Set up your test environment:

    a. Log in to the Business Center account using your test credentials.

    b. Switch to the test environment if not already in test mode.

    c. Set up a test website or application that integrates Unified Checkout.

2. Use test card numbers to simulate different payment scenarios.

    For more test payment data, see Unified Checkout Test Cards (on page 321).

3. Test different payment scenarios:

    - Successful transactions
    - Declined transactions
    - 3-D Secure authentication, if applicable
    - Different card brands
    - Digital wallet payments, if configured

4. Verify that the capture context object that you get from the sessions API is correct and that your integration can handle tokens.

    Ensure that your integration correctly handles the capture context and transient tokens throughout the payment process.

5. Test error handling and edge cases.

    Simulate various error scenarios to ensure your integration gracefully handles and reports errors to the user.

6. Verify webhook notifications, if configured.

    If you set up webhook notifications, ensure that your system correctly receives and processes them for various transaction events. For information about configuring your webhook notifications, see Webhooks Support (on page 278).

After completing these testing steps, you should have confidence in your integration. Remember to test in both the test and production environments before going live.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

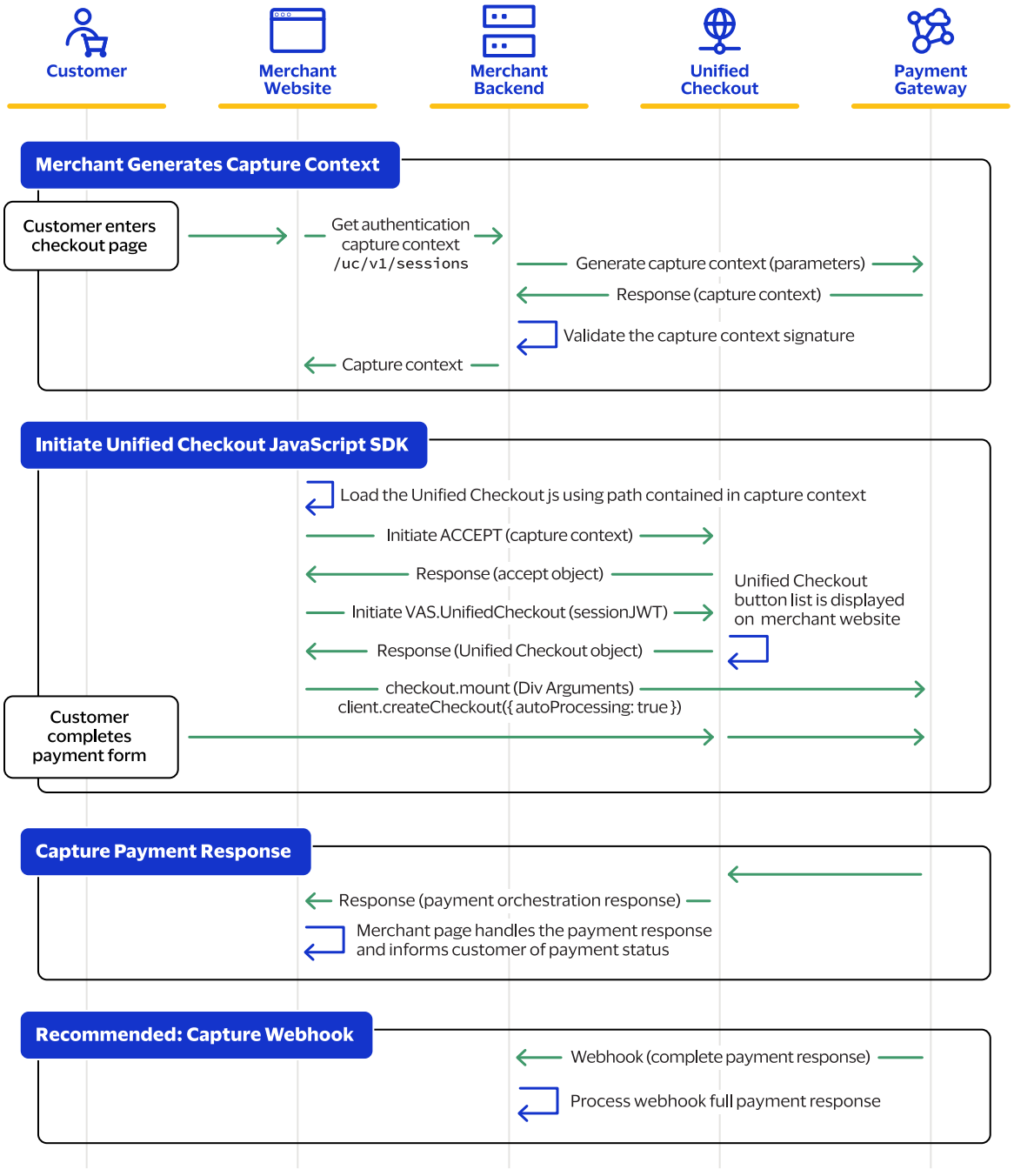
Unified Checkout Flow

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. You can find the detailed specifications of the APIs later in this document.

Information that is captured by Unified Checkout, including the billing and shipping address, can be retrieved using the Payment Details API.

The figure below shows the Unified Checkout payment flow using the Sessions API to generate the capture context:

Unified Checkout Payment Flow



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

- Sessions API (on page 280): This generates the capture context and determines what fields are displayed to the customer in the UI during checkout.
- Payment Details API (on page 338): This API can be used to retrieve personally identifiable information that is associated with a Unified Checkout transient token, such as the cardholder name and billing and shipping details, without retrieving payment credentials.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Payment Methods

This section describes the payment methods you can use in your Unified Checkout integration. After you successfully integrate one payment method, you can add another from the same category with minimal adjustments to your existing configuration.

These payment methods are available:

- Cards (on page 151)
- eCheck/ACH Service (on page 156)
- Apple Pay (on page 162)
- Google Pay (on page 165)
- Click to Pay (on page 169)
- Paze (on page 186)
- Online Bank Transfers (on page 190)
- Buy Now, Pay Later (on page 205)

- Post-Pay Reference Payments (on page 211)
- PayPal (on page 219)
- Venmo (on page 222)

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Cards

Unified Checkout accepts multiple card types including global networks such as Visa, Mastercard, and American Express. Unified Checkout also accepts local schemes such as Cartes Bancaires in France, EFTPOS in Australia, and PayPak in Pakistan.

Card Support

Support for card brands varies based on the payment method for these services:

- Payments
- Decision Manager
- Payer Authentication

This table shows which card types are accepted for each payment method and which region:

Card Brand by Region and Payment Method

Region	Card Brand	Manual Card Entry	Apple Pay	Click to Pay	Google Pay	Paze
Asia Pacific	China UnionPay	✓	✗	✗	✗	✗
Asia Pacific	EFTPOS	✓	✗	✗	✗	✗
Asia Pacific	JCB	✓	✗	✗	✗	✗
CEMEA	mada	✓	✗	✗	✗	✗
CEMEA	Meeza	✓	✗	✗	✗	✗
CEMEA	Jaywan	✓	✗	✗	✗	✗
CEMEA	PayPak	✓	✗	✗	✗	✗
Europe	Cartes Bancaires	✓	✗	✗	✗	✗
Global	American Express	✓	✓	✓	✓	✗
Global	Diners Club	✓	✗	✗	✗	✗
Global	Mastercard	✓	✓	✓	✓	✓
Global	Visa	✓	✓	✓	✓	✓

Card Brand by Region and Payment Method (continued)

Region	Card Brand	Manual Card Entry	Apple Pay	Click to Pay	Google Pay	Paze
Global and Europe	Maestro	✓	✗	✗	✗	✗
Latin America	Carnet	✓	✗	✗	✗	✗
Latin America	ELO	✓	✗	✗	✗	✗
US and Canada	Discover	✓	✗	✗	✗	✗
US and Canada	JCrew	✓	✗	✗	✗	✗

This table shows which card types are supported for each complete mandate feature by region.

Card Support for the Complete Mandate

Region	Card Brand	Authorization	Payer Authentication	Decision Manager	Token Create by Token Management Service
Asia Pacific	China UnionPay	✓	✓	✓	✓
Asia Pacific	EFTPOS	✓	✓	✓	✓
Asia Pacific	JCB	✓	✓	✓	✗
CEMEA	mada	✓	✓	✓	✓
CEMEA	Meeza	✓	✓	✓	✗
CEMEA	Jaywan	✓	✓	✓	✗
CEMEA	PayPak	✓	✓	✓	✗
Europe	Cartes Bancaires	✓	✓	✓	✓
Global	American Express	✓	✓	✓	✓
Global	Diners Club	✓	✓	✓	✓

Card Support for the Complete Mandate (continued)

Region	Card Brand	Authorization	Payer Authentication	Decision Manager	Token Create by Token Management Service
Global	Mastercard	✓	✓	✓	✓
Global	Visa	✓	✓	✓	✓
Global and Europe	Maestro	✓	✓	✓	✓
Latin America	Carnet	✓	✗	✓	✓
Latin America	ELO	✓	✓	✓	✓
US and Canada	Discover	✓	✓	✓	✓
US and Canada	JCrew	✓	✗	✓	✓

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Pay with Token

You can use Unified Checkout to pass through a single token ID to be shown within the Unified Checkout UI. To display a payment instrument in the Unified Checkout UI, you must include `TMS_TOKEN` as an allowed payment type in the **allowedPaymentTypes** field object and the details of the Token Management Service (TMS) token in the **paymentConfigurations** field object in the capture context request:

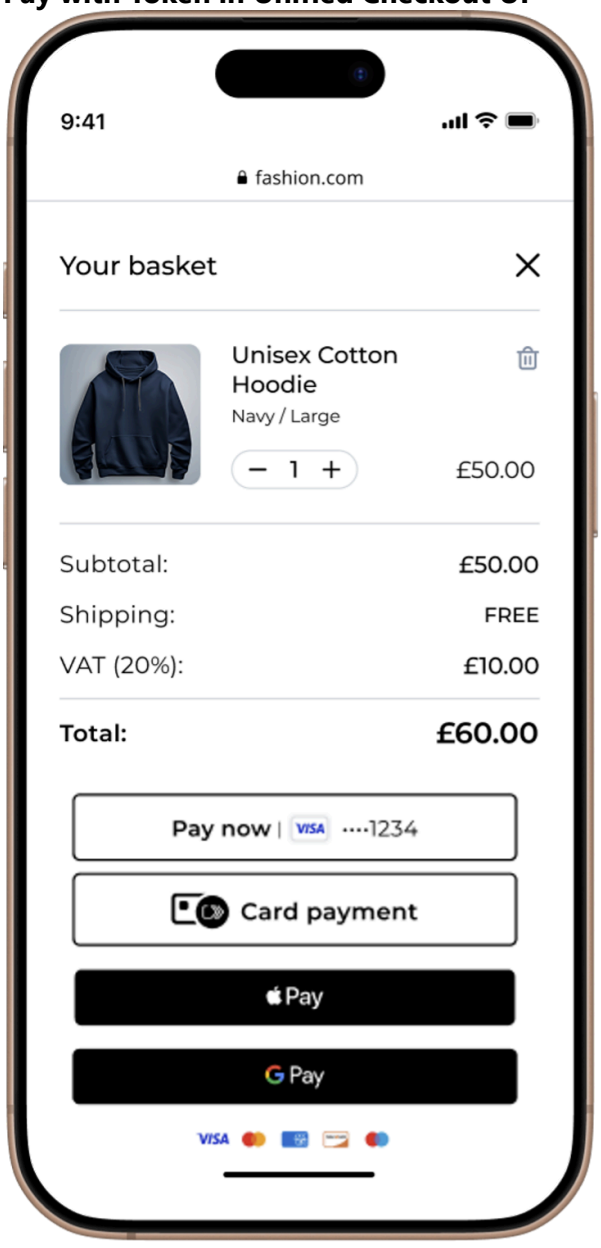
```
"allowedPaymentTypes": [
    "PANENTRY",
    "TMS_TOKEN"

},
  "paymentConfigurations": {
    "TMS_TOKEN": {
```

```
    "paymentInstruments": [
      {
        "id": "404352E77F6A66E7E0634136CF0ABCD7"
      }
```

This is an example UI with a payment instrument:

Pay with Token in Unified Checkout UI



You can use these token types to pay with a token in Unified Checkout:

Customer Tokens

When you include the customer token, your UI displays the default payment instrument that is inked to a customer. To display a customer token, you must include the **paymentConfigurations.TMS_TOKEN.customer.id** field in your Sessions API request.

Important: When you include a customer token ID here with **tokenCreate** for a **paymentInstrument** or **instrumentIdentifier**, the complete mandate creates a new payment instrument or instrument identifier within the level of the customer token that you provide.

```
"paymentConfigurations": {
    "TMS_TOKEN": {
      "customer": {
        "id": "404352E77F6A66E7E0634136CF0ABCD7"
       }
```

Instrument Identifier Tokens

When you include an instrument identifier token, your UI displays the payment instrument that is associated with the specified instrument identifier. To display an instrument identifier token, you must include the **paymentConfigurations.TMS_TOKEN.instrumentIdentifiers.id** field in your Sessions API request.

```
"paymentConfigurations": {
    "TMS_TOKEN": {
      "instrumentIdentifiers": [
        {
          "id": "4B1BCB328D52ED86E063AF598E0A99A5"
        }
```

Payment Instruments

When you include a payment instrument, your UI displays the payment instrument that is associated with the specified payment instrument token identifier. To display a payment instrument, you must include the **paymentConfigurations.TMS_TOKEN.paymentInstruments.id** field in your Sessions API request.

```
"paymentConfigurations": {
    "TMS_TOKEN": {
      "paymentInstruments": [
        {
          "id": "404352E77F6A66E7E0634136CF0ABCD7"
        }
```

Important: To make a new payment instrument or instrument identifier under an existing customer during the complete mandate, you must meet these requirements:

- You must include the customer token ID in the **paymentConfigurations** field object.
- `TMS_TOKEN` must be included in the **allowedPaymentTypes** field object.
- **tokenCreate** must be set to `true` and `paymentInstrument` and `instrumentIdentifier` must be included as values in the **tms.tokenTypes** field array. For example:

```
"tms": {
      "tokenCreate": true,
      "tokenTypes": [
        "paymentInstrument",
        "instrumentIdentifier",
      ]
    }
```

When you meet these requirements, a new payment instrument or instrument identifier is created under the specified customer token.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

eCheck/ACH Service

Unified Checkout supports the acceptance of eCheck information. Sensitive eCheck data is securely captured and replaced with a token. Acceptance of eCheck information enables merchants to collect funds from a customer's bank account through both the ACH service and eCheck service (US only) for either of these flows:

- ACH services are a set of connections composed of the legacy gateway solutions where Cybersource serves as the gateway.
- eCheck, the new service on Payments 2.0, is the acquirer solution where Cybersource is the acquirer.

Unified Checkout replaces these eCheck information fields in your payment input form:

- Routing number
- Account number
- Account type (non-sensitive)

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Enrolling in eCheck/ACH Services

Unified Checkout can accept bank account payments using the eCheck product. To accept eCheck payments through Unified Checkout, you must have the eCheck processing service enabled. To request access to eCheck processing and enable eCheck, you must submit an application in the Business Center. Once your application is approved, you can accept eCheck payments.

For step-by-step instructions on enrolling and enabling eCheck, see the "Getting Started with the eCheck Service" section of the eCheck Merchant Guide. If eCheck is not listed in the Available Products section in the Business Center, you must contact your portfolio owner to enable your account to apply for eCheck.

Important: If you have a business account or a financial relationship with Bank of America, Wells Fargo, or Chase, and you would like them to process your transactions, you must contact our Sales or Support team for more information on our ACH product.

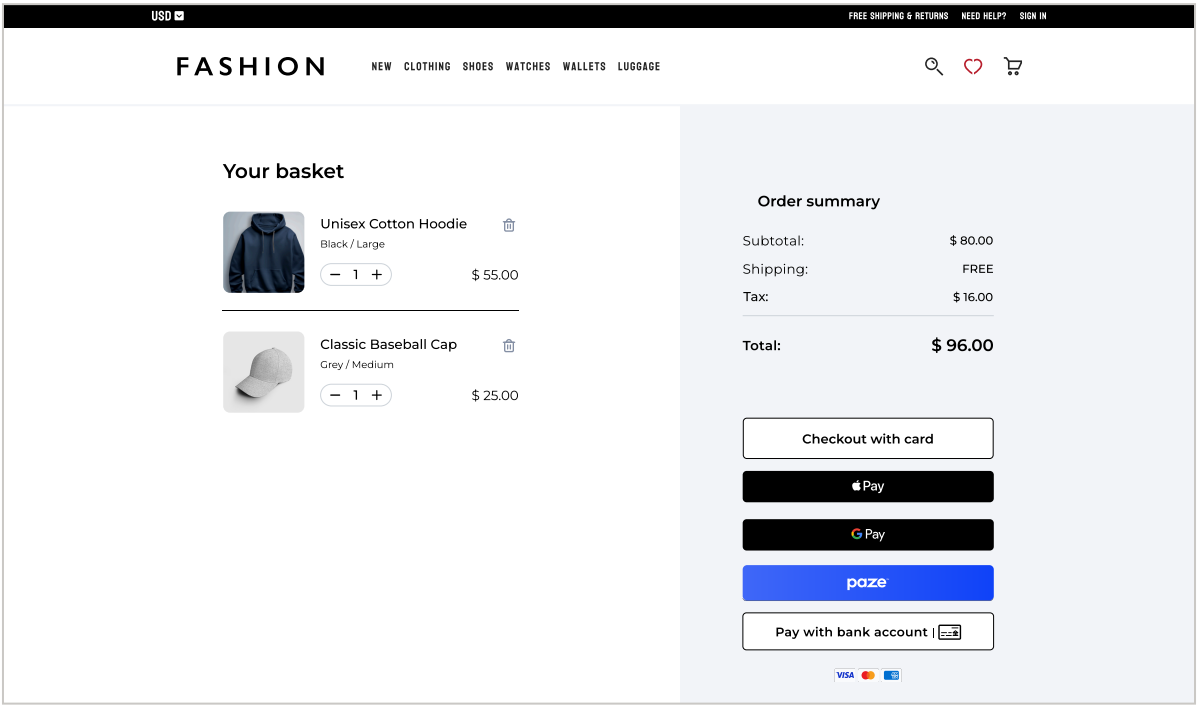
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

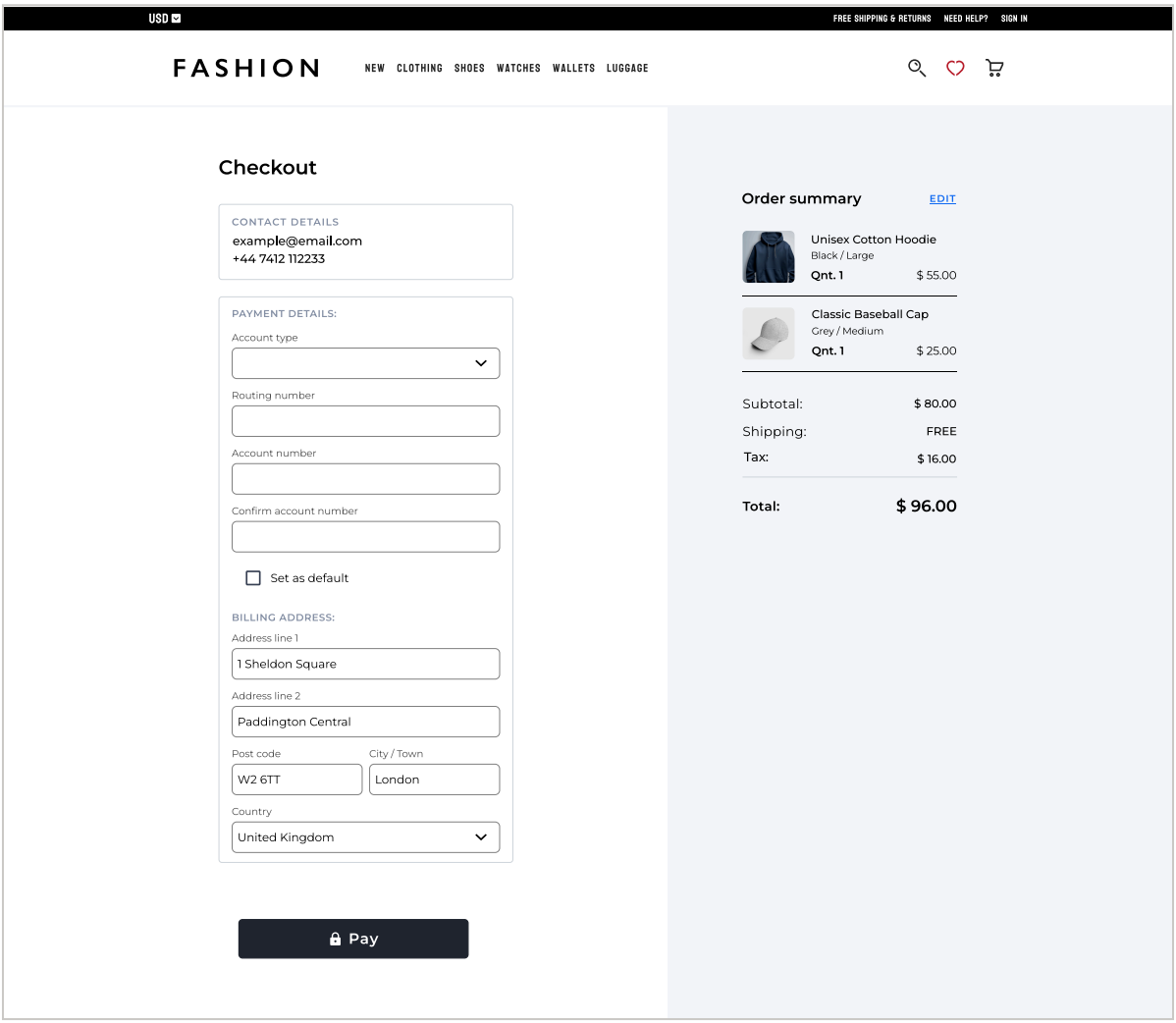
Pay with eCheck/ACH Service UI

These screen captures show the sequence of events your customer can expect when completing a payment with the eCheck/ACH service.

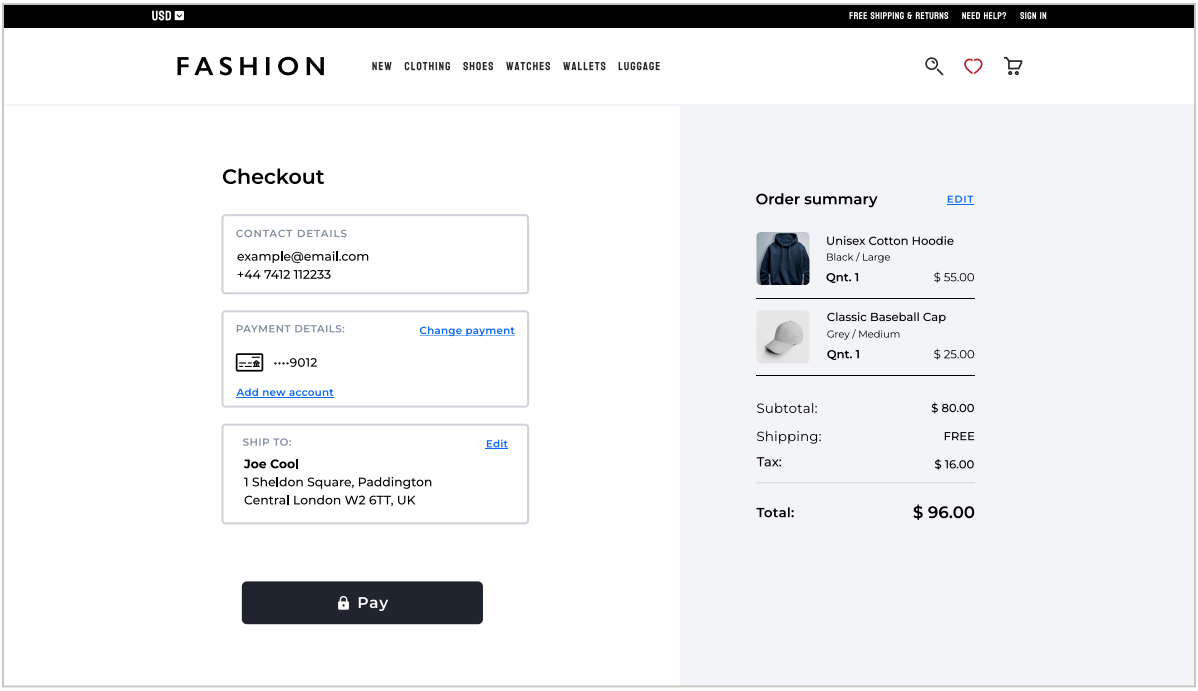
eCheck/ACH Service Account Order Summary



Pay with eCheck/ACH Service Checkout



Pay with eCheck/ACH Service Review and Confirm



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Digital Wallets

Digital wallets are secure applications or services that enable users to store payment details, such as debit or credit cards electronically.

Digital wallets such as Apple Pay, and Google Pay are accessible with smartphones, computers, or even directly in web browsers. Digital wallets allow customers to pay for goods and services both online and in physical stores without a physical card, often using biometrics, a device passcode, or wallet login, and approve the payment in just a few clicks.

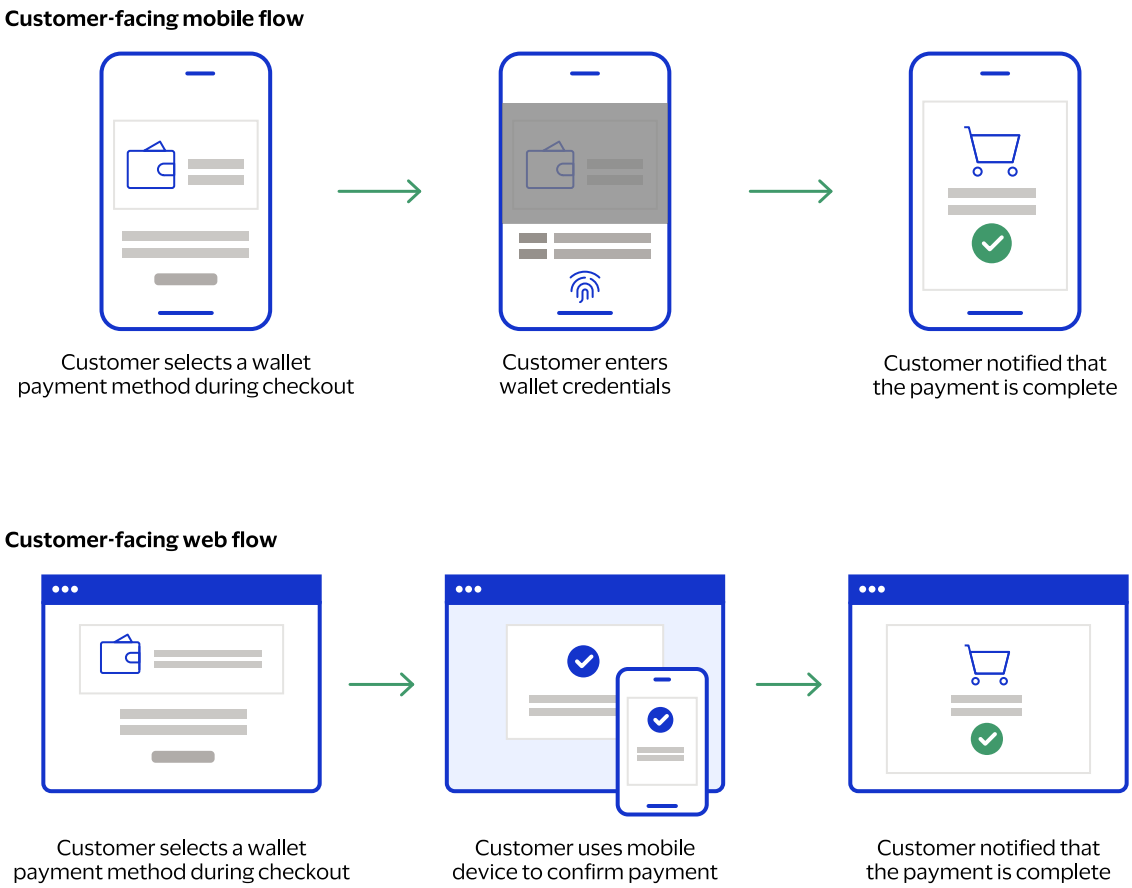
For online transactions, the wallet securely passes a payment token to the merchant or payment processor. This means that you do not handle sensitive customer data directly. This reduces friction at checkout, improves conversion rates, and enhances security through built-in authentication and tokenization.

Wallets are best suited for one-time online purchases and express checkout experiences. Support for subscriptions and recurring payments varies by wallet, so you must ensure compatibility if future charges or merchant-initiated transactions are required.

Unified Checkout supports these wallet-based payment options:

- Apple Pay
- Google Pay
- Click to Pay
- Paze

This is how payments with digital wallets work:



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Apple Pay

Apple Pay is a digital payment solution that enables your customers to make secure and convenient purchases without requiring them to enter their card details or shipping information. This section includes information about accepting Apple Pay payments with your Unified Checkout integration.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Enrolling in Apple Pay

Apple Pay is a digital payment service that enables users to make secure and convenient transactions using their Apple devices. Users can add their credit or debit cards to the Wallet app and use them to pay online or in apps in a safe and convenient consumer experience.

To enable Apple Pay you must first host a public certificate on your web page and then pass your merchant name and domain name to Apple. Apple crawls out to your web page to validate the presence of this certificate to ensure the web pages are properly vetted and registered with Apple.

Follow these steps to validate your domain and enroll in Apple Pay:

1. Navigate to **Payment Configuration > Unified Checkout**.
2. In the Apple Pay section, click **Set Up**.
3. Follow the link to download the certificate.
4. Upload the *apple-developer-merchantid-domain-association* certificate file to your web server at:
   `/.well-known/apple-developer-merchantid-domain-association`
   You must verify that the file is accessible through HTTPS. You can validate this by visiting `https://<your-domain>/.well-known/apple-developer-merchantid-domain-association`.

5. Click **Verify Domain**.

6. Enter the domain name where you are hosting Apple Pay. This must be the same domain to which you uploaded the public certificate.
   Your domain is now verified for Apple Pay.

Important: In order to run an end-to-end test of the Apple Pay service on Unified Checkout, you must perform additional setup steps. See Preparing a Device for Testing Apple Pay on Unified Checkout (on page 163).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Preparing a Device for Testing Apple Pay on Unified Checkout

To run an end-to-end test of the Apple Pay service on Unified Checkout, you must prepare an Apple test device by loading Apple Pay test cards onto the device.

Follow these steps to prepare your Apple test device for end-to-end testing:

1. Make sure your Apple Developer account is configured for Apple Pay.

2. Register your Apple Pay test device with Apple.

3. Load Apple Pay test cards onto your Apple test device.
   The Apple Developer center provides the instructions in the Sandbox Testing page for Apple Pay:

   a. Follow the steps described in *Create a Sandbox Tester Account*.

   b. Follow the steps described in *Adding a Test Card Number*.

Related information

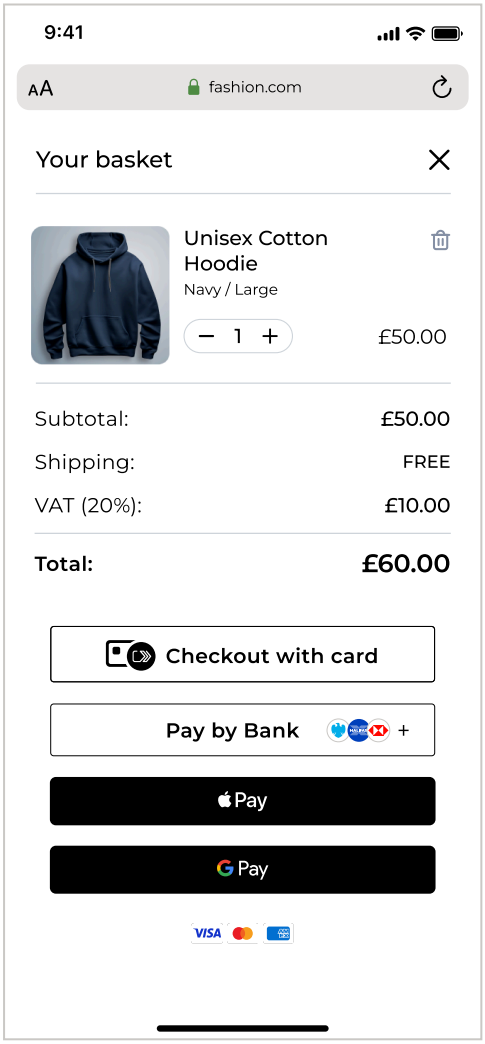
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

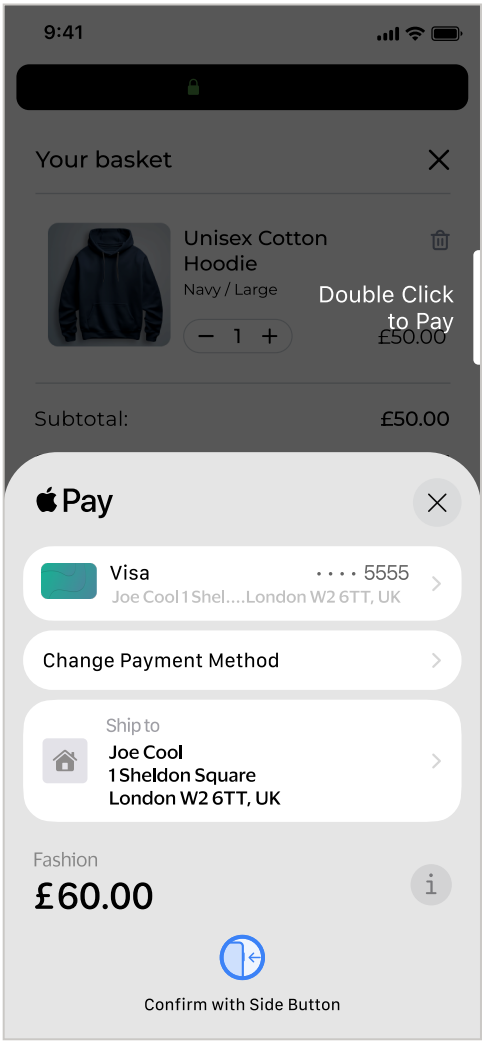
Apple Pay UI

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

Apple Pay UI



Apple Pay in the payment method list



Apple Pay checkout

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Google Pay

Google Pay is a simple, secure in-app mobile and Web payment solution. This section includes information about accepting Google Pay payments with your Unified Checkout integration.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Important: When you enable Google Pay on Unified Checkout, you can specify an optional parameter that defines the types of credentials that Google Pay sends you. See Managing Google Pay Authentication Types (on page 166).

Related information
Getting Started with REST
Response Codes
API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Managing Google Pay Authentication Types

Additional controls are available for Google Pay on Unified Checkout. When you enable Google Pay on Unified Checkout, you can specify optional parameters that define the types of card authentication you receive from Google Pay.

To manage the types of credentials that Google Pay sends, use this expanded payment type object within the **allowedPaymentTypes** section of the sessions request:

```
"paymentConfigurations": {
    "GOOGLEPAY": {
      "allowedAuthMethods": "<authentication type>"
    }
```

The expanded payment type object has these parameters:

- **type**: Defines the type of payment option.
- **options**: Contains specific payment types parameters.

  For Google Pay, use the new data element **allowedAuthMethods** within the **options** section of the payment types object to specify the authentication type you will receive from Google Pay. Possible values:

  - `PAN_ONLY`: Google returns primary account number (PAN) values
  - `CRYPTOGRAM_3DS`: Google returns fully authenticated network token values.

  By default, Google sends both authentication types.

  Important: When the complete mandate is used and Google Pay does not authenticate the transaction, then Unified Checkout completes the authentication request as part of the complete mandate.

REST Example: Specify Only PAN Authentication Accepted from Google

This sessions request example specifies that Google Pay is to send only PAN values.

```
"allowedPaymentTypes": [
  "GOOGLEPAY"
],
"paymentConfigurations": {
    "GOOGLEPAY": {
      "allowedAuthMethods": [
        "PAN_ONLY",
        "CRYPTOGRAM_3DS"
      ]
    }
  }
```

REST Example: Simple Google Pay Request

This sessions request example specifies that Google Pay can send all authentication types. This can be enabled and configured in the Business Center. For information about configuring your allowed payment types, see Configure Payment Options (on page 253).

```
"allowedPaymentTypes": [
  "PANENTRY",
  "GOOGLEPAY",
  "CLICKTOPAY",
  "PAZE",
  "CHECK"
]
```

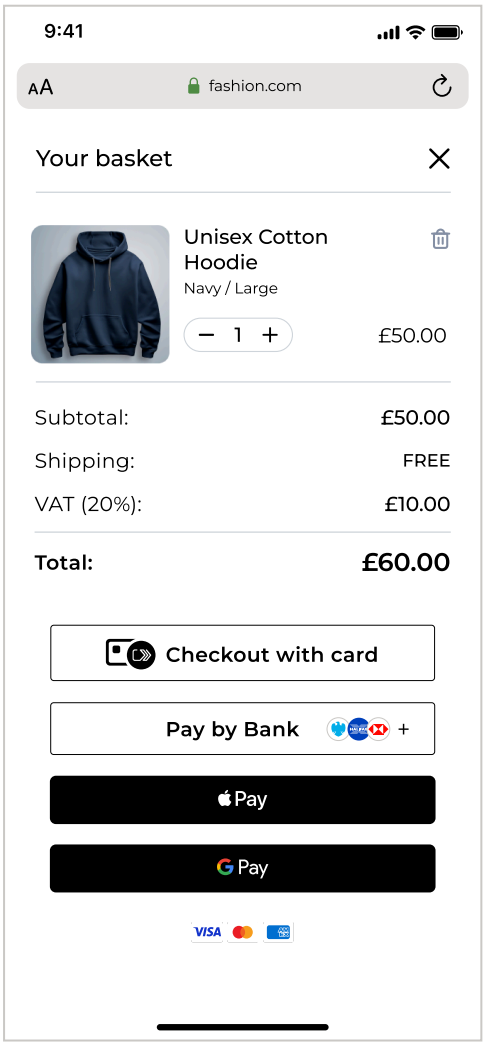
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

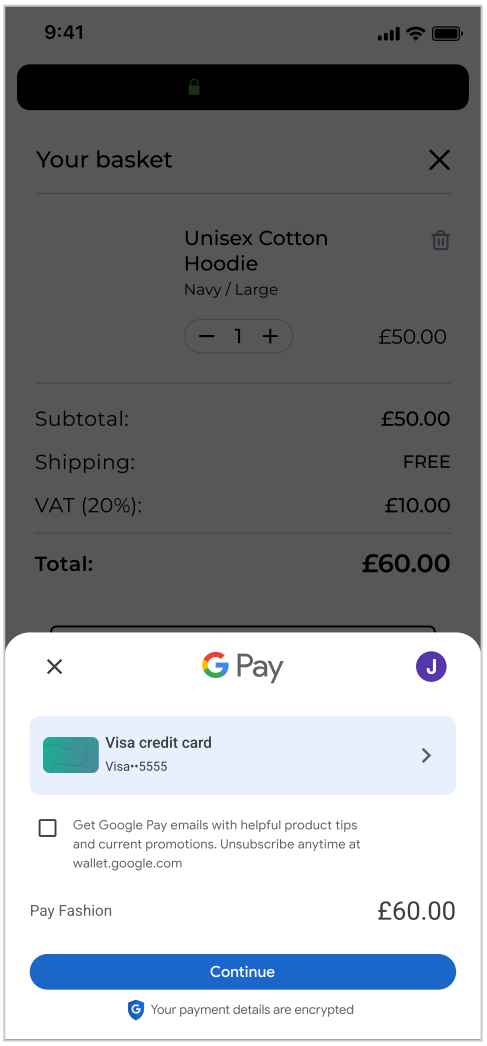
Google Pay UI

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

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Click to Pay

Click to Pay is a secure online checkout method that enables customers to make purchases without entering their payment details for every purchase. This section includes information about accepting Click to Pay payments with your Unified Checkout integration.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Enabling Click to Pay in the Business Center

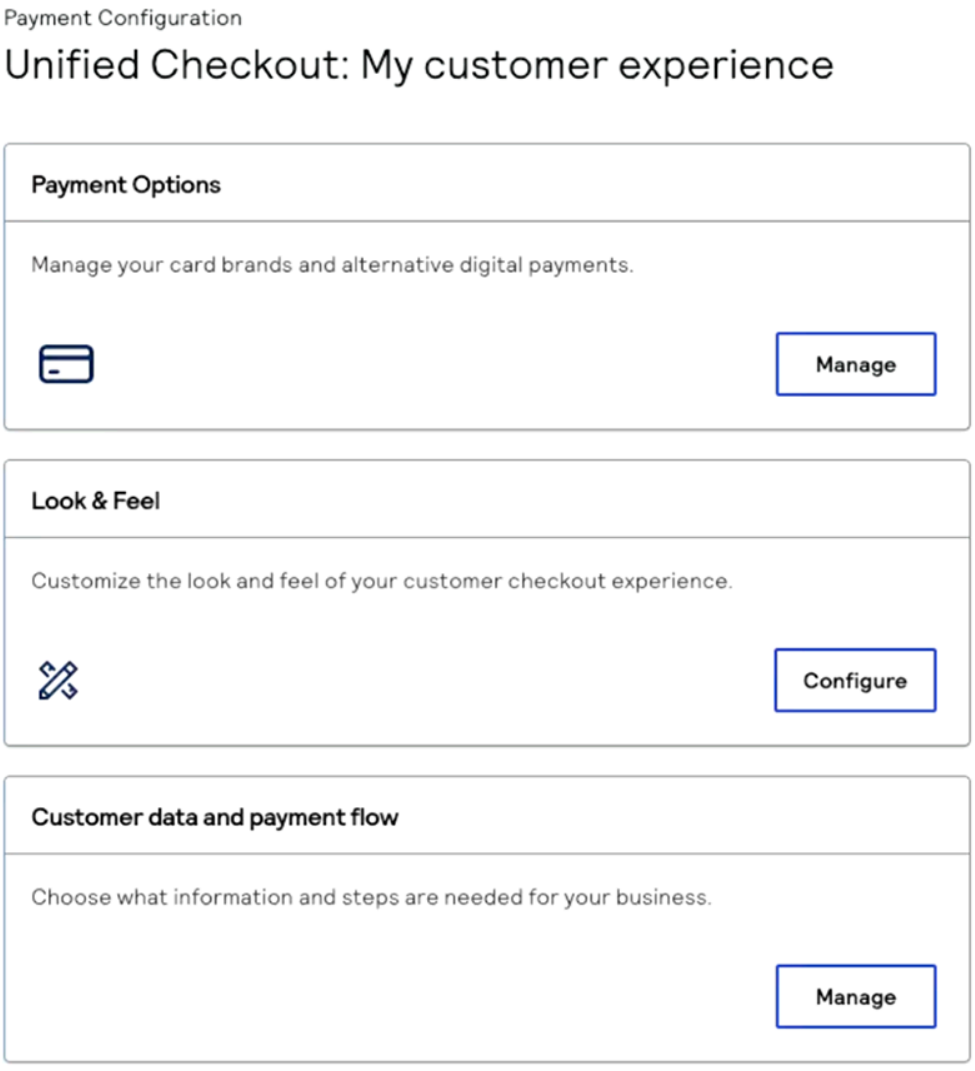
To begin your integration, you must first enable Click to Pay. Click to Pay is a digital payment solution that allows customers to pay with their preferred card network and issuer without entering their card details on every website. Customers can use Visa, Mastercard, and American Express cards to streamline 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. 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**. The Unified Checkout customer experience page appears:

Unified Checkout Customer Experience



3. In the Payment Options section, click **Manage**. The Payment Options page appears.

4. Click **Manage** next to Click to Pay. The Click to Pay configuration page appears.
5. Enter your business name and website URL.
6. Click **Submit**.

> **Important:** Click to Pay uses network tokenization for transactions. These network tokens are stored in the vault of the token requestor ID (TRID) for the card scheme.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Set Up Customer Authentication for Visa Click to Pay

Follow these steps to use the Business Center to enable customer authentication through Click to Pay. Authentication methods differ in each region and are dependent on the issuer, the cardholder device, and the Click to Pay configuration. These authentication methods are available:

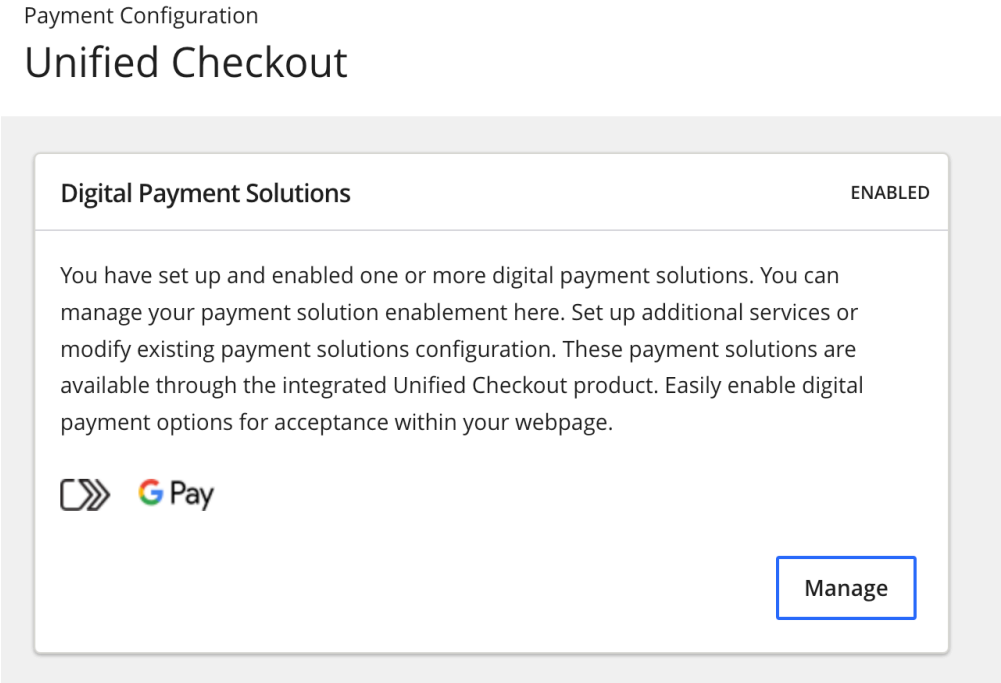
- 3-D Secure
- FIDO
- Card verification value (CVV)
- One-time password (OTP)

> **Important:** After you complete these steps, Visa determines which authentication method to use. When Visa determines that they will authenticate, they authenticate each Click to Pay transaction through the appropriate method. This may be a frictionless authentication or the customer may need to provide more information when required by the issuer. This is available only through Visa.

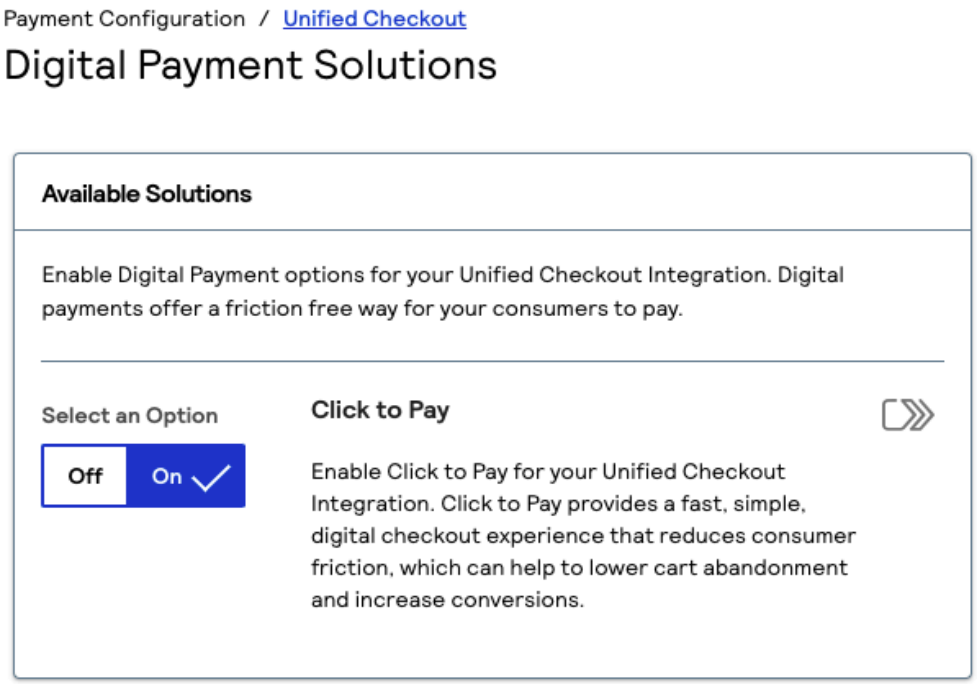
> **Important:** Visa Click to Pay authentication is not the same as consumer authentication using the complete mandate. See Test Authentication (on page 326).

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.

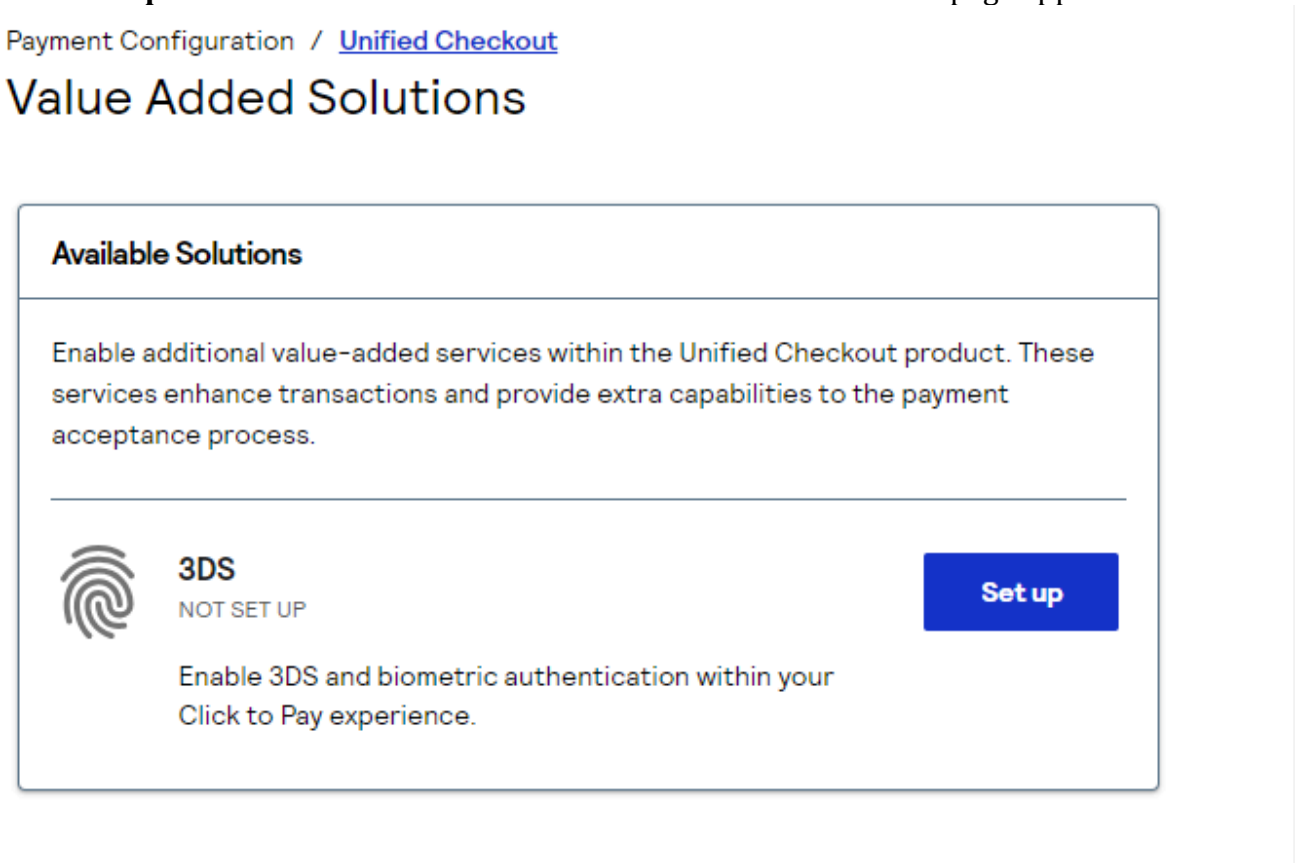
2. In the Business Center, go to the left navigation panel and choose **Payment Configuration** > **Unified Checkout**.
   You must have Click to Pay enabled as a digital payment method in order to use this method of authentication. Click **Manage** to view the digital payment methods that you have enabled.



   If Click to Pay is not enabled, click **On** next to Click to Pay.

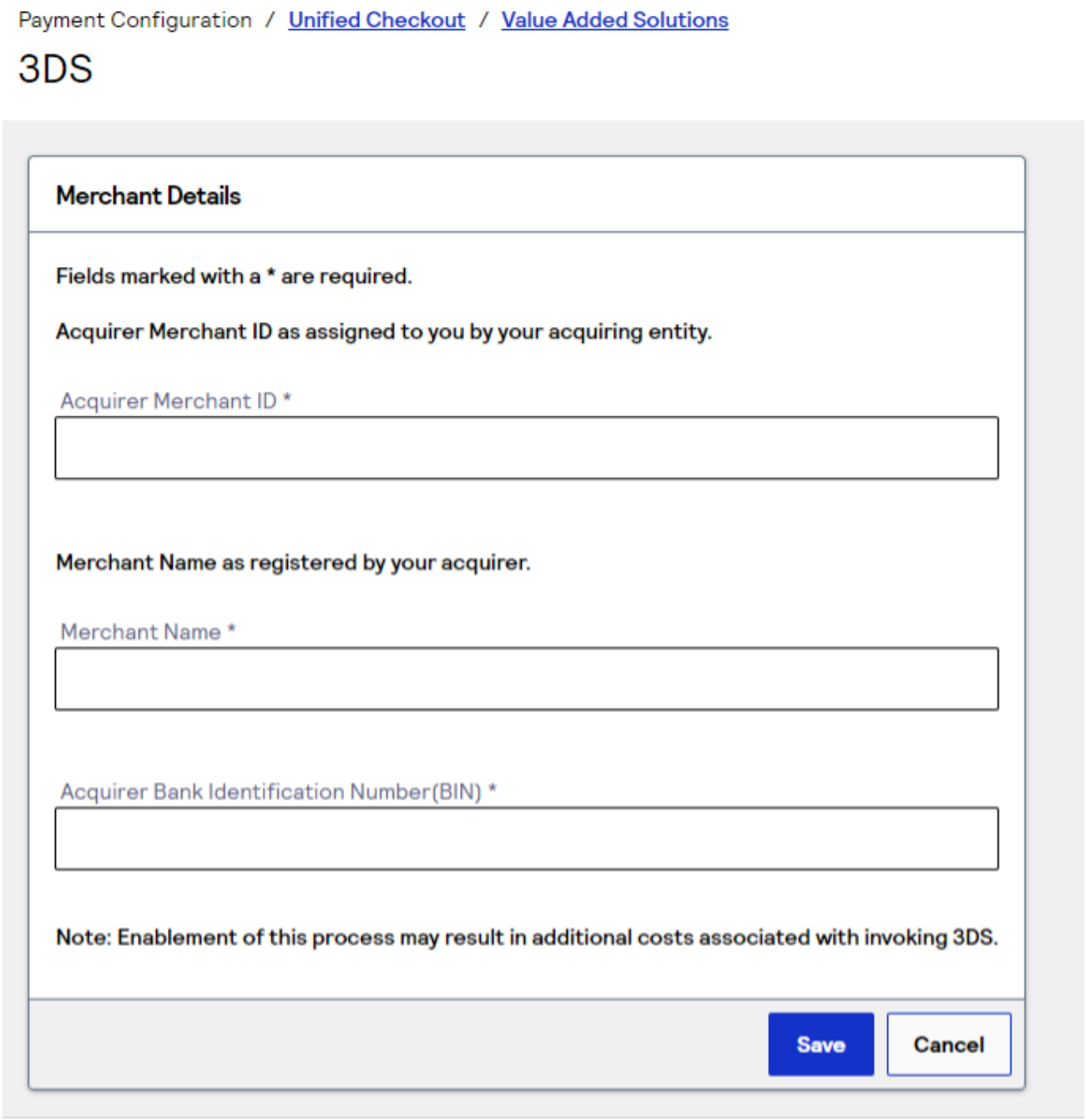


3. Click **Set up** under Value Added Solutions. The Value Added Solutions page appears.



4. Click **Set up** to set up 3-D Secure. The 3DS page appears.

5. Enter the required information in the Merchant Details section. You must enter the information that is provided to you by your acquirer or processor.



This completes the authentication setup for the entered acquirer merchant ID and BIN. If you do not know what these values are, you must contact your acquirer. Completing this information enables Cybersource to send Visa the information that is required for authentication.

Important: Charges for 3-D Secure may apply. You must speak with your acquirer for more information about the charges associated with 3-D Secure.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Click to Pay Customer Authentication

When you enable customer authentication through Click to Pay, you give Cybersource permission to send Visa the required authentication information for each transaction. When the customer completes a transaction using a Visa card that is already stored in Click to Pay, authentication is managed within Click to Pay.

Click to Pay authentication is only available for Visa branded cards that are tokenized with Click to Pay. If Click to Pay does not authenticate the transaction, but you are using the complete mandate with the **consumerAuthentication** field set to `true`, authentication is attempted as part of this request. When you do not use the complete mandate, you must check the result of the **cardholderAuthenticationStatus** field in the transient token and request Payer Authentication directly when it is required.

Important: American Express and Mastercard card brands cannot be authenticated through Click to Pay customer authentication.

Authentication Flow

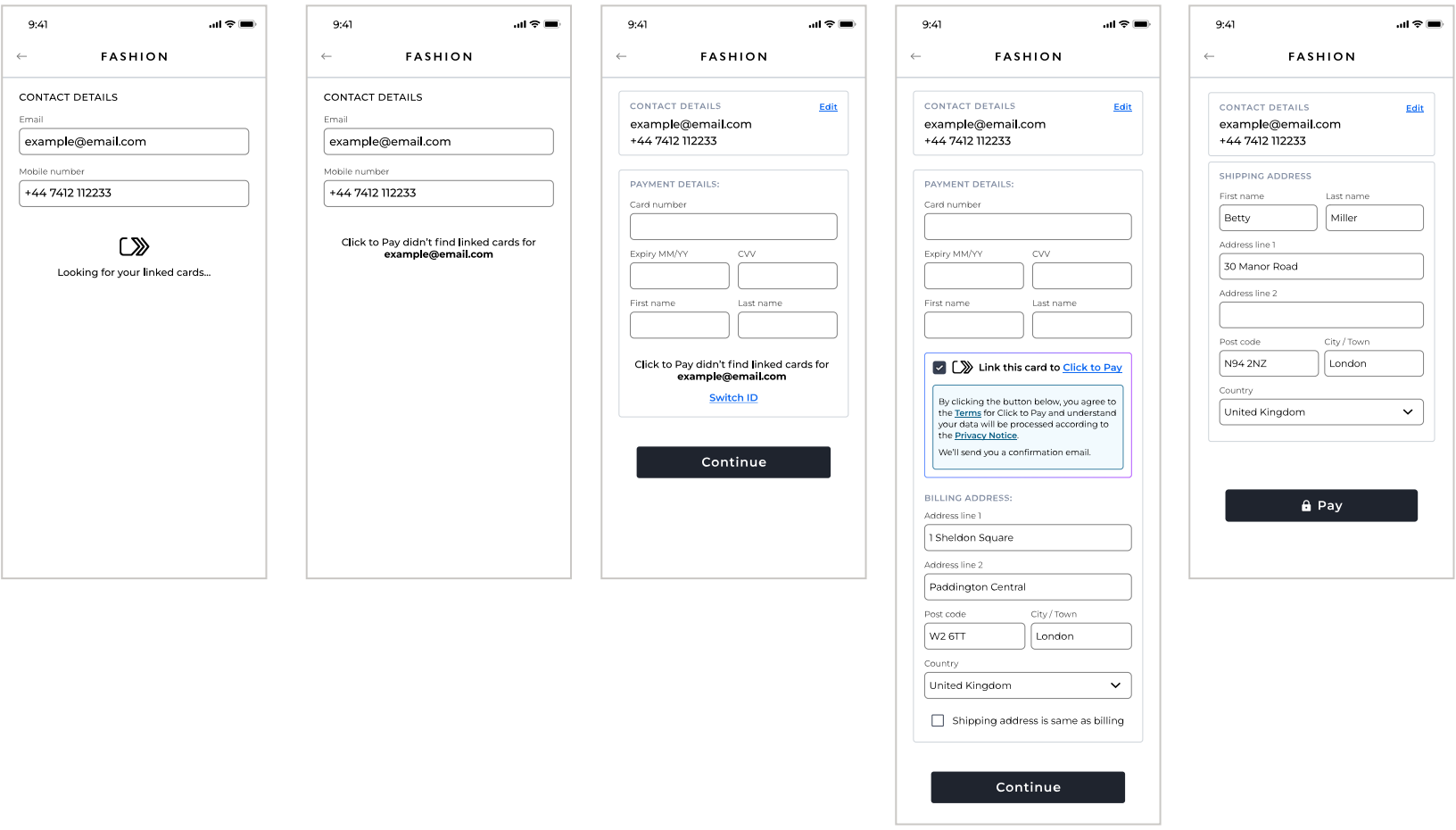
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Click to Pay UI Guidelines

The UI that is built in Unified Checkout for Click to Pay is built based on the EMV Click to Pay XC Guidelines V1.1. Unified Checkout has simplified the integration of the UI. The only UI work that you must complete is the placement of the payment option.

Important: You must include Click to Pay as one of the presented payment methods and not as a separate payment method.

Unified Checkout captures all card details that are manually entered by the cardholder. This enables the cardholder to enroll in Click to Pay and removes the requirement for the cardholder to manually enter their card details the next time they check out.

Unified Checkout provides a standard payment label in the Unified Checkout JavaScript that is loaded in your checkout page. One of these scenarios occurs when the cardholder selects the button:

- The cardholder is recognized.
- The cardholder is not recognized but has a Click to Pay account.
- The cardholder does not have a Click to Pay account.

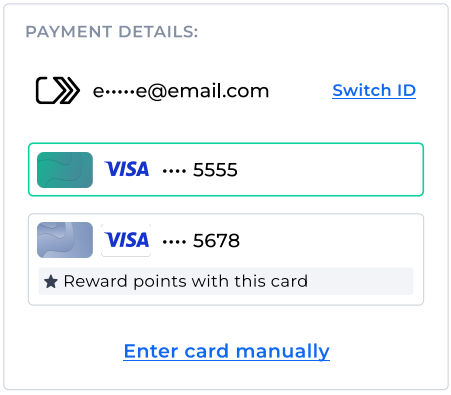
You can also trigger the Unified Checkout flow using a custom button. If you are using your own custom button, your payment button or widget must display the Click to Pay image for the cardholder. For information about a custom button, see JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry (on page 241).

Important: Your implementation consultant will ask you for a mock-up of your payment flow for confirmation that it is compliant with the Click to Pay UI design standards.

Recognized Click to Pay Customer

The cardholder is presented with their stored Click to Pay cards in the UI when they are on a recognized device:

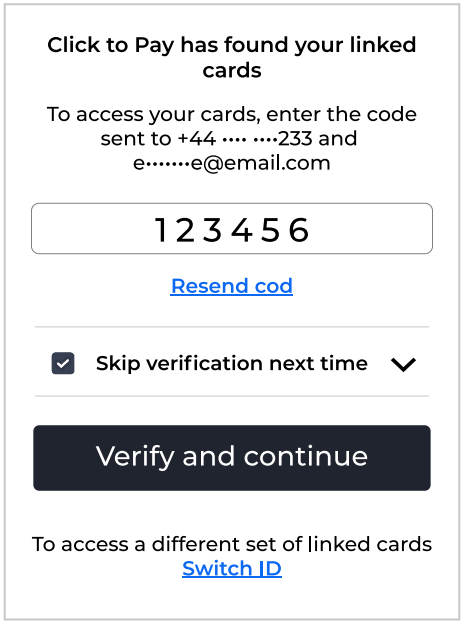
Recognized Click to Pay Customer UI



Unrecognized Click to Pay Customer

When the cardholder has a Click to Pay account but is not on a registered device, they receive a one-time password to their registered email address and phone number to authenticate their identity before their stored Click to Pay credentials are shown:

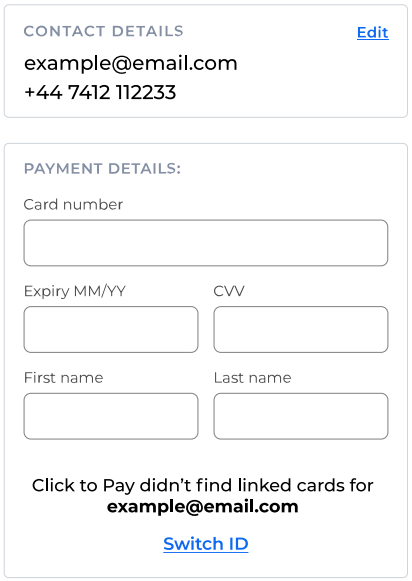
Unrecognized Click to Pay Customer on a Recognized Device UI



No Click to Pay Account

When the cardholder does not have a Click to Pay account, they can provide a new email address to perform a new lookup or they can choose to enter their card details manually. The cardholder can make a one-time payment or complete the payment and choose to create a Click to Pay account for future use:

No Click to Pay Account UI



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

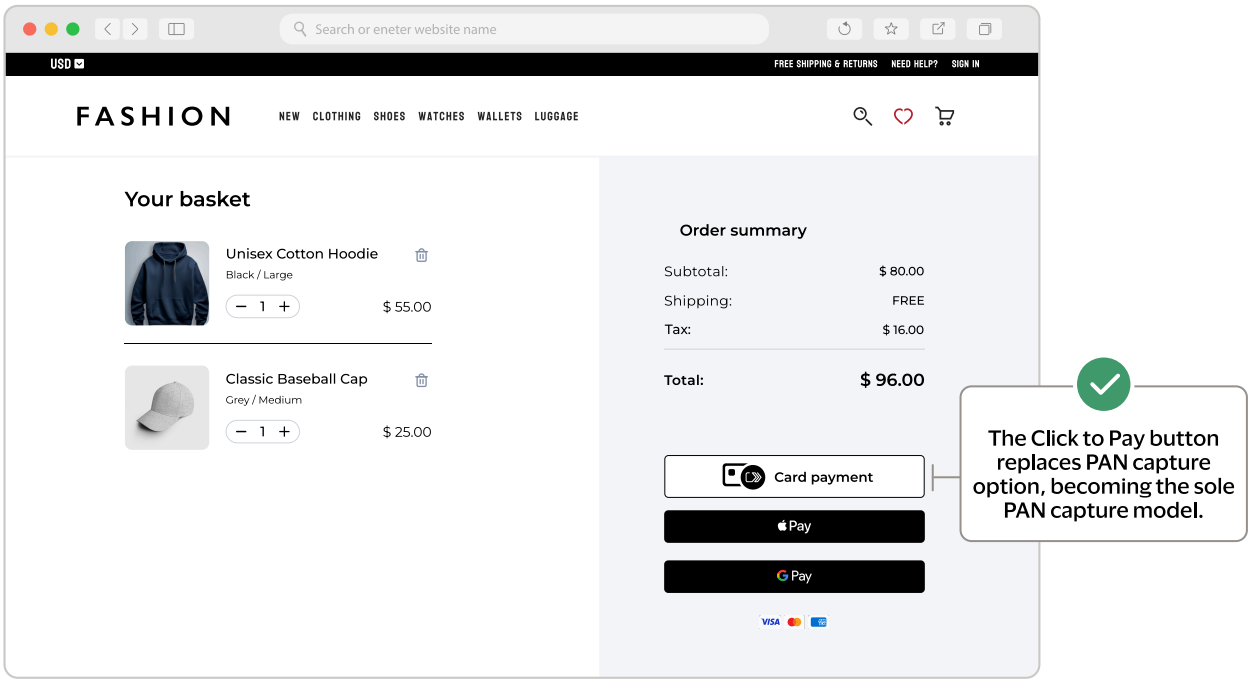
Click to Pay UI Examples

This section contains UI examples of how you should display Click to Pay on your payment page. For information about how to display the UI, see JavaScript API Reference (on page 341).

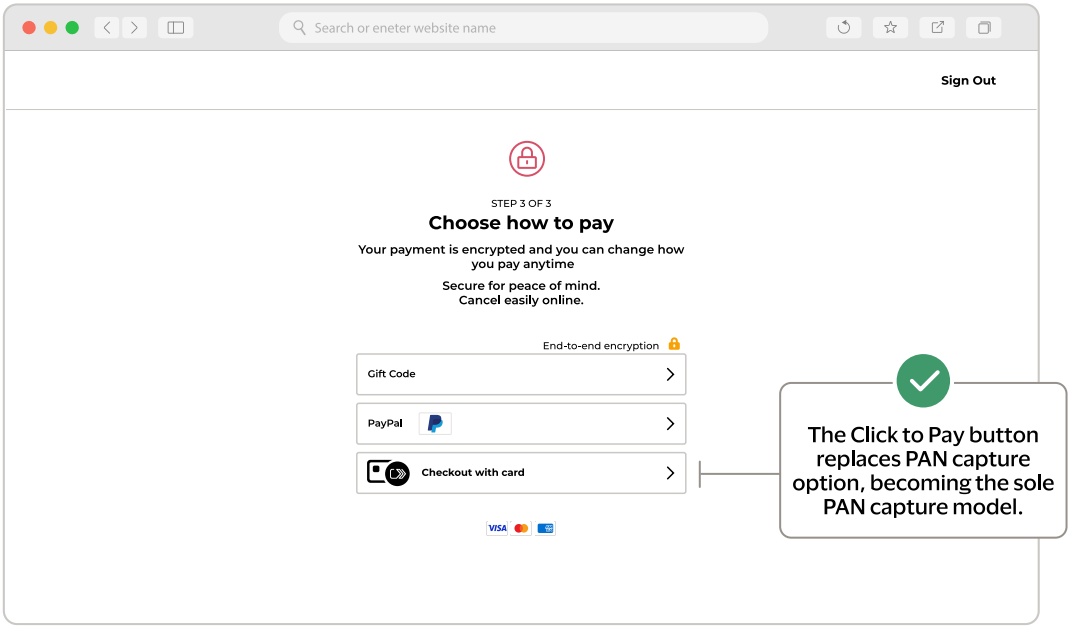
Click to Pay Replaces PAN Capture

Click to Pay is the card entry payment option within your payment page.

Click to Pay Replaces PAN Capture UI Example 1



Click to Pay Replaces PAN Capture UI Example 2

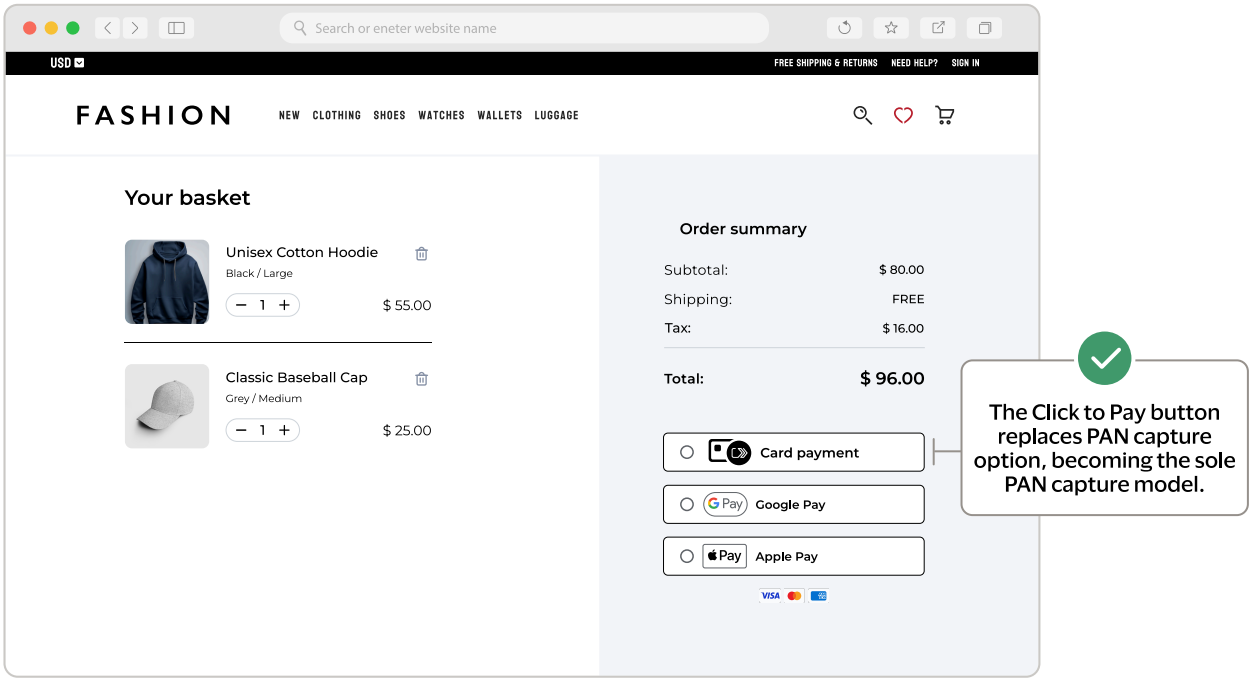


For information about how to configure this UI, see Loading the JavaScript Library (on page 238).

Click to Pay as Radio Button

Click to Pay is a radio button for the card entry payment option within your payment page. When the cardholder selects this option, the Click to Pay payment flow is loaded.

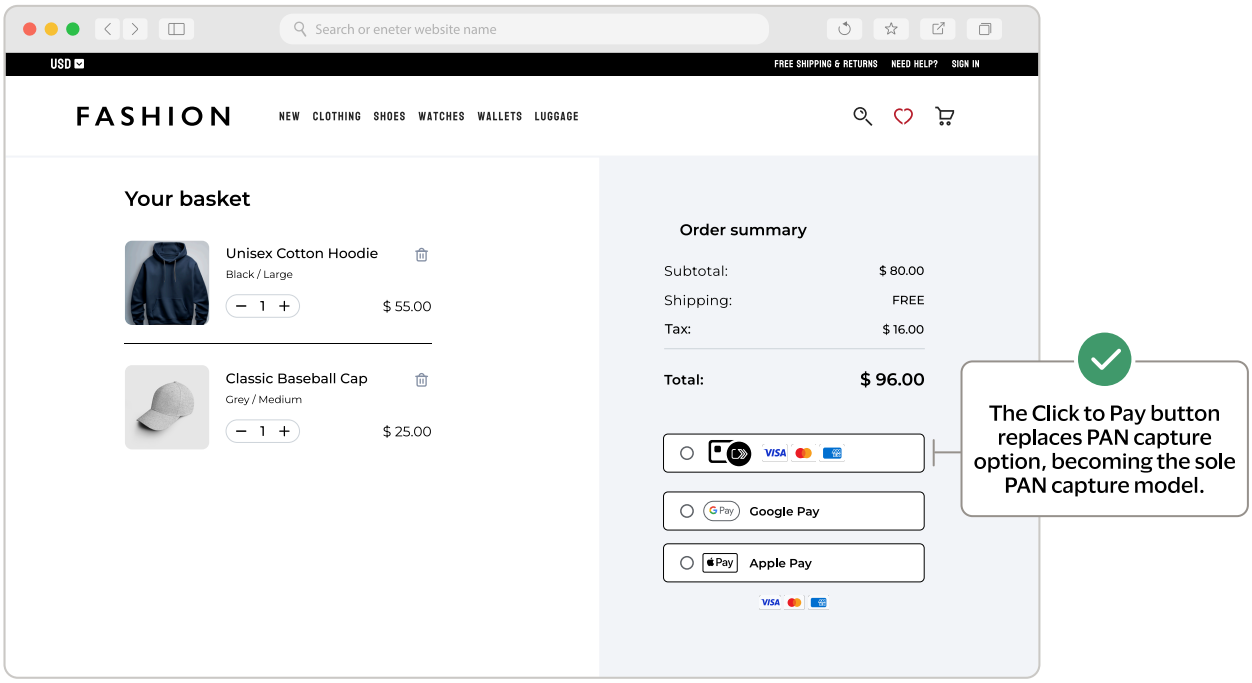
Click to Pay Radio Button Example UI



Click to Pay Icon on Radio Button

You can host the radio selection option for card payment with the Click to Pay icon displayed on the payment label. The Unified Checkout flow loads when the cardholder selects this option. For information about customizing how to trigger Unified Checkout, see JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry (on page 410).

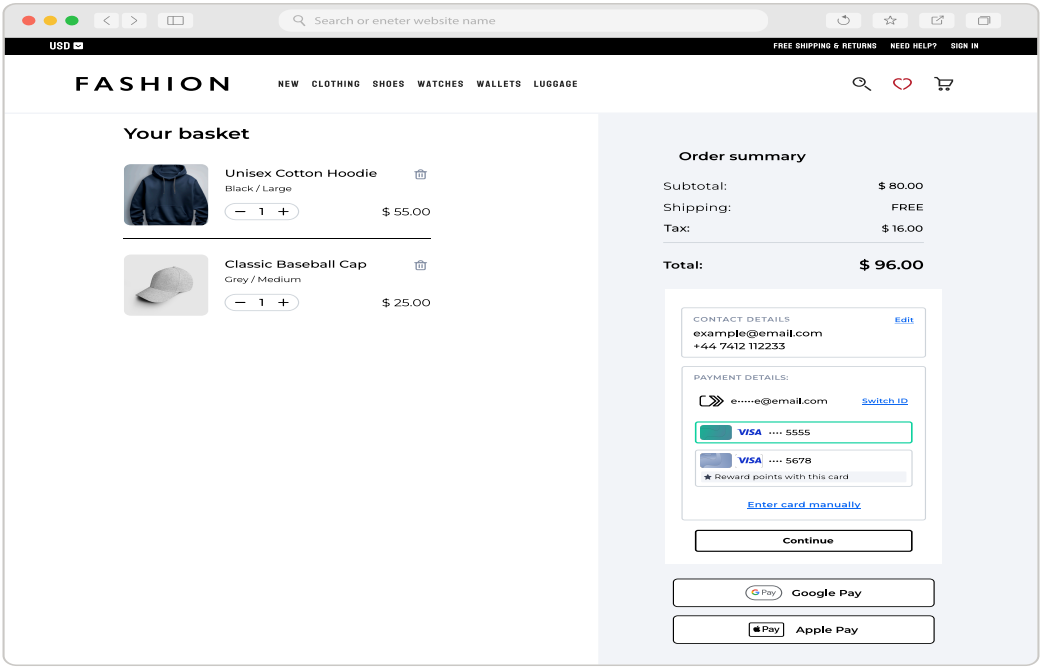
Click to Pay Icon on Radio Button Example UI



Load Click to Pay Automatically From Trigger

You can load the Unified Checkout JavaScript flow within your own payment button without requiring the cardholder to select a card payment option. This example shows a recognized user payment flow where the cardholder's information is shown automatically next to the other payment methods hosted within your payment page. For information about customizing how to trigger Unified Checkout, see JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry (on page 410).

Click to Pay Loaded Automatically From Trigger UI

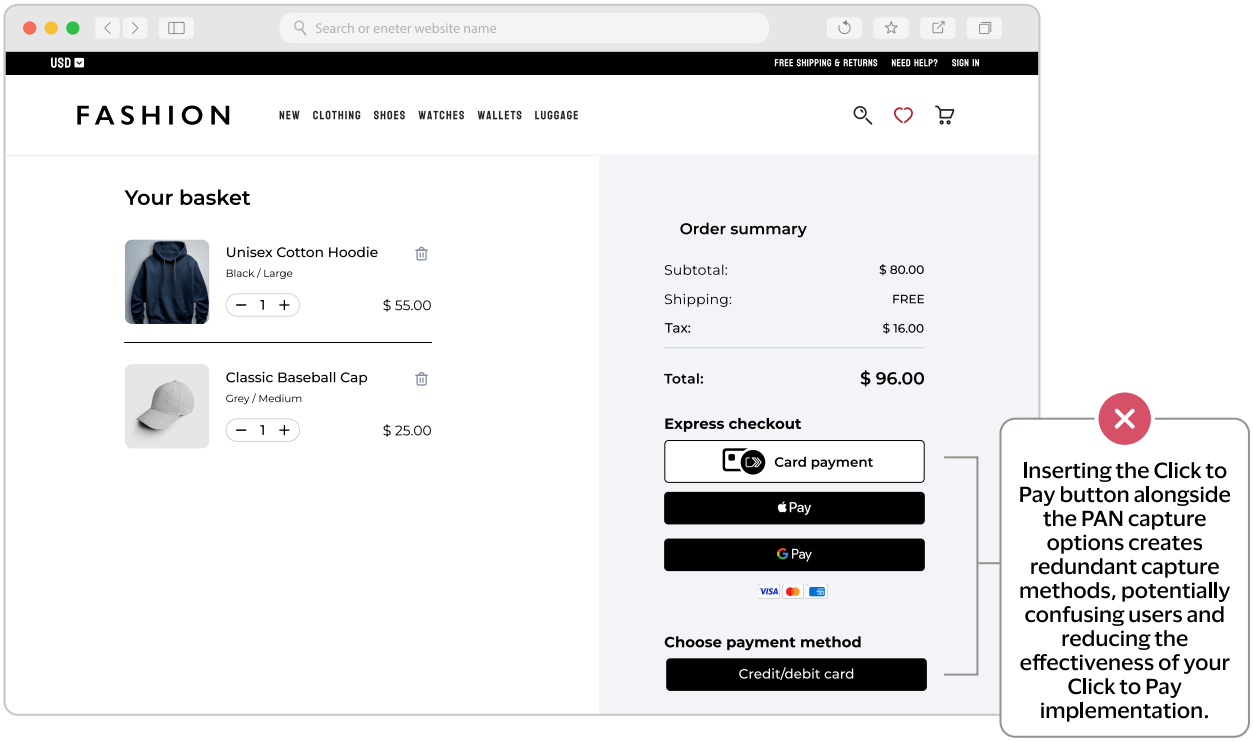


Card Payment Options with Click to Pay in UI

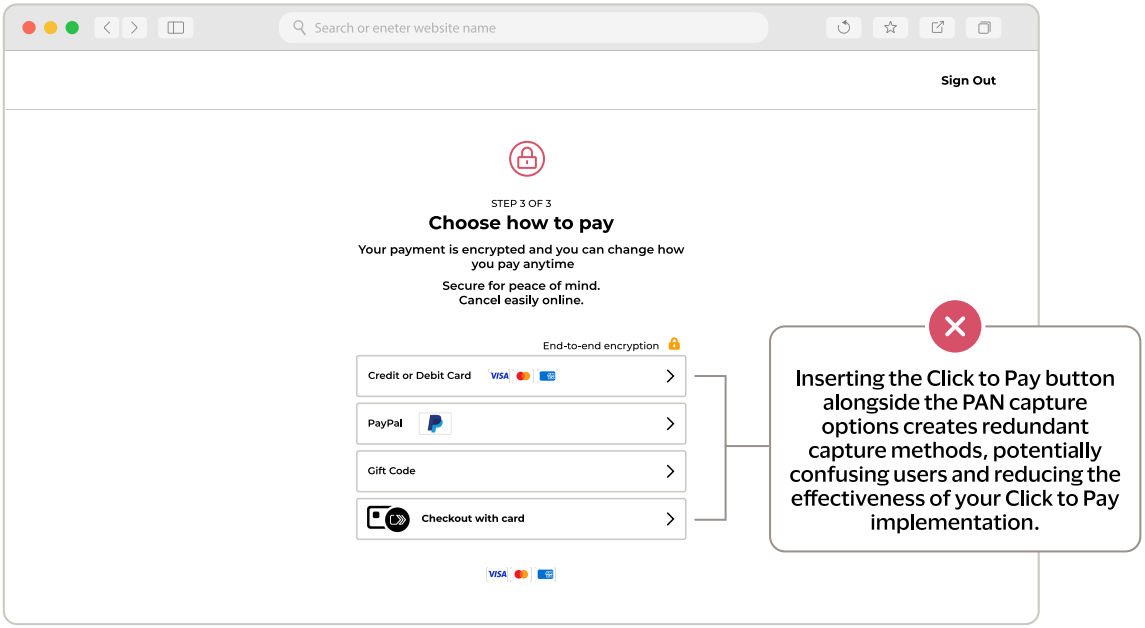
Do not present the Unified Checkout payment button as a separate payment method from the card payment button. If you do this, the cardholder is not prompted with their Click to Pay cards and must manually enter their payment details. They will also not have the option to store their card within Click to Pay for future use.

These examples show multiple card payment options and Click to Pay in a UI:

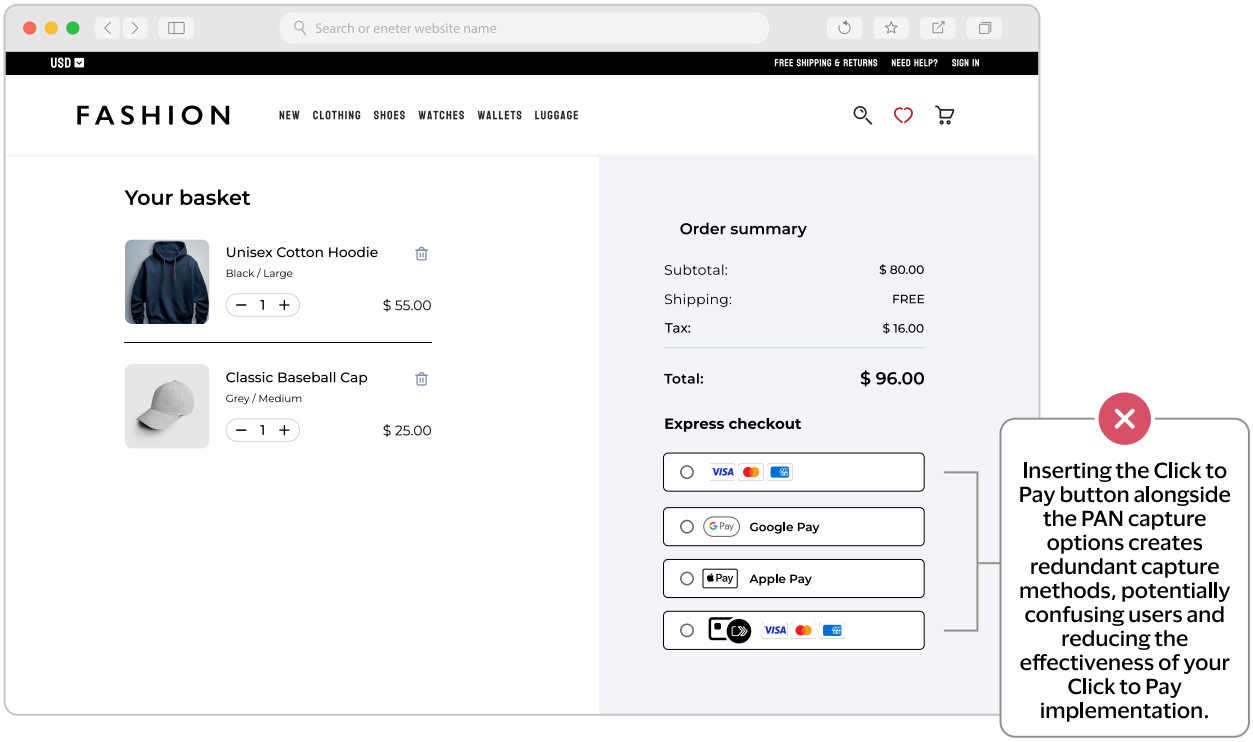
Multiple Card Payment Options in UI Example 1



Multiple Card Payment Options in UI Example 2



Multiple Card Payment Options in UI Example 3



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Paze

Paze is an online checkout option, or *digital wallet*, that enables you to offer customers a fast and secure way to make purchases online. This section includes information about accepting Paze payments with your Unified Checkout integration.

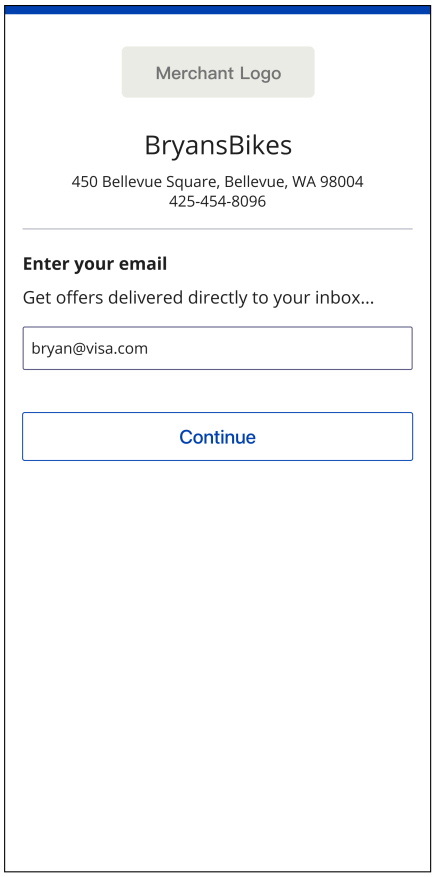
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

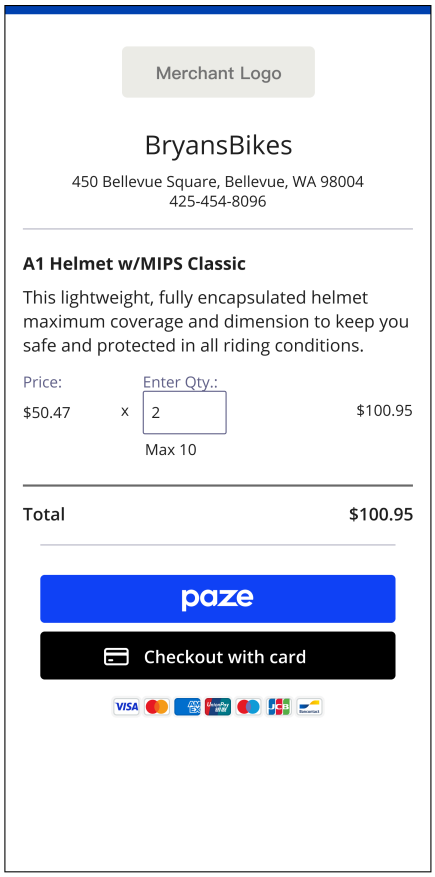
Paze UI

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

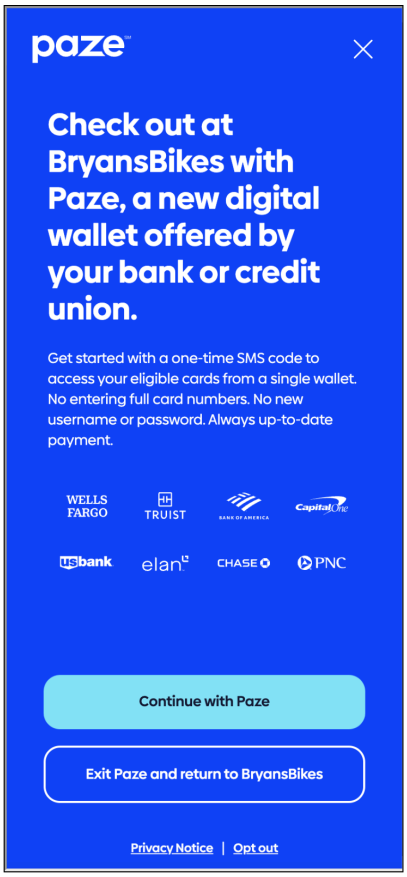
Paze UI



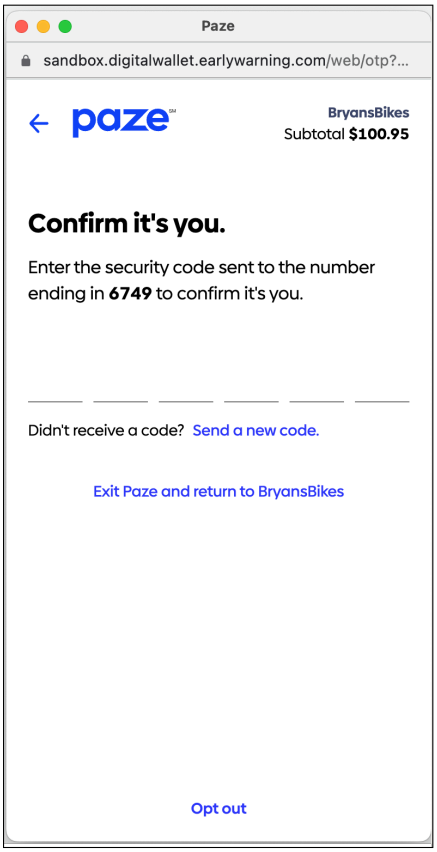
Collecting an email for Paze lookup



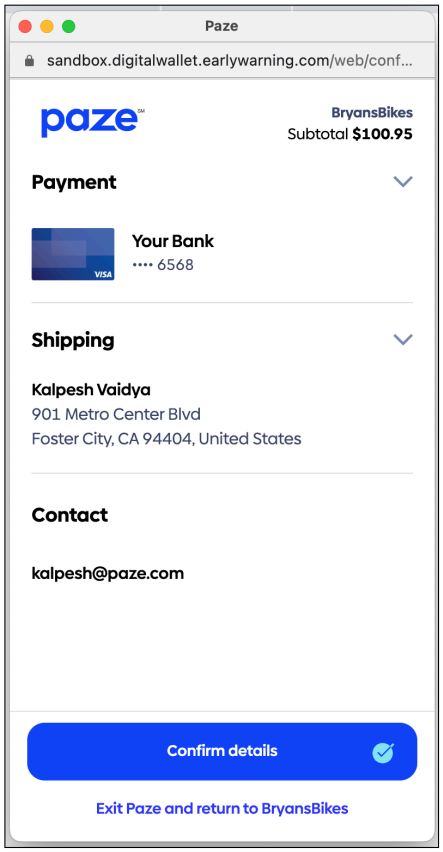
Paze in the payment method list



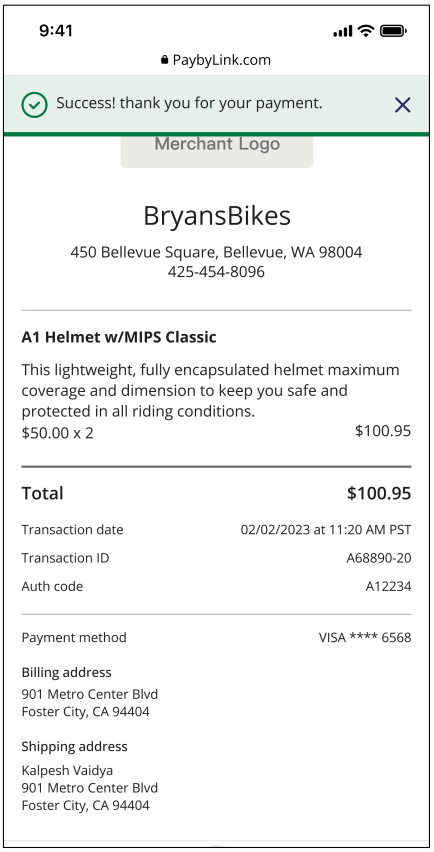
Paze explainer



Paze one-time password



Paze payment method selection



Paze payment confirmation

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Alternative Payment Methods

This section describes the alternative payment methods you can use in your Unified Checkout integration. After you successfully integrate one payment method, you can add another from the same category with minimal adjustments to your existing configuration.

- Online Bank Transfers (on page 190)
- Buy Now, Pay Later (on page 205)
- Post-Pay Reference Payments (on page 211)
- Alternative Payment Wallets (on page 216)

Related information

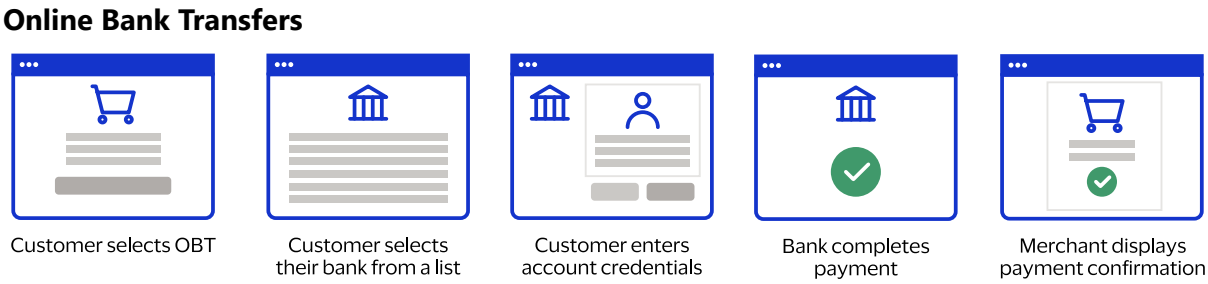
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Online Bank Transfers

Online bank transfers enable customers to complete their purchase by securely logging into their online banking environment. This method is secure, trusted, and widely used in many European countries.

Important: Before you can enroll in these alternative payment method on Unified Checkout, you must first be enabled for the alternative payment platform. Contact your portfolio administrator for more information.

This is how online bank transfers work:



1. The customer chooses online bank transfer as their payment method during checkout.
2. The customer chooses their bank from the list of available banks and is redirected to their bank's website or application where they are prompted to enter their account credentials.
3. The customer confirms their payment and completes the authorization process.
4. The customer is notified that the payment is complete.
5. The customer returns to your website for payment confirmation.

Unified Checkout supports these online bank transfer payment methods:

- Bancontact
- DragonPay
- iDeal
- Multibanco
- MyBank
- Przelewy24|P24
- Tink Pay By Bank

Online Bank Transfer Payment Methods

Payment Method	Capture Context allowedPaymentTypes	Capture Context completeMandate.type	Separate Capture?	Payment Confirmation	Customer Country (Country Code)	Customer ISO Currency Code
iDEAL	`IDEAL`	`CAPTURE` or `PREFER_AUTH`	No	Immediate	Netherlands (NL)	EUR
Multibanco	`MULTIBANCO`	`CAPTURE` or `PREFER_AUTH`	No	Immediate	Portugal (PT)	EUR
Przelewy24	`PRZELEWY24`	`CAPTURE` or `PREFER_AUTH`	No	Immediate	Poland (PL)	PLN
Bancontact	`BANCONTACT`	`CAPTURE` or `PREFER_AUTH`	No	Immediate	Belgium (BE)	EUR
MyBank	`MYBANK`	`CAPTURE` or `PREFER_AUTH`	No	Immediate	Italy (IT)	EUR
					Belgium (BE)	
					Portugal (PT)	
					Spain (ES)	
DragonPay	`DRAGONPAY`	`CAPTURE` or `PREFER_AUTH`	No	Immediate	Philippines (PH)	PHP
Tink Pay by Bank	`TINKPAYBYBANK`	`CAPTURE` or `PREFER_AUTH`	No	Immediate	France (FR)	EUR
					Germany (DE)	EUR
					Ireland (IE)	EUR
					Netherlands (NL)	EUR
					Spain (ES)	EUR
					United Kingdom (GB)	GBP

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Bancontact

Bancontact enables customers to make secure online and in-store purchases directly from their bank accounts. Bancontact is a leading payment method in Belgium.

When the total amount of the order is outside the range of accepted transaction amounts, the Bancontact payment button is not displayed in Unified Checkout. These are the accepted transaction amounts:

- **Minimum transaction amount**: EUR 0.01
- **Maximum transaction amount**: Not applicable

Opt in to Bancontact on Unified Checkout

Follow these steps to opt in to Bancontact on Unified Checkout:

1. Add Bancontact to your integration by adding `BANCONTACT` to the **allowedPaymentTypes** field object within the capture context request.
2. Set the **completeMandate.type** field value to `CAPTURE` or `PREFER_AUTH`.

   You can capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   If you support more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. The funds will be captured immediately for the online bank transfer.
3. Include these required fields for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.country**
   - **data.orderInformation.billTo.firstName**
   - **data.orderInformation.billTo.lastName**
4. Include this optional field for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.address1**
   - **data.orderInformation.billTo.email**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

DragonPay

DragonPay provides Filipino customers and businesses with a secure payment channel that does not require customers to be banked or have a credit card. Customers can make purchases online and pay by bank transfer.

When the total amount of the order is outside the range of accepted transaction amounts, the DragonPay payment button is not displayed in Unified Checkout. These are the accepted transaction amounts:

- **Minimum transaction amount**: PHP 50.01
- **Maximum transaction amount**: Not applicable

Opt in to DragonPay on Unified Checkout

Follow these steps to opt in to Multibanco on Unified Checkout:

1. Add DragonPay to your integration by adding `DRAGONPAY` to the **allowedPaymentTypes** field object within the capture context request.
2. Set the **completeMandate.type** field value to `CAPTURE` or `PREFER_AUTH`.

   You can capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. The funds are captured immediately for the online bank transfer.
3. Include these required fields for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.country**
   - **data.orderInformation.billTo.firstName**
   - **data.orderInformation.billTo.lastName**

4. Include this optional field for online bank transfers in the capture context request:
    - **data.orderInformation.billTo.address1**
    - **data.orderInformation.billTo.email**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Tink Pay By Bank

Tink is an alternative payment method that uses the *pay by bank* payment method. Tink Pay By Bank enables customers to make payments directly from their bank account to the seller's account and bypasses traditional payment methods such as credit cards.

When the total amount of the order is outside the range of accepted transaction amounts, Tink Pay By Bank is not displayed in Unified Checkout.

These are the accepted transaction amounts for the United Kingdom (GB):

- **Minimum transaction amount**: Not applicable
- **Maximum transaction amount**: GBP 8,500

These are the accepted transaction amounts for the European Union (EU):

- **Minimum transaction amount**: Not applicable
- **Maximum transaction amount**: EUR 10,000

Opt in to Tink Pay By Bank on Unified Checkout

You can enable Tink Pay By Bank from the Unified Checkout merchant experience section in the Business Center. For information about how to enable Tink Pay By Bank using the Business Center, see Configure Payment Options (on page 253).

You can also enable Tink Pay By Bank using the Unified Checkout `v1/sessions` API. Follow these steps to opt in to the Tink Pay By Bank payment method using the API:

1. Add Tink Pay by Bank to your integration by adding `TINKPAYBYBANK` to the **allowedPaymentTypes** field object within the capture context request.

2. To use Tink Pay By Bank in your selected country, you must set the **country** field value to the correct value in the capture context request:

   Tink Pay By Bank country Field Values

Country	country Field Value
France	`FR`
Germany	`DE`
Ireland	`IE`
Netherlands	`NL`
Spain	`ES`
United Kingdom	`GB`

3. Set the **completeMandate.type** field value to `CAPTURE` or `PREFER_AUTH`.

   You can capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. The funds are captured immediately for the online bank transfer.

4. To use Tink Pay By Bank in your selected country, you must set the **data.orderInformation.billTo.country** field value to the correct value in the capture context request:

   Tink Pay By Bank data.orderInformation.billTo.country Field Values

Country	data.orderInformation.billTo.country Field Value
France	`FR`
Germany	`DE`
Ireland	`IE`
Netherlands	`NL`
Spain	`ES`
United Kingdom	`GB`

5. To use Tink Pay By Bank in your selected country, you must set the **data.orderInformation.amountDetails.currency** field value to the correct value in the capture context request:

Tink Pay By Bank data.orderInformation.amountDetails.currency Field Values

Country	data.orderInformation.amount Details.currency Field Value
France	`EUR`
Germany	`EUR`
Ireland	`EUR`
Netherlands	`EUR`
Spain	`EUR`
United Kingdom	`GBP`

6. Include this optional field for online bank transfers in the capture context request:
    - **data.orderInformation.billTo.firstName**
    - **data.orderInformation.billTo.lastName**
    - **data.orderInformation.shipTo.address1**
    - **data.orderInformation.shipTo.address2**
    - **data.orderInformation.shipTo.country**
    - **data.orderInformation.shipTo.district**
    - **data.orderInformation.shipTo.firstName**
    - **data.orderInformation.shipTo.lastName**
    - **data.orderInformation.shipTo.locailty**
    - **data.orderInformation.shipTo.postalCode**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

iDeal

iDEAL enables customers to pay online through their mobile banking app or online bank account and provides you with a payment guarantee. iDEAL supports these banks:

- ABN AMRO
- ASN Bank
- bunq
- ING
- Knab
- Rabobank
- RegioBank
- Revolut
- SNS
- Svenska Handelsbanken
- Triodos Bank
- Van Lanschot

When the total amount of the order is outside the range of accepted transaction amounts, the iDeal payment button is not displayed in Unified Checkout. These are the accepted transaction amounts:

- **Minimum transaction amount**: EUR 0.01
- **Maximum transaction amount**: Subject to transaction approval from the customer's account.

Opt in to iDeal on Unified Checkout

Follow these steps to opt in to iDeal on Unified Checkout:

1. Add iDeal to your integration by adding `IDEAL` to the **allowedPaymentTypes** field object within the capture context request.
2. Set the **completeMandate.type** field value to `CAPTURE` or `PREFER_AUTH`.

   You can capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. The funds are captured immediately for the online bank transfer.

3. Include these required fields for online bank transfers in the capture context request:
    - **data.orderInformation.billTo.country**
    - **data.orderInformation.billTo.firstName**
    - **data.orderInformation.billTo.lastName**
4. Include this optional field for online bank transfers in the capture context request:
    - **data.orderInformation.billTo.address1**
    - **data.orderInformation.billTo.email**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

MyBank

MyBank enables customers to pay for their online purchases in an easy and safe way using real-time bank transfers. MyBank customers complete payments by selecting their bank and logging in with their online banking credentials.

When the total amount of the order is outside the range of accepted transaction amounts, the MyBank payment button is not displayed in Unified Checkout. These are the accepted transaction amounts:

- **Minimum transaction amount**: EUR 0.01
- **Maximum transaction amount**: EUR 999,999,999.99

Opt in to MyBank on Unified Checkout

Follow these steps to opt in to MyBank on Unified Checkout:

1. Add MyBank to your integration by adding `MYBANK` to the **allowedPaymentTypes** field object within the capture context request.
2. Set the **completeMandate.type** field value to `CAPTURE` or `PREFER_AUTH`.

   You can capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. The funds are captured immediately for the online bank transfer.
3. Include these required fields for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.country**
   - **data.orderInformation.billTo.firstName**
   - **data.orderInformation.billTo.lastName**
4. Include this optional field for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.address1**
   - **data.orderInformation.billTo.email**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Multibanco

Multibanco enables customers to pay for a range of goods and services by bank transfer. These services include e-commerce, licenses, and taxes post-purchase. Multibanco is supported by all banks in Portugal.

When the total amount of the order is outside the range of accepted transaction amounts, the Multibanco payment button is not displayed in Unified Checkout. These are the accepted transaction amounts:

- **Minimum transaction amount**: No minimum
- **Maximum transaction amount**: EUR 99,999

Opt in to Multibanco on Unified Checkout

Follow these steps to opt in to Multibanco on Unified Checkout:

1. Add Multibanco to your integration by adding `MULTIBANCO` to the **allowedPaymentTypes** field object within the capture context request.
2. Set the **completeMandate.type** field value to `CAPTURE` or `PREFER_AUTH`.

   You can capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. The funds are captured immediately for the online bank transfer.
3. Include these required fields for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.country**
   - **data.orderInformation.billTo.firstName**
   - **data.orderInformation.billTo.lastName**
4. Include this optional field for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.address1**
   - **data.orderInformation.billTo.email**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Przelewy24|P24

Przelewy24, or P24, is a Poland-based real-time online bank transfer payment method. P24 is one of the most popular payment methods in Poland covering all major consumer banks.

When the total amount of the order is outside the range of accepted transaction amounts, the P24 payment button is not displayed in Unified Checkout. These are the accepted transaction amounts:

- **Minimum transaction amount**: PLN 0.01, EUR 0.01
- **Maximum transaction amount**: PLN 55,000.00, EUR 12,500.00

Opt in to Przelewy24|P24 on Unified Checkout

Follow these steps to opt in to Przelewy24|P24 on Unified Checkout:

1. Add Przelewy24|P24 to your integration by adding `PRZELEWY24` to the **allowedPaymentTypes** field object within the capture context request.
2. Set the **completeMandate.type** field value to `CAPTURE` or `PREFER_AUTH`.

   You can capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   If you support more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. The funds will be captured immediately for the online bank transfer.
3. Include these required fields for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.country**
   - **data.orderInformation.billTo.email**
   - **data.orderInformation.billTo.firstName**
   - **data.orderInformation.billTo.lastName**
4. Include this optional field for online bank transfers in the capture context request:
   - **data.orderInformation.billTo.address1**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Verify Status for Online Bank Transfers

When the status of your payment request is `PENDING`, you can verify the status by using the URL method and the payload that is included in the **transactionStatus.url** field in the webhook response:

```
{
  "payload": {
    "transactionResult": {
      "id": "755775333723635790 4806",
      "rootId": "755775333723635790 4806",
      "reconciliationId": "XFZ40EJPGL5K",
      "submitTimeUTC": "2025-08-21T11:22:13Z",
      "merchantId": "uc_apm_tester004"
    },
    "transactionStatus": {
      "url":
 "https://apitest.cybersource.com/tss/v2/transactions/755775333723635790 4806",
      "method": "GET"
    }
  }
}
```

You can also send a request to this endpoint to verify the status:

Production: `GET https://api.cybersource.com/tss/v2/transactions/{id}`

Test: `GET https://apitest.cybersource.com/tss/v2/transactions/{id}`

The *{id}* is the ID that is returned in the webhook response. For more information, see Webhooks Support (on page 278).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Verify Status for Online Bank Transfers (Tink Pay By Bank)

When the status of your payment request is `PENDING`, you can verify the status by using the URL method and the payload that is included in the **transactionStatus.url** field in the webhook response:

```
"payload": {
    "transactionResult": {
      "submitTimeUtc": "2025-07-22T08:16:24Z",
      "reconciliationId": "KPUJHD4X2G31",
      "processorInformation": {
```

```
        "responseCode": "00004"
      },
      "id": "753117391851606420480 7",
      "message": "Request was processed successfully.",
      "status": "SETTLED"
    },
    "transactionStatus": {
      "url":
 "https://api.cybersource.com/pts/v2/refresh-payment-status/753117391851606420480
7",
      "method": "POST",
      "payload": {
        "clientReferenceInformation": {
          "applicationName": "unifiedCheckout"
        },
        "processingInformation": "processingInformation",
        "paymentInformation": {
          "paymentType": {
            "method": {
              "name": "tinkPayByBank"
            },
            "name": "bankTransfer"
          }
        }
      }
    }
    }
```


You can also send a request to this endpoint to verify the status:

Production: `POST https://api.cybersource.com/pts/v2/refresh-payment-status/{id}`

Test: `POST https://apitest.cybersource.com/pts/v2/refresh-payment-status/{id}`

The *{id}* is the ID that is returned in the webhook response. For more information, see Webhooks Support (on page 278).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Handle Responses

When Unified Checkout automatically processes a payment with `autoProcessing` is set to `true` or you have set `autoProcessing` to `false` and are using **checkout.Complete()**, you must handle both successful responses and various errors. After the payment is complete, the **completeResponse** field object contains information about the transaction outcome.

When a payment is processed successfully, you must parse the response to confirm the payment status, update their order records, and trigger any post-payment workflows. Post-payment workflows include sending confirmation emails or updating inventory. See JavaScript Example: Processing a Payment (on page 241).

Your error handling should account for specific cases such as `COMPLETE_TRANSACTION_CANCELED` and `COMPLETE_TRANSACTION_FAILED`. `COMPLETE_TRANSACTION_CANCELED` occurs when the user cancels the transaction and `COMPLETE_TRANSACTION_FAILED` indicates that the consumer's transaction failed.

For PPRO-enabled online bank transfers, only cancellation errors are returned, and Tink Pay By Bank returns failure and cancellation errors. For information about possible errors that can occur when calling the complete API see UnifiedCheckoutError (on page 329) in Handle Errors (on page 329).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Buy Now, Pay Later

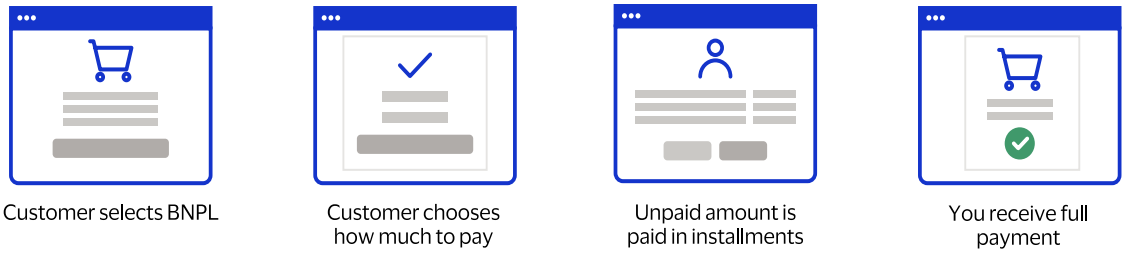
Buy Now, Pay Later payment methods enable customers to purchase goods or services immediately and pay in installments over time. With Buy Now, Pay Later, you are paid immediately and in full, while your customers pay nothing or only a portion of the total at the time of purchase. The remaining balance is typically spread over equal, often interest-free, payments.

Buy Now, Pay Later is increasingly popular for both online and in-store purchases.

Important: Before you can enroll in these alternative payment method on Unified Checkout, you must first be enabled for the alternative payment platform. Contact your portfolio administrator for more information.

This is how Buy Now, Pay Later works:

Buy Now, Pay Later



1. The customer chooses their Buy Now, Pay Later payment method during checkout.
2. The customer chooses how much they want to pay, such as nothing, installments, or the total amount.
3. The unpaid amount is divided into equal installments that are paid over a fixed amount of time.
4. You receive the full payment after the customer completes checkout, and the Buy Now, Pay Later provider collects the installment payments from your customer.

Unified Checkout supports the Afterpay/Clearpay and Paypal Buy Now, Pay Later payment methods.

Buy Now, Pay Later Payment Method Support

Payment Method	Capture Context allowedPaymentTypes	Capture Context completeMandate.type	Separate Capture?	Payment Confirmation	Customer Country (Country Code)	Customer ISO Currency Code
Afterpay	`AFTERPAY`	`CAPTURE`	No	Immediate	Canada (CA)	CAD
		`AUTH` or `PREFER_AUTH`	Yes	Delayed		

Buy Now, Pay Later Payment Method Support (continued)

Payment Method	Capture Context allowedPaymentTypes	Capture Context completeMandate.type	Separate Capture?	Payment Confirmation	Customer Country (Country Code)	Customer ISO Currency Code
		`CAPTURE`	No	Immediate	Australia (AU)	AUD
		`AUTH` or `PREFER_AUTH`	Yes	Delayed		
		`CAPTURE`	No	Immediate	New Zealand (NZ)	NZD
		`AUTH` or `PREFER_AUTH`	Yes	Delayed		
Cash App Afterpay		`CAPTURE`	No	Immediate	United States (US)	USD
		`AUTH` or `PREFER_AUTH`	Yes	Delayed		
Clearpay		`CAPTURE`	No	Immediate	Great Britain (GB)	GBP
		`AUTH` or `PREFER_AUTH`	Yes	Delayed		

For information on ISO country codes, see ISO Standard Country Codes.

For information on ISO currency codes, see ISO Standard Currency Codes.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Afterpay

Afterpay is a Buy Now, Pay Later service that allows customers to purchase items immediately and pay for them in four interest-free installments over a period of 6 weeks. Afterpay is also known as Clearpay in the UK, and Cash App Afterpay in the US. For more information, see the *Afterpay and Clearpay Developer Guide*.

When the total amount of the order is outside the range of accepted transaction amounts, the Afterpay/Clearpay payment button is not displayed in Unified Checkout. These are the accepted transaction amounts:

- **Minimum transaction amount**: 1 (CAD, AUD, NZD, USD, and GBP)
- **Maximum transaction amount**: Not applicable

Opt in to Afterpay on Unified Checkout

Follow these steps to opt in to the Afterpay/Clearpay payment method in Unified Checkout:

1. Add Afterpay to your integration by adding `AFTERPAY` to the **allowedPaymentTypes** field within the capture context request. The default field value is `AFTERPAY` even if you want to support Cash App Afterpay in the US or Clear Pay in the UK.
2. Set the **completeMandate.type** field value to `AUTH`, `CAPTURE` or `PREFER_AUTH`.

   You can perform a sale and capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   You can capture the funds later if you include the **completeMandate.type** field in the capture context request and set the value to `AUTH`. When you capture the funds later, you must perform a capture using the payments API. See Captures (on page 210).

   If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. You must perform a capture using the payments API when an authorization is performed. A capture is performed automatically if an authorization is not allowed by the payment type.
3. Include these required fields in the capture context request:
   - **data.orderInformation.billTo.email**
   - **data.orderInformation.billTo.firstName**
   - **data.orderInformation.billTo.lastName**
   - **data.orderInformation.billTo.address1**
   - **data.orderInformation.billTo.locality**
   - **data.orderInformation.billTo.postalCode**
   - **data.orderInformation.billTo.administrativeArea**
   - **data.orderInformation.billTo.country**
4. Include these optional fields in the capture context request:

   Important: These fields are required when the **requestShipping** field is set to `true`.

- **data.orderInformation.shipTo.firstName**
- **data.orderInformation.shipTo.lastName**
- **data.orderInformation.shipTo.address1**
- **data.orderInformation.shipTo.locality**
- **data.orderInformation.shipTo.postalCode**
- **data.orderInformation.shipTo.administrativeArea**
- **data.orderInformation.shipTo.country**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Verify Status for Afterpay

When the status of your payment request is `PENDING`, you can verify the status by sending a POST request to the URL that is included in the **transactionStatus.url** field in the webhook response:


```
{
  "payload": {
    "transactionResult": {
      "submitTimeUtc": "2025-07-22T08:16:24Z",
      "reconciliationId": "KPUJHD4X2G31",
      "processorInformation": {
        "responseCode": "00004"
      },
      "id": "7531173918516064204807",
      "message": "Request was processed successfully.",
      "status": "SETTLED"
    },
    "transactionStatus": {
      "url":
 "https://apitest.cybersource.com/pts/v2/refresh-payment-status/753117391851606420
4807",
      "method": "POST",
      "payload": {
```

```
        "clientReferenceInformation": {
          "applicationName": "unifiedCheckout"
        },
        "processingInformation": {
          "actionList": [
            "AP_STATUS"
          ]
        },
        "paymentInformation": {
          "paymentType": {
            "method": {
              "name": "AFTERPAY"
            },
            "name": "INVOICE"
          }
        }
      }
    }
  }
}
```

You can also send a request to this endpoint to verify the status:

Production: `POST https://api.cybersource.com/pts/v2/refresh-payment-status/{id}`

Test: `POST https://apitest.cybersource.com/pts/v2/refresh-payment-status/{id}`

The *{id}* is ID that is returned in the webhook response. For more information, see Webhooks Support (on page 278).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Handle Responses

When Unified Checkout automatically processes a payment with `autoProcessing` is set to `true` or you have set `autoProcessing` to `false` and are using **checkout.Complete()**, you must handle both successful responses and various errors. After the payment is complete, the **completeResponse** field object contains information about the transaction outcome.

When a payment is processed successfully, you must parse the response to confirm the payment status, update their order records, and trigger any post-payment workflows. Post-payment workflows include sending confirmation emails or updating inventory. See JavaScript Example: Processing a Payment (on page 241).

Your error handling should account for specific cases such as `COMPLETE_TRANSACTION_CANCELED` and `COMPLETE_TRANSACTION_FAILED`. `COMPLETE_TRANSACTION_CANCELED` occurs when the user cancels the transaction and `COMPLETE_TRANSACTION_FAILED` indicates that the consumer's transaction failed.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Captures

When you set the **completeMandate.type** field value to `AUTH` or `PREFER_AUTH`, you must send a request to capture an authorized payment. Full and partial captures are supported.

Endpoint

Production: `POST https://api.cybersource.com/pts/v2/payments/{id}/captures`

Test: `POST https://apitest.cybersource.com/pts/v2/payments/{id}/captures`

The *{id}* is the transaction ID returned in the authorization response.

Example: Authorization Response from Unified Checkout

```
{
  "details": {
    "clientReferenceInformation": {
      "code": "1753351101383"
    },
    "orderInformation": {
      "amountDetails": {
        "currency": "USD",
        "totalAmount": "21.00"
      }
    },
```

```
    "processorInformation": {
      "approvalCode": "AUTH456789",
      "responseCode": "00003",
      "responseDetails": "00003",
      "transactionId": "201601180815391001180815 3AUTH"
    },
    "reconciliationId": "04RYADD29YRO",
    "submitTimeUtc": "2025-07-24T09:58:21Z"
  },
  "id": "7533511014286971803092",
  "message": "Request processed successfully.",
  "outcome": "AUTHORIZED",
  "status": "AUTHORIZED"
}
```


Response Status

Cybersource responds to your capture request with one of these statuses:

- `FAILED`: The capture request failed.
- `PENDING`: The capture request is accepted but not captured. Send a request to the check status service to retrieve status updates.
- `SETTLED`: The capture request is settled for the amount requested.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

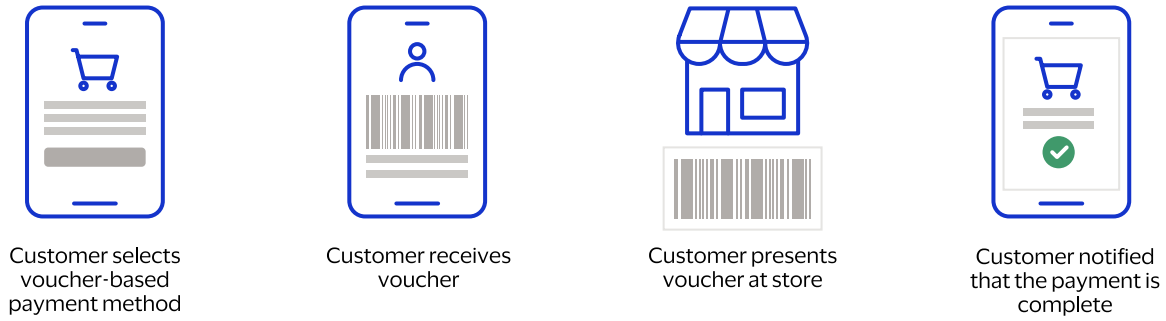
Post-Pay Reference Payments

Post-pay reference payments provide an alternative way for your customers to complete online purchases using cash. During checkout, customers select a voucher payment option and receive a unique code or payment slip. To finalize their purchase, they visit a designated offline location, such as a convenience store or payment kiosk, and pay the required amount in person. Once the payment is made, the merchant receives a notification that they can process the order.

Post-pay reference payments are widely used in countries where cash transactions are common or where many customers don't have access to credit or debit cards.

Important: Before you can enroll in these alternative payment method on Unified Checkout, you must first be enabled for the alternative payment platform. Contact your portfolio administrator for more information.

This is how post-pay reference payments work:



1. The customer selects a convenience store/voucher payment method during checkout.
2. The customer receives their payment code/voucher or QR code.
3. The customer presents the payment code/voucher or QR in-store to complete the payment.
4. The customer receives a notification that the payment is complete.

Unified Checkout supports the Konbini voucher-based payment option:

Post-Pay Reference Payment Support

Payment Method	Capture Context allowedPaymentTypes	Capture Context completeMandate.type	Separate Capture?	Payment Confirmation	Customer Country (Country Code)	Customer ISO Currency Code
Konbini	`KONBINI`	`CAPTURE` or `PREFER_AUTH`	No	Immediate	Japan (JP)	JPY

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Konbini

Konbini is used to make cash payments in Japan. Konbini payments enable your customers to pay for bills and online purchases at convenience stores in Japan.To complete a transaction, your customers receive payment codes for specific convenience stores and a confirmation number. Customers must then bring the information to a convenience store to make a cash payment. Your customers can pay at these convenience stores in Japan:

- 7-Eleven
- Family Mart
- Lawson
- Ministop
- Seicomart

You receive the payment confirmation immediately and the funds are available after 4 business days.

When the total amount of the order is outside the range of accepted transaction amounts, the Konbini payment button is not displayed in Unified Checkout. These are the accepted transaction amounts:

- **Minimum transaction amount**: JPY 1
- **Maximum transaction amount**: Not applicable

Opt in to Konbini on Unified Checkout

Follow these steps to opt in to the Konbini payment method in Unified Checkout:

1. Add Konbini to your integration by adding `KONBINI` to the **allowedPaymentTypes** field within the capture context request.
2. Set the **completeMandate.type** field value to `AUTH`, `CAPTURE` or `PREFER_AUTH`.

   You can perform a sale and capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. You must perform a capture using the payments API when an authorization is performed. A capture is performed automatically if an authorization is not allowed by the payment type.
3. Include these required fields in the capture context request:

- **data.orderInformation.billTo.country**
- **data.orderInformation.billTo.firstName**
- **data.orderInformation.billTo.lastName**
- **data.orderInformation.billTo.phoneNumber**

4. Include these optional fields in the capture context request:

> **Important:** These fields are required when the **requestShipping** field is set to `true`.

- **data.orderInformation.billTo.address1**
- **data.orderInformation.billTo.email**

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Verify Status for Post-Pay Reference

When the status of your payment request is `PENDING`, you can verify the status by sending a POST request to the URL that is included in the **transactionStatus.url** field in the webhook response:

```
{
  "payload": {
    "transactionResult": {
      "id": "755775333723635790 4806",
      "rootId": "755775333723635790 4806",
      "reconciliationId": "XFZ40EJPGL5K",
      "submitTimeUTC": "2025-08-21T11:22:13Z",
      "merchantId": "uc_apm_tester004"
    },
    "transactionStatus": {
      "url":
 "https://apitest.cybersource.com/tss/v2/transactions/755775333723635790 4806",
      "method": "GET"
```

```
    }
  }
}
```

You can also send a request to this endpoint to verify the status:

Production: `GET https://api.cybersource.com/tss/v2/transactions/{id}`

Test: `GET https://apitest.cybersource.com/tss/v2/transactions/{id}`

The `{id}` is the ID that is returned in the webhook response. For more information, see Webhooks Support (on page 278).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Handle Responses

When Unified Checkout automatically processes a payment with `autoProcessing` is set to `true` or you have set `autoProcessing` to `false` and are using **checkout.Complete()**, you must handle both successful responses and various errors. After the payment is complete, the **completeResponse** field object contains information about the transaction outcome.

When a payment is processed successfully, you must parse the response to confirm the payment status, update their order records, and trigger any post-payment workflows. Post-payment workflows include sending confirmation emails or updating inventory. See JavaScript Example: Processing a Payment (on page 241).

Your error handling should account for specific cases such as `COMPLETE_TRANSACTION_CANCELED` which is returned when the user cancels the transaction.

For PPRO-enabled online bank transfers, only cancellation errors are returned. For information about possible errors that can occur when calling the complete API see UnifiedCheckoutError (on page 329) in Handle Errors (on page 329).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Alternative Payment Wallets

Digital wallets are secure applications or services that enable users to store payment details, such as debit or credit cards electronically.

For online transactions, the wallet securely passes a payment token to the merchant or payment processor. This means that you do not handle sensitive customer data directly. This reduces friction at checkout, improves conversion rates, and enhances security through built-in authentication and tokenization.

Wallets are best suited for one-time online purchases and express checkout experiences. Support for subscriptions and recurring payments varies by wallet, so you must ensure compatibility if future charges or merchant-initiated transactions are required.

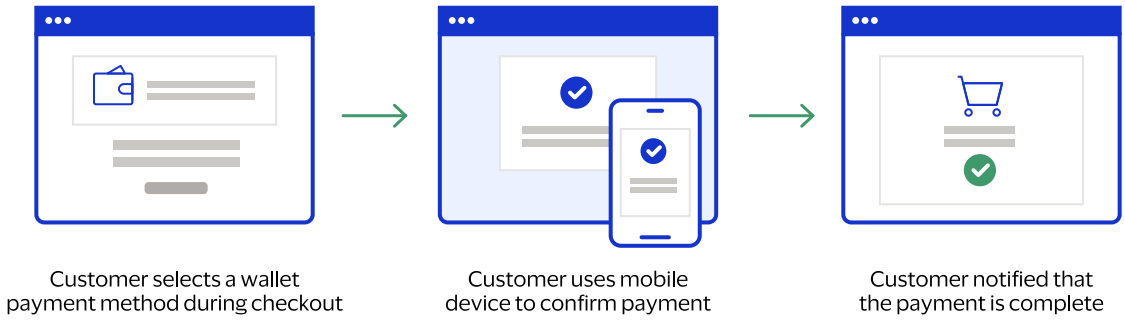
Unified Checkout supports these wallet-based payment options:

- PayPal
- Venmo

This is how payments with digital wallets work:

Customer-facing mobile flow

Customer-facing web flow



Digital Wallet Payment Support

Payment Method	Capture Context allowedPaymentTypes	Capture Context completeMandate.type	Separate Capture?	Payment Confirmation	Customer Country (Country Code)	Customer ISO Currency Code
PayPal	`PAYPAL`	`AUTH` `PREFER_AUTH`	Yes	Delayed	Global	AUD, CAD, CHF, CZK, DKK, EUR, GBP, HKD, NOK, NZD, PLN, SEK, SGD, and USD
PayPal		`CAPTURE`	No	Immediate		
Venmo	`VENMO`	`AUTH` `PREFER_AUTH`	Yes	Delayed	United States (US)	USD
Venmo		`CAPTURE`	No	Immediate		

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

PayPal

PayPal is a secure and convenient payment service that your customers can use to make payments without directly using their bank accounts or credit cards. PayPal gives yours customers the option to pay in installments over time with PayPal Pay Later while also giving you the full payment immediately. These installments are available:

- **Pay in 3**: Pay in three installments. Available in the UK.
- **Pay in 4**: Pay in four installments. Available in the US.
- **Pay Monthly**: Pay in monthly recurring installments.

Opt in to Paypal on Unified Checkout

You can enable Paypal from the Unified Checkout merchant experience section in the Business Center. For information about how to enable Paypal using the Business Center, see Configure Payment Options (on page 253).

You can also enable Paypal using the Unified Checkout `v1/sessions` API. Follow these steps to opt in to the Paypal payment method using the API:

1. Add Paypal to your integration by adding `PAYPAL` to the **allowedPaymentTypes** field within the capture context request.
2. Set the **completeMandate.type** field value to `AUTH`, `CAPTURE` or `PREFER_AUTH`.

   You can perform a sale and capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   You can capture the funds later if you include the **completeMandate.type** field in the capture context request and set the value to `AUTH`. When you capture the funds later, you must perform a capture using the payments API. See Captures (on page 228).

   If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. You must perform a capture using the payments API when an authorization is performed. A capture is performed automatically if an authorization is not allowed by the payment type.
3. Include these required fields in the capture context request:
   - **orderInformation.amountDetails.currency**
   - **orderInformation.amountDetails.taxAmount**
     - This field is the sum of all **orderInformation.lineItems[].taxAmount** values.
   - **orderInformation.amountDetails.totalAmount**

        - This field is the sum of (**orderInformation.lineItems[].quantity** × **orderInformation.lineItems[].unitPrice**) + **orderInformation.lineItems[].taxAmount** for all line items.

    - **orderInformation.lineItems[].productDescription**
    - **orderInformation.lineItems[].productName**
    - **orderInformation.lineItems[].productSKU**
    - **orderInformation.lineItems[].quantity**
    - **orderInformation.lineItems[].taxAmount**

      Important: This field is required for all line items when you include **orderInformation.lineItems[].taxAmount** for one line item.

        - You can set this field value to `0.00`.

    - **orderInformation.lineItems[].typeOfSupply**
    - **orderInformation.lineItems[].unitPrice**

4. Include these optional fields in the capture context request:

    - **buyerInformation.dateOfBirth**
    - **buyerInformation.language**
    - **buyerInformation.personalIdentification[].id**
    - **buyerInformation.personalIdentification[].type**
    - **clientReferenceInformation.reconciliationId**
    - **merchantInformation.merchantDescriptor.name**

        - This field affects the name that the customers see on their card statement. If the merchant is Candy Shop and passes this as their name - customer will see `PAYPAL * Candy Shop` in their bank statement.

    - **orderInformation.amountDetails.taxDetails.taxId**
    - **orderInformation.amountDetails.taxDetails.type**
    - **data.orderInformation.billTo.address1**
    - **data.orderInformation.billTo.company.name**
    - **data.orderInformation.billTo.country**

- **data.orderInformation.billTo.email**
- **data.orderInformation.billTo.firstName**
- **data.orderInformation.billTo.lastName**
- **data.orderInformation.billTo.locality**
- **data.orderInformation.billTo.phoneNumber**
- **orderInformation.invoiceDetails.invoiceNumber**
- **orderInformation.invoiceDetails.productDescription**
- **orderInformation.lineItems[].totalAmount**
- **data.orderInformation.shipTo.country**
- **data.orderInformation.shipTo.locality**
- **data.orderInformation.shipTo.postalCode**
- **paymentInformation.customer.customerid**
- **processingInformation.processingInstruction**
  - Set this field to `ORDER_SAVED_EXPLICITLY` to save the customer's payment credentials using the save an order follow-on request.

Example: Multiple Line-Items

This example includes multiple line items in the **amountDetails** field object:

```
{
  "amountDetails": {
    "totalAmount": "221.91",
    "currency": "USD",
    "taxAmount": "14.42"
  },
  "lineItems": [
    {
      "productName": "501 Original Fit Jeans Blues",
      "productSku": "00501019403432",
      "quantity": 1,
      "unitPrice": "100.0",
      "totalAmount": "100.00",
      "taxAmount": "8.63"
    },
    {
```

```
      "productName": "Levi Blue shirt Plaid",
      "productSku": "0094784142",
      "quantity": 1,
      "unitPrice": "100.00",
      "totalAmount": "100.00",
      "taxAmount": "5.79"
    },
    {
      "productName": "shipping_and_handling",
      "productSku": "shipping_and_handling",
      "quantity": 1,
      "unitPrice": "7.49",
      "totalAmount": "7.49",
      "taxAmount": "0.00"
    }
  ]
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Venmo

Venmo is a mobile payment service that is owned by PayPal and enables your customers to make payments from their Venmo mobile application. Customers link their Venmo accounts to their bank accounts, debit cards, and credit cards to send and receive payments. Venmo is a secure and convenient payment service for your customers to make payments without directly using their bank accounts or credit cards.

Opt in to Venmo on Unified Checkout

You can enable Venmo from the Unified Checkout merchant experience section in the Business Center. For information about how to enable Venmo using the Business Center, see Configure Payment Options (on page 253).

You can also enable Venmo using the Unified Checkout `v1/sessions` API. Follow these steps to opt in to the Venmo payment method using the API:

1. Add Venmo to your integration by adding `VENMO` to the **allowedPaymentTypes** field object within the capture context request:

2. Set the **completeMandate.type** field value to `CAPTURE` or `PREFER_AUTH`.

   You can capture the funds immediately if you include the **completeMandate.type** field in the capture context request and set the value to `CAPTURE`.

   If you accept more than one payment type and must perform an authorization where funds are collected at a later time, set the **completeMandate.type** field to `PREFER_AUTH`. The funds are captured immediately for the online bank transfer.

3. Include these required fields for online bank transfers in the capture context request:

   - **orderInformation.amountDetails.currency**
   - **orderInformation.amountDetails.taxAmount**
     - This field is the sum of all **orderInformation.lineItems[].taxAmount** values.
   - **orderInformation.amountDetails.totalAmount**
     - This field is the sum of (**orderInformation.lineItems[].quantity** × **orderInformation.lineItems[].unitPrice**) + **orderInformation.lineItems[].taxAmount** for all line items.
   - **orderInformation.lineItems[].productDescription**
   - **orderInformation.lineItems[].productName**
   - **orderInformation.lineItems[].productSKU**
   - **orderInformation.lineItems[].quantity**
   - **orderInformation.lineItems[].taxAmount**

     Important: This field is required for all line items when you include **orderInformation.lineItems[].taxAmount** for one line item.

     - You can set this field value to `0.00`.
   - **orderInformation.lineItems[].unitPrice**

4. Include these optional fields in the capture context request:

   - **buyerInformation.dateOfBirth**
   - **buyerInformation.language**
   - **buyerInformation.personalIdentification[].id**

- **buyerInformation.personalIdentification[].type**
- **clientReferenceInformation.reconciliationId**
- **merchantInformation.merchantDescriptor.name**
    - This field affects the name that the customers see on their card statement. If the merchant is Candy Shop and passes this as their name - customer will see `PAYPAL * Candy Shop` in their bank statement.
- **orderInformation.amountDetails.taxDetails.taxId**
- **orderInformation.amountDetails.taxDetails.type**
- **data.orderInformation.billTo.address1**
- **data.orderInformation.billTo.company.name**
- **data.orderInformation.billTo.country**
- **data.orderInformation.billTo.email**
- **data.orderInformation.billTo.firstName**
- **data.orderInformation.billTo.lastName**
- **data.orderInformation.billTo.locality**
- **data.orderInformation.billTo.phoneNumber**
- **orderInformation.invoiceDetails.invoiceNumber**
- **orderInformation.invoiceDetails.productDescription**
- **orderInformation.lineItems[].totalAmount**
- **orderInformation.lineItems[].typeOfSupply.type**
- **data.orderInformation.shipTo.country**
- **data.orderInformation.shipTo.locality**
- **data.orderInformation.shipTo.postalCode**
- **paymentInformation.customer.customerid**
- **processingInformation.processingInstruction**
    - Set this field to `ORDER_SAVED_EXPLICITLY` to save the customer's payment credentials using the save an order follow-on request.

Example: Multiple Line-Items

This example includes multiple line items in the **amountDetails** field object:

```
{
  "amountDetails": {
    "totalAmount": "221.91",
    "currency": "USD",
    "taxAmount": "14.42"
  },
  "lineItems": [
    {
      "productName": "501 Original Fit Jeans Blues",
      "productSku": "00501019403432",
      "quantity": 1,
      "unitPrice": "100.0",
      "totalAmount": "100.00",
      "taxAmount": "8.63"
    },
    {
      "productName": "Levi Blue shirt Plaid",
      "productSku": "0094784142",
      "quantity": 1,
      "unitPrice": "100.00",
      "totalAmount": "100.00",
      "taxAmount": "5.79"
    },
    {
      "productName": "shipping_and_handling",
      "productSku": "shipping_and_handling",
      "quantity": 1,
      "unitPrice": "7.49",
      "totalAmount": "7.49",
      "taxAmount": "0.00"
    }
  ]
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Verify Status for PayPal and Venmo

When the status of your payment request is `PENDING`, you can verify the status by sending a POST request to the URL that is included in the **transactionStatus.url** field in the webhook response:


```
{
  "transactionStatus": {
    "url":
 "https://apitest.cybersource.com/pts/v2/refresh-payment-status/7640632277149132395899488",
    "method": "POST",
    "payload": {
      "clientReferenceInformation": {
        "applicationName": "unifiedCheckout"
      },
      "processingInformation": {
        "actionList": [
          "AP_STATUS"
        ]
      },
      "paymentInformation": {
        "paymentType": {
          "method": {
            "name": "payPal"
          },
          "name": "eWallet"
        }
      }
    }
  },
  "transactionResult": {
    "details": {
      "submitTimeUtc": "2025-11-25T09:31:14Z",
      "processorInformation": {
        "sellerProtection": {
          "eligibility": "ELIGIBLE",
          "disputeCategories": [
            "ITEM_NOT_RECEIVED",
            "UNAUTHORIZED_TRANSACTION"
          ]
        },
        "transactionId": "14815899TN877504A",
        "orderStatus": "COMPLETED",
        "orderId": "74902340T07048209",
        "updateTimeUtc": "2025-11-25T09:33:48Z",
        "expirationTimeUtc": "2025-12-24T09:33:48Z"
      },
      "orderInformation": {
        "amountDetails": {
```

```
          "totalAmount": "0.11",
          "currency": "USD"
        },
        "billTo": {
          "email": "jane.doe@visa.com",
          "lastName": "Doe",
          "firstName": "Jane"
        }
      },
      "updateTimeUtc": "2025-11-25T09:33:48Z",
      "buyerInformation": {
        "merchantCustomerId": "JS4EXZRT68ED8"
      },
      "message": "Successful",
      "createTimeUtc": "2025-11-25T09:33:48Z",
      "clientReferenceInformation": {
        "code": "default"
      },
      "reconciliationId": "76406307371191323955899488",
      "status": "COMPLETED",
      "id": "76406322771491323955899488"
    },
    "id": "76406322771491323955899488",
    "message": "Successful",
    "outcome": "COMPLETED",
    "status": "COMPLETED"
  }
}
```


You can also send a request to this endpoint to verify the status:

Production: `POST https://api.cybersource.com/pts/v2/refresh-payment-status/{id}`

Test: `POST https://apitest.cybersource.com/pts/v2/refresh-payment-status/{id}`

The *{id}* is ID that is returned in the webhook response. For more information, see Webhooks Support (on page 278).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Handle Responses

When Unified Checkout automatically processes a payment with `autoProcessing` is set to `true` or you have set `autoProcessing` to `false` and are using **checkout.Complete()**, you must handle both successful responses and various errors. After the payment is complete, the **completeResponse** field object contains information about the transaction outcome.

When a payment is processed successfully, you must parse the response to confirm the payment status, update their order records, and trigger any post-payment workflows. Post-payment workflows include sending confirmation emails or updating inventory. See JavaScript Example: Processing a Payment (on page 241).

Your error handling should account for specific cases such as `COMPLETE_TRANSACTION_CANCELED` and `COMPLETE_TRANSACTION_FAILED`. `COMPLETE_TRANSACTION_CANCELED` occurs when the user cancels the transaction and `COMPLETE_TRANSACTION_FAILED` indicates that the consumer's transaction failed.

For wallet based payments, only cancellation errors are returned. For information about possible errors that can occur when calling the complete API, see UnifiedCheckoutError (on page 329) in Handle Errors (on page 329).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Captures

When you set the **completeMandate.type** field value to `AUTH` or `PREFER_AUTH`, you must send a request to capture an authorized payment. Full and partial captures are supported.

Endpoint

Production: `POST https://api.cybersource.com/pts/v2/payments/{id}/captures`

Test: `POST https://apitest.cybersource.com/pts/v2/payments/{id}/captures`

The *{id}* is the transaction ID returned in the authorization response.

Example: Authorization Response from Unified Checkout


```
{
  "details": {
    "clientReferenceInformation": {
      "code": "1753351101383"
    },
    "orderInformation": {
      "amountDetails": {
        "currency": "USD",
        "totalAmount": "21.00"
      }
    },
    "processorInformation": {
      "approvalCode": "AUTH456789",
      "responseCode": "00003",
      "responseDetails": "00003",
      "transactionId": "201601180815391001180815 3AUTH"
    },
    "reconciliationId": "04RYADD29YRO",
    "submitTimeUtc": "2025-07-24T09:58:21Z"
  },
  "id": "7533511014286971803092",
  "message": "Request processed successfully.",
  "outcome": "AUTHORIZED",
  "status": "AUTHORIZED"
}
```


Response Status

Cybersource responds to your capture request with one of these statuses:

- `FAILED`: The capture request failed.
- `PENDING`: The capture request is accepted but not captured. Send a request to the check status service to retrieve status updates.
- `SETTLED`: The capture request is settled for the amount requested.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

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 362).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Capture Context

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 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 on JWTs see JSON Web Tokens (on page 362).

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. The request provides authentication, one-time keys, the target origin to the Unified Checkout integration in addition to allowed card networks and payment types.

Browser Support

Unified Checkout supports these browser versions:

- Safari 16
- Firefox 121
- Google Chrome/Chium-based browsers 118
- Microsoft Edge 118

Capture Context Example

Use the **targetOrigins** and the **allowedPaymentTypes** fields to define the target origin and the accepted digital payment methods in your capture context. Use the **completeMandate** to orchestrate follow-on services such as Payments, Decision Manager, Payer Authentication, and TMS. The data that is included in the capture context are configured in the merchant experience. For information about how to configure these fields, see Configure Payment Options (on page 253) Configure Customer Data and Payment Flow (on page 266). For information about what fields can be overridden by what is included in the capture context, see Capture Context Fields in the Business Center (on page 268).

This example shows a capture context with the minimum required fields:

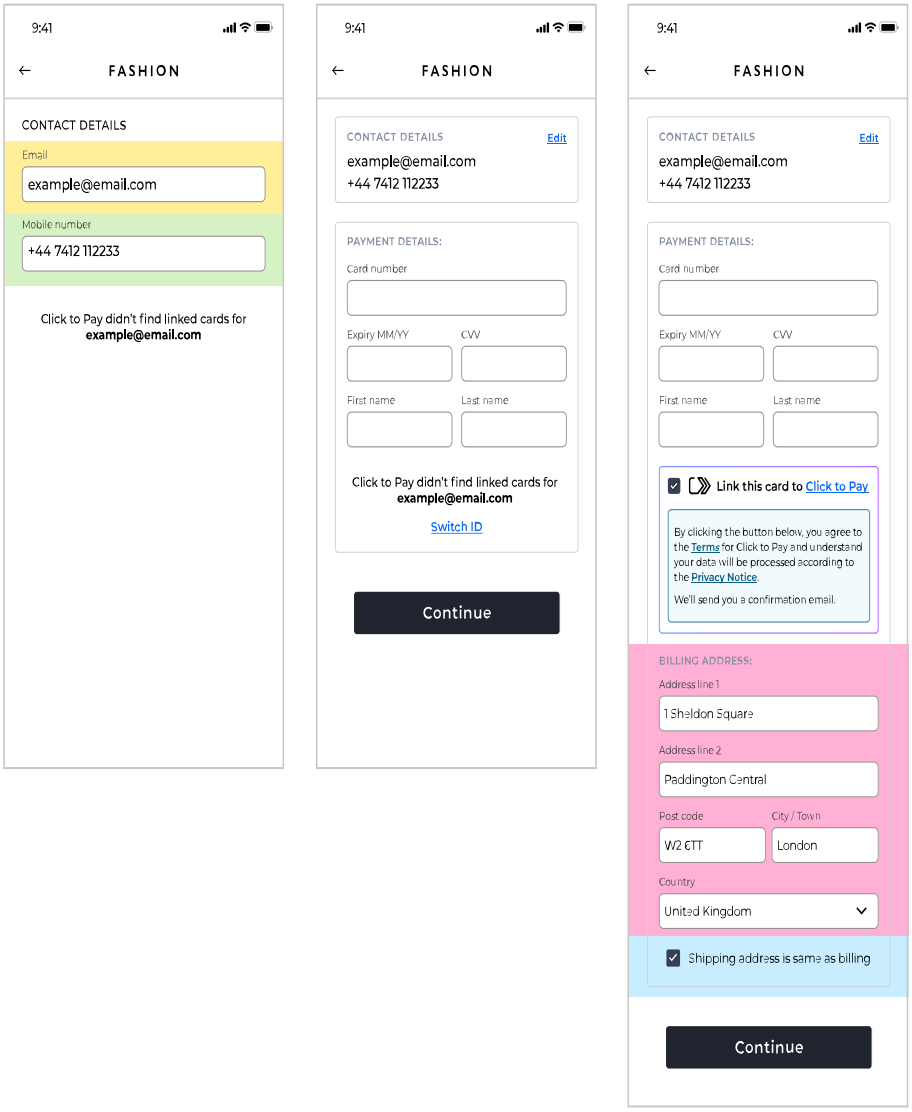
```
{
  "targetOrigins": ["https://merchant.com", "https://reseller.com:8443"],
  "locale": "en_US",
  "country": "US",
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "21.00",
      "currency":  "USD"
    }
  }
}
```

Card Entry Form

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" : "1.0",
  "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" : "example@email.com",
      "firstName" : "Joe",
      "lastName" : "Cool",
      "middleName" : "S",
      "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" : "Cool"
    }
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Versioning

Unified Checkout uses Semantic Versioning (SemVer). Version numbers use the `MAJOR.MINOR.PATCH` format:

- **MAJOR**: breaking changes that require code modifications
- **MINOR**: new features that are backwards compatible
- **PATCH**: bug fixes and improvements that are backwards compatible

The server controls which SDK version is loaded for each session that you request. When your server creates a session, the response JWT includes a **clientLibrary** field that contains the full URL to the correct version of the SDK. Your server parses the JWT, extracts the URL, and passes it to the frontend to load dynamically.

Important: The `clientVersion` field in the session request is optional. When you do not include this field, the server automatically resolves the appropriate version for every session. This ensures that the client-side library and server-side features are compatible.Cybersource recommends that you do not include the `clientVersion` field in your request and that you use the most recent version. When you do this, your integration benefits from new features, payment methods, and improvements and there are no code changes required.

Pin to a Version

If you must set your integration to a specific version, you can set the `clientVersion` field to a `MAJOR` version such as `1`, or a `MAJOR.MINOR` version such as `1.2`. The server uses the latest compatible patch release within that range. This ensures that you continue to receive security fixes and bug fixes.

Important: You cannot pin to a specific patch version (`MAJOR.MINOR.PATCH`). This ensures that all integrations receive critical patches. Cybersource recommends omitting **clientVersion** from your request unless you have a specific need for pinned behavior.

For information about the latest releases, see Client Version History (on page 235).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Client Version History

Below is a list of client versions and the features that are included in each version.

Important: Cybersource recommends that you use the most recent client version in your integration.

0.23

Accepts these card networks in the **allowedCardNetworks** field for manual card entry:

- Carnet
- Cartes Bancaires
- China UnionPay with card verification value (CVV)
- EFTPOS
- ELO
- JCrew PLCC
- mada
- Meeza

Ordering controls for the **allowedPaymentTypes** button.

De-coupling of PANENTRY from other payment types in the **allowedPaymentTypes** field.

0.24

Support for enabling combo cards in the capture context.

Support for eight-digit BINs.

Support for enabling card save in the capture context.

0.25

Addition of **Skip Verification next time** in the Click to Pay payment flow.

Support for CPF in the capture context.

0.26

Support for auto-lookup in Click to Pay when an email is included in the capture context.

Inclusion of the **cardDetails** field object in the transient token response.

Support for the **cardholderAuthenticationStatus** field object in the transient token response.

Support for the complete mandate.

0.28

Complete mandate enhancement to support Payer Authentication for manual card entry for Visa, Mastercard, American Express, Discover, JCB, Cartes Bancaires, China UnionPay, and ELO card brands.

Support for Afterpay as an **allowedPaymentType**.

Support for PayPak as an **allowedCardNetwork**.

Auto-enrollment for Click to Pay in supported markets.

Removal of the confirm or continue screen for specific use cases.

Static button for Click to Pay flows.

0.30

Support for iDeal, Multibanco, and Przelewy24|P24.

Complete mandate enhancement to support Payer Authentication for Google Pay and Click to Pay.

Support for Pakistan locales (en_PK and ur_PK).

New look and feel of Unified Checkout in line with EMVCO best practices.

0.31

Addition of the **data** object of the **orderInformation** field object and pass-through fields.

Support for **tokenCreate** in the complete mandate.

Support of pass-through fields, including challenge codes and data only, for Payer Authentication.

Support for Jaywan as an **allowedCardNetwork**.

Updated the payment details response to return detected card types. Multiple card types are shown when more than one card type is detected.

Support for Bancontact, Dragonpay, MyBank, and Tink Pay By Bank.

0.32

Support for KCP and UATP in the **allowedCardNetwork** field.

Support for Konbini as a post-pay reference payment method.

A radio button in the UI for Cartes Bancaires dual-branded cards.

0.33

Support for mobile as identity for Click to Pay accounts.

Japanese language translation updates.

UX captures billing and shipping information when they are not included in the capture context.

0.34

Iframes are used instead of pop-ups to reduce pop-up blocking and streamlining mobile deployment.

Additional BIN range for Jaywan card types.

0.35

Look and feel customization.

1.0

Configure payment options.

Configure customer data and payment flow.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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 embedded 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 362).
3. Initialize the unified payment object with optional parameters.
4. Show the button list or payment acceptance page or both.
5. Process the payment request using the instructions included within the capture mandate.

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

For information about handling the errors that may occur on the client-side, see Handle Errors (on page 329).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Loading the JavaScript Library

Use the client library asset path and client library integrity value that is returned by the capture context response to invoke Unified Checkout on your page.

You must retrieve these values from the **clientLibrary** and **clientLibraryIntegrity** fields that are returned in the JWT from `https://apitest.cybersource.com/uc/v1/sessions`. You can use these values to create your script tags.

You must perform this process for each transaction, as these values are unique for each transaction. You must avoid hard-coding values for the **clientLibrary** and **clientLibraryIntegrity** fields to prevent client-side errors.

For example, a response from `https://apitest.cybersource.com/uc/v1/sessions` would include:

```
"data": {
    "clientLibrary":"[EXTRACT clientLibrary VALUE from here]",
    "clientLibraryIntegrity": "[EXTRACT clientLibraryIntegrity VALUE from here]"
}
```

Below is an example script tag:

```
<script src="[INSERT clientLibrary VALUE HERE]"
    integrity="[INSERT clientLibraryIntegrity VALUE HERE]"
    crossorigin="anonymous"></script>
```

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

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

For information about the client-side API, see JavaScript API Reference (on page 341).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Initializing the SDK

```
const client = await VAS.UnifiedCheckout(sessionJWT);
```

In this example, `sessionJWT` refers to the capture context JWT.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Displaying the Button List

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

```
// Sidebar
const result = await checkout.mount('#buttons');

// Embedded
const result = await checkout.mount({
  paymentSelection: '#buttons',
  paymentScreen: '#form'
});
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry

When you display `CLICKTOPAY` or `PANENTRY` as allowed payment types, you can load the UI without displaying the Unified Checkout checkout button. You can do this by creating a trigger that defines what event loads the UI.

You can create a trigger only for `CLICKTOPAY` or `PANENTRY` payment methods:

```
//PAN Entry

const trigger = client.createTrigger('PANENTRY');

//Click to Pay

const trigger = client.createTrigger('CLICKTOPAY');
```

Important: When you use the `client.createTrigger()` method for Click to Pay, you must create a custom UI. See Click to Pay UI Guidelines (on page 177).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Processing a Payment

Payment is initiated when Unified Checkout captures the customer's payment information by calling the `client.createCheckout()`. When `autoProcessing` is set to `true`, the payment is completed. When `autoProcessing` is set to `false`, you can manually complete the payment using `checkout.complete(token)`. See Authorizations with a Transient Token (on page 318).

```
// Automatic (default when completeMandate is in session)
const checkout = await client.createCheckout({ autoProcessing: true });
const result = await checkout.mount('#buttons');
// result is the completed transaction – no need to call complete()

// Manual - similar to v0
```

```
const checkout = await client.createCheckout({ autoProcessing: false });
const token = await checkout.mount('#buttons');
const result = await checkout.complete(token);
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Authorization

Collect payment information and process an authorization. You must initiate a separate capture request to move funds and complete the transaction.

```
async function launchCheckout() {
  try {
    const client = await VAS.UnifiedCheckout(sessionJWT);
    const checkout = await client.createCheckout();
    const result = await checkout.mount('#payment-buttons');

    // result contains the completed payment result JWT
    // Send result to your server for verification
    sendToServer(result);
  } catch (error) {
    if (error.name === 'UnifiedCheckoutError') {
      handleError(error.reason, error.message);
    }
  } finally {
    checkout.destroy();
    client.destroy();
  }
}

launchCheckout();
```

Because the session includes **completeMandate**, `autoProcessing` defaults to `true` and `mount()` returns the completed payment result directly.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Sale

Collect payment information and process a sale. A sale is a combined authorization and capture in a single step.

```
async function launchCheckout() {
  try {
    const client = await VAS.UnifiedCheckout(sessionJWT);
    const checkout = await client.createCheckout();
    const result = await checkout.mount('#payment-buttons');

    // result contains the completed payment result JWT
    // Send result to your server for verification
    sendToServer(result);
  } catch (error) {
    if (error.name === 'UnifiedCheckoutError') {
      handleError(error.reason, error.message);
    }
  } finally {
    checkout.destroy();
    client.destroy();
  }
}

launchCheckout();
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Sale with Decision Manager

Collect payment information and process a sale while running Decision Manager fraud screening before the payment is initiated.

```
async function launchCheckout() {
  try {
    const client = await VAS.UnifiedCheckout(sessionJWT);
    const checkout = await client.createCheckout();
    const result = await checkout.mount('#payment-buttons');

    // result contains the completed payment result JWT
    // Send result to your server for verification
    sendToServer(result);
  } catch (error) {
    if (error.name === 'UnifiedCheckoutError') {
      handleError(error.reason, error.message);
    }
  } finally {
    checkout.destroy();
    client.destroy();
  }
}

launchCheckout();
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: No Service Orchestration

Collect payment information and receive a transient token. Your server handles payment authorization and any follow-on services directly.

```
async function launchCheckout() {
  try {
    const client = await VAS.UnifiedCheckout(sessionJWT);
    const checkout = await client.createCheckout({ autoProcessing: false });
    const transientToken = await checkout.mount('#payment-buttons');
```

```
    // transientToken is a JWT — send it to your server
    // Your server uses the transient token to authorize the payment
    sendToServer(transientToken);
  } catch (error) {
    if (error.name === 'UnifiedCheckoutError') {
      handleError(error.reason, error.message);
    }
  } finally {
    checkout.destroy();
    client.destroy();
  }
}

launchCheckout();
```

Without a **completeMandate**, `autoProcessing` defaults to `false` and the `mount()` call returns a transient token that you pass to your server for payment authorization. For information about how to use the transient token in an authorization request, see Transient Tokens (on page 313).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Setting Up with Full Sidebar

```
<html>
<head>
  <script
    src="[INSERT clientLibrary VALUE HERE]"
    integrity="[INSERT clientLibraryIntegrity VALUE HERE]”
    crossorigin=”anonymous"
  ></script>
</head>
<body>
  <h1>Unified Checkout Integration</h1>
  <input
    type="hidden"
    name="sessionJWT"
    value="[INSERT sessionJWT HERE]"
  />
```

```
  <script type="text/javascript">
    const sessionJWT = document.getElementById("sessionJWT").value;

    async function launchCheckout() {
      try {
        const client = await VAS.UnifiedCheckout(sessionJWT);
        const checkout = await client.createCheckout();
        const result = await checkout.mount('#payment-buttons');

        // result contains the completed payment result JWT
        // Send result to your server for verification
        sendToServer(result);
      } catch (error) {
        if (error.name === 'UnifiedCheckoutError') {
          handleError(error.reason, error.message);
        }
      } finally {
        checkout.destroy();
        client.destroy();
      }
    }

    launchCheckout();
  </script>
</body>
</html>
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Setting Up with the Embedded Component

The main difference between using an embedded component and the sidebar is that the `VAS.UnifiedCheckout(sessionJWT)` object is set to `false`, and the location of the payment screen is passed in the containers argument.

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

```
<html>
<head>
  <script
    src="[INSERT clientLibrary VALUE HERE]"
    integrity="[INSERT clientLibraryIntegrity VALUE HERE]"
    crossorigin="anonymous"
  ></script>
</head>
<body>
  <h1>Unified Checkout Integration</h1>
  <input
    type="hidden"
    id="sessionJWT"
    name="sessionJWT"
    value="[INSERT sessionJWT HERE]"
  />
  <script type="text/javascript">
    const sessionJWT = document.getElementById("sessionJWT").value;

    async function launchCheckout() {
      let client;
      let checkout;

      try {
        client = await VAS.UnifiedCheckout(sessionJWT);
        checkout = await client.createCheckout();
        const result = await checkout.mount('#payment-buttons');

        // result contains the completed payment result JWT
        // Send result to your server for verification
        sendToServer(result);
      } catch (error) {
        if (error.name === 'UnifiedCheckoutError') {
          handleError(error.reason, error.message);
        }
      } finally {
        if (checkout) {
          checkout.destroy();
        }

        if (client) {
          client.destroy();
        }
      }
    }

    launchCheckout();
```

```
  </script>
</body>
</html>
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Complete Integration Examples

These examples show how to integrate Unified Checkoutwith different payment scenarios. Each example includes the session configuration and client-side JavaScript.

For information about session fields, see Sessions API (on page 280).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Configuration

This section contains the information required to configure Unified Checkout.

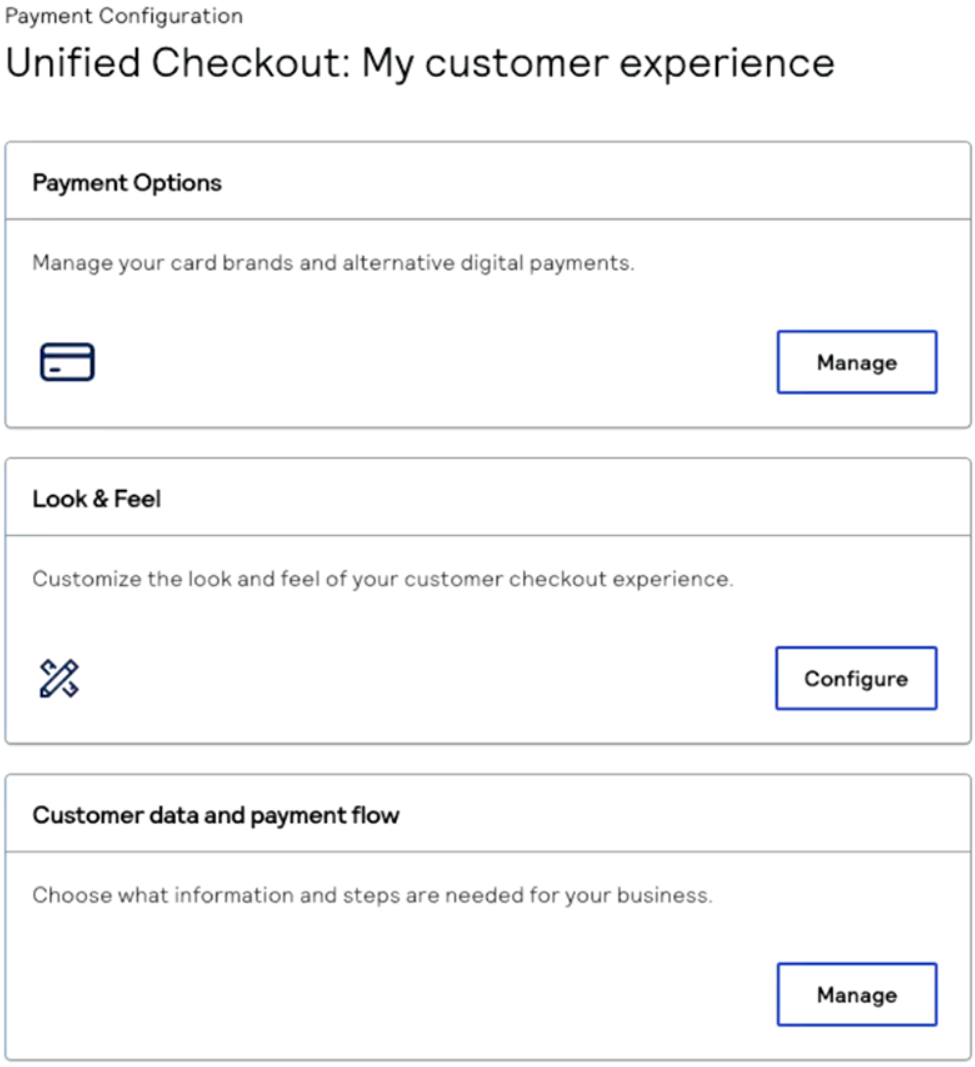
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Configure the Unified Checkout Merchant Experience

The Unified Checkout merchant experience interface provides complete control over your checkout experience by using dedicated configuration screens accessible through the Business Center:

Unified Checkout Customer Experience



This option is low-code and provides a user-friendly approach to customization of your Unified Checkout UI.

Configure each of these components of the checkout experience:

- **Payment Options**: Add, remove, and arrange payment methods. For information about configuring payment methods, see Configure Payment Options (on page 253) and Process Payments with Unified Checkout (on page 276).
- **Look & feel**: Configure visual appearance and branding. For information about configuring the look and feel, see Customize the Unified Checkout Look and Feel (on page 255).
- **Customer information and payment flow**: Control data collection and manage payment processing service. For information about configuring customer data, see Configure Customer Data and Payment Flow (on page 266).

You can also manage permissions as a direct merchant or as a portfolio administrator. For information about managing permissions, see Manage Permissions (on page 273).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

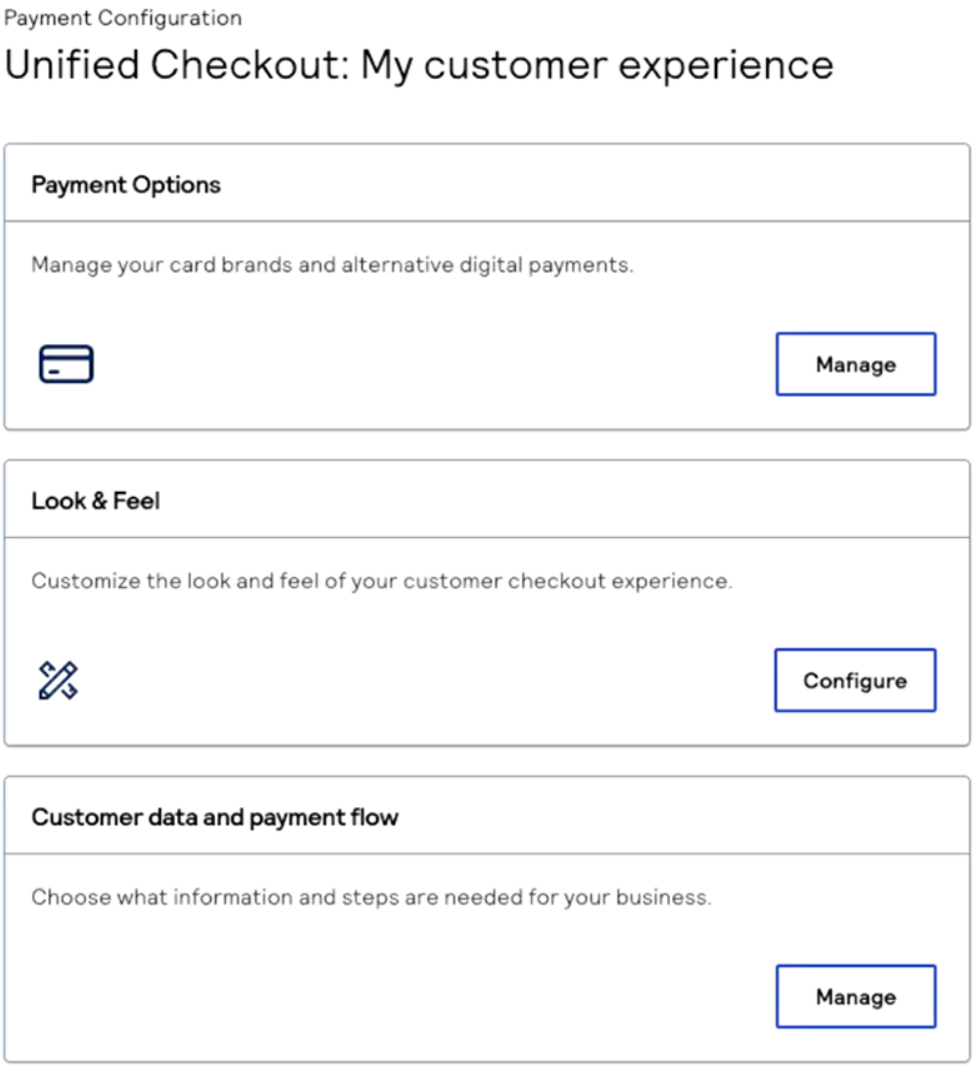
Enable Unified Checkout

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**. The Unified Checkout customer experience page appears:

Unified Checkout Customer Experience



3. The Unified Checkout configuration interface provides complete low-code control over your checkout experience by using dedicated configuration screens accessible through the Business Center. The configuration interface is organized into separate screens, each accessible from the *My customer experience* page. Configure each of these components of the checkout experience:

- **Payment Options**: Add, remove, and arrange payment methods. For information about configuring payment methods, see Configure Payment Options (on page 253).
- **Look & feel**: Configure visual appearance and branding. For information about configuring the look and feel, see Customize the Unified Checkout Look and Feel (on page 255).
- **Customer information and payment flow**: Control data collection and manage payment processing service. For information about configuring customer data, see Configure Customer Data and Payment Flow (on page 266).

Related information

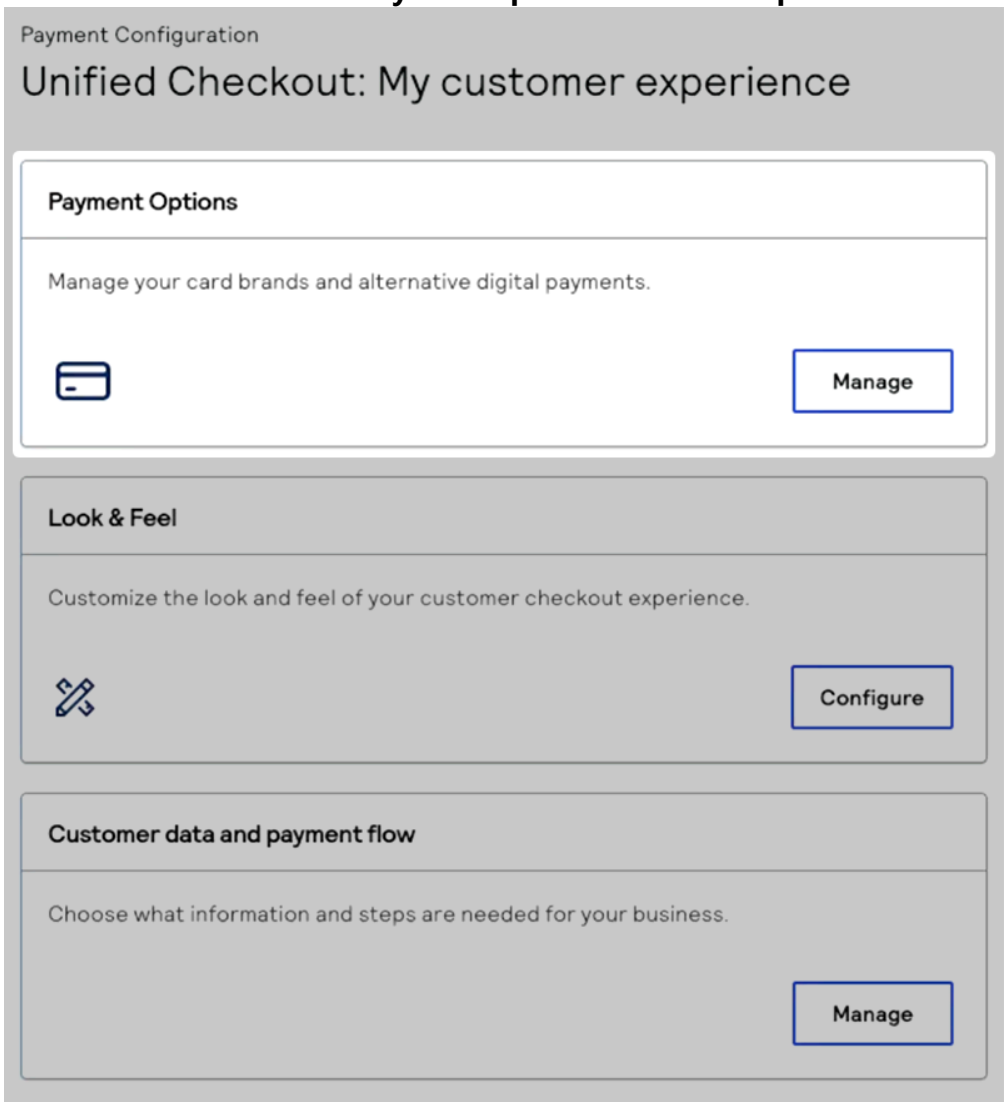
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Configure Payment Options

Payment Options in the Business Center enable you to control which payment methods appear in your checkout and in what order. Follow these steps to customize the available payment options in Unified Checkout in the Business Center:

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**. The Unified Checkout customer experience page appears:

Unified Checkout Payment Options Merchant Experience



3. In the Payment Options section, click **Manage**. The Payment Options page appears.

4. Under Payment options, click the checkbox next to each payment method that you want to display in your checkout UI. Click the drag icon (⠿) to rearrange the order of the payment options.

> **Important:** Some payment options are set at the portfolio level and cannot be reordered. When you see a pin icon next to a payment option, you cannot move that payment option in the list of available methods. You must contact your portfolio administrator if you want to reorder the list of available payment options.

These payment options are supported by Unified Checkout:

- Apple Pay (on page 162)
- Click to Pay (on page 170)
- Google Pay (on page 165)
- eCheck (on page 157)
- Konbini (on page 213)
- Bancontact (on page 192)
- DragonPay (on page 193)
- iDeal (on page 197)
- MyBank (on page 198)
- Multibanco (on page 199)
- Przelewy24|P24 (on page 200)
- Tink Pay By Bank (on page 194)
- Afterpay (on page 206)

5. Click **Manage** next to each payment type that you want to configure. The configuration page for the selected payment type appears.
6. Under card brands, slide the toggle ( ) to display the card brand logos in your button list. Click the checkbox next to each card brand that you want to display in your checkout UI. Click the drag icon ( ) to rearrange the order of the card brands.
7. Click **Save and publish** to save your payment options configuration settings.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

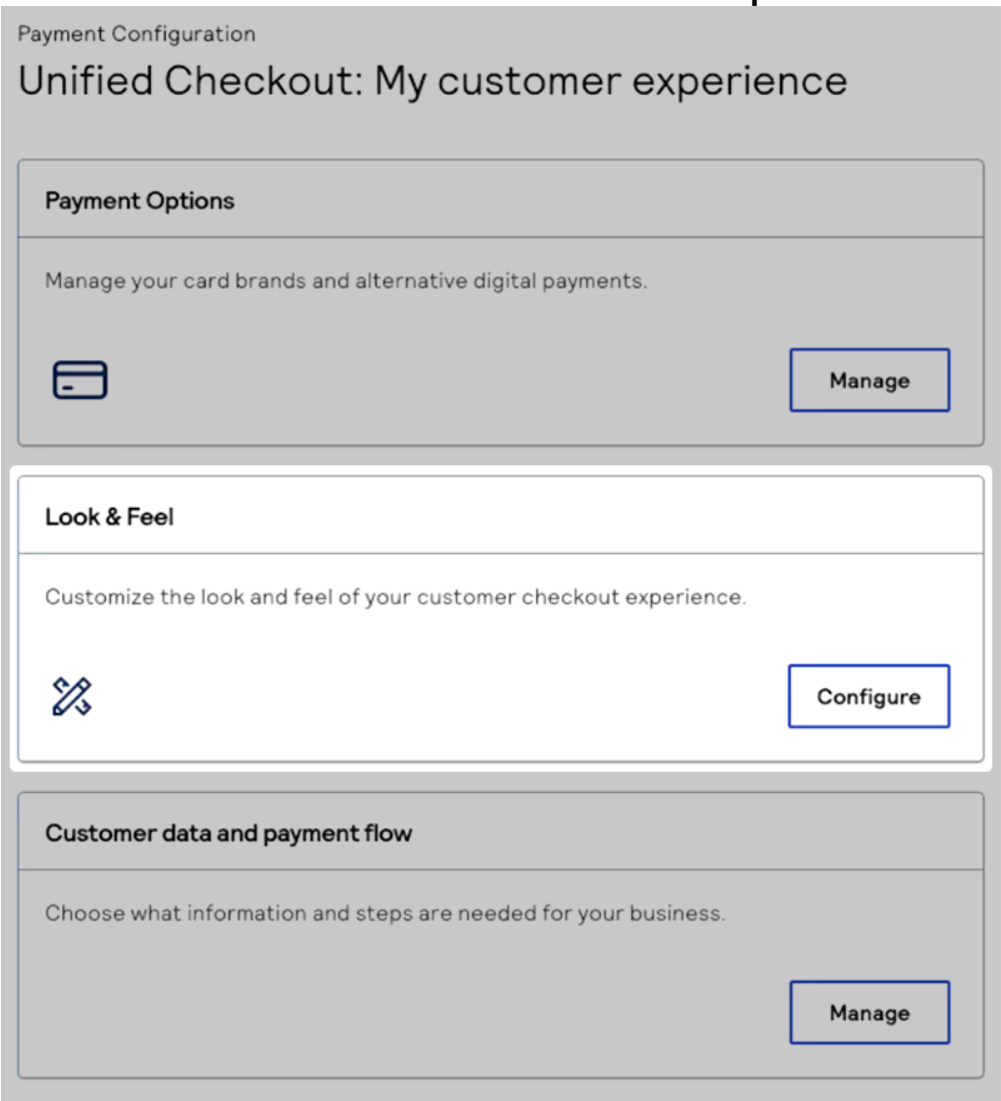
Customize the Unified Checkout Look and Feel

Follow these steps to customize the appearance of Unified Checkout in the Business Center:

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**. The Unified Checkout customer experience page appears:

Unified Checkout Look and Feel Merchant Experience



3. In the Look & Feel section, click **Configure**. The Look and feel page appears.

4. If you want to use AI to determine the look and feel, under AI brand studio, click **Browse** the upload a screenshot of your website. These components are updated using the colors from the screenshot you provide:

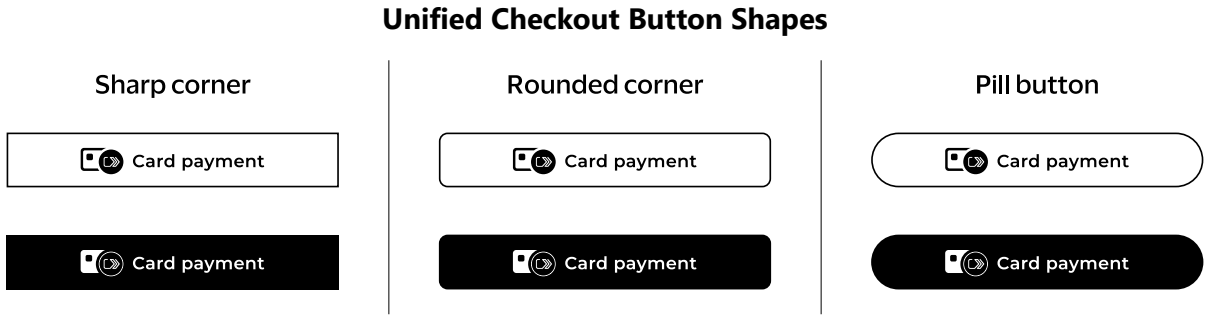
   - Button list background color
   - Card checkout outline and text colors
   - Header and checkout font background colors

Proceed to the next steps to manually customize these and other components of the Unified Checkout appearance.

5. Under Button customizations, select the options for your button list. These customizations are available:

Universal Button Shape

Use the Button shape drop-down menu to select if you want a sharp corner, rounded corner, or pill button:



Button List

Use the color selector or enter a HEX code in the Button list background color text box to customize the background color of your button list. To review the button list preview, see Button List (on page 260).

If you uploaded a screenshot in the AI brand studio section, a color is entered in this space based on the colors present in your screenshot.

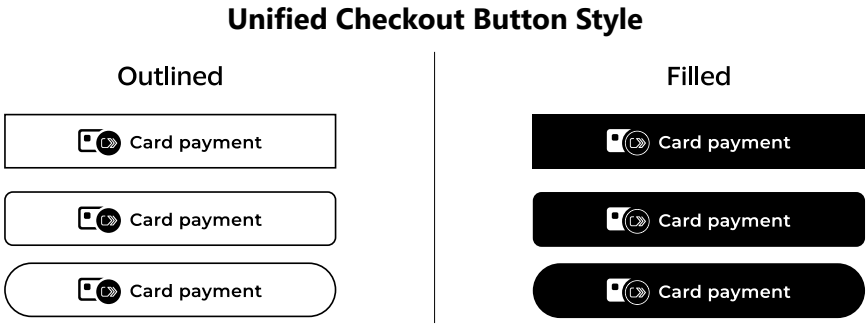
Card Checkout and Click to Pay Button

Use the card checkout button label drop-down menu to select the text that appears on the checkout button. These text options are available:

- Pay with card
- Card payment
- Checkout with card (default)
- Debit/Credit payment
- Donate with card
- Subscribe with card

Card Checkout Buttons

Use the button style drop-down menu to select if you want an outlined or filled checkout button. Select the button fill and text colors using the color selector or HEX code:



If you uploaded a screenshot in the AI brand studio section, a color is entered in this space based on the colors present in your screenshot.

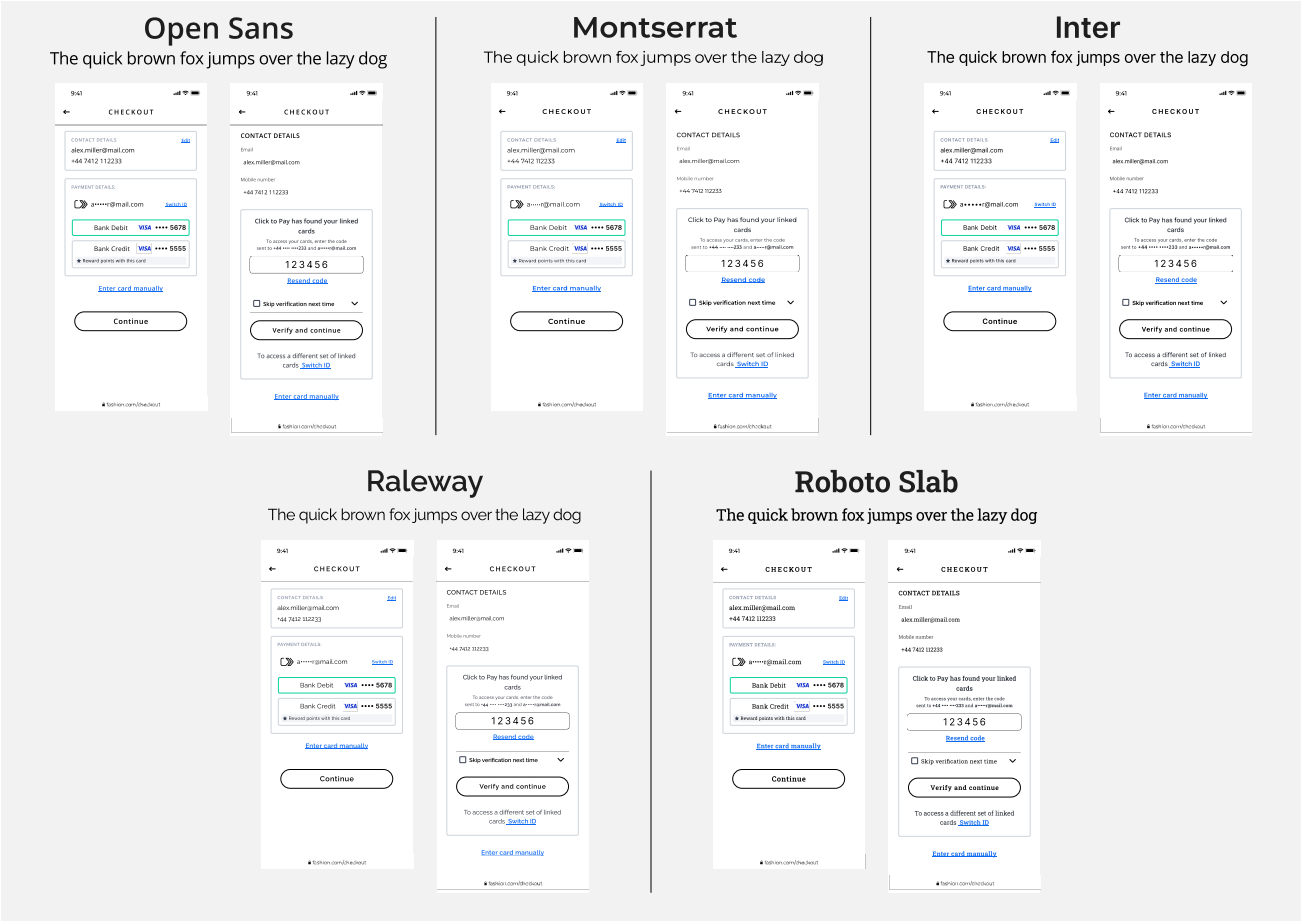
6. Under Checkout customizations, select the options for your checkout experience. These customizations are available:

Font

Use the font drop-down menu to select the font you want to use. These fonts are available:

- Inter
- Monserrat
- Open Sans
- Raleway
- Roboto Slab

Unified Checkout Fonts



Select the background color using the color selector or HEX code.

If you uploaded a screenshot in the AI brand studio section, a color is entered in this space based on the colors present in your screenshot.

Header Customization

Select the color you want to use in your header using the color selector or HEX code.

If you uploaded a screenshot in the AI brand studio section, a color is entered in this space based on the colors present in your screenshot.

7. Click **Save and publish** to save your look and feel customization. The Ready to publish popup window appears.

8. If you are done editing, click **Publish now** to publish your changes. If you need to make more changes, click **Keep editing** to return to the Look and feel page. To review your changes, click through the example screens. For more information about the UI previews, see Look and Feel UI Examples (on page 260).

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Look and Feel UI Examples

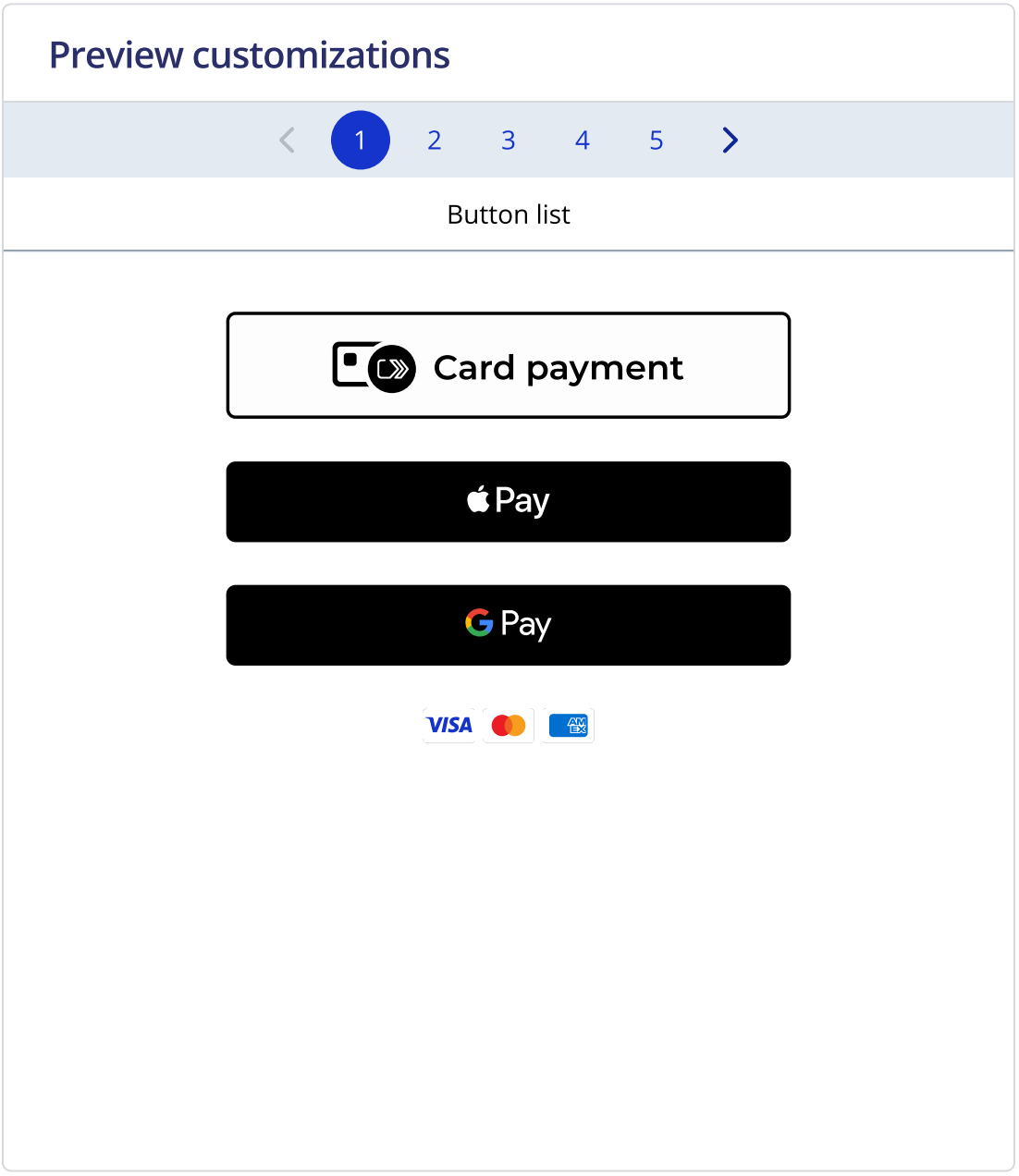
These examples show the different preview screens for each look and feel customization feature:

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Button List

This example shows the button list:

Screen 1: Button List



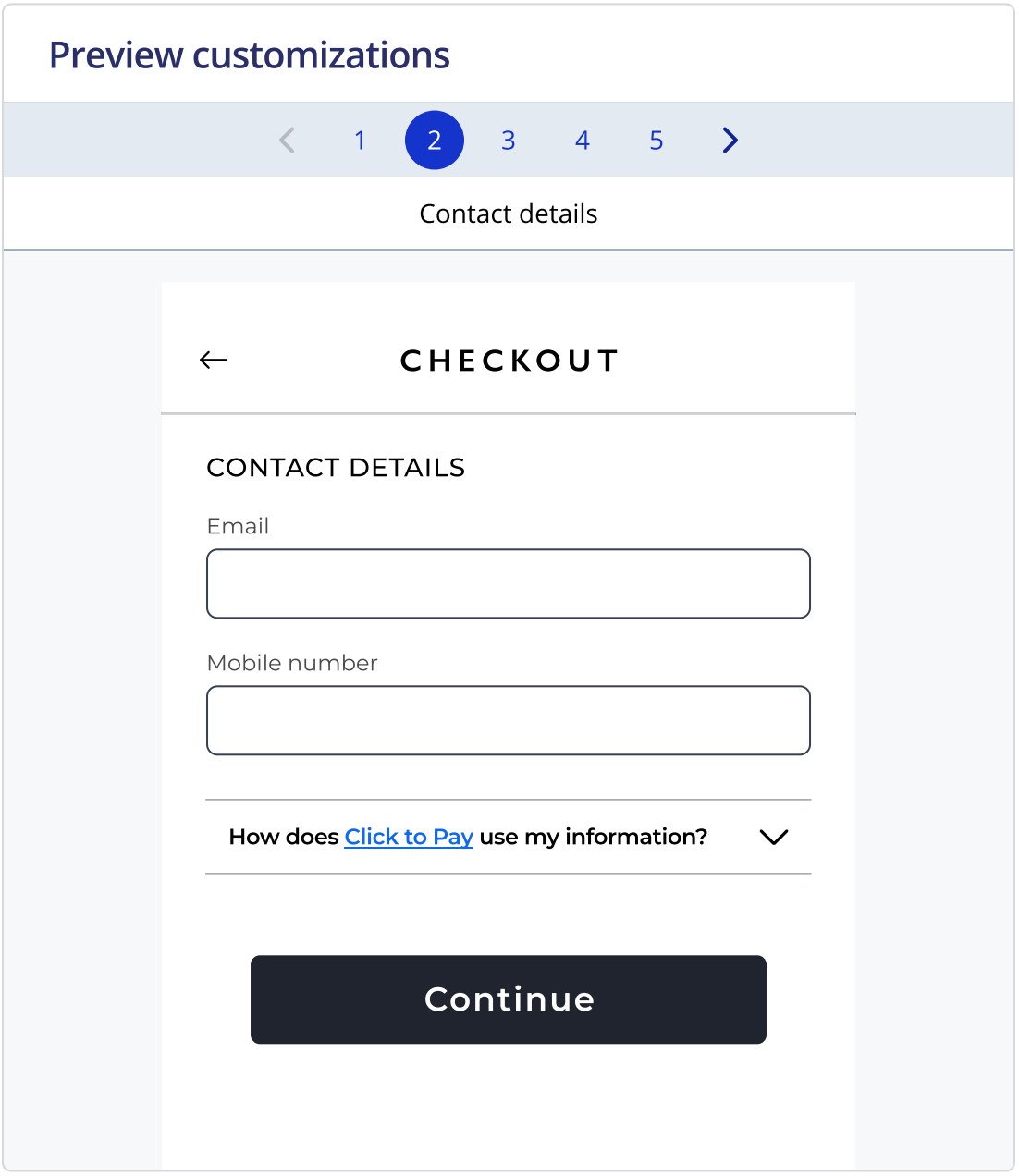
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Contact Details

This example shows the contact details that appear during checkout:

Screen 2: Contact Details



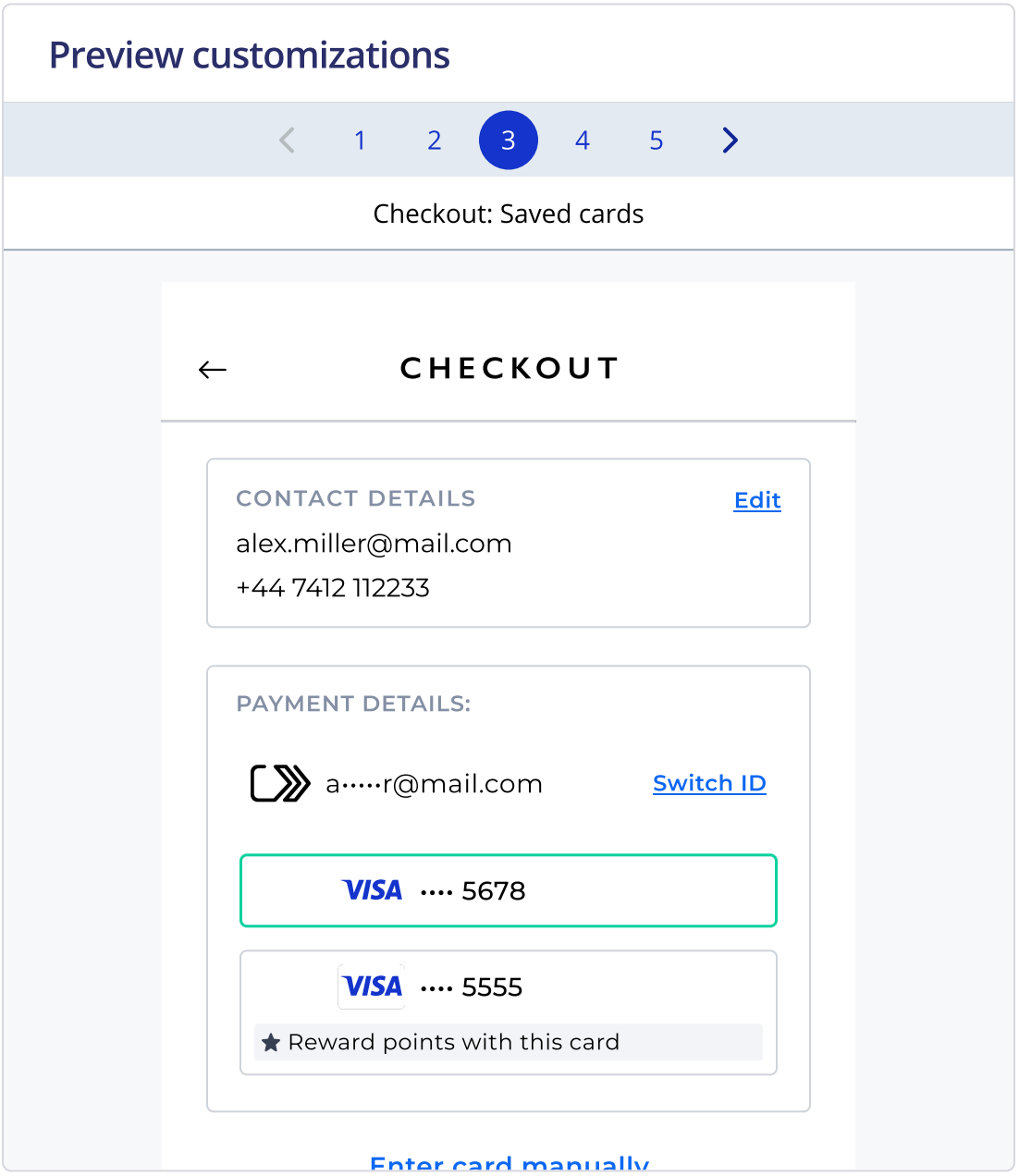
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Saved Cards

This example shows the saved card payment details that appear during checkout:

Screen 3: Saved Cards



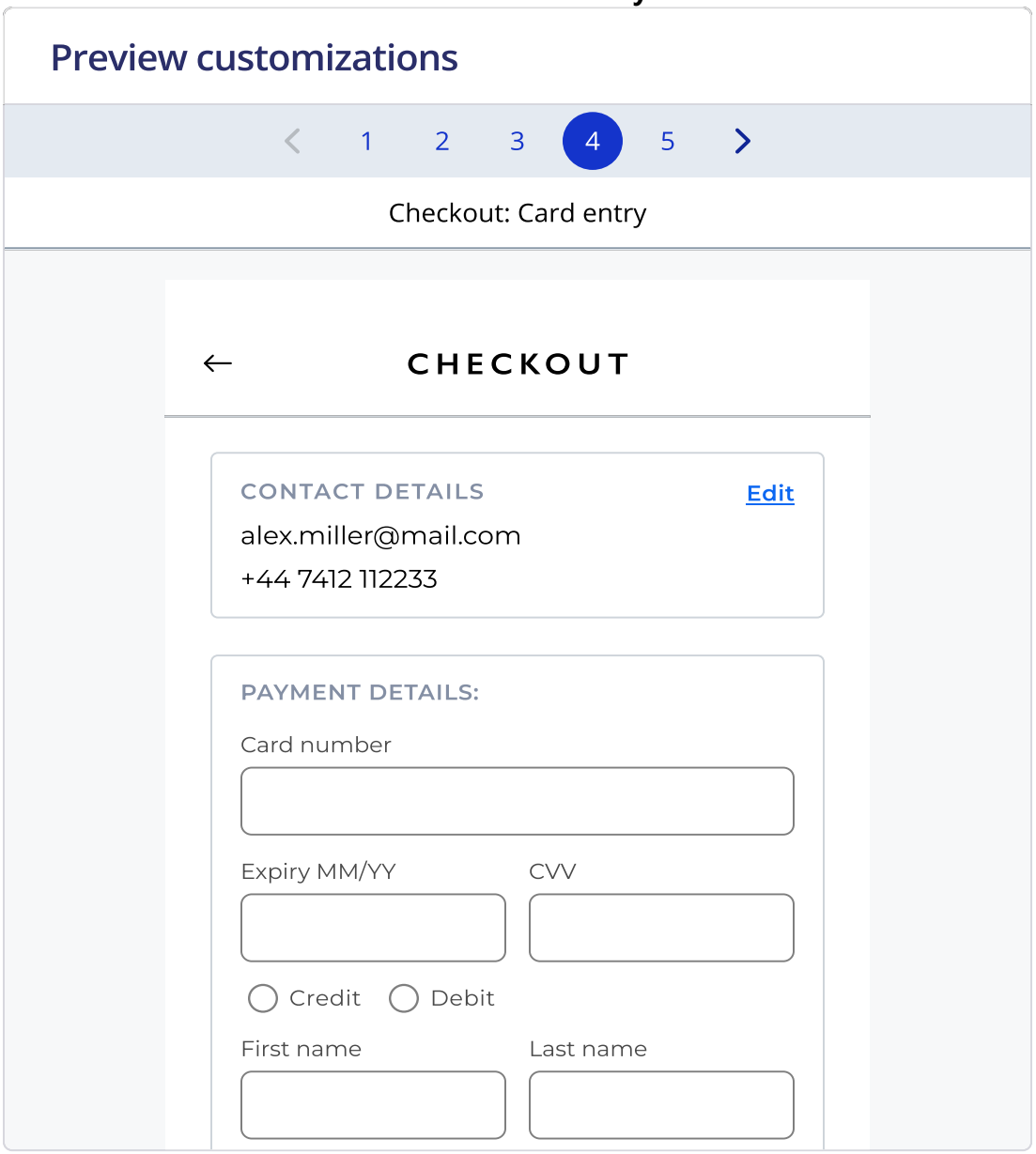
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Card Entry

This example shows the card entry payment details that appear during checkout:

Screen 4: Card Entry



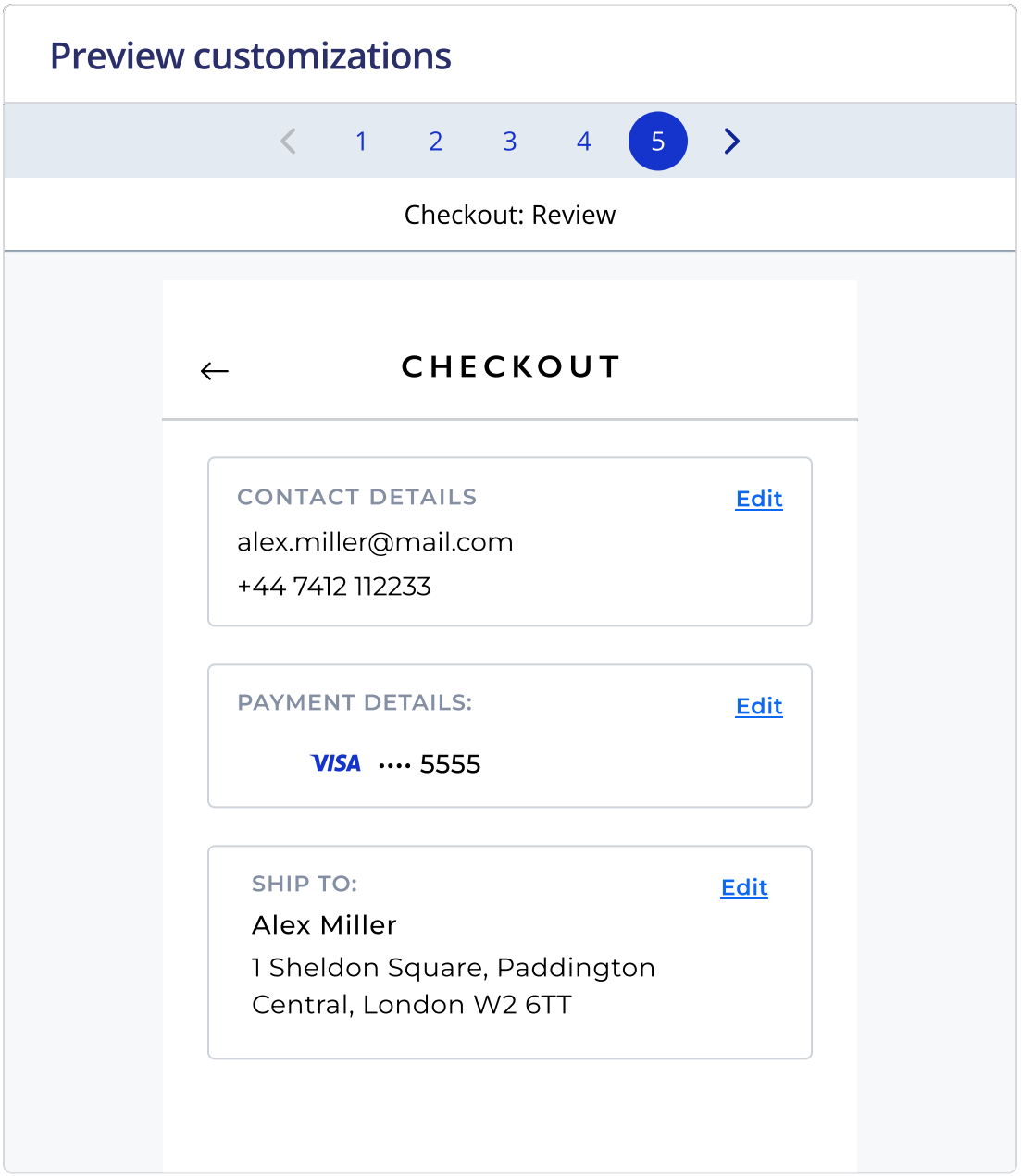
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Review Details

This example shows the review contact details that appear at the end of checkout:

Screen 5: Review



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

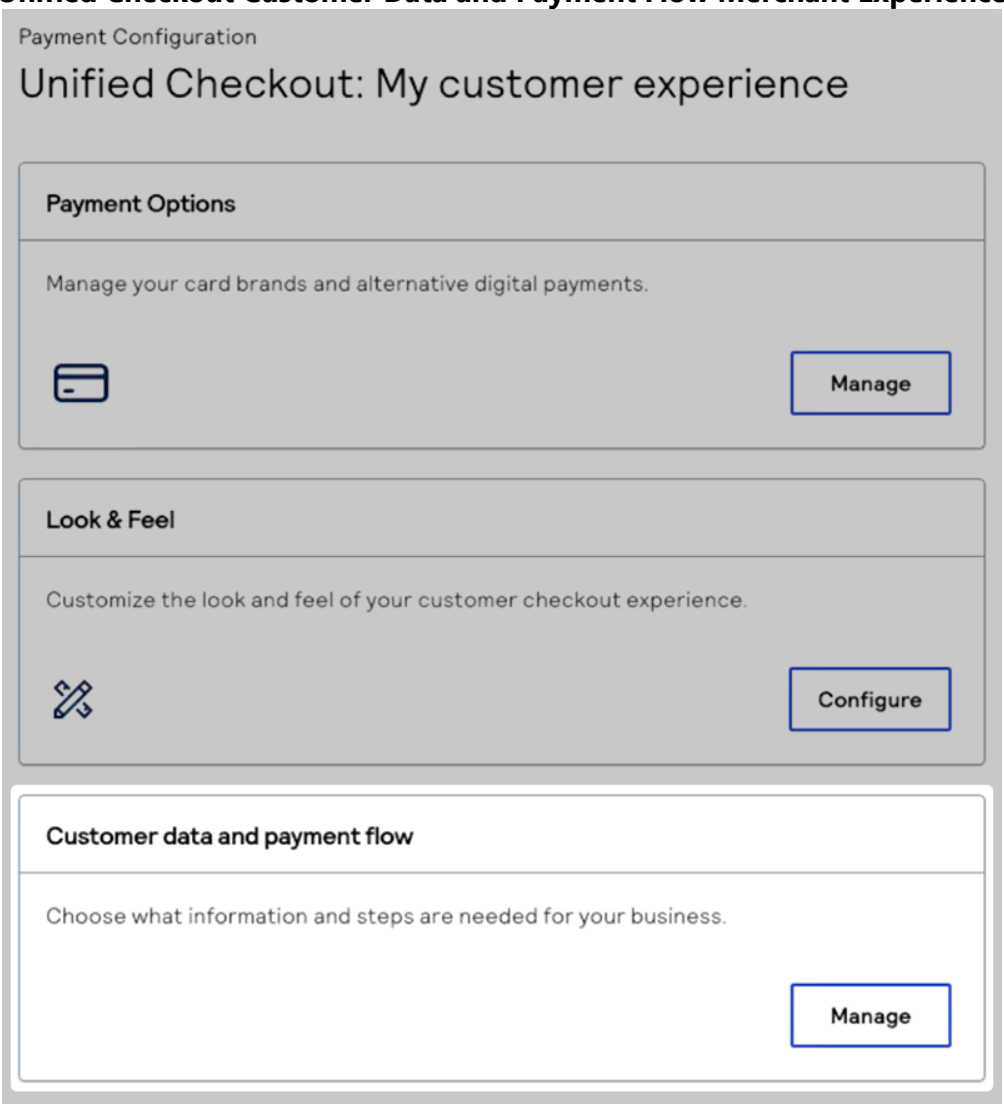
Configure Customer Data and Payment Flow

Use the Customer data and payment flow section of the Business Center to configure the information that you want to collect during checkout. Follow these steps to customize the appearance of Unified Checkout in the Business Center:

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**. The Unified Checkout customer experience page appears:

Unified Checkout Customer Data and Payment Flow Merchant Experience



3. In the Customer data and payment flow section, click **Manage**. The Customer information and payment flow page appears.

4. Under Contact details, slide the toggle ( ) next to Email address and Mobile phone number to collect the customer email address and phone number during checkout.

5. Under Payment details, slide the toggle ( ) next to Billing address to collect the cardholder billing address.

6. Under Payment details, in the Billing address drop-down menu, select the billing address information for payment verification:

    - **Full address**: Request the full cardholder billing address during checkout.

    - **Zip code only**: Request only the zip code of the cardholder billing address during checkout.

7. Under Payment details, slide the toggle ( ) next to Shipping address to collect the cardholder shipping address.

8. Under Payment details, in the Shipping address drop-down menu, select the countries that you ship to.

9. Under Checkout review step, slide the toggle ( ) to display a review page for the customer to confirm the payment and shipping address.

10. Under payment processing and & service orchestration, slide the toggle ( ) in the Payment processing section to control how to process transactions:
    You can turn payment processing on or off:

    - **Payment Processing ON**: Unified Checkout handles the complete payment for you automatically. When payment processing is on, you can select how transactions are processed:

        - **Preferred auth**: Choose this option to authorize the payment when possible.

        - **Sale**: Choose this option to capture the funds immediately.

    - **Payment Processing OFF**: You must complete the payment independently using VISA or another selected gateway.

    For information about enabling or disabling payment processing in Unified Checkout, see Process Payments with Unified Checkout (on page 276).

11. Under save customer information in payment processing and & service orchestration, slide the toggle to the right to prompt the user to save their payment information for future use.

    Saving customer payment details can significantly improve the checkout experience for returning customers. There are two approaches that you can use:

    - **Ask for consent to save payment information for future use**: This option displays a checkbox during checkout that asks customers if they want their payment details saved. You can enable this option regardless of if payment processing is enabled at any time.

    - **Store payment details securely in your TMS vault**: This option saves payment information using secure encryption and links directly to your TMS vault. This enables faster checkout for returning customers by using tokens rather than raw card data.

Important: If you already have cardholder consent (for example, if it was obtained during account creation or through another verified process) or consent is not required for your flow, you should not display or request consent again in the UI.

12. Under Fraud detection in payment processing and & service orchestration, use the drop-down menu to turn fraud detection with Decision Manager or Fraud Management Essentials **On** or **Skip**.
13. Under Localized payment settings, slide the toggle ( ) to enable these features:
    - **Combo cards**: Enable cardholders to decide how to process their payments.
    - **Brazil tax ID**: Collect a customer's CPF or CNPJ at checkout.

Important: Combo cards and tax IDs are only available in Brazil.

14. Click **Save and publish** to save your customer payment settings.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Capture Context Fields in the Business Center

There are some components of Unified Checkout that are available in the API, the Business Center or both. This section describes which components of the Sessions API capture context object can be configured using the Business Center. For a complete list of fields that are available, see the API Reference in the Cybersource Developer Center.

API Parameters

Parameter	Business Center	API	Notes
allowedCardNetworks	Yes	Override	Optional. This is the override priority order:

API Parameters (continued)

Parameter	Business Center	API	Notes
			1. API 2. Merchant experience profile 3. Portfolio profile
allowedPaymentTypes	Yes	Override	Optional. This is the priority order: 1. API (payment types that are not enabled are removed) 2. Merchant experience profile (payment types that are not enabled are removed) 3. Portfolio profile (all payment types are enabled at the profile level) **Important:** `SRCVISA`, `SRCMASTERCARD`, and `SRCAMEX` are not support

API Parameters (continued)

Parameter	Business Center	API	Notes
			ed. You must use `CLICKTOPAY`.
paymentConfigurations	Partial	Yes	Allows per-transaction payment type configuration overrides. This is available only in **clientVersion** `1.0` or later.
paymentConfigurations.CLICKTOPAY	Yes	Override	Default values can be set in the Business Center.
paymentConfigurations.CLICKTOPAY.autoCheckEnrollment	Yes	Override	This payment configuration is part of the merchant experience.
paymentConfigurations.GOOGLEPAY	Yes	Override	Default values can be set in the Business Center.
paymentConfigurations.GOOGLEPAY.allowedAuthMethods	Yes	Override	This payment configuration is part of the merchant experience.

Capture Mandate

Capture Mandate Parameters

Parameter	Business Center	API	Default Value
showConfirmationStep	Yes	Override	
billingType	Yes	Override	

Capture Mandate Parameters (continued)

Parameter	Business Center	API	Default Value
requestEmail	Yes	Overr ide	
requestPhone	Yes	Overr ide	
requestShipping	Yes	Overr ide	
shipToCountries	Yes	Overr ide	
showAcceptedNetworkIc ons	Yes	Overr ide	
comboCard	Yes	Overr ide	
requestSaveCredentials	Yes	Overr ide	
cpf	Yes	Overr ide	
cpf.required	Yes	Overr ide	

Complete Mandate

Complete Mandate Parameters

Parameter	Business Center	API	Default Value
type	Yes	Overr ide	`SALE`
decisionManager	Yes	Overr ide	`true`
consumerAuthenticat ion	Yes	Overr ide	`false`
tms	Yes	Optional	
tms.tokenCreate	Yes	Overr ide	`true`

Appearance Variables

Top-Level Appearance Parameters and Variables

Parameter / Variable	Business Center	API	Default Value	Example
buttonType	Yes	Override	`CHECKOUT`	"CHECKOUT"
variables	Partial	Override		
backgroundColor	Yes	Override		"#FFFFFF"
textColor	Yes	Override		"#000000"
headerBackground	Yes	Override		"#1A237E"
headerForeground	Yes	Override		"#FFFFFF"
buttonBackground	Yes	Override		"#E0E0E0"
buttonForeground	Yes	Override		"#333333"
buttonBorderRadius	Yes	Override		"4px"
fontFamily	Yes	Override		"Roboto Slab, serif"
paymentSelectionBackground	Yes	Override		"#F5F5F5"

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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 275).

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 274).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Process Payments with Unified Checkout

Payment processing is a payment completion option in your merchant configuration. Your configuration determines if your checkout system automatically handles and finalizes customer payments. When payment processing is enabled, Unified Checkout handles the complete payment for you automatically. When payment processing not enabled, you complete payments independently using your selected gateway.

Payment Processing Enabled

Cybersource recommends that you enable payment processing if you meet these requirements:

- Your integration is configured with the payment completion step. For more information about **completeMandate()** see Complete Mandate (on page 289).
- You do not have a designated technical team to process payments.
- Your integration includes any of these features:
  - Fraud checks with Decision Manager or Fraud Management Essentials
  - 3-D Secure/ Payer Authentication
  - Stored customer credentials with the Token Management Service(TMS)
  - Multiple payment methods

When payment processing is enabled, you can choose how payments should be handled:

- **Preferred Authorization**: The payment is authorized first and captured at a later time. This method works well for businesses that ship products to their customers. For example, you authorize the payment when the customer places and order and capture the payment when you ship it.
- **Sale**: The payment is captured immediately. This method works well for service businesses, digital products, and immediate purchases.

Important: Not all payment methods are compatible with all processing types. Different payment methods work with different payment processing types. You must configure your payment processing settings to be compatible with your integration and review which payment methods are compatible. For information about payment methods and their processing compatibility, see Payment Methods (on page 149).

Payment Processing Disabled

Cybersource recommends that you disable payment processing if you meet these requirements:

- You want to process payments on our platform using your own API requests instead of relying on the automatic payment completion step.
- You want full control over your checkout and payment orchestration flow.
- You only use Unified Checkout to collect encrypted payment information and you handle the completion phase through our platform APIs or your custom back-end. The completion phase includes authorization, sale, fraud checks, 3-D Secure and card storage.

When disable payment processing, the checkout system will collect payment information from your customers but you will need to process the actual payment on your end. When payment processing is disabled, you must consider the following:

- Customer payments are not completed automatically.
- Fraud checks and 3-D Secure using automatic orchestration are disabled.
- Saved credentials and token management using automatic processing are disabled.
- You must handle payment completion yourself.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Webhooks Support

Unified Checkout supports webhooks. You can use webhooks to obtain the complete response from the **completeMandate** call. To receive a webhook notification, you must first subscribe to the webhook.

Prerequisite

Webhook payloads are encrypted. In order to receive a Unified Checkout webhook notification, you must enabled message-level encryption (MLE). For information about enabling MLE, see Enable Message-Level Encryption in the *Getting Started with REST Developer Guide*.

Webhook Events

Unified Checkout Webhook Events

Product ID	Event Types	Description
`unifiedCheckout`	`uc.orders.transactionresults`	Full payload response from the payment service call made by Unified Checkout

Set Up Webhook Subscriptions

For information on setting up a webhook for the `unifiedCheckout` product, see the How to Set Up Webhook Subscriptions section of the *Webhooks Developer Guide*.

Example Webhook Payload

Example: Webhooks Request for Unified Checkout Events


```
{
  "organizationId": "your_merchant_id",
  "webhookId": "2d55e648-d96c-d727-e063-3cb8d30a938e",
  "productId": "unifiedCheckout",
  "eventType": "uc.orders.transactionresults",
  "eventDate": "2025-03-27T08:44:55",
  "payload": {
    "id": "743518889935640500 3091",
    "status": "AUTHORIZED",
    "outcome": "AUTHORIZED",
    "details": {
      "processorInformation": {
        "transactionId": "20160118081539100118081 53AUTH"
```

```
      },
      "paymentInformation": {
        "card": {
          "type": "001"
        }
      },
      "riskInformation": {
        "score": {
          "result": "42"
        }
      }
    }
  }
}
```

The `payload` field object contains the same fields as the response from a direct payment authorization request. Use the `id` field for capture requests or to look up a transaction.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Sessions API

Use the Sessions API to generate a capture context. The capture context contains all of the merchant-specific parameters that tell the front-end JavaScript library what to do within your payment experience.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Capture Context Components

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.

There are some components of Unified Checkout that are available in the API, the Business Center or both. For information about how to configure Unified Checkout using the Business Center, see Configure the Unified Checkout Merchant Experience (on page 249). For information about which fields are available using the API or the Business Center, and when one is overridden by the other, see Capture Context Fields in the Business Center (on page 268). For a full capture context with all possible fields, see Example: Unified Checkout Complete Capture Context (on page 301).

Use these required fields to request the capture context:

allowedPaymentTypes

clientVersion

country

locale

data.orderInformation.amountDetails.currency

data.orderInformation.amountDetails.totalAmount

targetOrigins

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

This example shows the minimum fields that must be included in the capture context:

```
{
  "targetOrigins": [
    "http://localhost:8080"
  ],
  "country": "US"
  "locale": "en_US"
  "data": {
    "orderInformation": {
      "amountDetails": {
        "totalAmount": "21.00",
        "currency": "USD"
      }
    }
  }
}
```

For information on JSON Web Tokens, see JSON Web Tokens (on page 362).

Important:

Cybersource recommends that you dynamically parse the response for the fields that you are looking for when you integrate with Cybersource APIs. Cybersource may add additional fields in the future.

You must ensure that your integration can handle new fields that are returned in the response. Even though the underlying data structures do 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.

Related Information

- API field reference guide for the REST API

Endpoint

Production: `POST https://api.cybersource.com/uc/v1/sessions`

Test: `POST https://apitest.cybersource.com/uc/v1/sessions`

Production in Saudi Arabia: `POST https://api.sa.cybersource.com/uc/v1/sessions`

Test in Saudi Arabia: `POST https://apitest.sa.cybersource.com/uc/v1/sessions`

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Allowed Card Networks

Use the **allowedCardNetworks** field to define the card types.

These card networks are available for card entry:

- American Express
- Cartes Bancaires
- Carnet
- China UnionPay
- Diners Club
- Discover
- EFTPOS
- ELO
- Jaywan
- JCB
- JCrew
- KCP
- mada
- Maestro

- Mastercard
- Meeza
- PayPak
- UATP
- Visa

To support dual-branded or co-badged cards, you must list your supported card type values for the **allowedCardNetworks** field based on your preference for processing card numbers. For example, if a card is dual-branded as Visa and Cartes Bancaires, and Cartes Bancaires is listed first, the card type is set to Cartes Bancaires after the card number is entered in your Unified Checkout card collection form. For information on dual-branded or co-badged cards, see Dual-Branded Cards (on page 317).

Important:

Some card types, such as KCP and UATP, do not have security codes (CVV or CVN). If you include only card types that do not have security codes in the **allowedCardNetworks** field, Unified Checkout does not display the security code field in the UI.

If you include card types that do not have security codes and cards types that do have security codes in the **allowedCardNetworks** field, Unified Checkout displays the security code field in the UI. The field is disabled in the UI when the cardholder enters a card number for a card type with no security code

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Target Origins

The target origin is defined by the scheme (protocol), hostname (domain), and port number (if used).

You must use the https:// protocol. Sub domains must also be included in the target origin.

Any valid top-level domains, such as .com, .co.uk, and .gov.br, are supported. Wildcards are not supported.

For example, if you are launching Unified Checkout on example.com, the target origin could be any of the following:

- https://example.com
- https://subdomain.example.com
- https://example.com:8080

When you use Unified Checkout in an iframe, you must include the domain for the URL that loads the iframe and the iframe URL in the **targetOrigins** field.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Allowed Payment Types

You can specify the type of Unified Checkout digital payment methods that you want to accept in the capture context.

Use the **allowedPaymentTypes** field to define the payment type:

- `AFTERPAY`
- `APPLEPAY`
- `BANCONTACT`
- `CHECK`
- `CLICKTOPAY`
- `DRAGONPAY`
- `GOOGLEPAY`
- `IDEAL`
- `KONBINI`
- `MULTIBANCO`

- `MYBANK`
- `P24`
- `PANENTRY`
- `PAZE`
- `TINKPAYBYBANK`

Important: Click to Pay accepts American Express, Mastercard, and Visa for saved cards. Visa and Mastercard tokenize payment credentials using network tokenization for all Click to Pay requests. Click to Pay uses Click to Pay Token Requester IDs (TRIDs) rather than your existing TRIDs to generate network tokens.

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

- Enrolling in Apple Pay (on page 162)
- Enabling Click to Pay in the Business Center (on page 170)
- Enrolling in Google Pay (on page 165)
- Alternative Payment Methods (on page 189)

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Client Version

This field is used to specify the Unified CheckoutAPI version that your integration should use.

Cybersource recommends that you do no include this field in your `uc/v1/sessions` API request. When you do not include this field, Unified Checkout automatically uses the latest available version. This ensures access to the most recent enhancements and updates without requiring integration changes.

When you include this field, the value must be provided in `MAJOR.MINOR` format (for example, `1.1` or `1.2`). For information about semantic versioning, see Versioning (on page 233).

Important: This field cannot be configured through the merchant experience screens in the Business Center.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Auto-check Enrollment

You can have the Click to Pay box pre-checked when a user is manually entering their card details and Click to Pay is enabled. The customer can uncheck the box if necessary, which means the request is processed as a one-time manual PAN transaction. This is available when you set the **billingType** field to `PARTIAL` or `FULL` in the capture context. This ensures that the customer's billing country can be validated in the UI.

Click to Pay enrollment pre-check is available in these countries:

- Argentina
- Brazil
- Chile
- Colombia
- Kuwait
- Mexico
- Peru
- Qatar
- Saudi Arabia
- South Africa
- Ukraine
- United Arab Emirates

```
"paymentConfigurations": {
    "CLICKTOPAY": {
      "autoCheckEnrollment": true
    }
  }
```

Related information

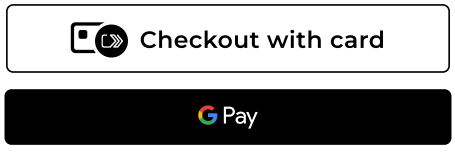
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Button Type

When Unified Checkout loads, the payment buttons displayed are based on what you include in the **allowedPaymentTypes** object in the capture context. Unified Checkout enables you to customize the text on the payment buttons. You can do this by setting the **buttonType** field object in the capture context to one of these values:

- `ADD_CARD`
- `CARD_PAYMENT`
- `CHECKOUT_AND_CONTINUE`
- `DEBIT_CREDIT`
- `DONATE`
- `PAY`
- `PAY_WITH_CARD`
- `SUBSCRIBE_WITH_CARD`

If you do not include the **buttonType** field in your request, the payment button text defaults to **Checkout with card**. For example:



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Customize Button Text

Use the **buttonType** field to customize the text on payment buttons:

Button Text Options

buttonType Value	Button Display Text
`ADD_CARD`	Add card
`CARD_PAYMENT`	Card payment
`CHECKOUT_AND_CONTINUE`	Checkout and continue
`DEBIT_CREDIT`	Debit or credit
`DONATE`	Donate
`PAY`	Pay
`PAY_WITH_CARD`	Pay with card
`SUBSCRIBE_WITH_CARD`	Subscribe with card

When you do not include this field in your request, the default button text is “Checkout with card.”

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Complete Mandate

The complete mandate feature provides service orchestration within Unified Checkout and simplifies your integration. Service orchestration enables Unified Checkout to orchestrate services on your behalf. The complete mandate feature provides instructions to the **unifiedPayment.complete()** method in the JavaScript. You must include both the **unifiedPayment.complete()** object in the Javascript and the **completeMandate** field object in your capture context to enable Unified Checkout to initiate services on your behalf from the browser.

Important: If you are updating an existing Unified Checkout configuration to use the complete mandate, you must update your JavaScript to include the **unifedPayment.complete()** function.

Important: When the **billingType** field is set to `NONE` you must include the required fields within the capture context request to ensure that the required fields are included for payment processing. For information about the fields that are required for payment services, see the Payments Developer Guide.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

completeMandate.type

This field is required to run the complete mandate and is used to indicate how a payment should be processed.

Possible values:

- `AUTH`: Authorize the payment and capture the funds at a later date.
- `CAPTURE`: Perform a sale. A sale is a combined authorization and capture in a single request.
- `PREFER_AUTH`: Perform an authorization if possible. If a payment method requires the funds to be captured immediately, then Unified Checkout captures the payment.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

completeMandate.decisionManager

This field determines whether Decision Manager is run. Set this field to `true` and include **completeMandate.type** in your request to run Decision Manager and device fingerprinting services. When Decision Manager runs, it uses the associated Decision Manager configuration based on the merchant ID that is included in the request.

When this field is set to `false` or is not included in the request, Decision Manager and device fingerprinting services do not run.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

completeMandate.consumerAuthentication

This field determines whether Payer Authentication should be used. When this field set to `3DS`, Payer Authentication runs. When this field is set to `NONE` or is not included in the request, Payer Authentication does not run.

When you use Unified Checkout with Payer Authentication, device data is collected through Payer Authentication setup and Unified Checkout completes all calls that are associated with Payer Authentication.

For information about challenge codes, see **consumerAuthenticationInformation.challengeCode** in the REST API Field Reference. Unified Checkout supports 3-D Secure data only when your payment processor supports it. For information see Visa Data Only in the *Payer Authentication Developer Guide*.

To test Payer Authentication, you must use Payer Authentication test cards. See Test Cases for 3-D Secure 2.x in the *Payer Authentication Developer Guide*.

Consumer authentication is available for these card types:

- American Express
- Cartes Bancaires
- China UnionPay
- Diners Club
- Discover
- EFTPOS
- ELO
- Jaywan
- JCB
- mada
- Maestro
- Mastercard
- Visa

Consumer authentication runs for these payment methods when they are supported in your Unified Checkout configuration:

- `PANENTRY`
- `CLICKTOPAY` when the transaction is not authenticated with Click to Pay.
- `GOOGLEPAY` when the transaction is not authenticated with Google Pay.

Unified Checkout does not attempt to authenticate for Click to Pay and Google Pay if the transaction has already been authenticated when it is received by Unified Checkout. For information about testing authentication, see Test Authentication (on page 326).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

completeMandate.tms

completeMandate.tms.tokenCreate: This field determines if a TMS token is created for the customer's selected payment method. When this field is set to `true`, a token is created. When this field is set to `false` or not included in the request, a token is not created.

Important: To make a new payment instrument or instrument identifier under an existing customer during the complete mandate, you must meet these requirements:

- You must include the customer token ID in the **paymentConfigurations** field object.
- `TMS_TOKEN` must be included in the **allowedPaymentTypes** field object.
- **tokenCreate** must be set to `true` and `paymentInstrument` and `instrumentIdentifier` must be included as values in the **tms.tokenTypes** field array. For example:

```
"tms": {
      "tokenCreate": true,
      "tokenTypes": [
        "paymentInstrument",
        "instrumentIdentifier",
      ]
    }
```

When you meet these requirements, a new payment instrument or instrument identifier is created under the specified customer token.

completeMandate.tms.tokenTypes: This is an optional field that you can use to indicate the token type for the token that is created. When this field is not included in the request, a token is created based on your TMS vault configuration. You can set this field to these values:

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

If you want Unified Checkout to capture the cardholder's consent to save the card before a request to create a token is completed, then you must set **captureMandate.requestSaveCredentials** to `true`. When this field is set to `true`, Unified Checkout presents a **Save card for future payments** checkbox within the UI and enables the cardholder to give consent. Do not include **captureMandate.requestSaveCredentials** in your request if you have already gained cardholder consent to create a TMS token or do not require consent.

This table indicates if a token is created given the requested payment method:

Payment Method	Capture Context	Result
PAN Entry and Click to Pay	**completeMandate.tms.token Create** = `true`	TMS token is created at the token level(s) specified in the request or based on the default for the token vault.
	completeMandate.tms.token Create = `true` and **captureMandate.requestSave Credentials** = `true`	Cardholder can check **Save Payment Information** in Unified Checkout. The request to create a token is made when the cardholder checks this field in the UI. When it is not checked, ni token is created.
Apple Pay, Google Pay, and Paze	**completeMandate.tms.token Create** = `true`	TMS token is created at the token level(s) specified in the request or based on the default for the token vault.
	completeMandate.tms.token Create = `true` and **captureMandate.requestSave Credentials** = `true`	Unified Checkout cannot obtain consent to create a token and no token is created when the customer completes the payment.
Echeck	**completeMandate.tms.token Create** = `true`	TMS token is created at the token level(s) specified in the request or based on the default for the token vault.
	completeMandate.tms.token Create = `true` and **captureMandate.requestSave Credentials** = `true`	Unified Checkout cannot obtain consent to create a token and no token is created when the customer completes the payment.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Capture Mandate

The capture mandate enables you to define which fields are captured within Unified Checkout. You must include the fields and set the values in the capture context based on the information that you want Unified Checkout to collect. This enables the cardholder to review and edit their details where the UI includes these fields. When the UI is used to capture cardholder information, all captured information is available within the Payment Details API response. When you want the cardholder to review existing address data, you can include the known customer data in the capture context and this information is pre-filled in the Unified Checkout UI. For information about the Payment Details API, see Payment Details API (on page 338).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.comboCard

A combo card is a single card in Brazil that functions as both a debit and a credit card. Unified Checkout enables the cardholder to choose whether to pay for a transaction using a debit or credit card. The cardholder can choose the card that they want to use when they enter their card details or when they choose a stored Visa card from their Click to Pay wallet during checkout. While in the card details section of the payment form, the cardholder is prompted for a debit or credit card. Credit is the default option.

To enable combo cards during checkout, you must include the **comboCard** field in your capture context request and set the field value to `true`. When the **comboCard** field value is set to `true`, the option to use a debit or credit card appears for all Visa cards that are entered in Unified Checkout and for all cards that are already stored in Click to Pay. If you do not want to offer a combo card at checkout, do not include the **comboCard** field in your capture context request:

```
"captureMandate" : {
      "comboCard": true
}
```

Important: This feature is available only in Brazil.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.CPF

The Cadastro de Pessoas Físicas (CPF) Brazilian tax ID feature is for customers in Brazil and provides your customers with a way to include their Consumer National Identifier when it is requested at checkout. Include this field in the capture context to display this field within the flow for manual card entry and Click to Pay transactions:

```
"captureMandate" : {
    "CPF": {
        "required": true
    }
}
```

Important: This feature is available only in Brazil.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.requestSaveCredentials

This feature enables you to display a consent option in the Unified Checkout UI for the cardholder to save their payment details for future use. If you use the complete mandate to create a token, see Sessions API (on page 280).

When you use this field without using the complete mandate, the transient token payload includes the **consumerPreference.saveCard** field with the value set to `true` when the cardholder has checked to save the payment information for future purchases:

```
"captureMandate" : {
      "requestSaveCredentials": true
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.showConfirmationStep

When **showConfirmstionStep** is set to `false`, you can remove the final summary confirmation screens from the checkout experience. When the UI displays cardholder data, the cardholder can review and, if necessary, edit their payment details before checkout is complete.

```
{
  "captureMandate": {
    "showConfirmationStep": false
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.billingType

`PARTIAL`: Only the billing postal code and billing country are collected in the UI. Set to this value when you use relaxed address verification services (AVS). This includes markets where postal code and billing country are enough for successful payment processing.

`NONE`: No fields are shown in the UI to capture cardholder billing details. If you are using the Complete Mandate, you must provide billing details in the capture context. All information that is collected from these fields is tokenized in the transient token and sent for payment processing. For information about which fields are required for payment processing, see the Payments Developer Guide.

`FULL`: These fields are shown in the UI to capture cardholder billing details. When you include the billing details in the capture context, these details are pre-filled in the Unified Checkout UI. All information that is collected from these fields are tokenized in the transient token and sent for payment processing where the Complete Mandate is used.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.requestEmail

`false`: No email address is shown in the UI. If you are using Click to Pay, this email address is used to find the cardholder's Click to Pay account and it appears in the UI when **requestEmail** is set to `false`.

`true`: The email address is shown and captured in the UI. If you are using Click to Pay, this email address is used to find the cardholder's Click to Pay account.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.requestPhone

`false`: No phone number is shown or captured in the UI.

`true`: The phone number is shown and captured in the UI.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.requestShipping

`false`: No shipping information is captured in the UI. When shipping details are required for payment processing and are used for follow on services such as Decision Manager, you can include these fields in the capture context. These details are tokenized and passed through.

`true`: Shipping fields are shown in the UI and are collected by Unified Checkout. When you include the shipping details in the capture context, the information appears prefilled in the UI.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.shipToCountries

When the **requestShipping** field is set to `true`, only the countries that are included in this field can be selected by the cardholder for their shipping address.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Include Card Prefix

You can control the length of the card number prefix to be received in the response to the capture context `/sessions` request:

- Six digits
- Eight digits
- No prefix

To specify your preferred card number prefix length, include or exclude the **transientTokenResponseOptions.includeCardPrefix** field in the capture context `/sessions` request.

To receive a six-digit card number prefix in the response, follow this step:

Do not include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context `/sessions` request.

This example shows how a six-digit card number prefix `411111` is returned in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111”,
                        "bin" : "411111"
```

To receive an eight-digit card number prefix in the response, follow this step:

Include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request, and set the value to `true`.

Important: This PCI DSS requirement applies only to card numbers longer than 15 digits and only for Discover, JCB, Mastercard, UnionPay, and Visa brands.

- If the card type entered is not part of these brands, a six-digit card number prefix is returned instead.
- If the card type entered is not part of these brands but is *co-branded* with these brands, an eight-digit card number prefix is returned.

This example shows how an eight-digit card prefix `41111102` is returned in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111”,
                        "prefix" : "41111102"
```

To not receive a card number prefix in the response, follow this step:

Include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request, and set the value to `false`.

This example shows how a card number is returned without a card number prefix in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111"
```

Best practice: If your application does not require card number prefix information for routing or identification, Cybersource recommends that you include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request and set its value to `false`. Doing so limits the exposure of payment data to only what is necessary for your processing needs.

For more information about PCI DSS, see *Frequently Asked Questions* on the PCI Security Standards Council site.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Email Autolookup

When you include Click to Pay as an **allowedPaymentType**, an automatic email lookup occurs when an email address is included in the capture context request. If the user has a Click to Pay account but is not on a recognized device, a one-time password (OTP) screen appears and the user is prompted to enter their OTP. If the user does not have a Click to Pay account, the user must enter their card information manually. They will have the option to create a Click to Pay account.

To enable email autolookup, you must include `CLICKTOPAY` as a value in the **allowedPaymentTypes** field and include an email address in the capture context.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Mobile as Identity for Click to Pay

Click to Pay supports mobile numbers as way to identify a user. This enables cardholders to use their mobile number instead of their email address in certain markets for Visa and Mastercard transactions.

When the **requestEmail** field is set to `false` and the **requestPhone** field is set to `true`, the cardholder is identified using the provided mobile number. When the **requestEmail** field is set to `true` and the **requestPhone** field is set to `false`, the cardholder is identified using the provided email address. When the **requestEmail** field is set to `true` and the **requestPhone** field is also set to `true`, the cardholder is identified using the provided email address first and then the mobile number if there is no match.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Example: Unified Checkout Complete Capture Context

Capture Context Request

```
{
  "country": "US",
  "locale": "en_GB",
  "targetOrigins": [
    "https://merchant.com",
    "https://reseller.com:8443"
  ],
```

```
  "clientVersion": "1.0",
  "allowedCardNetworks": [
    "VISA",
    "MASTERCARD",
    "AMEX",
    "JCB",
    "DISCOVER",
    "DINERSCLUB",
    "CARTESBANCAIRES",
    "EFTPOS",
    "JCREW",
    "MEEZA",
    "CUP",
    "CARNET",
    "MADA",
    "ELO",
    "MAESTRO",
    "PAYPAK",
    "JAYWAN",
    "KCP",
    "UATP"
  ],
  "allowedPaymentTypes": [
    "PANENTRY",
    "GOOGLEPAY",
    "CLICKTOPAY",
    "APPLEPAY",
    "PAZE",
    "CHECK",
    "AFTERPAY",
    "IDEAL",
    "MULTIBANCO",
    "PRZELEWY24",
    "MYBANK",
    "KONBINI",
    "DRAGONPAY",
    "BANCONTACT",
    "TINKPAYBYBANK"
  ],
  "appearance": {
    "theme": "LIGHT",
    "variables": {
      "primaryColor": "#007bff",
      "secondaryColor": "#6c757d",
      "fontFamily": "Arial, sans-serif",
      "fontSize": "14px",
      "borderRadius": "4px"
    }
  },
```

```
  "buttonType": "CHECKOUT",
  "captureMandate": {
    "showConfirmationStep": true,
    "billingType": "FULL",
    "requestEmail": true,
    "requestPhone": true,
    "requestShipping": true,
    "shipToCountries": [
      "US",
      "GB",
      "CA"
    ],
    "showAcceptedNetworkIcons": true,
    "comboCard": true,
    "requestSaveCredentials": true,
    "CPF": {
      "required": true
    }
  },
  "completeMandate": {
    "type": "CAPTURE",
    "decisionManager": true,
    "consumerAuthentication": "3DS",
    "tms": {
      "tokenCreate": true,
      "tokenTypes": [
        "customer",
        "paymentInstrument",
        "instrumentIdentifier",
        "shippingAddress"
      ]
    }
  },
  "paymentConfigurations": {
    "PANENTRY": {
      "customer": "existing_customer_token_123"
    },
    "GOOGLEPAY": {
      "allowedAuthMethods": [
        "PAN_ONLY",
        "CRYPTOGRAM_3DS"
      ]
    },
    "CLICKTOPAY": {
      "autoCheckEnrollment": true
    }
  },
  "transientTokenResponseOptions": {
    "includeCardPrefix": true
```

```
  },
  "data": {
    "orderInformation": {
      "amountDetails": {
        "totalAmount": "102.21",
        "currency": "USD",
        "surcharge": {
          "amount": "2.50"
        },
        "discountAmount": "2.00",
        "serviceFeeAmount": "5.00",
        "taxAmount": "10.00",
        "taxDetails": [
          {
            "taxId": "1234",
            "type": "N"
          },
          {
            "taxId": "5678",
            "type": "S"
          }
        ]
      },
      "billTo": {
        "address1": "123 Main Street",
        "address2": "Apt 4B",
        "address3": "Building C",
        "address4": "Floor 3",
        "administrativeArea": "CA",
        "buildingNumber": "123",
        "country": "US",
        "district": "Downtown",
        "locality": "San Francisco",
        "postalCode": "94105",
        "email": "jane.doe@visa.com",
        "firstName": "John",
        "middleName": "Michael",
        "lastName": "Doe",
        "phoneNumber": "+1-123456789",
        "phoneType": "night",
        "nameSuffix": "Mr",
        "title": "Software Engineer",
        "company": {
          "name": "Visa Inc",
          "address1": "900 Metro Center Blvd",
          "address2": "Suite 200",
          "country": "US",
          "administrativeArea": "CA",
          "postalCode": "94404",
```

```
      "locality": "Foster City"
    }
  },
  "shipTo": {
    "address1": "456 Oak Avenue",
    "address2": "Suite 100",
    "address3": "Building A",
    "address4": "Level 2",
    "administrativeArea": "NY",
    "buildingNumber": "456",
    "country": "US",
    "district": "Midtown",
    "locality": "New York",
    "postalCode": "10001",
    "firstName": "Jane",
    "lastName": "Smith"
  },
  "lineItems": [
    {
      "productCode": "WIDGET-001",
      "productName": "Premium Widget",
      "productSku": "WID-PRE-001",
      "quantity": 2,
      "unitPrice": "45.50",
      "unitOfMeasure": "EA",
      "totalAmount": "91.00",
      "taxAmount": "7.28",
      "taxRate": "0.08",
      "taxAppliedAfterDiscount": "y",
      "taxStatusIndicator": "N",
      "taxTypeCode": "1234",
      "amountIncludesTax": true,
      "typeOfSupply": "12",
      "commodityCode": "COMM-001",
      "discountAmount": "5.00",
      "discountApplied": true,
      "discountRate": "0.05",
      "invoiceNumber": "INV-2024-001",
      "taxDetails": [
        {
          "type": "STATE",
          "amount": "3.64",
          "rate": "0.04",
          "code": "1234",
          "taxId": "TAX-001",
          "applied": true,
          "exemptionCode": "1"
        },
        {
```

```
            "type": "LOCAL",
            "amount": "3.64",
            "rate": "0.04",
            "code": "5678",
            "taxId": "TAX-002",
            "applied": true,
            "exemptionCode": "2"
          }
        ],
        "fulfillmentType": "SHIP",
        "weight": "500",
        "weightIdentifier": "N",
        "weightUnit": "mg",
        "referenceDataCode": "REF-001",
        "referenceDataNumber": "REF-NUM-001",
        "unitTaxAmount": "3.64",
        "productDescription": "High-quality premium widget with extended
warranty",
        "giftCardCurrency": "USD",
        "shippingDestinationTypes": "residential",
        "gift": false,
        "passenger": {
          "type": "ADT",
          "status": "confirmed",
          "phone": "+1-123456789",
          "firstName": "Robert",
          "lastName": "Johnson",
          "id": "PASS-001",
          "email": "jane.doe@visa.com",
          "nationality": "US"
        }
      },
      {
        "productCode": "GADGET-002",
        "productName": "Digital Gadget",
        "productSku": "GAD-DIG-002",
        "quantity": 1,
        "unitPrice": "29.99",
        "unitOfMeasure": "EA",
        "totalAmount": "29.99",
        "taxAmount": "2.40",
        "taxRate": "0.08",
        "taxAppliedAfterDiscount": "n",
        "taxStatusIndicator": "Y",
        "taxTypeCode": "5678",
        "amountIncludesTax": false,
        "typeOfSupply": "11",
        "commodityCode": "COMM-002",
        "discountAmount": "3.00",
```

```
        "discountApplied": true,
        "discountRate": "0.10",
        "invoiceNumber": "INV-2024-002",
        "taxDetails": [
          {
            "type": "FEDERAL",
            "amount": "2.40",
            "rate": "0.08",
            "code": "9012",
            "taxId": "TAX-003",
            "applied": true,
            "exemptionCode": "0"
          }
        ],
        "fulfillmentType": "DIGITAL",
        "weight": "0",
        "weightIdentifier": "Y",
        "weightUnit": "g",
        "referenceDataCode": "REF-002",
        "referenceDataNumber": "REF-NUM-002",
        "unitTaxAmount": "2.40",
        "productDescription": "Advanced digital gadget with cloud sync",
        "giftCardCurrency": "EUR",
        "shippingDestinationTypes": "commercial",
        "gift": true,
        "passenger": {
          "type": "CHD",
          "status": "pending",
          "phone": "+1-123456789",
          "firstName": "Emily",
          "lastName": "Williams",
          "id": "PASS-002",
          "email": "jane.doe@visa.com",
          "nationality": "GB"
        }
      }
    ],
    "invoiceDetails": {
      "invoiceNumber": "INV-MAIN-2024-001",
      "productDescription": "Multiple items including widgets and gadgets"
    }
  },
  "buyerInformation": {
    "personalIdentification": [
      {
        "type": "CPF",
        "id": "01234567890"
      }
    ],
```

```
    "merchantCustomerId": "CUST-12345",
    "companyTaxId": "123456789",
    "dateOfBirth": "19901215",
    "language": "en"
  },
  "clientReferenceInformation": {
    "code": "TAGX001",
    "partner": {
      "developerId": "DEV-1234",
      "solutionId": "SOL-4567"
    }
  },
  "consumerAuthenticationInformation": {
    "challengeCode": "01",
    "messageCategory": "01",
    "acsWindowSize": "01"
  },
  "merchantInformation": {
    "merchantDescriptor": {
      "name": "Jane Sales",
      "alternateName": "BIG SALES INC",
      "locality": "New York",
      "phone": "+1-123456789",
      "country": "US",
      "postalCode": "170056",
      "administrativeArea": "NY",
      "address1": "123 47TH STREET"
    }
  },
  "processingInformation": {
    "reconciliationId": "01234567",
    "authorizationOptions": {
      "aftIndicator": true,
      "authIndicator": "Y",
      "ignoreCvResult": true,
      "ignoreAvsResult": true,
      "initiator": {
        "credentialStoredOnFile": true,
        "merchantInitiatedTransaction": {
          "reason": "1"
        }
      }
    },
    "businessApplicationId": "AA",
    "commerceIndicator": "recurring",
    "processingInstruction": "ORDER_SAVED_EXPLICITLY"
  },
  "recipientInformation": {
    "firstName": "John",
```

```
      "middleName": "A",
      "lastName": "Buyer",
      "country": "GB",
      "accountId": "acc0123567",
      "administrativeArea": "GB",
      "accountType": "01",
      "dateOfBirth": "19901215",
      "postalCode": "170056"
    },
    "merchantDefinedInformation": [
      {
        "key": "promo_code",
        "value": "DISCOUNT20"
      },
      {
        "key": "customer_tier",
        "value": "gold"
      }
    ],
    "deviceInformation": {
      "ipAddress": "192.168.1.100"
    },
    "paymentInformation": {
      "card": {
        "typeSelectionIndicator": "0"
      }
    }
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Validating the Capture Context

The capture context that you generate is a JSON Web Token (JWT) data object. The JWT is digitally signed using a public key and confirms the validity of the JWT and that it comes from Cybersource. When you do not have a key in the JWT header, Cybersource recommends that you follow cryptography best practices and validate the capture context signature.

To validate a JWT, you must obtain its public key. This public RSA key is in JSON Web Key (JWK) format. The 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, you must retrieve the key ID associated with the public key and then pass the key ID to the `/flex/v2/public-keys` endpoint:

1. From the header of the capture context, get the key ID (**kid**):

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

2. Send a GET request to the `/flex/v2/public-keys` endpoint and include the key ID. For example:

   - **Test:** `GET https://apitest.cybersource.com/flex/v2/public-keys/{3g}`
   - **Production:** `GET https://api.cybersource.com/flex/v2/public-keys/{3g}`
   - **Production in Saudi Arabia:** `GET https://api.sa.cybersource.com/flex/v2/public-keys/{3g}`
   - **Test in Saudi Arabia:** `GET https://apitest.sa.cybersource.com/flex/v2/public-keys/{3g}`

   Depending on the cryptographic method you use to validate the public key, you might need to convert the key to privacy-enhanced mail (PEM) format.

3. The resource returns the public key:

```
eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsImRhdGEiOiI2bUFLNTNPNVpGTUk5Y3Rob
WZmd2doQUFFRGNqNU5QYzcxelErbm8reDN6WStLOTVWQ2c5bThmQWs4czlTRXBtT21zMmVhbEx5Nk
hHZ29oQ0JEWjVlN3ZUSGQ5YTR5a2tNRDlNVHhqK3ZoWXVDUmRDaDhVY1dwVUNZWlZnbTE1UXVFMkE
iLCJvcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbSIsImp3ayI6eyJrdHki
OiJSU0EiLCJlIjoiQVFBQiIsInVzZSI6ImVuYyIsIm4iOiJyQmZwdDRjeGlkcVZwT0pmVTlJQXcwU
1JCNUZqN0xMZjA4U0R0VmNyUjlaajA2bEYwTVc1aUpZb3F6R3ROdnBIMnFZbFN6LVRsSDdybVNTUE
ZIeTFJQ3BfZ0I3eURjQnJ0RWNEanpLeVNZSTVCVjNsNHh6Qk5CNzRJdnB2Smtqcnd3QVZvVU4wM1R
aT3FVc0pfSy1jT0xpYzVXV0ZhQTEyOUthWFZrZFd3N3c3LVBLdnMwNmpjeGwyV05STUIzTS1ZQ0xO
b3FCdkdCSk5oYy1uM1lBNU5hazB2NDdiYUswYWdHQXRfWEZ0ZGItZkphVUVUTW5WdW9fQmRhVm90d
1NqUFNaOHFMOGkzWUdmemp2MURDTUM2WURZRzlmX0tqNzJjTi1OaG9BRURWUlZyTUtiZ3QyRDlwWk
J1d2gzZlNfS3VRclFWTVdPelRnT3AzT2s3UVFGZ1EiLCJraWQiOiIwOEJhWXMxbjdKTUhjSDh1bkc
xc1NDUVdxN2VveWQ1ZyJ9fSwiY3R4IjpbeyJkYXRhIjp7InRhcmdldE9yaWdpbnMiOlsiaHR0cHM6
Ly93d3cudGVzdC5jb20iXSwibWZPcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlL
mNvbSJ9LCJ0eXBlIjoibWYtMC4xMS4wIn1dLCJpc3MiOiJGbGV4IEFQSSIsImV4cCI6MTYxNjc3OT
A5MSwiaWF0IjoxNjE2Nzc4MTkxLCJqdGkiOiJ6SG1tZ25uaTVoN3ptdGY0In0.GvBzyw6JKl3b2Pz
```

```
tHb9rZXawx2T817nYqu6goxpe4PsjqBY1qeTo19R-CP_DkJXov9hdJZgdlzlNmRY6yoiziSZnGJdp
nZ-pCqIlC06qrpJVEDob3O_efR9L03Gz7F5JlLOiTXSj6nVwC5mRlcP032ytPDEx5TMI9Y0hmBadJ
YnhEMwQnn_paMm3wLh2v6rfTkaBqd8n6rPvCNrWMOwoMdoTeFxku-
```

Use this public RSA key to validate the capture context.

4. Parse the JWT capture context to get the **kid** from its header:

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

5. Send a GET request to retrieve the 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"
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Session Validation

The session JWT is digitally signed using RS256. You must confirm that it was issued by Cybersource and has not been tampered with. Follow these steps to validate the signature:

1. Parse the session JWT header to extract the key ID (`kid`):

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

2. Retrieve the public key by sending a request to the `/flex/v2/public-keys/{kid}` endpoint:
   - **Test**: GET `apitest.cybersource.comflex/v2/public-keys/{kid}`
   - **Production**: GET `api.cybersource.comflex/v2/public-keys/{kid}`
3. Use the returned RSA public key in JSON Web Key format to verify the JWT signature.

> **Important:** Depending on the cryptographic library that tou use, you may need to convert the key to Privacy-Enhanced Mail (PEM) format.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Transient Tokens

The response to a successful customer interaction with Unified Checkout is a transient token. This is returned in the response from the **checkout.mount()** function. The transient token is a reference to the payment data collected on your behalf. Transient tokens allow secure card payments to occur without risk of exposure to sensitive payment information. The transient token is a short-term token that expires after 15 minutes. This reduces your PCI burden/responsibility and ensures that sensitive information is not exposed to your back-end systems.

Transient tokens can be included requests sent to the Payment Details API for the customer payment data that is collected.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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 362).

The payload portion of the token is a Base64URL-encoded JSON string and contains various claims. For more information, see JSON Web Tokens (on page 362).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Example: Transient Token Format

Transient Token Payload

Important: The empty field values in the transient token indicate which fields were captured by the application without exposing you to personally identifiable information directly.


```
{
  "metadata" : {
    "sequenceNumber" : "1",
    "cardholderAuthenticationStatus" : false,
    "paymentType" : "PANENTRY"
  },
  "iss" : "Flex/00",
  "exp" : 1762870464,
  "type" : "gda-0.10.0",
  "iat" : 1762869564,
  "jti" : "1D4Q8FJSSZ9ASKQ9ZCJ7E13IFOITOOH2GGHY6TRZ3O28TUQ1BN8H691344C098CA",
  "content" : {
    "deviceInformation" : {
      "fingerprintSessionId" : { }
    },
    "orderInformation" : {
      "billTo" : {
        "country" : { },
        "lastName" : { },
        "firstName" : { },
        "phoneNumber" : { },
        "address1" : { },
        "postalCode" : { },
        "locality" : { },
        "buildingNumber" : { },
        "company" : {
          "name" : { }
        },
        "administrativeArea" : { },
        "email" : { }
      },
      "amountDetails" : {
        "totalAmount" : { },
        "currency" : { }
      },
      "shipTo" : {
        "firstName" : { },
```

```
        "lastName" : { },
        "country" : { },
        "address1" : { },
        "postalCode" : { },
        "locality" : { },
        "buildingNumber" : { },
        "administrativeArea" : { }
      }
    },
    "paymentInformation" : {
      "card" : {
        "expirationYear" : {
          "value" : "2027"
        },
        "number" : {
          "maskedValue" : "XXXXXXXXXXXX1111",
          "bin" : "411111"
        },
        "securityCode" : { },
        "expirationMonth" : {
          "value" : "03"
        },
        "typeSelectionIndicator" : {
          "value" : "1"
        },
        "type" : {
          "value" : "001"
        }
      }
    }
  }
}
```

PAN BIN in `metadata` Object

The `cardDetails` object, including the PAN BIN, is included in the transient token `metadata` when a Click to Pay network token is used as a payment method. This allows you to display information about the card on invoices and see the BIN details that are linked to the underlying card.

```
"metadata": {
  "cardDetails": {
    "suffix": "9876",
    "prefix": "123456",
    "expirationMonth": "MM",
    "expirationYear": "YYYY"
  }
}
```

Authentication Status in **metadata** Object

The `cardholderAuthenticationStatus` object is included in the `metadata` and enables you to determine if the payload is fully authenticated. When `cardholderAuthenticationStatus` is set to `true`, the payload is fully authenticated. When `cardholderAuthenticationStatus` is set to `false`, the transaction is not authenticated.

If you are using Unified Checkout with **unifiedPayment.complete()** and **consumerAuthentication** is set to `true` in the complete mandate request, then Payer Authentication is called automatically if it is available for the selected payment method and card network. If you use a transient token to request follow-on services directly, the value of this field indicates if the transaction has been authenticated.

```
"metadata": {
  "cardholderAuthenticationStatus": "true"
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Dual-Branded Cards

Unified Checkout accepts dual-branded cards. To use this feature, you must include the card networks that have overlapping BIN ranges in the capture context request. For example:

```
"allowedCardNetworks": ["VISA", "MASTERCARD", "AMEX", "CARTESBANCAIRES"]
```

When a card number within an overlapping BIN range is entered, the network that is listed first in the value array for the **allowedCardNetworks** field is used. Based on the previous example, if the card number 403550XXXXXXXXXX is entered, the payment network for payment processing is Visa.

During the transaction, the card type is populated with the first network in the list, and the **detectedCardTypes** field returned in the transient token includes all of the detected card types in the transient token.

The **detectedCardTypes** field is returned in the transient token response only when more than one card type is detected.

If you include Cartes Bancaires as a supported dual-branded card type, Unified Checkout displays a radio button with Visa and Mastercard options at checkout. This enables the customer to select which payment scheme they want to use to process the payment. The radio button defaults to the card type that you specify in the capture context request, but the payment is processed using the option selected by the customer during checkout.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Authorizations with a Transient Token

This section provides the information required in order to perform a successful authorization with a Unified Checkout transient token. You can use this method to construct more complex payment scenarios that are not supported by the `unifiedPayments.complete()` payment method.

Important: When you process payments through Unified Checkout using **unifiedPayments.complete()**, Unified Checkout invokes service orchestration directly. When you send an authorization request using a transient token, you must request the follow-on services that you want to use. For information about the required fields for the payment services that you request, see the Payments Developer Guide.

The transient token is a short-term token that expires after 15 minutes. Doing so eliminates the need to send sensitive payment data along with the request. For more information on transient tokens, see Transient Tokens (on page 313).

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

This example shows a transient token in the context of an authorization request:

```
"tokenInformation": {
    "transientTokenJwt":
 "eyJraWQiOiIwOG4zUnVsRTJGQXJDRktycVRkZFlkWGZSWFhMNXFoNSIsImFsZyI6IlJTMjU2In0.eyJp
c3MiOiJGbGV4LzA3IiwiZXhwIjoxNTk3MDg0ODk3LCJ0eXBlIjoiZ2RhLTAuMS4xIiwiaWF0IjoxNTk3MD
gzOTk3LCJqdGkiOiIxQzI2VlpSkVJUU1PTzVIMDUwNEtINDdJMEFNMklaRkM0M1Y1TDU0MUhCTE45Q09JM
0w3NUYzMTk0RTE5NkExIn0.SNm1VZaZr3DkTqUg9CdV0F5arRe-uQU9oUWPKfWIpbIzIPZutRokv5DSDcM
7asZIKNJyNIBx5DLsl_yQPrKgzhwQxZ8qbhto7cu3t-v8DHG2yO951plPQVQnj7x-vEDcXkLUL1F8sqY23
R5HW-xSDAQ3AFLawCckn7Q2eudRGeuMhLWH742Gflf9Hz3KyKnmeNKA3o9yW2na16nmeVZaYGqbUSPVITd
l5cMA0o9lEob8E3OQH0HHdmIsu5uMA4x7DeBjfTKD1rQxFP3JBNVcv30AIMLkNcw0pHbtHDVzKBWxUVxvn
m3zFEdiBuSAco2uWhC9zFqHrrp64ZvzxZqoGA"
}
```

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 338).

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`

Production in Saudi Arabia: `POST https://api.sa.cybersource.com/pts/v2/payments`

Test in Saudi Arabia: `POST https://apitest.sa.cybersource.com/pts/v2/payments`

Required Field for an Authorization with a Transient Token

tokenInformation.transientTokenJwt

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

REST Example: Requesting an Authorization with a Transient Token

```
{
  "clientReferenceInformation": {
    "code": "TC50171_3"
  },
  "processingInformation": {
    "commerceIndicator": "internet"
  },
  "tokenInformation": {
    "transientTokenJwt":
 "eyJraWQiOiIwOG4zUnVsRTJGQXJDRktycVRkZFlkWGZSWFhMNXFoNSIsImFs
ZyI6IlJTMjU2In0.eyJpc3MiOiJGbGV4LzA3IiwiZXhwIjoxNTk3MDg0ODk3LCJ0eXBlIjoiZ2RhLTAuMS
4xIi
wiaWF0IjoxNTk3MDgzOTk3LCJqdGkiOiIxQzI2VlpSRkVJUU1PTzVIMDUwNEtINDdJMEFNMklaRkM0M1Y1
TDU0
MUhCTE45Q09JM0w3NUYzMTk0RTE5NkExIn0.SNm1VZaZr3DkTqUg9CdV0F5arRe-uQU9oUWPKfWIpbIzIP
ZutR
okv5DSDcM7asZIKNJyNIBx5DLsl_yQPrKgzhwQxZ8qbhto7cu3t-v8DHG2yO951plPQVQnj7x-vEDcXkLU
L1F8
sqY23R5HW-xSDAQ3AFLawCckn7Q2eudRGeuMhLWH742Gflf9Hz3KyKnmeNKA3o9yW2na16nmeVZaYGqbUS
PVIT
dl5cMA0o9lEob8E3OQH0HHdmIsu5uMA4x7DeBjfTKD1rQxFP3JBNVcv30AIMLkNcw0pHbtHDVzKBWxUVxv
nm3z
FEdiBuSAco2uWhC9zFqHrrp64ZvzxZqoGA"
  },
```

```
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "21.00",
      "currency": "USD"
    },
    "billTo": {
      "firstName": "John",
      "lastName": "Doe",
      "address1": "1Market St",
      "address2": "Address 2",
      "locality": "san francisco",
      "administrativeArea": "CA",
      "postalCode": "94105",
      "country": "US",
      "email": "test@cybs.com",
      "phoneNumber": "4158880000"
    }
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Test Your Configuration

Use these test payment credentials to test your Unified Checkout configuration:

- Unified Checkout Test Cards (on page 321)
- Visa and Mastercard Click to Pay Test Cards (on page 323)
- Test Cards for Authentication by Click to Pay (on page 325)
- Echeck Test Values (on page 326)

You can handle errors and test your configuration using these topics:

- Handle Errors (on page 329)
- Reason Codes (on page 335)

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Unified Checkout Test Cards

Use these test card numbers to test your Unified Checkout configuration.

Combine the BIN with the card number when sending to Unified Checkout.

To test Payer Authentication, you must use Payer Authentication test cards. See Test Cases for 3-D Secure 2.x in the *Payer Authentication Developer Guide*.

Test Card Numbers

Card Brand	BIN	Card Number	Expiration Date	CVV
Visa	411111	1111111111	12/2026	123
Mastercard	555555	5555554444	02/2026	265
American Express	378282	246310005	03/2026	7890

Test Card Numbers (continued)

Card Brand	BIN	Card Number	Expiration Date	CVV
Cartes Bancaires	436000	0001000005	04/2040	123
Carnet	506221	0000000009	04/2026	123
China UnionPay	627988	6248094966	04/2040	123
Diners Club	305693	09025904	04/2040	123
Discover	644564	4564456445	04/2040	123
JCB	353011	13333 0000	04/2040	123
Jaywan	679009	0000002009	04/2040	123
Jaywan	669000	0000000000	04/2040	123
KCP	949022	0011669217	04/2040	
Paypak	220543	0000003002	04/2040	123
Maestro	675964	9826438453	04/2040	123
mada	446404	0000000007	04/2040	123
ELO	451416	0000000003	04/2040	123
JCrew	515997	1500000005	04/2040	123
EFTPOS	401795	000000000009	04/2040	123
Meeza	507808	3000000002	04/2040	123
UATP	148512	345678905	04/2040	

Test Cards for Click to Pay Authentication by Unified Checkout

se these test cards to test when you use a Click to Pay card with authentication performed outside Click to Pay and the **consumerAuthentication** field is set to `true` in the capture context.

Replace the X in the card number with 4.

To manage Visa test cards for customer authentication, contact your implementation consultant or technical account manager.

Important: These test cards are not valid for testing in production. To test in production, you must leverage production credentials.

Test Card Numbers for Authentication Outside Click to Pay Flow

Card Brand	Card Number	Expiration Date	CVV
Visa	46229431231X2X56	12/2026	432
	46229431231X232X	12/2026	581
Mastercard	512X35X1XXX64578	Any future date	Any
	512X35X1XXX64552	Any future date	Any

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Visa and Mastercard Click to Pay Test Cards

Visa Test Cards

These Visa test cards can be added to your Click to Pay wallet.

Replace the X in the card number with 4.

You can manage your Visa Click to Pay test cards and account here:

- **Production:** https://src.visa.com/login
- **Test:** https://sandbox.src.visa.com/login

To manage Visa test cards for customer authentication, contact your implementation consultant or technical account manager.

Important: These test cards are not valid for testing in production. To test in production, you must leverage production credentials.

Visa Test Card Numbers

Card Number	Expiration Date	CVV
x6229x3123123755	12/2029	728
x6229x3123123763	12/2029	605
x6229x3123123771	12/2029	694
x6229x3123123789	12/2029	881
x6229x3123123797	12/2029	678
x6229x3123123805	12/2029	084
x6229x3123123813	12/2029	127
x6229x3123123821	12/2029	218
x6229x3123123839	12/2029	114
x6229x31231238x7	12/2029	867
x6229x312312385x	12/2029	301

Mastercard Test Cards

Mastercard test cards can be added to your Click to Pay wallet. You must retrieve Mastercard test cards from their Click to Pay test page: https://developer.mastercard.com/mastercard-checkout-solutions/documentation/testing/test_cases/click_to_pay_case/#test-cards

Mastercard has different test cards for retrieving tokenized and non-tokenized data. Cybersource recommends that you use these test cards as follows:

- Test cards to retrieve PAN data: Use these cards when the customer is completing checkout as a one-time guest and does not have a Click to Pay account or want to create one.
- Test cards to retrieve token data: Use these cards for tokenized Click to Pay transactions.

You can manage your Mastercard Click to Pay test cards and account here:

- **Live:** https://src.mastercard.com/profile/enroll
- **SBX:** https://sandbox.src.mastercard.com/profile/enroll

Mastercard authentication test cards are available on the *Mastercard Checkout Solutions* page in the Mastercard Developer Center.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Test Cards for Authentication by Click to Pay

Use these cards when authentication is performed by Click to Pay within the Click to Pay flow.

Replace the X in the card number with 4.

To manage Visa test cards for customer authentication, contact your implementation consultant or technical account manager.

Important: These test cards are not valid for testing in production. To test in production, you must leverage production credentials.

Click to Pay Test Card Numbers for Authentication in Click to Pay Flow

Card Brand	Card Number	Expiration Date	CVV
Visa	43958XXX0449X11X	12/2025	509
	439584XXX282X11X	12/2025	693
	439584XX91X1XX11	12/2025	676
	439584XX9119XX11	12/2025	789

For information about testing authentication, seeTest Authentication (on page 326). For information about enabling Visa customer authentication, see Set Up Customer Authentication for Visa Click to Pay (on page 171).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Echeck Test Values

These eCheck test values can be used to process a test eCheck transactions:

- **Routing number:** Set to 071923284
- **Account number:** Set to any supported value. For example, 1234567890.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Test Authentication

Use this table to determine how to test your authentication method.

Authentication Testing by Product

Payment Method	Authentication	Minimum Follow-On Actions	Prerequisites	Test Cards	Details
PAN Entry	Payer Authentication through Unified Checkout	Authorization and Payer Authentication	The transacting MID must be enabled for Payer Authentication and the complete mandate is used with the **consumerAut**	See Testing Payer Authentication in the *Payer Authentication Developer Guide.*	When the complete mandate is not used, Unified Checkout does not initiate authentication and you must perform authentication within your own environment.

Authentication Testing by Product (continued)

Payment Method	Authentication	Minimum Follow-On Actions	Prerequisites	Test Cards	Details
			hentication field set to `true`.		
Click to Pay	Payer Authentication through Unified Checkout	Authorization and Payer Authentication	The transacting MID must be enabled for Payer Authentication and the complete mandate is used with the **consumerAuthentication** field set to `true`. Authentication for Click to Pay must not be configured.	See Unified Checkout Test Cards (on page 321).	When authentication is not enabled for Click to Pay or Click to Pay is not able to perform authentication for Click to Pay, Unified Checkout performs authentication using Payer Authentication when the complete mandate is used with the **consumerAuthentication** field set to `true`.
Click to Pay	Visa Click to Pay	Authorization and Payer Authentication	You must configure the authentication for Click to Pay. Click to Pay performs authentication only if it is a tokenized Visa card.	See Test Cards for Authentication by Click to Pay (on page 325).	When authentication is enabled for Click to Pay, authentication is attempted for all Click to Pay transactions for Visa cards that are stored in Click to Pay. For information about setting up authentication

Authentication Testing by Product (continued)

Payment Method	Authentication	Minimum Follow-On Actions	Prerequisites	Test Cards	Details
					for Visa Click to Pay, see Set Up Customer Authentication for Visa Click to Pay (on page 171).
Google Pay	Google Pay	Authorization	A Google device must be used with biometric authentication for Google authentication.		A user authenticates themselves on a Google device with a tokenized Google Pay credential – the returned payload from Google will be Authenticated
Google Pay	Payer Authentication through Unified Checkout	Authorization and Payer Authentication	You must use a device, such as a web browser, that does not authenticate the cardholder as part of the authorization process.		Google will return an un-authenticated payload to Unified Checkout . Unified Checkout will step in and process Authentication via Payer Authentication when the Complete Mandate function is used with consumerAuthentication

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Handle Errors

The Unified Checkout SDK uses a structured error object for all error scenarios. Errors are returned as exceptions from asynchronous methods and are also returned as events for centralized handling.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

UnifiedCheckoutError

All SDK errors are instances of `UnifiedCheckoutError` with these properties:

`UnifiedCheckoutError` Properties

Property	Type	Description
`correlationId`	`string?`	The correlation ID from an underlying API call, when applicable.
`details`	`unknown?`	Additional error-specific information. This is often an array of objects.
`informationLink`	`string?`	The URL linked to the online documentation for this error.
`message`	`string`	This property is a human-readable description of the error.
`name`	`string`	The value is always `"UnifiedCheckoutError"`.
`reason`	`string`	This property is a machine-readable error code, such as `"CAPTURE_CONTEXT_INVALID"`.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Detect Errors

Errors may be serialized through `postMessage`. Cybersource recommends that you use the `name` property instead of `instanceof`:

```
try {
  const result = await checkout.mount('#buttons');
} catch (error) {
  if (error.name === 'UnifiedCheckoutError') {
    // Access error.reason, error.message, error.details
  }
}
```

You can also write a helper function that can be reused:

```
function isUnifiedCheckoutError(obj) {
  return (
    obj !== null &&
    typeof obj === 'object' &&
    obj.name === 'UnifiedCheckoutError' &&
    typeof obj.reason === 'string' &&
    typeof obj.message === 'string'
  );
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Error Handling Patterns

Try/Catch

```
try {
  const client = await VAS.UnifiedCheckout(sessionJWT);
  const checkout = await client.createCheckout();
  const result = await checkout.mount('#buttons');
} catch (error) {
  console.error(error.reason, error.message);
}
```

Promise .catch()

```
VAS.UnifiedCheckout(sessionJWT)
  .then(client => client.createCheckout())
  .then(checkout => checkout.mount('#buttons'))
  .catch(error => console.error(error.reason, error.message));
```

Centralized Error Logging via Events

Errors from all integrations are applicable up to the client level. Use `client.on('error')` for centralized logging:

```
const client = await VAS.UnifiedCheckout(sessionJWT);

client.on('error', (err) => {
  errorReporter.send({
    source: err.source,   // "checkout", "trigger", "button", or "client"
    code: err.code,
    message: err.message
  });
});
```

Cybersource recommends that you do this as it catches errors from all checkouts, triggers, and buttons created from this client instance.

Related information

Getting Started with REST

Response Codes

API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Error Codes

Initialization Errors

These errors are returned during `VAS.UnifiedCheckout(sessionJWT)`:

Initialization Reason Values

Reason	Description
`CAPTURE_CONTEXT_EXPIRED`	The supplied JWT has expired. Generate a new session.
`CAPTURE_CONTEXT_INVALID`	The session JWT is not valid. For example, it has a bad signature or is malformed.
`UNUSED_TARGET_ORIGINS`	One or more `targetOrigins` in the session do not match the current page origin. The `details` array lists the unused origins.

Mount Errors

These errors are returned during `checkout.mount()` or `trigger.mount()`:

Mount Reason Values

Reason	Description
`CHECKOUT_ALREADY_MOUNTED`	The checkout or trigger is already mounted. Call `unmount()` first, or create a new instance.
`MOUNT_CONTAINER_SELECTOR`	The CSS selector does not match any Document Object Model (DOM) element. Check that the container exists before calling `mount()`.
`MOUNT_ERROR`	A problem occurred loading the payment iframe.
`MOUNT_INVALID_CONTAINER`	The supplied container parameter is not a valid CSS selector string or `HTMLElement`.
`MOUNT_PAYMENT_TIMEOUT`	A payment method timed out during initialization.
`MOUNT_PAYMENT_UNAVAILABLE`	No payment types could be presented to the customer. This may be due to browser or device support, or errors during checkout initialization.

Mount Reason Values (continued)

Reason	Description
`MOUNT_SIDEBAR_OPTIONS`	The supplied container parameter is invalid for sidebar mode.
`MOUNT_TOKEN_TIMEOUT`	Token creation timed out during mount. This may indicate a network issue.
`MOUNT_TOKEN_XHR_ERROR`	A network error occurred during token creation. Check the customer's connectivity.

Complete Errors

These errors are returned during `checkout.complete()` or `trigger.complete()`:

Complete Reason Values

Reason	Description
`COMPLETE_AUTHENTICATION_CANCELED`	The customer cancelled the 3-D Secure authentication step-up.
`COMPLETE_AUTHENTICATION_FAILED`	The 3-D Secure authentication step-up failed.
`COMPLETE_ERROR`	A general error occurred during transaction completion.
`COMPLETE_IN_PROGRESS`	**complete()** has already been called and has not yet finished. Wait for the current call to resolve.
`COMPLETE_NOT_ALLOWED`	Complete is not allowed for this transaction, such as when `autoProcessing` is set to `true`.
`COMPLETE_TRANSACTION_CANCELLED`	The customer cancelled the transaction.
`COMPLETE_TRANSACTION_FAILED`	The transaction failed during processing.
`COMPLETE_VALIDATION_ERROR`	The parameters supplied to **complete()** have a validation error. Check the `details` for specifics

Checkout Errors

Checkout Reason Values

Reason	Description
`CHECKOUT_ERROR`	A general checkout error occurred.

Checkout Reason Values (continued)

Reason	Description
`CHECKOUT_PAYMENT_PARAMETERS`	One or more payment parameters have a validation error.
`CHECKOUT_VALIDATION_PARAMS`	One or more checkout parameters have a validation error.

Trigger Errors

Trigger Reason Values

Reason	Description
`TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED`	The specified payment type cannot be used with a trigger. Only `PANENTRY` and `CLICKTOPAY` values are supported.

Payment-Specific Errors

Payment-Specific Reason Values

Reason	Description
`CLICK_TO_PAY_SDK_LOAD_ERROR`	The Click to PaySDK failed to load.
`ENCRYPT_CARD_FOR_SRC_ENROLMENT_ERROR`	Card encryption for Click to Pay enrollment failed.
`GOOGLEPAY_CHECKOUT_ERROR`	A Google Pay checkout error occurred.
`LAUNCH_SRC_CHECKOUT_ERROR`	Launching the Click to Pay checkout failed.
`TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED`	The payment type is not supported for triggers.

General Errors

Reason Code	Description
`UNKNOWN_ERROR`	An unknown error has occurred.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Reason Codes

This section describes the server-side HTTP status codes and reason values that are returned when you send requests to Unified Checkout. For information about client-side SDK error codes, see Handle Errors (on page 329).

HTTP Status Codes

Code	Description
`200`	Request processed successfully
`201`	Session created
`400`	Bad request. The response body contains a reason value with details
`404`	Resource not found
`500`	Unexpected server error

Reason Values (HTTP `400`)

When the API returns a status code value of `400`, the response body includes the **reason** field. This section lists all possible values for the **reason** field:

Initialization Reason Values

Reason	Description
`CAPTURE_CONTEXT_EXPIRED`	The session JWT is expired. Generate a new session.
`CAPTURE_CONTEXT_INVALID`	The session JWT is not valid. For example, it has a bad signature or is malformed.
`INVALID_APIKEY`	The API key is not valid.

Checkout Reason Values

Reason	Description
`CHECKOUT_ERROR`	A general checkout error occurred.

Checkout Reason Values (continued)

Reason	Description
`UNIFIED_PAYMENTS_ALREADY_SHOWN`	The checkout is already displayed.
`UNIFIED_PAYMENTS_PAYMENT_PARAMETERS`	One or more payment parameters have a validation error.
`UNIFIED_PAYMENTS_VALIDATION_FIELDS`	One or more fields have a validation error.
`UNIFIED_PAYMENTS_VALIDATION_PARAMS`	One or more checkout parameters have a validation error.

Mount Reason Values

Reason	Description
`SHOW_LOAD_CONTAINER_SELECTOR`	The CSS selector does not match any element.
`SHOW_LOAD_ERROR`	A problem occurred loading the payment iframe.
`SHOW_LOAD_INVALID_CONTAINER`	The container parameter is not valid.
`SHOW_LOAD_SIDEBAR_OPTIONS`	The container parameter is not valid for sidebar mode.
`SHOW_PAYMENT_TIMEOUT`	A payment method timed out during initialization.
`SHOW_PAYMENT_UNAVAILABLE`	No payment types could be presented to the customer.
`SHOW_TOKEN_TIMEOUT`	Token creation timed out during mount.
`SHOW_TOKEN_XHR_ERROR`	A network error occurred during token creation.

Complete Reason Values

Reason	Description
`COMPLETE_AUTHENTICATION_CANCELLED`	The customer cancelled 3-D Secure authentication.
`COMPLETE_AUTHENTICATION_FAILED`	3-D Secure authentication failed.
`COMPLETE_ERROR`	A general error occurred during completion.
`COMPLETE_IN_PROGRESS`	A complete call is already in progress.
`COMPLETE_NOT_ALLOWED`	Complete is not allowed, such as when auto-processing is enabled.
`COMPLETE_TRANSACTION_CANCELLED`	The customer cancelled the transaction.
`COMPLETE_TRANSACTION_FAILED`	The transaction failed during processing.
`COMPLETE_VALIDATION_ERROR`	Parameters supplied to complete have a validation error.

Tokenization Reason Values

Reason	Description
`CREATE_TOKEN_TIMEOUT`	Token creation request timed out.
`CREATE_TOKEN_XHR_ERROR`	A network error occurred during token creation.
`SDK_XHR_ERROR`	A general SDK network error occurred.
`TOKENIZATION_ERROR`	Tokenization of payment data failed.

Payment-Specific Reason Values

Reason	Description
`CLICK_TO_PAY_SDK_LOAD_ERROR`	The Click to PaySDK failed to load.
`ENCRYPT_CARD_FOR_SRC_ENROLMENT_ERROR`	Card encryption for Click to Pay enrollment failed.
`GOOGLEPAY_CHECKOUT_ERROR`	A Google Pay checkout error occurred.
`LAUNCH_SRC_CHECKOUT_ERROR`	Launching the Click to Pay checkout failed.
`TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED`	The payment type is not supported for triggers.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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, and they are described in the *Getting Started with REST Developer Guide*:

- Set Up a JSON Web Token Message
- Set Up HTTP Signature Message

Important:

Cybersource recommends that you dynamically parse the response for the fields that you are looking for when you integrate with Cybersource APIs. Cybersource may add additional fields in the future.

You must ensure that your integration can handle new fields that are returned in the response. Even though the underlying data structures do 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-details/{jti}`

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

Production in Saudi Arabia: `GET https://api.sa.cybersource.com/flex/v2/payment-details/{jti}`

Test in Saudi Arabia: `GET https://apitest.sa.cybersource.com/flex/v2/payment-details/{jti}`

The `{jti}` is the ID of the JWT within the transient token that is returned by Unified Checkout. The transient token is a JWT object that you retrieved as part of a successful capture of payment information from a cardholder.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

REST Example: Retrieving Transient Token Payment Details

Request

```
GET https://apitest.cybersource.com/flex/v2/payment-details/{jti}
```

Response to Successful Request

```
{
  "paymentInformation": {
    "card": {
      "expirationYear": "2026",
      "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"
    }
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript API Reference

This reference provides details about the JavaScript API for creating the Unified Checkout v1 payment form.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

VAS.UnifiedCheckout(sessionJWT)

This is a factory function that initializes the SDK. It returns a frozen, immutable client interface.

VAS.UnifiedCheckout(sessionJWT) Parameters

Name	Type	Required?	Description
`sessionJWT`	`string`	Yes	Signed JSON Web Token (JWT) from the server-side session endpoint

Returns

`Promise<UnifiedCheckoutInterface>`

Errors

Returns `UnifiedCheckoutError` with reason `CAPTURE_CONTEXT_INVALID` if the JWT signature is invalid, or `UNUSED_TARGET_ORIGINS` if the current page origin is not in the JWT's `targetOrigins` list.

Example

```
const client = await VAS.UnifiedCheckout(sessionJWT);
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

UnifiedCheckoutInterface

The client object returned by `VAS.UnifiedCheckout()`. All methods throw an `Error` if called after `destroy()`.

client.createCheckout(options?)

client.createCheckout(options?) Parameters

Name	Type	Required?	Description
`options`	`CreateCheckoutOptions`	No	Configuration for the checkout

`CreateCheckoutOptions` Properties

Property	Type	Default	Description
`autoProcessing`	`boolean`	Inferred from capture context.	• `true`: `mount()` returns completed payment result. Defaults to `true` when **completeMandate** is included in the capture context. • `false`: `mount()` returns transient token.

Returns

`Promise<Checkout>`

Example

```
const checkout = await client.createCheckout({ autoProcessing:
 false });
```

client.createTrigger(paymentType, options?)

client.createTrigger(paymentType, options?) Parameters

Name	Type	Required?	Description
`paymentType`	`AllowedPaymentType`	Yes	Only `PANENTRY` and `CLICKTOPAY` are supported.
`options`	`CreateTriggerOptions`	No	The configuration for the trigger.

`CreateTriggerOptions` Properties

Property	Type	Default	Description
`autoProcessing`	`boolean`	Inferred from session	Same as checkout `autoProcessing`

Returns

`Trigger`

Errors

Returns `UnifiedCheckoutError` with reason `TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED` when the payment type cannot be used with a trigger.

Example

```
const trigger = client.createTrigger('PANENTRY');
```

client.createButton(paymentType, options?) (Experimental)

Creates an individual payment method button.

client.createButton(paymentType, options?) Parameters

Name	Type	Required?	Description
`paymentType`	`AllowedPaymentType`	Yes	Payment type for the button. For example, `GOOGLEPAY` and `APPLEPAY`.
`options`	`CreateButtonOptions`	No	The configuration for the button.

`CreateButtonOptions` **Properties**

Property	Type	Default	Description
`autoProcessing`	`boolean`	Inferred from session	Same as checkout `autoProcessing`

Returns

`PaymentButton`

Example

```
const button = client.createButton('GOOGLEPAY');
```

client.on(event, callback)

Subscribes to a client-level event and returns an unsubscribe function.

client.on(event, callback) Parameters

Name	Type	Required?	Description
`event`	`string`	Yes	Event name. Possible values: • `*` • `created` • `destroyed` • `error`
`callback`	`function`	Yes	Handler function that receives event-specific payload.

`CreateButtonOptions` **Properties**

Property	Type	Default	Description
`autoProcessing`	`boolean`	Inferred from session	Same as checkout `autoProcessing`

Returns

`Unsubscribe`: A function that removes the handler when called.

Errors

Returns `Error` when `event` is not a valid event name with reason `TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED` when the payment type cannot be used with a trigger.

Example

```
const unsubscribe = client.on('error', (err) => {
  console.error(err.source, err.code, err.message);
});

// Later
unsubscribe();
```

client.off(event, callback?)

Removes an event handler. This method is permissive — calling it with an unknown event or callback does not throw.

client.off(event, callback?) Parameters

Name	Type	Required?	Description
`event`	`string`	Yes	Event name to unsubscribe from
`callback`	`function`	No	Specific handler to remove. When this is not included, all handlers for the event are removed.

client.destroy()

Permanently destroys the client. Returns a `destroyed` event, clears all event listeners, and marks the instance as destroyed.

You can call `destroy()` multiple times.

client.isDestroyed()

Returns a value of `true` if **client.destroy()** is called.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Checkout

This field is returned by `client.createCheckout()` and manages the full checkout UI lifecycle.

checkout.mount(target)

Subscribes to a client-level event and returns an unsubscribe function.

checkout.mount(target) Parameters

Name	Type	Required?	Description
`target`	`string` or `CheckoutContainers`	No	CSS selector string for sidebar mode, or an object with `paymentSelection` and `paymentScreen` for embedded mode. Omit for full sidebar

`CheckoutContainers` Properties

Property	Type	Default	Description
`paymentSelection`	`string`	Yes	CSS selector for the button list container
`paymentScreen`	`string`	No	CSS selector for the payment form container. If omitted, payment screens appear in sidebar mode

Returns

`Promise<string>`: This is a transient token JWT when `autoProcessing: false` or completed payment result JWT when `autoProcessing: true`.

Errors

Returns `UnifiedCheckoutError`. For information about how to handle mount error codes, see Handle Errors (on page 329).

Example

```
// Sidebar
const result = await checkout.mount('#buttons');

// Embedded
const result = await checkout.mount({
  paymentSelection: '#buttons',
  paymentScreen: '#form'
});
```

checkout.unmount()

Removes the payment UI from the page. The checkout is not destroyed — you can call `mount()` again.

checkout.complete(transientToken)

Manually completes the payment flow. This is only available when `autoProcessing` is set to `false`.

checkout.complete(transientToken) Parameters

Name	Type	Required?	Description
`transientToken`	`string`	Yes	The transient token JWT that is returned by `mount()`.

`CheckoutContainers` Properties

Property	Type	Default	Description
`paymentSelection`	`string`	Yes	CSS selector for the button list container
`paymentScreen`	`string`	No	CSS selector for the payment form container. If omitted, payment screens appear in sidebar mode

Returns

`Promise<string>`: This is the completed payment result JWT.

Errors

Returns `UnifiedCheckoutError`. For information about how to handle mount error codes, see Handle Errors (on page 329).

Example

```
const token = await checkout.mount('#buttons');
const result = await checkout.complete(token);
```

checkout.isMounted()

Returns `true` when the checkout UI is mounted.

checkout.isDestroyed()

Returns `true` when `destroy()` is called.

checkout.on(event, handler)

Subscribes to a checkout-level event and returns an unsubscribe function.

Valid events:

- `mounted`
- `ready`
- `unready`
- `unmounted`
- `destroyed`
- `paymentMethodSelected`
- `paymentMethodCancelled`
- `paymentMethodUpdate"`
- `error`
- `*`

checkout.off(event, handler?)

Removes a checkout event handler.

checkout.destroy()

Permanently destroys the checkout. This field removes the payment UI, cleans up iframes, and emits a `destroyed` event.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Trigger

The trigger is returned by `client.createTrigger()` and programmatically launches a specific payment method.

trigger.mount(target?)

Launches the payment method UI.

trigger.mount(target?) Parameters

Name	Type	Required?	Description
`target`	`string`	No	CSS selector for embedded mode. Omit for sidebar mode.

Returns

`Promise<string>`: A transient token or completed payment result.

Example

```
const result = await trigger.mount('#payment-screen');
```

trigger.unmount()

Hides the payment method UI. The trigger is not destroyed.

trigger.complete(transientToken)

Manually completes the payment. Same interface as `checkout.complete()`.

trigger.isMounted()

Returns a boolean value.

trigger.isDestroyed()

Returns a boolean value.

trigger.on(event, handler)

Subscribes to trigger events. Same event names and payloads as checkout events.

trigger.off(event, handler?)

Removes a trigger event handler.

trigger.destroy()

Permanently destroys the trigger.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

PaymentButton

This is returned by `client.createButton()`. It renders an individual payment method button.

button.mount(container)

Mounts the button into a container.

button.mount(container) Parameters

Name	Type	Required?	Description
`container`	`string` or `HTMLElement`	Yes	CSS selector string or Document Object Model (DOM) element for the button container

Returns

`Promise<string>`: A transient token or completed payment result.

button.unmount()

Removes the button from the page. The button is not destroyed.

button.isMounted()

Returns a boolean value if the button is mounted (`true`) or not mounted (`false`).

button.isDestroyed()

Returns a boolean value if the button is destroyed (`true`) or not destroyed (`false`).

button.on(event, callback)

Subscribes to button events. Returns `void`. Use `button.off()` to remove handlers.

Important: The button event system is under development. Event names and payloads may change in future releases.

button.off(event, callback?)

Removes a button event handler.

button.destroy()

Permanently destroys the button. Idempotent.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Events

Unified Checkout provides a type-safe event system for monitoring the payment lifecycle. Events are emitted at the client and integration levels.

Subscribe to Events

Use `on()` to subscribe to events. this returns an unsubscribe function:

```
const unsubscribe = checkout.on('ready', (data) => {
  console.log('Ready:', data.availablePaymentMethods);
});

// Later, remove the handler
unsubscribe();
```

You can use `off()` to remove a specific handler:

```
function onReady(data) { /* ... */ }

checkout.on('ready', onReady);
checkout.off('ready', onReady);
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Update to Unified Checkout Version 1

The version 1 (v1) SDK simplifies your integration with fewer lines of code, a streamlined API, and enhancements such as auto-processing and a full event system. The core flow is the same in v1, and migrating to v1 involves only straightforward method renames.

Summary of Changes

Aspect	v0	v1
Initialization	`new Accept(session).unifiedPayments()`	`VAS.UnifiedCheckout(session)`
Display the payment UI	`up.show(options)`	`checkout.mount(target)`
Completing transactions	`up.complete(token)`	`checkout.complete(token)` or automatic using `autoProcessing`
Events	None	Full event system on client and checkout
Cleanup	`up.dispose()`	`checkout.destroy()` + `client.destroy()`
Hide UI	`up.hide()`	`checkout.unmount()`
Target Origin	Multiple non-usable URLs can be included in the request.	If any origins are absent or mismatched, for example, they are not presented in Unified Checkout, the system prevents Unified Checkout from loading and displays a client-side error message.

Summary of v0 and v1 Changes

Feature	Pre V1 Support	V1 Support	Description
Status	✓	Business Center	
Capture context endpoint	`/up/v1/capture-contexts`	`/up/v1/sessions`	
Capture context management	API only	API only or API and Business Center	Business Center configuration is at the merchant level.
Unified Checkout Look and Feel in Business Center	✗	✓	Business Center configuration is at the merchant level.
Unified Checkout Look and Feel Using the API	✗	✓	Configure the look and feel in a Sessions API request.
Payment methods	API only	API or API and Business Center	Business Center configuration is at the merchant level.
Real-time preview in Business Center	✗	✓	Business Center configuration is at the merchant level.
Three-decimal currency support	✗	✓	
SDK	Legacy Unified Payments SDK supported (link)	New UC SDK (link)	
Payment Details API	`/up/v1/payment-details/{id}`	JTI used in place of transient token	JTI is located in the transient token
Future enhancements	Manual opt-in is required.	Automatic when the **clientVersion** is not included in the Sessions API request.	Legacy versions receive critical updates only.

Initialization

Initialization with Unified Checkout v1 is done in a single asynchronous factory call. There is no intermediate `Accept` object:

v0 Initialization

```
const accept = new Accept(sessionJWT);
const up = accept.unifiedPayments();
```

v1 Initialization

```
const client = await VAS.UnifiedCheckout(sessionJWT);
```

Unified Checkout v1 validates the JWT signature and target origins during initialization.

Display the Payment UI

Unified Checkout v1 passes your UI payment selectors directly to `mount()`.

When `autoProcessing` is enabled, `mount()` returns the completed payment result rather than a transient token.

v0 Display Payment UI with `show()`

```
// Sidebar
const token = await up.show({
  containers: {
    paymentSelection: '#buttons'
  }
});

// Embedded
const token = await up.show({
  containers: {
    paymentSelection: '#buttons',
    paymentScreen: '#form'
  }
});
```

v1 Display Payment UI with `mount()`

```
// Sidebar
const result = await checkout.mount('#buttons');

// Embedded
```

```
const result = await checkout.mount({
  paymentSelection: '#buttons',
  paymentScreen: '#form'
});
```

Completing Transactions

When `autoProcessing` is enabled in Unified Checkout v1, `mount()` returns the completed payment result and you do not need to send a separate **complete()** request. When `autoProcessing` is disabled in Unified Checkout v1, you must complete transactions manually.

v0 Complete Transactions Manually

```
const token = await up.show({
  containers: {
    paymentSelection: '#buttons'
  }
});
const result = await up.complete(token);
```

v1 Complete Transactions Manually or Automatically

```
// Automatic (default when completeMandate is in session)
const checkout = await client.createCheckout({ autoProcessing: true });
const result = await checkout.mount('#buttons');
// result is the completed transaction — no need to call complete()

// Manual - similar to v0
const checkout = await client.createCheckout({ autoProcessing: false });
const token = await checkout.mount('#buttons');
const result = await checkout.complete(token);
```

Events

Unified Checkout v0 does not include an event system, as the integration resolution or rejection from `show()` and **complete()**. v1 includes a full event system as the client and integration levels.

v1 Full Event System

```
// Client-level — centralized error tracking
client.on('error', (err) => {
  console.error(`[${err.source}] ${err.code}: ${err.message}`);
});

// Checkout-level — granular lifecycle events
checkout.on('ready', (data) => {
  console.log('Available methods:', data.availablePaymentMethods);
});

checkout.on('paymentMethodSelected', (data) => {
  console.log('Selected:', data.type);
});

checkout.on('error', (err) => {
  console.error('Checkout error:', err.code);
});
```

Cleanup

Unified Checkout v1 distinguishes between `unmount()`, which is reversible, and `destroy()`, which is permanent. Before a cleanup, `client.destroy()` sends a `destroyed` event.

v0 Cleanup

```
up.hide();     // Hide UI
up.dispose();  // Clean up resources
```

v1 Cleanup

```
checkout.unmount();   // Remove UI from page (can remount later)
checkout.destroy();   // Permanent cleanup

client.destroy();     // Destroy client and clear all event listeners
```

Handle Errors

The `UnifiedCheckoutError` class and its reason codes are the same in v0 and v1:

v0 Error Handling

```
try {
  const token = await up.show({
    containers: {
      paymentSelection: '#buttons'
    }
  });
} catch (err) {
  console.error(err.reason, err.message);
}
```

v1 Error Handling

```
// Same error class, same properties
try {
  const result = await checkout.mount('#buttons');
} catch (err) {
  console.error(err.reason, err.message);
}
```

Migrate Triggers

If your Unified Checkout v0 integration uses triggers, the migration is similar to checkout. In v1, triggers are created from the `client.createTrigger`, not from `UnifiedPayments` as in v0. In v1, `show()` is renamed to `mount()`.

v0 Triggers

```
const trigger = up.createTrigger('CLICKTOPAY', {
  containers: { paymentScreen: '#screen' }
});
const token = await trigger.show();
```

v1 Triggers

```
const trigger = client.createTrigger('CLICKTOPAY');
const result = await trigger.mount('#screen');
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Update Reason Codes

Some reason codes were renamed in v1. This table shows the v0 reason code name and the corresponding name in the v1 client-side SDK:

v0 Reason Code	v1 Reason Code
`SHOW_LOAD_CONTAINER_SELECTOR`	`MOUNT_CONTAINER_SELECTOR`
`SHOW_LOAD_ERROR`	`MOUNT_ERROR`
`SHOW_LOAD_INVALID_CONTAINER`	`MOUNT_INVALID_CONTAINER`
`SHOW_LOAD_SIDEBAR_OPTIONS`	`MOUNT_SIDEBAR_OPTIONS`
`SHOW_PAYMENT_TIMEOUT`	`MOUNT_PAYMENT_TIMEOUT`
`SHOW_PAYMENT_UNAVAILABLE`	`MOUNT_PAYMENT_UNAVAILABLE`
`SHOW_TOKEN_TIMEOUT`	`MOUNT_TOKEN_TIMEOUT`
`SHOW_TOKEN_XHR_ERROR`	`MOUNT_TOKEN_XHR_ERROR`
`UNIFIED_PAYMENTS_ALREADY_SHOWN`	`CHECKOUT_ALREADY_MOUNTED`
`UNIFIED_PAYMENTS_PAYMENT_PARAMETERS`	`CHECKOUT_PAYMENT_PARAMETERS`
`UNIFIED_PAYMENTS_VALIDATION_PARAMS`	`CHECKOUT_VALIDATION_PARAMS`

Important: The server-side API continues to return these v0 reason codes. The v1 reason codes listed here are used only in the client-side SDK. For all v1 client-side error codes, see Handle Errors (on page 329).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Version 1 Update Checklist

You must complete these tasks before you can complete your migration from Unified Checkout v0 to v1:

- Replace `new Accept(session).unifiedPayments()` with `await VAS.UnifiedCheckout(session)`.
- Replace `up.show(options)` with `checkout = await client.createCheckout(); checkout.mount(target)`.
- Update container options: `{ containers: { paymentSelection, paymentScreen } }` becomes direct arguments to `mount()`.
- Replace `up.complete(token)` with `checkout.complete(token)` or use `autoProcessing: true` to complete transactions automatically
- Replace `up.hide()` with `checkout.unmount()`.
- Replace `up.dispose()` with `checkout.destroy()` and `client.destroy()`.
- Add event listeners for observability. For example, `client.on('error')` and `checkout.on('ready')`.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Appendix

This section contains supplementary information for Unified Checkout.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

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

```
<Header>.<Payload>.<Signature>
```

The header and payload is **Base64-encoded JSON** and contains these claims:

- **Header**: The algorithm and token type. For example:

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

- **Payload**: The claims of what the token represents. For example:

  ```
  {
    "sub": "1234567890",
    "name": "John Doe",
    "iat": 1516239022
  }
  ```

- **Signature**: The signature is computed from the header and payload using a secret or private key.

Important: When working with JWTs, Cybersource recommends that you use a well-maintained JWT library to ensure proper decoding and parsing of the JWT.

Important: When parsing the JWT's JSON payload, you must ensure that you implement a robust solution for transversing JSON. Additional elements can be added to the JSON in future releases. Follow JSON parsing best practices to ensure that you can handle the addition of new data elements in the future.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Supported Countries for Digital Payments

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Supported Countries for Digital Payments A-D

Supported Countries (A through D)

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

Supported Countries (A through D) (continued)

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

Supported Countries (A through D) (continued)

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

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Supported Countries for Digital Payments E-K

Supported Countries (E through K)

Country	Apple Pay	Click to Pay	Google Pay
Ecuador	✗	✓	✓
Egypt	✗	✗	✓

Supported Countries (E through K) (continued)

Country	Apple Pay	Click to Pay	Google Pay
El Salvador	✓	✓	✓
Equatorial Guinea	✗	✗	✓
Eritrea	✗	✗	✓
Estonia	✓	✗	✓
Eswatini	✗	✗	✓
Ethiopia	✗	✗	✓
Faroe Islands	✓	✗	✗
Fiji	✗	✗	✓
Finland	✓	✓	✓
France	✓	✓	✓
Gabon	✗	✗	✓
Gambia	✗	✗	✓
Georgia	✓	✗	✓
Germany	✓	✓	✓
Ghana	✗	✗	✓
Gibraltar	✗	✗	✗
Greece	✓	✓	✓
Greenland	✓	✗	✗
Guernsey	✓	✗	✗
Grenada	✗	✗	✓
Guatemala	✓	✓	✓
Guinea	✗	✗	✓
Guinea-Bissau	✗	✗	✓

Supported Countries (E through K) (continued)

Country	Apple Pay	Click to Pay	Google Pay
Guyana	✗	✗	✓
Haiti	✗	✗	✓
Honduras	✓	✓	✓
Hong Kong	✓	✓	✓
Hungary	✓	✓	✓
Iceland	✓	✗	✓
Indonesia	✗	✓	✗
Iraq	✗	✗	✓
Ireland	✓	✓	✓
Isle of Man	✓	✗	✗
Israel	✓	✗	✓
Italy	✓	✓	✓
Jamaica	✗	✗	✓
Japan	✓	✓	✓
Jersey	✓	✗	✗
Jordan	✓	✓	✓
Kazakhstan	✓	✗	✓
Kenya	✗	✗	✓
Kiribati	✗	✗	✓
Kuwait	✓	✓	✓
Kyrgyzstan	✗	✗	✓

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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	✓	✗	✓
Mongolia	✗	✗	✓
Montenegro	✓	✗	✓
Morocco	✗	✗	✓
Mozambique	✗	✗	✓
Myanmar	✗	✗	✓
Namibia	✗	✗	✓
Nauru	✗	✗	✓
Nepal	✗	✗	✓
Netherlands	✓	✓	✓
New Zealand	✓	✓	✓
Nicaragua	✗	✓	✓
Niger	✗	✗	✓
Nigeria	✗	✗	✓
North Macedonia	✗	✗	✓
Norway	✓	✓	✓
Oman	✗	✗	✓
Pakistan	✗	✗	✓
Palau	✗	✗	✓
Palestinian Territories	✓	✗	✓
Panama	✓	✓	✓
Papua New Guinea	✗	✗	✓
Paraguay	✗	✓	✓

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	✗	✗	✓

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Supported Countries for Digital Payments S-Z

Supported Countries (S through Z)

Country	Apple Pay	Click to Pay	eCheck	Google Pay	Paze
Saint Kitts and Nevis	✗	✗	✗	✓	✗
Saint Lucia	✗	✗	✗	✓	✗
Saint Vin cent 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	✗	✗	✗	✓	✗

Supported Countries (S through Z) (continued)

Country	Apple Pay	Click to Pay	eCheck	Google Pay	Paze
Spain	✓	✓	✗	✓	✗
Sri Lanka	✗	✗	✗	✓	✗
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	✓	✓	✗	✓	✗

Supported Countries (S through Z) (continued)

Country	Apple Pay	Click to Pay	eCheck	Google Pay	Paze
United Kingdom	✓	✓	✗	✓	✗
United States	✓	✓	✓	✓	✓
Uruguay	✗	✓	✗	✓	✗
Uzbekistan	✗	✗	✗	✓	✗
Vanuatu	✗	✗	✗	✓	✗
Vatican City (Holy See)	✓	✗	✗	✓	✗
Venezuela	✗	✗	✗	✓	✗
Vietnam	✓	✓	✗	✓	✗
Yemen	✗	✗	✗	✓	✗
Zambia	✗	✗	✗	✓	✗
Zimbabwe	✗	✗	✗	✓	✗

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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. Set the **locale** field in your session request to display the checkout UI in the customer's language.

Unified Checkout supports these locales:

Supported Locales

Locale	ISO Language	ISO Region
`ar_AE`	Arabic	United Arab Emirates
`bg_BG`	Bulgarian	Bulgaria
`ca_ES`	Catalan	Spain
`cs_CZ`	Czech	Czechia
`da_DK`	Danish	Denmark
`de_AT`	German	Austria
`de_DE`	German	Germany
`el_GR`	Greek	Greece
`en_AU`	English	Australia
`en_CA`	English	Canada
`en_GB`	English	United Kingdom
`en_IE`	English	Ireland
`en_NZ`	English	New Zealand
`en_PK`	English	Pakistan
`en_US`	English	United States
`es_AR`	Spanish	Argentina
`es_CL`	Spanish	Chile
`es_CO`	Spanish	Colombia
`es_ES`	Spanish	Spain
`es_MX`	Spanish	Mexico
`es_PE`	Spanish	Peru
`es_US`	Spanish	United States
`fi_FI`	Finnish	Finland
`fr_CA`	French	Canada

Supported Locales (continued)

Locale	ISO Language	ISO Region
fr_FR	French	France
he_IL	Hebrew	Israel
hr_HR	Croatian	Croatia
hu_HU	Hungarian	Hungary
id_ID	Indonesian	Indonesia
it_IT	Italian	Italy
ja_JP	Japanese	Japan
km_KH	Khmer	Cambodia
ko_KR	Korean	South Korea
lo_LA	Lao	Laos
ms_MY	Malay	Malaysia
nb_NO	Norwegian Bokmål	Norway
nl_NL	Dutch	Netherlands
pl_PL	Polish	Poland
pt_BR	Portuguese	Brazil
ro_RO	Romanian	Romania
ru_RU	Russian	Russia
sk_SK	Slovak	Slovakia
sl_SI	Slovenian	Slovenia
sv_SE	Swedish	Sweden
th_TH	Thai	Thailand
tl_PH	Tagalog	Philippines
tr_TR	Turkish	Türkiye
uk_UA	Ukrainian	Ukraine
ur_PK	Urdu	Pakistan
vi_VN	Vietnamese	Vietnam
zh_CN	Chinese	China
zh_HK	Chinese	Hong Kong
zh_MO	Chinese	Macao

Supported Locales (continued)

Locale	ISO Language	ISO Region
`zh_SG`	Chinese	Singapore
`zh_TW`	Chinese	Taiwan

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Security Recommendations

Unified Checkout is compliant with Payment Card Industry (PCI) Self-Assessment Questionnaire A (SAQ-A). Cybersource recommends that you consider these security policies so you can maintain a secure integration.

Content Security Policy

Implement a Content Security Policy (CSP) to mitigate cross-site scripting (XSS) attacks. Add these directives for Unified Checkout:

CSP Directives

Directive	Test	Production
`connect-src`	https://apitest.cybersource.com	https://api.cybersource.com
`frame-src`	https://apitest.cybersource.com	https://api.cybersource.com
`child-src`	https://apitest.cybersource.com	https://api.cybersource.com
`script-src`	https://apitest.cybersource.com	https://api.cybersource.com

These directives enable the SDK to load secure iframes and communicate with Cybersource services.

Important: When you use additional payment methods such as Google Pay or PayPal, you must also add their respective domains to your CSP directives.

Iframe Isolation

Unified Checkout renders all payment UI inside cross-origin iframes hosted by Cybersource. This architecture provides several security benefits:

- **Data isolation**: Your page cannot access payment data within the iframe due to the browser's same-origin policy.
- **Reduced attack surface**: Attackers cannot extract card data from the isolated iframe if your merchant page is compromised.
- **Origin verification**: The SDK validates that the hosting page origin matches the `targetOrigins` that is declared in the session before displaying any UI.

Do not attempt to access or manipulate the contents of the payment iframes. The browser blocks cross-origin access by design.

Token Security

A session is a signed JWT with a short lifespan. Cybersource recommends that you follow these practices:

- Generate a new session for each checkout. Do not reuse sessions across checkouts or customers.
- Keep the session server-side until needed. Pass it to the client only when the customer is ready to pay.
- Set `targetOrigins` to only the domains that host the SDK. Do not use wildcard origins or include domains that do not need access.

The transient token returned by `mount()` or **complete()** expires after 15 minutes. Cybersource recommends that you follow these practices:

- Send the transient token to your server immediately after receiving it.
- Verify the token signature using the public key from the session before authorizing the payment. For verification details, see Transient Tokens (on page 313).
- Do not store transient tokens in browser storage (`localStorage`, `sessionStorage`, or cookies). Process them server-side and discard.

Immutable API

The client interface that is returned by `VAS.UnifiedCheckout()` is frozen with `Object.freeze()`. This prevents runtime tampering, which means that no properties can be added, removed, or modified on the client, checkout, trigger, or button objects. Do not attempt to modify or extend the SDK objects. If you need custom behavior, use the event system to react to SDK state changes.

Cleanup

You must always call `destroy()` on the client when the payment flow is complete or the customer navigates away. This removes all iframes and clears internal state:

```
checkout.destroy();
client.destroy();
```

If you do not destroy the client, you could leave payment iframes in the page after they are no longer needed.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

PCI Compliance

The least burdensome level of Payment Card Industry (PCI) compliance is Self-Assessment Questionnaire A (SAQ-A). To be compliant with SAQ-A, you must securely capture sensitive payment data with a validated payment provider. Unified Checkout meets this requirement by rendering secure iframes hosted by Cybersource. Payment data is submitted directly to Cybersource and never touches your systems.

Security Architecture

Unified Checkout uses many layers of protection to be compliant with PCI SAQ-A guidelines:

- **Iframe isolation**: All payment UI renders inside cross-origin iframes hosted by Cybersource. Your page cannot access payment data within the iframe due to the browser's same-origin policy.
- **Origin verification**: The SDK validates that the hosting page origin matches the `targetOrigins` declared in the session.
- **Immutable API**: The client interface returned by `VAS.UnifiedCheckout()` is frozen with `Object.freeze()`. This prevents runtime tampering.
- **Closure-based privacy**: The internal SDK state is not accessible from outside the SDK. There are no public properties that expose session data or credentials.
- **Short-lived tokens**: The session and transient tokens expire after a short period, limiting the window for misuse

Because Unified Checkout handles payment data capture within secure iframes, your page never receives, processes, or stores cardholder data. This means that you qualify for SAQ-A over the more burdensome SAQ A-EP or SAQ D and your PCI audit scope is significantly reduced compared to direct API integrations.

Even with all that Unified Checkout handles, you must still do the following to remain SAQ-A compliant:

- All pages that load the SDK must use Transport Layer Security (TLS).
- You must restrict which domains can load scripts and frames. For information about the required directives, see Security Recommendations (on page 378).
- You must generate a new session for each checkout and restrict `targetOrigins` to only your domains.
- You must send transient tokens to your server over HTTPS and verify their signatures before authorizing payments.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Client Version History

Below is a list of client versions and the features that are included in each version.

Important: Cybersource recommends that you use the most recent client version in your integration.

0.23

Accepts these card networks in the **allowedCardNetworks** field for manual card entry:

- Carnet
- Cartes Bancaires
- China UnionPay with card verification value (CVV)
- EFTPOS
- ELO
- JCrew PLCC
- mada
- Meeza

Ordering controls for the **allowedPaymentTypes** button.

De-coupling of PANENTRY from other payment types in the **allowedPaymentTypes** field.

0.24

Support for enabling combo cards in the capture context.

Support for eight-digit BINs.

Support for enabling card save in the capture context.

0.25

Addition of **Skip Verification next time** in the Click to Pay payment flow.

Support for CPF in the capture context.

0.26

Support for auto-lookup in Click to Pay when an email is included in the capture context.

Inclusion of the **cardDetails** field object in the transient token response.

Support for the **cardholderAuthenticationStatus** field object in the transient token response.

Support for the complete mandate.

0.28

Complete mandate enhancement to support Payer Authentication for manual card entry for Visa, Mastercard, American Express, Discover, JCB, Cartes Bancaires, China UnionPay, and ELO card brands.

Support for Afterpay as an **allowedPaymentType**.

Support for PayPak as an **allowedCardNetwork**.

Auto-enrollment for Click to Pay in supported markets.

Removal of the confirm or continue screen for specific use cases.

Static button for Click to Pay flows.

0.30

Support for iDeal, Multibanco, and Przelewy24|P24.

Complete mandate enhancement to support Payer Authentication for Google Pay and Click to Pay.

Support for Pakistan locales (en_PK and ur_PK).

New look and feel of Unified Checkout in line with EMVCO best practices.

0.31

Addition of the **data** object of the **orderInformation** field object and pass-through fields.

Support for **tokenCreate** in the complete mandate.

Support of pass-through fields, including challenge codes and data only, for Payer Authentication.

Support for Jaywan as an **allowedCardNetwork**.

Updated the payment details response to return detected card types. Multiple card types are shown when more than one card type is detected.

Support for Bancontact, Dragonpay, MyBank, and Tink Pay By Bank.

0.32

Support for KCP and UATP in the **allowedCardNetwork** field.

Support for Konbini as a post-pay reference payment method.

A radio button in the UI for Cartes Bancaires dual-branded cards.

0.33

Support for mobile as identity for Click to Pay accounts.

Japanese language translation updates.

UX captures billing and shipping information when they are not included in the capture context.

0.34

Iframes are used instead of pop-ups to reduce pop-up blocking and streamlining mobile deployment.

Additional BIN range for Jaywan card types.

0.35

Look and feel customization.

1.0

Configure payment options.

Configure customer data and payment flow.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Introduction to the Click to Pay Drop-In UI

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. The Click to Pay Drop-In UI handles manual card entry for the non-Click to Pay payment schemes called out in this guide. 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 set of server-side APIs and a client-side JavaScript library.

The server-side APIs authenticate your merchant identity, instruct the system to act within your payment environment, and provide a way to retrieve the payment data following a successful Click to Pay Drop-In UI interaction.

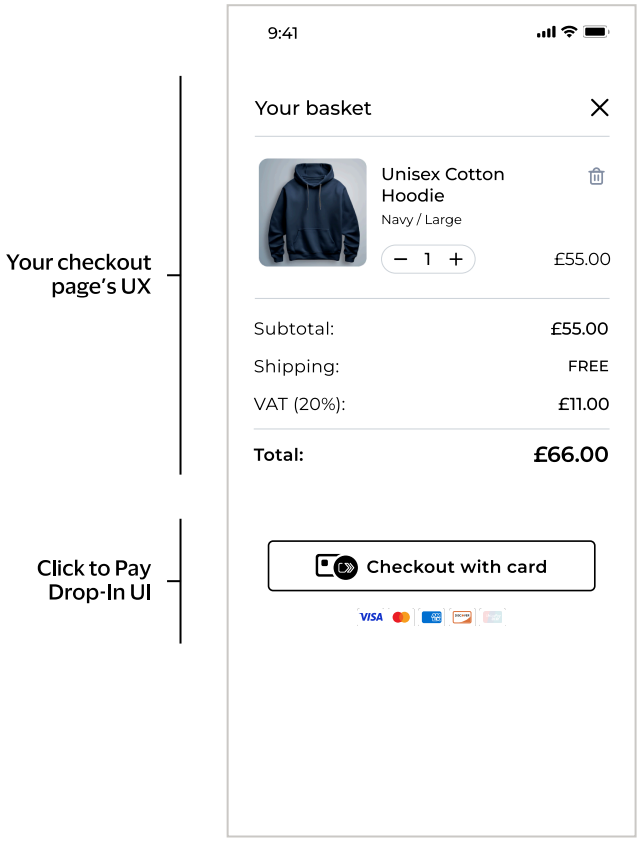
The provided JavaScript library enables you to place a payment application within your e-commerce environment. This embedded component offers Click to Pay and card entry to your customers.

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. All UI / UX must follow the UI/UX guidelines. For information about configuring your UI/UX, see Click to Pay UI Examples (on page 499).

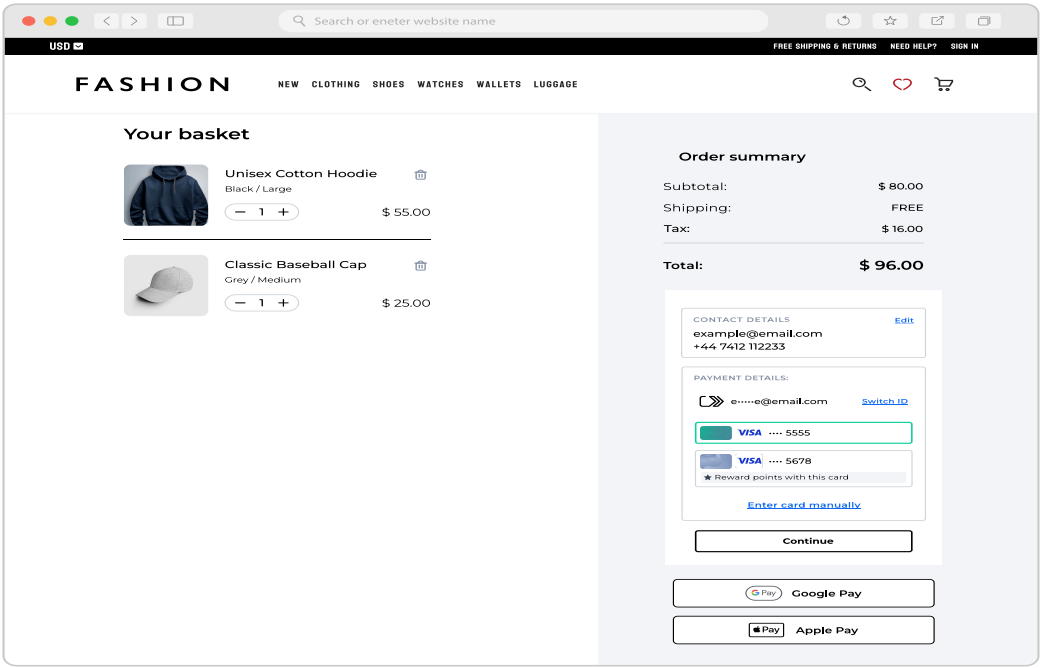
The Click to Pay Drop-In UI enables a portfolio to receive an encrypted payload and send a request to the API to retrieve the payment details. The format of the decrypted payment details are determined by the transaction type. The details are either a network token and cryptogram or the PAN, expiration details, and card verification value (CVV).

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

Unified Checkout UI with Card Payment Button



Unified Checkout UI without Card Payment Button



Important: Each request that you send to Cybersource requires header information. For information about constructing the headers for your request, see the *Getting Started with REST Developer Guide*.

Browser Support

Unified Checkout supports these browser versions:

- Firefox 121
- GoogleChrome/Chium-based browsers 118
- MicrosoftEdge 118
- Safari16

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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 388)
- Unrecognized Click to Pay Customer (on page 389)
- Guest Customer (on page 390)

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Recognized Click to Pay Customer

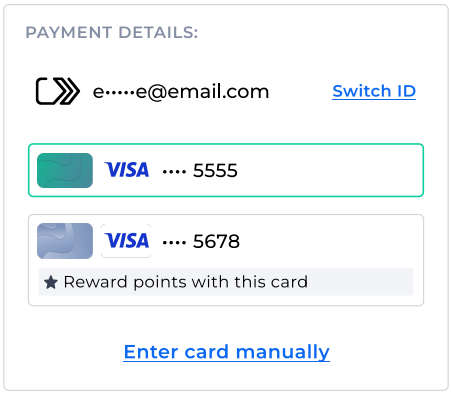
This section provides an overview of the Click to Pay Drop-In UI recognized experience. This interaction occurs when a customer’s device is recognized by the Click to Pay Drop-In UI.

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 or when they enter their one-time password (OTP).

  The cardholder is presented with their stored Click to Pay cards in the UI when they are on a recognized device:

Recognized Click to Pay Customer UI



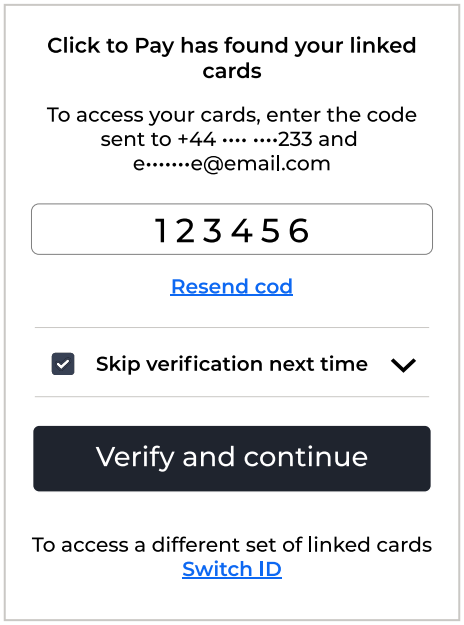
Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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 opted to have their details stored on the device. In this flow, the customer receives an OTP on their registered mobile device or their email address. The OTP can be received from any of the supported card networks, but will return cards that are stored in Click to Pay across all of the user's supported card networks. A Visa cardholder will receive an OTP to their registered email address and phone number to authenticate their identity. A Mastercard cardholder will receive an OTP on their registered phone number. After the user's identity is authenticated, their stored Click to Pay credentials are shown:

Unrecognized Click to Pay Customer on a Recognized Device UI



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

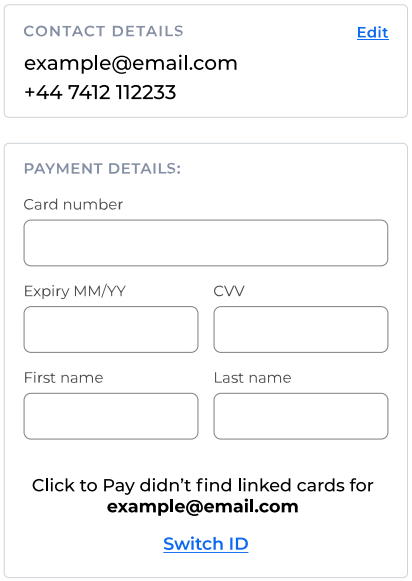
Guest Customer

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

In the guest experience, Click to Pay Drop-In UI captures the PAN details and the cardholder chooses to create a Click to Pay account or to check out as a guest. In both cases, the payment credentials are available for processing transactions using your payment gateway.

The cardholder can make a one-time payment or complete the payment and choose to create a Click to Pay account for future use using their chosen email address and phone number combination. A user can select **Switch ID** or **Edit** within the contact details tab in order to look up new payment details:

Guest UI



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Click to Pay Drop-In UI Flow

To integrate Click to Pay Drop-In UI into your platform, you must follow several integration steps. This section gives a high-level overview of how to integrate and launch Click to Pay Drop-In UI on your webpage and process a transaction. You can find the detailed specifications of the APIs later in this document.

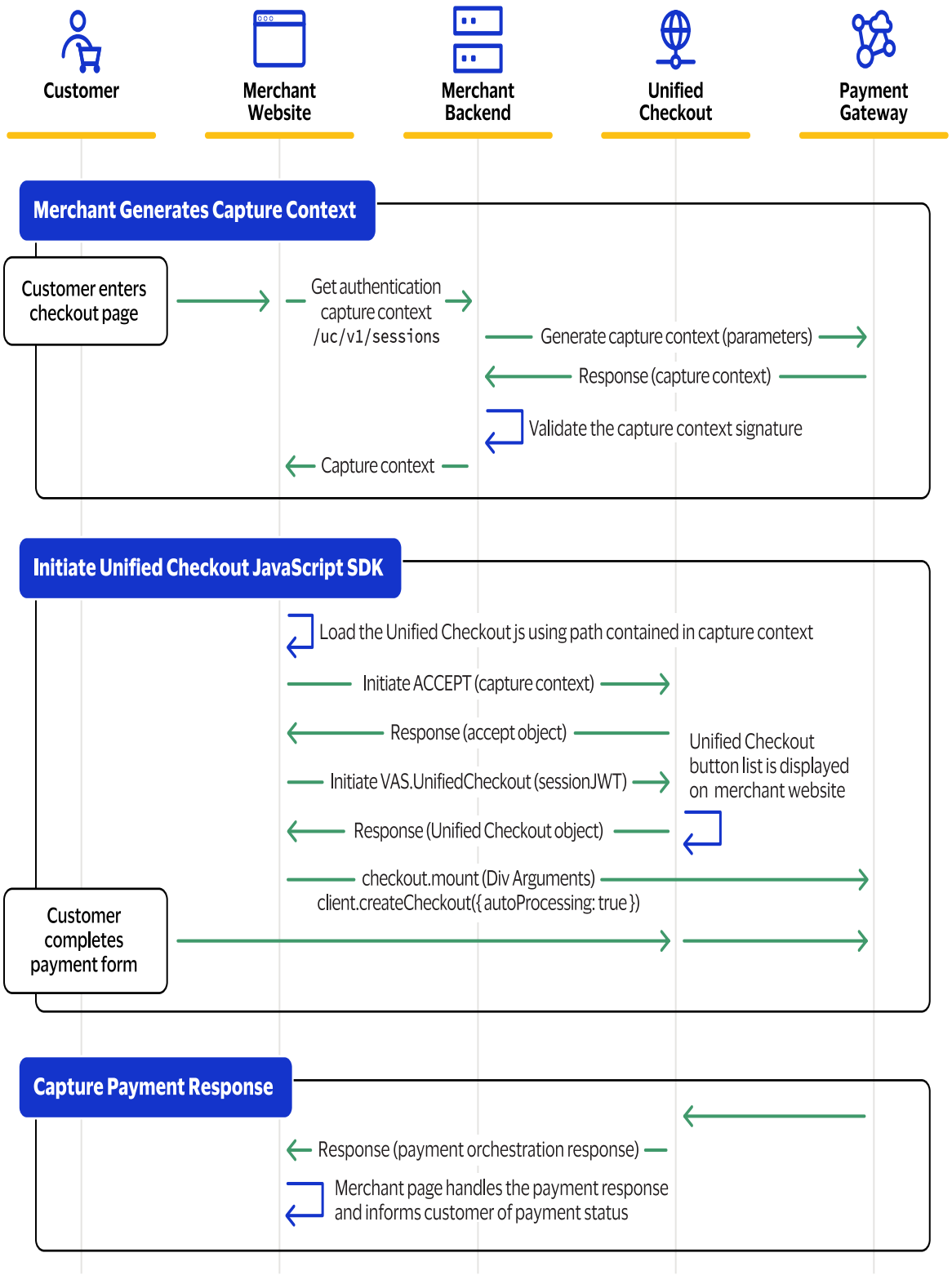
1. You send a server-to-server API request for a capture context. This request is fully authenticated and returns a JSON Web Token (JWT) that is necessary to invoke the frontend JavaScript library. For information on setting up the server side, see Server-Side Set Up (on page 404).

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 407).

3. You use the response from the Click to Pay Drop-In UI to retrieve payment credentials for payment processing or other steps.

This figure illustrates the system's payment flow.

Click to Pay Payment Flow



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

- Sessions API - Capture Context (on page 414)
- Payment Details API (on page 451)
- Payment Credentials API (on page 454)

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Add Click to Pay to a Merchant Account

Follow these steps to add Click to Pay to an organization:

1. In the left navigation panel, click **Portfolio Management**.
2. Under Merchants, click **Manage Merchants**. The Manage Merchants page appears.
3. Click **+ Add Merchant**.
4. Select where you want to board your merchant:
   - Select **Board a new merchant account** to create a new merchant account.
   - Select **Add to an existing account** to add a transacting merchant to an existing merchant organization.

   Click **Next**.
5. If you are adding a transacting organization to an existing merchant account, search for the merchant account in the Boarding Presets section.
6. If you have more than one boarding package, choose a boarding package from the drop-down menu, or enter text in the search field to find one. Click **Next**. If you have only one boarding package, the Boarding Package section does not display.
7. Click **Start** in the Merchant Account Information section to enter account information. For more information, see Add Merchant Account Information (on page).
8. Optional: click **Skip** in the Hierarchy Details section to skip the hierarchy step.

9. Click **Start** in the Transacting Organization and Products section to set up a transacting organization and configure products for it. The Transacting Organization and Products page appears.
10. Under Transacting Organization Details, enter the transacting organization name and the organization ID.
11. Under Product Enablement, find Unified Checkout and select **Enabled** under the Enablement drop-down menu.
12. Click **Configure** to configure Unified Checkout.
13. Under Payment methods, select Click to Pay.
14. Under Card Brands, select the card brands you want to enable in Unified Checkout. These card brands are supported by Click to Pay:
    - American Express
    - Mastercard
    - Visa

    You can select **Allow All** to enable all card brands for your merchants. When you select **Allow All**, future additions to supported card brands are automatically available.

    Important: You must select at least one card brand to support.

15. Under Integrated services, select **Retrieve Sensitive Information at Portfolio Level**.
16. Click **Apply** to save your configuration.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Follow these steps to enable Click to Pay for Unified Checkout using the API or in the Business Center. You must follow these steps for each transacting merchant for which you want to enable Click to Pay.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Enabling Click to Pay Drop-In UI Using the API

This section shows you how to enable Click to Pay Drop-In UI using the Boarding Registration Service (BRS) API.

To enable Click to Pay 3DS authentication, you must include these fields in your request:

- **productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.acquirerBIN**
- **productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.acquirerName**

For information about enabling Click to Pay authentication, see Enable Click to Pay Customer Authentication Using the API (on page 481).

Endpoint

Production: `POST https://api.cybersource.com/boarding/v1/registrations`

Test: `POST https://apitest.cybersource.com/boarding/v1/registrations`

Related information

Getting Started with REST
Response Codes
API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Required Fields for Enabling Click to Pay Drop-In UI

organizationInformation.businessInformation.merchantCategoryCode

organizationInformation.businessInformation.name

organizationInformation.businessInformation.websiteUrl

organizationInformation.organizationId

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.merchantName

Required if the **enrollmentData** object is included in the request.

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.merchantURL

Required if the **enrollmentData** object is included in the request.

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.portfolioAccessofSensitiveData.merchantAccessofSensitiveData

Set to `false`.

productInformation.selectedProducts.payments.unifiedCheckout.subscriptionInformation.enabled

Required to enable Unified Checkout.

productInformation.selectedProducts.payments.unifiedCheckout.subscriptionInformation.features.clickToPay.enabled

Set to `true`.

productInformation.selectedProducts.payments.unifiedCheckout.subscriptionInformation.features.portfolioAccessofSensitiveData.enabled

Set to `true`.

Related Information

- API Field Reference for the REST API

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Optional Fields for Enabling Click to Pay Drop-In UI

organizationInformation.businessInformation.address

organizationInformation.businessInformation.address.address1

organizationInformation.businessInformation.address.administrativeArea

organizationInformation.businessInformation.address.country

organizationInformation.businessInformation.address.locality

organizationInformation.businessInformation.address.postalCode

organizationInformation.configurable

organizationInformation.parentOrganizationId

This value is dependent on your organization hierarchy rules.

organizationInformation.status

organizationInformation.type

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.acquirerBIN

Required when you want to enroll in 3-D Secure.

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.acquirerName

Required when you want to enroll in 3-D Secure.

registrationInformation.boardingFlow

registrationInformation.boardingPackageId

registrationInformation.mode

Related Information

- API Field Reference for the REST API

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

REST Example: Enabling Click to Pay Drop-In UI

Enable Click to Pay Drop-In UI

```
{
  "registrationInformation": {
    "boardingFlow": "ENTERPRISE",
    "mode": "COMPLETE",
    "boardingPackageId": "74921204027"
  },
  "organizationInformation": {
    "organizationId": "testucctp001",
    "status": "TEST",
    "businessInformation": {
      "name": "testucctp",
      "websiteUrl": "https://www.test.com",
      "merchantCategoryCode": "0742",
      "address": {
        "country": "US",
        "address1": "Test Dr",
        "postalCode": "78641",
        "administrativeArea": "TX",
        "locality": "Austin"
      }
    },
    "parentOrganizationId": "testucctp",
    "type": "TRANSACTING",
    "configurable": false
  },
  "productInformation": {
    "selectedProducts": {
      "payments": {
        "unifiedCheckout": {
          "subscriptionInformation": {
            "enabled": true,
            "features": {
              "clickToPay": {
                "enabled": true
              },
              "portfolioAccessofSensitiveData ": {
                "enabled": true
              }
            }
          },
          "configurationInformation": {
            "configurations": {
              "features": {
                "clickToPay": {
```

```
                  "enrollmentData": {
                    "merchantName": "testucctp",
                    "merchantURL": "https://www.test.com",
                    "acquirerBIN": "123456",
                    "acquirerName": "ExampleAcquirer"
                  }
                },
                "portfolioAccessofSensitiveData ": {
                  " merchantAccessofSensitiveData ": false
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Enabling Click to Pay in the Business Center

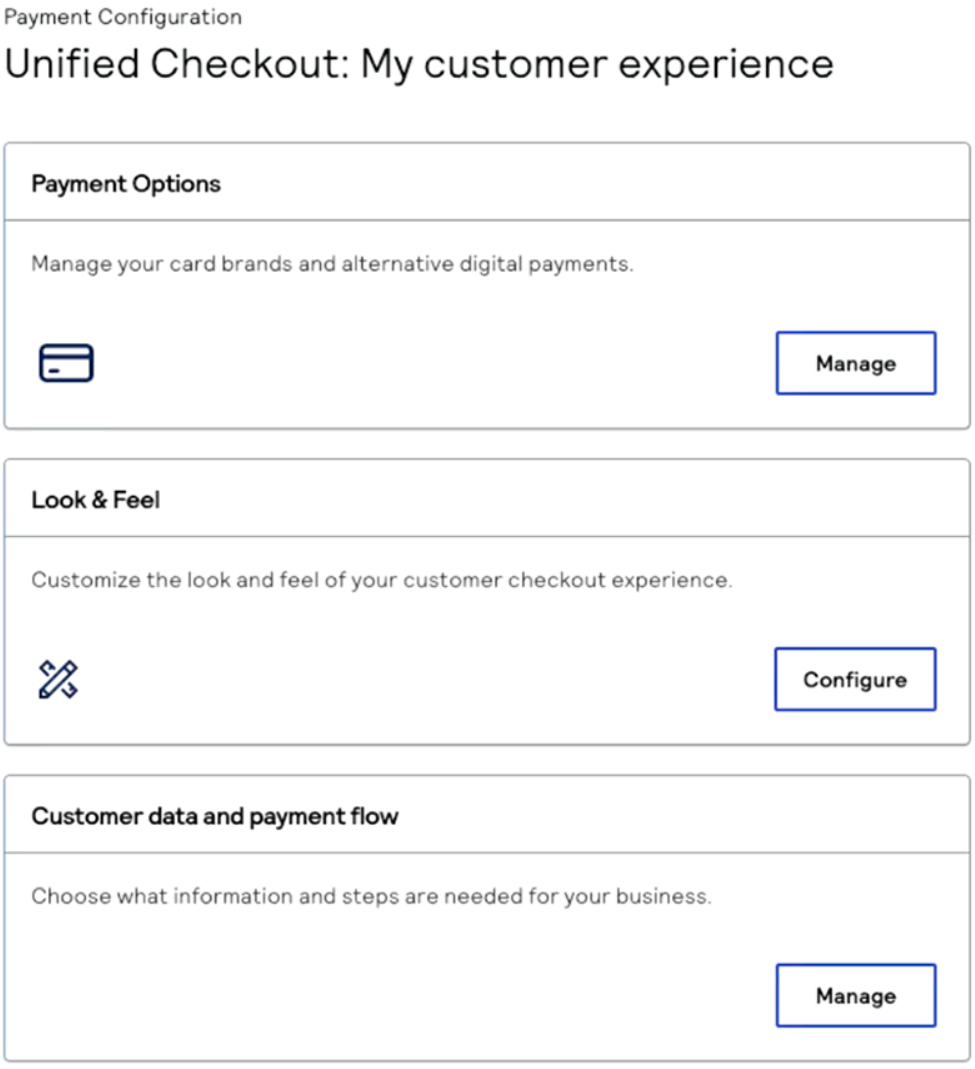
To begin your integration, you must first enable Click to Pay. Click to Pay is a digital payment solution that allows customers to pay with their preferred card network and issuer without entering their card details on every website. Customers can use Visa, Mastercard, and American Express cards to streamline 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. 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**. The Unified Checkout customer experience page appears:

Unified Checkout Customer Experience



3. In the Payment Options section, click **Manage**. The Payment Options page appears.

4. Click **Manage** next to Click to Pay. The Click to Pay configuration page appears.
5. Enter your business name and website URL.
6. Click **Submit**.

> **Important:** Click to Pay uses network tokenization for transactions. These network tokens are stored in the vault of the token requestor ID (TRID) for the card scheme.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Server-Side Set Up

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

The server-side component provides this information:

- A transaction-specific public key that 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 522).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Capture Context Using the Sessions API

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 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 on JWTs see JSON Web Tokens (on page 362).

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. The request provides authentication, one-time keys, the target origin to the Unified Checkout integration in addition to allowed card networks and payment types. The capture context request includes these elements:

- **allowedCardNetworks**
- **allowedPaymentTypes**
- **clientVersion**
- **targetOrigin**

Use the **targetOrigins** and the **allowedPaymentTypes** fields to define the target origin and the accepted digital payment methods in your capture context.

When you configure the merchant settings using the Merchant Experience section of the Business Center, your request to the `sessions` API must include these required fields. All other values are determined from the settings that are configured in the Business Center:

```
{
  "targetOrigins": ["https://merchant.com", "https://reseller.com:8443"],
  "locale": "en_US",
  "country": "US",
    "data": {
        "orderInformation": {
          "amountDetails": {
            "totalAmount": "21.00",
            "currency": "USD"
      }
    }
  }
}
```

When you use only the sessions API to generate the capture context, your request must include these required fields:

```
{
  "targetOrigins": [
    "https://yourCheckoutPage.com"
  ],
  "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
  },
  "data": {
    "orderInformation": {
      "amountDetails": {
        "totalAmount": "21.00",
        "currency": "USD"
      }
    }
  }
}
```

For information about requesting the capture context using the `sessions` API, see the API Reference in the Cybersource Developer Center.

For more information on requesting the capture context, see Sessions API - Capture Context (on page 414).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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 presents Click to Pay to the customer. There are different options available to display this to your customers. See these topics: XXX
- The payment acceptance page, which captures payment information from the cardholder. You can embed the payment acceptance page within 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 522).
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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Loading the JavaScript Library and Invoking the Accept Function

Use the client library asset path and client library integrity value that is returned by the capture context response to invoke Unified Checkout on your page.

You can retrieve these values from the **clientLibrary** and **clientLibraryIntegrity** fields that are returned in the JWT from POST `https://api.cybersource.com/uc/v1/sessions`. You can use these values to create your script tags.

You must perform this process for each transaction, as these values may be unique for each transaction. You must avoid hard-coding values for the **clientLibrary** and **clientLibraryIntegrity** fields to prevent client-side errors.

For example, a response from `https://apitest.cybersource.com/uc/v1/sessions` would include:

```
"data": {
    "clientLibrary":"[EXTRACT clientLibrary VALUE from here]",
    "clientLibraryIntegrity": "[EXTRACT clientLibraryIntegrity VALUE from here]"
}
```

Below is an example script tag:

```
<script src="[INSERT clientLibrary VALUE HERE]"
    integrity="[INSERT clientLibraryIntegrity VALUE HERE]"
    crossorigin="anonymous"></script>
```

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

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

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Initializing the SDK

```
async function launchCheckout() {
  try {
 const client = await VAS.UnifiedCheckout(sessionJWT);
```

```
 const checkout = await client.createCheckout({ autoProcessing: false });
 const token = await checkout.mount('#buttons');
    // result contains the Transient Token
    // Send result to your server for retrieval of payment information
    sendToServer(token);
  } catch (error) {
    if (error.name === 'UnifiedCheckoutError') {
      handleError(error.reason, error.message);
    }
  } finally {
    checkout.destroy();
    client.destroy();
  }
}

launchCheckout();
```

In this example, `sessionJWT` refers to the capture context JWT.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Displaying the Button List

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

```
// Sidebar
const result = await checkout.mount('#buttons');

// Embedded
const result = await checkout.mount({
  paymentSelection: '#buttons',
  paymentScreen: '#form'
});
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry

When you display `CLICKTOPAY` or `PANENTRY` as allowed payment types, you can load the UI without displaying the Unified Checkout checkout button. You can do this by creating a trigger that defines what event loads the UI.

You can create a trigger only for `CLICKTOPAY` or `PANENTRY` payment methods:

```
//PAN Entry

const trigger = client.createTrigger('PANENTRY');

//Click to Pay

const trigger = client.createTrigger('CLICKTOPAY');
```

Important: When you use the `client.createTrigger()` method for Click to Pay, you must create a custom UI to trigger Click to Pay. See Click to Pay UI Examples (on page 499).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Setting Up with Full Sidebar

```
<html>
<head>
  <script
    src="[INSERT clientLibrary VALUE HERE]"
    integrity="[INSERT clientLibraryIntegrity VALUE HERE]”
    crossorigin=”anonymous"
  ></script>
</head>
<body>
  <h1>Unified Checkout Integration</h1>
  <input
    type="hidden"
    name="sessionJWT"
    value="[INSERT sessionJWT HERE]"
  />
  <script type="text/javascript">
    const sessionJWT = document.getElementById("sessionJWT").value;

    async function launchCheckout() {
      try {
        const client = await VAS.UnifiedCheckout(sessionJWT);
        const checkout = await client.createCheckout();
        const result = await checkout.mount('#payment-buttons');
```

```
        // result contains the completed payment result JWT
        // Send result to your server for verification
        sendToServer(result);
      } catch (error) {
        if (error.name === 'UnifiedCheckoutError') {
          handleError(error.reason, error.message);
        }
      } finally {
        checkout.destroy();
        client.destroy();
      }
    }

    launchCheckout();
  </script>
</body>
</html>
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript Example: Setting Up with the Embedded Component

The main difference between using an embedded component and the sidebar is that the `VAS.UnifiedCheckout(sessionJWT)` object is set to `false`, and the location of the payment screen is passed in the containers argument.

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

```
<html>
<head>
  <script
    src="[INSERT clientLibrary VALUE HERE]"
    integrity="[INSERT clientLibraryIntegrity VALUE HERE]"
    crossorigin="anonymous"
```

```
  ></script>
</head>
<body>
  <h1>Unified Checkout Integration</h1>
  <input
    type="hidden"
    id="sessionJWT"
    name="sessionJWT"
    value="[INSERT sessionJWT HERE]"
  />
  <script type="text/javascript">
    const sessionJWT = document.getElementById("sessionJWT").value;

    async function launchCheckout() {
      let client;
      let checkout;

      try {
        client = await VAS.UnifiedCheckout(sessionJWT);
        checkout = await client.createCheckout();
        const result = await checkout.mount('#payment-buttons');

        // result contains the completed payment result JWT
        // Send result to your server for verification
        sendToServer(result);
      } catch (error) {
        if (error.name === 'UnifiedCheckoutError') {
          handleError(error.reason, error.message);
        }
      } finally {
        if (checkout) {
          checkout.destroy();
        }

        if (client) {
          client.destroy();
        }
      }
    }

    launchCheckout();
  </script>
</body>
</html>
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Sessions API - Capture Context

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

The capture context request includes these elements:

- **allowedCardNetworks**
- **allowedPaymentTypes**
- **clientVersion**
- **targetOrigins**

For information on JSON Web Tokens, see JSON Web Tokens (on page 522).

Target Origin

The target origin is defined by the scheme (protocol), hostname (domain) and port number (if used).

You must use the https:// protocol. Sub domains must also be included in the target origin.

Any valid top-level domains, such as .com, .co.uk, and .gov.br, are supported. Wildcards are not supported.

For example, if you are launching Click to Pay on example.com, the target origin could be any of the following:

- https://example.com
- https://subdomain.example.com
- https://example.com:8080

You can supply up to seven origins within the **targetOrigins** field for nested iframes. To do this, you must do the following:

- Compare the list of origins in the `v1/sessions` **targetOrigins** field against the **location.ancestorOrigins** of the browser.
- Ensure that the count of origins and their content matches in the **targetOrigins** field against the **location.ancestorOrigins** of the browser. If any origins are missing or mismatched, the system prevents Unified Checkout from loading and displays a client-side error message.

You must::

Allowed Card Networks

Use the **allowedCardNetworks** field to define the card types. Click to Pay supports American Express, Mastercard, and Visa. The Click to Pay Drop-In UI manually captures the other card types that are listed in the capture context request. This enables you to process the payment through the chosen gateway but the cardholder is not able to enroll these cards in Click to Pay.

These card networks are available for card entry:

- American Express
- Carnet
- Cartes Bancaires
- China UnionPay
- Diners Club
- Discover
- EFTPOS
- ELO
- Jaywan
- JCB

- KCP
- mada
- Maestro
- Mastercard
- Meeza
- PayPak
- UATP
- Visa

To support dual-branded or co-badged cards, you must list your supported card types values for the **allowedCardNetworks** field based on your preference for processing card numbers. For example, if a card is dual-branded as Visa and EFTPOS and EFTPOS is listed first, the card type is set to EFTPOS after the card number is entered in your Unified Checkout card collection form. For information on dual-branded or co-badged cards, see Dual-Branded Cards (on page 450).

When a Cartes Bancaires dual-branded card is entered in the Click to Pay Drop-In UI, the Click to Pay Drop-In UI provides a radio selector button to enable the cardholder to select which scheme they want to use to process the payment. The radio selector defaults to the card scheme that appears first in the **allowedCardNetworks** field.

Cartes Bancaires is not supported for Click to Pay. If a cardholder selects to process a payment with Cartes Bancaires it is processed as a one-time guest checkout and the user is not enrolled in Click to Pay. If a cardholder chooses to process with Visa or Mastercard instead of Cartest Bancaires, they are given the option to enroll their card in Click to Pay.

Important: Some card types, such as KCP and UATP, do not have security codes (CVV or CVN). If you include only card types that do not have security codes in the **allowedCardNetworks** field, Unified Checkout does not display the security code field in the UI.

If you include card types that do not have security codes and cards types that do have security codes in the **allowedCardNetworks** field, Unified Checkout displays the security code field in the UI. The field is disabled in the UI when the cardholder enters a card number for a card type with no security code.

Include Card Prefix

You can control the length of the card number prefix to be received in the response to the capture context `/sessions` request:

- 6 digits
- 8 digits
- no prefix at all

> **Important:** When you request the card number prefix for a Click to Pay tokenized credential, 6 digits are returned. Click to Pay does not return 8 digits.

To specify your preferred card number prefix length, include or exclude the **transientTokenResponseOptions.includeCardPrefix** field in the capture context `/sessions` request.

If you want to receive a 6-digit card number prefix in the response

- Do not include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context `/sessions` request.
- This example shows how a 6-digit card number prefix `411111` is returned in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111”,
                    "bin" : "411111"
```

If you want to receive an 8-digit card number prefix in the response

- Include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request, and set the value to `true`.

> **Important:** Per PCI DSS requirements, this requirement applies only to card numbers longer than 15 digits and for Discover, JCB, Mastercard, UnionPay, and Visa brands.
>
> - If the card type entered is not part of these brands, a 6-digit card number prefix is returned instead.
> - If the card type entered is not part of these brands but is *co-branded* with these brands, an 8-digit card number prefix is returned.

- This example shows how an 8-digit card prefix `41111102` is returned in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111”,
                    "prefix" : "41111102"
```

If you do not want to receive a card number prefix in the response

- Include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request, and set the value to `false`.
- This example shows how a card number is returned without a card number prefix in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111"
```

Best practice: If your application does not require card number prefix information for routing or identification purposes, Cybersource recommends that you include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request and set its value to `false`. Doing so limits the exposure of payment data to only what is necessary for your processing needs.

For more information about PCI DSS, see *Frequently Asked Questions* on the PCI Security Standards Council site.

Allowed Payment Types

You can specify the type of Unified Checkout digital payment methods that you want to accept in the capture context.

Use the **allowedPaymentTypes** field to define the payment type. The Click to Pay Drop-In UI accepts these payment types:

- `CLICKTOPAY`
- `PANENTRY`

Important: When you include `CLICKTOPAY`, it supports both Click to Pay and `PANENTRY` in the UI.

Important: When integrating with Cybersource APIs, Cybersource insists 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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Features

This section includes information on the features that are supported in Click to Pay.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Allowed Card Networks

Use the **allowedCardNetworks** field to define the card types.

These card networks are available for card entry:

- American Express (supported on Click to Pay)
- Cartes Bancaires
- Carnet

- China UnionPay
- Diners Club
- Discover
- EFTPOS
- ELO
- Jaywan
- JCB
- JCrew
- KCP
- mada
- Maestro
- Mastercard (supported on Click to Pay)
- Meeza
- PayPak
- UATP
- Visa (supported on Click to Pay)

To support dual-branded or co-badged cards, you must list your supported card type values for the **allowedCardNetworks** field based on your preference for processing card numbers. For example, if a card is dual-branded as Visa and Cartes Bancaires, and Cartes Bancaires is listed first, the card type is set to Cartes Bancaires after the card number is entered in your Unified Checkout card collection form. For information on dual-branded or co-badged cards, see Dual-Branded Cards (on page 450).

Important:

Some card types, such as KCP and UATP, do not have security codes (CVV or CVN). If you include only card types that do not have security codes in the **allowedCardNetworks** field, Unified Checkout does not display the security code field in the UI.

If you include card types that do not have security codes and cards types that do have security codes in the **allowedCardNetworks** field, Unified Checkout displays the security code field in the UI. The field is disabled in the UI when the cardholder enters a card number for a card type with no security code

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Target Origins

The target origin is defined by the scheme (protocol), hostname (domain), and port number (if used).

You must use the https:// protocol. Sub domains must also be included in the target origin.

Any valid top-level domains, such as .com, .co.uk, and .gov.br, are supported. Wildcards are not supported.

For example, if you are launching Unified Checkout on example.com, the target origin could be any of the following:

- https://example.com
- https://subdomain.example.com
- https://example.com:8080

When you use Unified Checkout in an iframe, you must include the domain for the URL that loads the iframe and the iframe URL in the **targetOrigins** field.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Allowed Payment Types

You can specify the type of Unified Checkout digital payment methods that you want to accept in the capture context.

Use the **allowedPaymentTypes** field to define the payment type:

- `CLICKTOPAY`
- `PANENTRY`

Important: Click to Pay accepts American Express, Mastercard, and Visa for saved cards. Visa and Mastercard tokenize payment credentials using network tokenization for all Click to Pay requests. Click to Pay uses Click to Pay Token Requester IDs (TRIDs) rather than your existing TRIDs to generate network tokens.

For more information on enabling and managing Click to Pay, see Enabling Click to Pay in the Business Center (on page 170).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Auto-check Enrollment

You can have the Click to Pay box pre-checked when a user is manually entering their card details and Click to Pay is enabled. The customer can uncheck the box if necessary, which means the request is processed as a one-time manual PAN transaction. This is available when you set the **billingType** field to `PARTIAL` or `FULL` in the capture context. This ensures that the customer's billing country can be validated in the UI.

Click to Pay enrollment pre-check is available in these countries:

- Argentina
- Brazil
- Chile
- Colombia
- Kuwait
- Mexico

- Peru
- Qatar
- Saudi Arabia
- South Africa
- Ukraine
- United Arab Emirates

```
"paymentConfigurations": {
    "CLICKTOPAY": {
      "autoCheckEnrollment": true
    }
  }
```

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

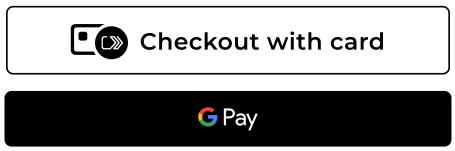
Button Type

When Unified Checkout loads, the payment buttons displayed are based on what you include in the **allowedPaymentTypes** object in the capture context. Unified Checkout enables you to customize the text on the payment buttons. You can do this by setting the **buttonType** field object in the capture context to one of these values:

- `ADD_CARD`
- `CARD_PAYMENT`
- `CHECKOUT_AND_CONTINUE`
- `DEBIT_CREDIT`
- `DONATE`

- `PAY`
- `PAY_WITH_CARD`
- `SUBSCRIBE_WITH_CARD`

If you do not include the **buttonType** field in your request, the payment button text defaults to **Checkout with card**. For example:



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Customize Button Text

Use the **buttonType** field to customize the text on payment buttons:

Button Text Options

buttonType Value	Button Display Text
`ADD_CARD`	Add card
`CARD_PAYMENT`	Card payment
`CHECKOUT_AND_CONTINUE`	Checkout and continue
`DEBIT_CREDIT`	Debit or credit
`DONATE`	Donate
`PAY`	Pay
`PAY_WITH_CARD`	Pay with card
`SUBSCRIBE_WITH_CARD`	Subscribe with card

When you do not include this field in your request, the default button text is “Checkout with card.”

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Capture Mandate

The capture mandate enables you to define which fields are captured within Unified Checkout. You must include the fields and set the values in the capture context based on the information that you want Unified Checkout to collect. This enables the cardholder to review and edit their details where the UI includes these fields. When the UI is used to capture cardholder information, all captured information is available within the payment details API response. When you want the cardholder to review existing address data, you can include the known customer data in the capture context and this information is pre-filled in the Unified Checkout UI. For information about the payment details API, see Payment Details API (on page 451).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.comboCard

A combo card is a single card in Brazil that functions as both a debit and a credit card. Unified Checkout enables the cardholder to choose whether to pay for a transaction using a debit or credit card. The cardholder can choose the card that they want to use when they enter their card details or when they choose a stored Visa card from their Click to Pay wallet during checkout. While in the card details section of the payment form, the cardholder is prompted for a debit or credit card. Credit is the default option.

To enable combo cards during checkout, you must include the **comboCard** field in your capture context request and set the field value to `true`. When the **comboCard** field value is set to `true`, the option to use a debit or credit card appears for all Visa cards that are entered in Unified Checkout and for all cards that are already stored in Click to Pay. If you do not want to offer a combo card at checkout, do not include the **comboCard** field in your capture context request:

```
"captureMandate" : {
      "comboCard": true
}
```

Important: This feature is available only in Brazil.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.CPF

The Cadastro de Pessoas Físicas (CPF) Brazilian tax ID feature is for customers in Brazil and provides your customers with a way to include their Consumer National Identifier when it is requested at checkout. Include this field in the capture context to display this field within the flow for manual card entry and Click to Pay transactions:

```
"captureMandate" : {
    "CPF": {
        "required": true
    }
}
```

Important: This feature is available only in Brazil.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.requestSaveCredentials

This feature enables you to display a consent option in the Unified Checkout UI for the cardholder to save their payment details for future use.

When you use this field without using the complete mandate, the transient token payload includes the **consumerPreference.saveCard** field with the value set to `true` when the cardholder has checked to save the payment information for future purchases:

```
"captureMandate" : {
      "requestSaveCredentials": true
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.showConfirmationStep

When **showConfirmstionStep** is set to `false`, you can remove the final summary confirmation screens from the checkout experience. When the UI displays cardholder data, the cardholder can review and, if necessary, edit their payment details before checkout is complete.

```
{
  "captureMandate": {
    "showConfirmationStep": false
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.billingType

`PARTIAL`: Only the billing postal code and billing country are collected in the UI. Set to this value when you use relaxed address verification services (AVS). This includes markets where postal code and billing country are enough for successful payment processing.

`NONE`: No fields are shown in the UI to capture cardholder billing details. If you are using the Complete Mandate, you must provide billing details in the capture context. All information that is collected from these fields is tokenized in the transient token and sent for payment processing. For information about which fields are required for payment processing, see the Payments Developer Guide.

`FULL`: These fields are shown in the UI to capture cardholder billing details. When you include the billing details in the capture context, these details are pre-filled in the Unified Checkout UI. All information that is collected from these fields are tokenized in the transient token and sent for payment processing where the Complete Mandate is used.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.requestEmail

`false`: No email address is shown in the UI. If you are using Click to Pay, this email address is used to find the cardholder's Click to Pay account and it appears in the UI when **requestEmail** is set to `false`.

`true`: The email address is shown and captured in the UI. If you are using Click to Pay, this email address is used to find the cardholder's Click to Pay account.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.requestPhone

`false`: No phone number is shown or captured in the UI.

`true`: The phone number is shown and captured in the UI.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.requestShipping

`false`: No shipping information is captured in the UI. When shipping details are required for payment processing and are used for follow on services such as Decision Manager, you can include these fields in the capture context. These details are tokenized and passed through.

`true`: Shipping fields are shown in the UI and are collected by Unified Checkout. When you include the shipping details in the capture context, the information appears prefilled in the UI.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

captureMandate.shipToCountries

When the **requestShipping** field is set to `true`, only the countries that are included in this field can be selected by the cardholder for their shipping address.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Include Card Prefix

You can control the length of the card number prefix to be received in the response to the capture context `/sessions` request:

- Six digits
- Eight digits
- No prefix

To specify your preferred card number prefix length, include or exclude the **transientTokenResponseOptions.includeCardPrefix** field in the capture context `/sessions` request.

To receive a six-digit card number prefix in the response, follow this step:

Do not include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context `/sessions` request.

This example shows how a six-digit card number prefix `411111` is returned in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111”,
                        "bin" : "411111"
```

To receive an eight-digit card number prefix in the response, follow this step:

Include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request, and set the value to `true`.

Important: This PCI DSS requirement applies only to card numbers longer than 15 digits and only for Discover, JCB, Mastercard, UnionPay, and Visa brands.

- If the card type entered is not part of these brands, a six-digit card number prefix is returned instead.
- If the card type entered is not part of these brands but is *co-branded* with these brands, an eight-digit card number prefix is returned.

This example shows how an eight-digit card prefix `41111102` is returned in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111”,
                        "prefix" : "41111102"
```

To not receive a card number prefix in the response, follow this step:

Include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request, and set the value to `false`.

This example shows how a card number is returned without a card number prefix in the transient token response:

```
"maskedValue" : "XXXXXXXXXXXX1111"
```

Best practice: If your application does not require card number prefix information for routing or identification, Cybersource recommends that you include the **transientTokenResponseOptions.includeCardPrefix** field in the capture context request and set its value to `false`. Doing so limits the exposure of payment data to only what is necessary for your processing needs.

For more information about PCI DSS, see *Frequently Asked Questions* on the PCI Security Standards Council site.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Email Autolookup

When you include Click to Pay as an **allowedPaymentType**, an automatic email lookup occurs when an email address is included in the capture context request in the **data.billTo.email** field.. If the user has a Click to Pay account but is not on a recognized device, a one-time password (OTP) screen appears and the user is prompted to enter their OTP. If the user does not have a Click to Pay account, the user must enter their card information manually. They will have the option to create a Click to Pay account.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Mobile as Identity for Click to Pay

Click to Pay supports mobile numbers as way to identify a user. This enables cardholders to use their mobile number instead of their email address in certain markets for Visa and Mastercard transactions.

When the **requestEmail** field is set to `false` and the **requestPhone** field is set to `true`, the cardholder is identified using the provided mobile number. When the **requestEmail** field is set to `true` and the **requestPhone** field is set to `false`, the cardholder is identified using the provided email address. When the **requestEmail** field is set to `true` and the **requestPhone** field is also set to `true`, the cardholder is identified using the provided email address first and then the mobile number if there is no match.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

UI/UX Customization Look and Feel

UI/UX Customization Look and Feel

Click to Pay Drop-In UI supports appearance customization using the **appearance** field object. You can customize the theme, button configuration, color styling, input states, and typography. All customization fields are optional and can be configured in the API in the **appearance.variables** field object. For a complete list of customizable fields, see Customization Matrix (on page 516).

This is an example JSON configuration:

```
{
  "appearance": {
    "theme": "LIGHT",
    "buttonType": "CHECKOUT",
    "variables": {
      "backgroundColor": "#FFFFFF",
      "textColor": "#000000",
      "headerBackground": "#1A237E",
      "headerForeground": "#FFFFFF",
      "inputBackground": "#FAFAFA",
      "buttonBackground": "#E0E0E0",
      "buttonForeground": "#333333",
      "fontFamily": "Roboto Slab, serif"
    }
  }
}
```

This is an example sessions capture context request with UI/UX customization:

```
{
  "targetOrigins": [
    "https://yourCheckoutPage.com"
  ],
  "allowedCardNetworks": [
    "VISA",
    "MASTERCARD",
    "AMEX"
  ],
  "allowedPaymentTypes": [
    "CLICKTOPAY"
  ],
  "appearance": {
    "variables": {
      "backgroundColor": "#FFFFFF",
      "textColor": "#1A1A1A",
      "headerBackground": "#0A3450",
      "headerForeground": "#FFFFFF",
      "headerAvatarBackgroundColor": "#E6EEF8",
      "headerAvatarForegroundColor": "#0A2540",
      "inputBackground": "#FFFFFF",
      "inputColor": "#1A1A1A",
      "inputPlaceholderColor": "#6B7280",
      "inputBorderColor": "#D1D5DB",
      "inputBorderStyle": "solid",
      "inputBorderRadius": "6px",
      "inputHoverBackground": "#FFFFFF",
      "inputHoverColor": "#111827",
```

```
      "inputHoverPlaceholderColor": "#6B7280",
      "inputHoverBorderColor": "#9CA3AF",
      "inputHoverBorderStyle": "solid",
      "inputFocusedBackground": "#FFFFFF",
      "inputFocusedColor": "#111827",
      "inputFocusedPlaceholderColor": "#6B7280",
      "inputFocusedBorderColor": "#2563EB",
      "inputFocusedBorderStyle": "solid",
      "inputActiveBackground": "#FFFFFF",
      "inputActiveColor": "#111827",
      "inputActivePlaceholderColor": "#6B7280",
      "inputActiveBorderColor": "#2563EB",
      "inputActiveBorderStyle": "solid",
      "inputPressedBackground": "#F9FAFB",
      "inputPressedColor": "#111827",
      "inputPressedPlaceholderColor": "#9CA3AF",
      "inputPressedBorderColor": "#2563EB",
      "inputPressedBorderStyle": "solid",
      "inputErrorBackground": "#FEF2F2",
      "inputErrorColor": "#991B1B",
      "inputErrorPlaceholderColor": "#B91C1C",
      "inputErrorBorderColor": "#DC2626",
      "inputErrorBorderStyle": "solid",
      "inputValidBackground": "#F0FDF4",
      "inputValidColor": "#065F46",
      "inputValidPlaceholderColor": "#047857",
      "inputValidBorderColor": "#16A34A",
      "inputValidBorderStyle": "solid",
      "buttonBackground": "#2563EB",
      "buttonForeground": "#FFFFFF",
      "buttonShape": "rect",
      "buttonBorderColor": "#2563EB",
      "buttonBorderStyle": "solid",
      "buttonBorderRadius": "8px",
      "buttonHoverBackground": "#1D4ED8",
      "buttonHoverForeground": "#FFFFFF",
      "buttonHoverBorderColor": "#1D4ED8",
      "buttonHoverBorderStyle": "solid",
      "buttonFocusBackground": "#1E40AF",
      "buttonFocusForeground": "#FFFFFF",
      "buttonFocusBorderColor": "#1E40AF",
      "buttonActiveBackground": "#1E3A8A",
      "buttonActiveForeground": "#FFFFFF",
      "buttonActiveBorderColor": "#1E3A8A",
      "buttonActiveBorderStyle": "solid",
      "buttonDisabledBackground": "#E5E7EB",
      "buttonDisabledForeground": "#9CA3AF",
      "buttonDisabledBorderColor": "#E5E7EB",
      "fontFamily": "Inter, Arial, sans-serif",
```

```
      "borderRadius": "8px",
      "paymentSelectionBackground": "#F9FAFB"
    }
  },
  "country": "US",
  "locale": "en_US",
  "captureMandate": {
    "billingType": "FULL",
    "requestEmail": true,
    "requestPhone": true,
    "requestShipping": true,
    "shipToCountries": [
      "US",
      "GB"
    ],
    "showAcceptedNetworkIcons": true
  },
  "data": {
    "orderInformation": {
      "amountDetails": {
        "totalAmount": "21.00",
        "currency": "USD"
      }
    }
  }
}
```

Requesting the Capture Context Using the Sessions API

This section shows you how to request the capture context.

Endpoint

Production: `POST https://api.cybersource.com/uc/v1/sessions`

Test: `POST https://apitest.cybersource.com/uc/v1/sessions`

Production in Saudi Arabia: `POST https://api.sa.cybersource.com/uc/v1/sessions`

Test in Saudi Arabia: `POST https://apitest.sa.cybersource.com/uc/v1/sessions`

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Required Fields for Requesting the Capture Context

Use these required fields to request the capture context:

Required Fields for Requesting the Capture Context

Your capture context request must include these fields:

allowedPaymentTypes

Set to `CLICKTOPAY`.

country

data.orderInformation.amountDetails.currency

data.orderInformation.amountDetails.totalAmount

locale

targetOrigins

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

Related Information

- API field reference guide for the REST API

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

REST Example: Requesting the Capture Context

Request

```
{
    "targetOrigins": [
      "https://unified-payments.appspot.com"
    ],
    "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
    },
    "data": {
      "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

eyJraWQiOiJ6dSIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsImNvbXBsZXRlUC
vVKrasZtxHFGQUz-3F16j8y85p2fNElUwzl1s12dbPaA6IgsvZ47k2QGBmYcVq7aBAb9ia4zhORqHPb8B_gWuZoa
vEOaepZTJUcjH4g8DXlaOlwO0h8bHQTpLnjvHGyBU0ltNdePR-FoPUn0ZjnE22H-
Mg0vncpE1V2qymtBMEXiESZjrz1SbYx3Wp1oYhCOvnikyxs4yH1eJPqrlRUw7eyPyRGjjecTe1S3aoA

Decrypted Capture Context Header

```
{
          "kid": "zu",
          "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": // taken dynamically from response ,
        "clientLibraryIntegrity": //taken dynamically from response:
 "sha256-cQ1t6GQcN5El4ml1H10eaSV+TuS/hFryblLLl9s/xjY="
      },
      "type": "gda-0.10.0"
    }
  ],
  "iss": "Flex API",
  "exp": 1765827144,
  "iat": 1765826244,
  "jti": "k7oy3rhyKnLr44pf"
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Validating the Capture Context

The capture context that you generate is a JSON Web Token (JWT) data object. The JWT is digitally signed using a public key and confirms the validity of the JWT and that it comes from Cybersource. When you do not have a key in the JWT header, Cybersource recommends that you follow cryptography best practices and validate the capture context signature.

To validate a JWT, you must obtain its public key. This public RSA key is in JSON Web Key (JWK) format. The 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, you must retrieve the key ID associated with the public key and then pass the key ID to the `/flex/v2/public-keys` endpoint:

1. From the header of the capture context, get the key ID (**kid**):

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

2. Send a GET request to the `/flex/v2/public-keys` endpoint and include the key ID. For example:

   - **Test:** `GET https://apitest.cybersource.com/flex/v2/public-keys/{3g}`
   - **Production:** `GET https://api.cybersource.com/flex/v2/public-keys/{3g}`
   - **Production in Saudi Arabia:** `GET https://api.sa.cybersource.com/flex/v2/public-keys/{3g}`
   - **Test in Saudi Arabia:** `GET https://apitest.sa.cybersource.com/flex/v2/public-keys/{3g}`

   Depending on the cryptographic method you use to validate the public key, you might need to convert the key to privacy-enhanced mail (PEM) format.

3. The resource returns the public key:

   ```
   eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsImRhdGEiOiI2bUFLNTNPNVpGTUk5Y3Rob
   WZmd2doQUFFRGNqNU5QYzcxelErbm8reDN6WStLOTVWQ2c5bThmQWs4czlTRXBtT21zMmVhbEx5Nk
   hHZ29oQ0JEWjVlN3ZUSGQ5YTR5a2tNRDlNVHhqK3ZoWXVDUmRDaDhVY1dwVUNZWlZnbTE1UXVFMkE
   iLCJvcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbSIsImp3ayI6eyJrdHki
   ```

```
OiJSU0EiLCJlIjoiQVFBQiIsInVzZSI6ImVuYyIsIm4iOiJyQmZwdDRjeGlkcVZwT0pmVTlJQXcwU
1JCNUZqN0xMZjA4U0R0VmNyUjlaajA2bEYwTVc1aUpZb3F6R3ROdnBIMnFZbFN6LVRsSDdybVNTUE
ZIeTFJQ3BfZ0I3eURjQnJ0RWNEanpLeVNZSTVCVjNsNHh6Qk5CNzRJdnB2Smtqcnd3QVZvVU4wM1R
aT3FVc0pfSy1jT0xpYzVXV0ZhQTEyOUthWFZrZFd3N3c3LVBLdnMwNmpjeGwyV05STUIzTS1ZQ0xO
b3FCdkdCSk5oYy1uM1lBNU5hazB2NDdiYUswYWdHQXRfWEZ0ZGItZkphVUVUTW5WdW9fQmRhVm90d
1NqUFNaOHFMOGkzWUdmemp2MURDTUM2WURZRzlmX0tqNzJjTi1OaG9BRURWUlZyTUtiZ3QyRDlwWk
J1d2gzZlNfS3VRclFWTVdPelRnT3AzT2s3UVFGZ1EiLCJraWQiOiIwOEJhWXMxbjdKTUhjSDh1bkc
xc1NDUVdxN2VveWQ1ZyJ9fSwiY3R4IjpbeyJkYXRhIjp7InRhcmdldE9yaWdpbnMiOlsiaHR0cHM6
Ly93d3cudGVzdC5jb20iXSwibWZPcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlL
mNvbSJ9LCJ0eXBlIjoibWYtMC4xMS4wIn1dLCJpc3MiOiJGbGV4IEFQSSIsImV4cCI6MTYxNjc3OT
A5MSwiaWF0IjoxNjE2Nzc4MTkxLCJqdGkiOiJ6SG1tZ25uaTVoN3ptdGY0In0.GvBzyw6JKl3b2Pz
tHb9rZXawx2T817nYqu6goxpe4PsjqBY1qeTo19R-CP_DkJXov9hdJZgdlzlNmRY6yoiziSZnGJdp
nZ-pCqIlC06qrpJVEDob3O_efR9L03Gz7F5JlLOiTXSj6nVwC5mRlcP032ytPDEx5TMI9Y0hmBadJ
YnhEMwQnn_paMm3wLh2v6rfTkaBqd8n6rPvCNrWMOwoMdoTeFxku-
```

Use this public RSA key to validate the capture context.

4. Parse the JWT capture context to get the **kid** from its header:

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

5. Send a GET request to retrieve the 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"
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox

Business Center Test
Business Center Production
Customer Support

Session Validation

The session JWT is digitally signed using RS256. You must confirm that it was issued by Cybersource and has not been tampered with. Follow these steps to validate the signature:

1. Parse the session JWT header to extract the key ID (`kid`):

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

2. Retrieve the public key by sending a request to the `/flex/v2/public-keys/{kid}` endpoint:
   - **Test**: GET `apitest.cybersource.comflex/v2/public-keys/{kid}`
   - **Production**: GET `api.cybersource.comflex/v2/public-keys/{kid}`
3. Use the returned RSA public key in JSON Web Key format to verify the JWT signature.

> **Important:** Depending on the cryptographic library that tou use, you may need to convert the key to Privacy-Enhanced Mail (PEM) format.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Transient Tokens

The response to a successful customer interaction with Unified Checkout is a transient token. This is returned in the response from the **checkout.mount()** function. The transient token is a reference to the payment data collected on your behalf. Transient tokens allow secure card payments to occur without risk of exposure to sensitive payment information. The transient token is a short-term token that expires after 15 minutes. This reduces your PCI burden/responsibility and ensures that sensitive information is not exposed to your back-end systems.

Transient tokens can be included requests sent to the Payment Details API for the customer payment data that is collected.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Transient Token Format

The transient token is issued as a JSON Web Token (JWT) (RFC 7519). The payload portion of the token is a Base64URL-encoded JSON string and contains various claims. For more information, see JSON Web Tokens (on page 522).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Example: Transient Token Format

Encrypted Transient Token JWT


```
eyJraWQiOiIwMEl1NWJDT2NINVpPWjFNYldsQktodzFZeFFjSkVlZSIsImFsZyI6IlJTMjU2In0.eyJtZX
RhZGF0YSI6eyJzZXF1ZW5jZU51bWJlciI6IjEiLCJjYXJkaG9sZGVyQXV0aGVudGljYXRpb25TdGF0dXMi
OmZhbHNlLCJwYXltZW50VHlwZSI6IlNSQ01BU1RFUkNBUkQifSwiaXNzIjoiRmxleC8wMCIsInBheW1lbn
RDcmVkZW50aWFsc1JlZmVyZW5jZSI6eyJ1Y19hZ25vc3RpY19wb3J0Zm9saW9fdmFsaWQiOiJXaWUyM2NB
S3RGblFlSXpqeDRFUXgifSwiZXhwIjoxNzY1ODg1MDcyLCJ0eXBlIjoiZ2RhLTAuMTAuMCIsImlhdCI6MT
c2NTg4NDE3MywianRpIjoiMUQwMlk4Q09FQURYS1k5VjRHTjRNUDJOSVNMVjNQM1dSUUNEV0VZNDJHUjNB
RzIzTUI3WjY5NDE0NDkwQkI1MSIsImNvbnRlbnQiOnsiZGV2aWNlSW5mb3JtYXRpb24iOnsiZmluZ2VycH
JpbnRTZXNzaW9uSWQiOnt9fSwicHJvY2Vzc2luZ0luZm9ybWF0aW9uIjp7InBheW1lbnRTb2x1dGlvbiI6
eyJ2YWx1ZSI6IjAyNyJ9fSwib3JkZXJJbmZvcm1hdGlvbiI6eyJiaWxsVG8iOnsiY291bnRyeSI6e30sIm
xhc3ROYW1lIjp7fSwiZmlyc3ROYW1lIjp7fSwicGhvbmVOdW1iZXIiOnt9LCJhZGRyZXNzMSI6e30sInBv
c3RhbENvZGUiOnt9LCJsb2NhbGl0eSI6e30sImFkbWluaXN0cmF0aXZlQXJlYSI6e30sImVtYWlsIjp7fX
0sImFtb3VudERldGFpbHMiOnsidG90YWxBbW91bnQiOnt9LCJjdXJyZW5jeSI6e319LCJzaGlwVG8iOnsi
Zmlyc3ROYW1lIjp7fSwiY291bnRyeSI6e30sImxhc3ROYW1lIjp7fSwiYWRkcmVzczEiOnt9LCJwb3N0YW
xDb2RlIjp7fSwibG9jYWxpdHkiOnt9LCJhZG1pbmlzdHJhdGl2ZUFyZWEiOnt9fX0sInBheW1lbnRJbmZv
cm1hdGlvbiI6eyJjYXJkIjp7ImV4cGlyYXRpb25ZZWFyIjp7InZhbHVlIjoiMjAyNyJ9LCJudW1iZXIiOn
sibWFza2VkVmFsdWUiOiJYWFhYWFhYWFg5OTA4IiwiYmluIjoiNTE4NjAwIn0sImV4cGlyYXRpb25N
b250aCI6eyJ2YWx1ZSI6IjAzIn0sInR5cGUiOnsidmFsdWUiOiIwMDIifSwidHlwZVNlbGVjdGlvbkluZG
ljYXRvciI6eyJ2YWx1ZSI6IjEifX19fSwiXyI6IkwrYnlQL2FIRnhEUDRPc0NGNFI5RkhjbElkZkFXR0g4
VkF3NGdta0NubGRyZzQ3WDdnR1hseUZRTWM4aU5KWjBzSlNhTlFOTzM5RHVCVWdWaW1SeC9zRUJJS2VQYX
FvRVRoWDZpRjFrWWs1ZkxjWDRqU1gvTmtzb1U0bGQzU0RNTnJHcS8ybDRoM0laTjd1WEVON1g5azA5K3Fa
RFNaWVk3S1dzaVJXeXNwcHpFZE5uQy9ORGYxY1ZqNUJHZTZnN3Jjckt0THYyT1VzYUIyb3hicVkwL2VuUF
Y4N0JHakMrUk9lSmFYa3VGeml0RDVrQ0paZHdoYnorRStRVmtnejdBVTZnNG5pa0dsWDFNUklkeXVVNnY4
QlJsWG42ZEF3c252TW5pZ3Yvc05NUE4zV0hSaUhZWHZubTdramVOd2k2YW1EbkdTSzVqY3RFMmJPWDZUU0
RmWG5RSE01eVhWcFdMRHZxa0VQOVZzM2NBNmhTbTlZcHFaaXRSSjZ5OWtMSGFNemsydm9TWlhUV1JQU210
UkZ2TWdcdTAwM2RcdTAwM2QifQ.M1ttoaMyKz9NjQ7nYfhGqrt7Gga1YvUph8FH4-0aV98tNbZilEqF4AN
QHKFjNQavJ5_EKB_4cDayuwa7xyZzrz2WNXSlRS97EJYfvFAYza8cq2SpvHlR1DvJdMuYsyui-fZafdkxq
TudsAUUYJErWezliWOvCw2gi18hb3bS3V_evt8zznRdgbwd7Q1BgSmQwgnIDI-H4wdZMByMbpG1zC8Ujbv
yPB5OUQxOTCljmbsiAquSI_8LFJoasRUK9txVjezO49E_DX1ClETbnzuiUlJ6MzBlTNAtdbxGB5ELjuf8-
SSj4ojlZZTMWARllskZsx_DUtqLBUdNXKpPKEJtzg
```


Decoded Transient Token JWT - Visa PAN


```
{
  "metadata": {
    "sequenceNumber": "1",
    "ccJti": "onNpuEB7TmCjcga2",
    "cardholderAuthenticationStatus": false,
    "paymentType": "PANENTRY"
  },
  "iss": "Flex/08",
  "paymentCredentialsReference": {
```

```
    "na_partner_ctp2": "P6RXHH8Lc79oT1DKS1_Rs"
  },
  "exp": 1778665181,
  "type": "uc-1.0.0",
  "iat": 1778664281,
  "jti": "1E3QZQ85MKIWE5EU8QUT4AVC2Y4J0Z8ZEQ4C7QDKZ28171QG12DD6A0446DD7118",
  "content": {
    "clientReferenceInformation": {
      "applicationVersion": {},
      "applicationUser": {},
      "code": {},
      "applicationName": {}
    },
    "deviceInformation": {
      "fingerprintSessionId": {}
    },
    "orderInformation": {
      "billTo": {
        "country": {},
        "lastName": {},
        "firstName": {},
        "phoneNumber": {},
        "address2": {},
        "address1": {},
        "postalCode": {},
        "locality": {},
        "buildingNumber": {},
        "administrativeArea": {},
        "email": {}
      },
      "amountDetails": {
        "totalAmount": {},
        "currency": {}
      },
      "shipTo": {
        "firstName": {},
        "lastName": {},
        "country": {},
        "address1": {},
        "postalCode": {},
        "locality": {},
        "buildingNumber": {},
        "administrativeArea": {}
      }
    },
    "paymentInformation": {
      "card": {
        "expirationYear": {
          "value": "2029"
```

```
        },
        "number": {
          "maskedValue": "XXXXXXXXXXXX1111",
          "bin": "411111"
        },
        "securityCode": {},
        "expirationMonth": {
          "value": "12"
        },
        "typeSelectionIndicator": {
          "value": "1"
        },
        "type": {
          "value": "001"
        }
      }
    }
  }
}
```

Decoded Transient Token JWT - Visa Network Token

```
{
  "metadata": {
    "sequenceNumber": "1",
    "tokenizedCard": {
      "card": {
        "expirationYear": "2033",
        "maskedValue": "XXXXXXXXXXXX2700",
        "prefix": "462294",
        "expirationMonth": "08"
      }
    },
    "ccJti": "9M2EooZC767DNhvC",
    "cardholderAuthenticationStatus": false,
    "paymentType": "SRCVISA"
  },
  "iss": "Flex/08",
  "paymentCredentialsReference": {
    "na_partner_ctp2": "U3lO1Flys1fcQmAWLzchn"
  },
  "exp": 1778665930,
  "type": "uc-1.0.0",
  "iat": 1778665030,
  "jti": "1E0T0CPF4S96D8P30Y89ID69U51WRHRWCTW896BX7K75ROSI6TMV6A0449CA2D1B",
  "content": {
    "clientReferenceInformation": {
```

```
      "applicationVersion": {},
      "applicationUser": {},
      "code": {},
      "applicationName": {}
    },
    "deviceInformation": {
      "fingerprintSessionId": {}
    },
    "processingInformation": {
      "paymentSolution": {
        "value": "027"
      }
    },
    "orderInformation": {
      "billTo": {
        "lastName": {},
        "country": {},
        "firstName": {},
        "phoneNumber": {},
        "address2": {},
        "address1": {},
        "postalCode": {},
        "locality": {},
        "buildingNumber": {},
        "administrativeArea": {},
        "email": {}
      },
      "amountDetails": {
        "totalAmount": {},
        "currency": {}
      },
      "shipTo": {
        "country": {},
        "firstName": {},
        "lastName": {},
        "address1": {},
        "postalCode": {},
        "locality": {},
        "buildingNumber": {},
        "administrativeArea": {}
      }
    },
    "paymentInformation": {
      "tokenizedCard": {
        "expirationYear": {
          "value": "2033"
        },
        "transactionType": {},
        "number": {
```

```
          "maskedValue": "XXXXXXXXXXXX3278",
          "bin": "432312"
        },
        "expirationMonth": {
          "value": "08"
        },
        "type": {
          "value": "001"
        },
        "cryptogram": {}
      },
      "card": {
        "typeSelectionIndicator": {
          "value": "1"
        },
        "useAs": {
          "value": "D"
        }
      }
    }
  }
}
```

Authentication Status in **metadata** Object

The `cardholderAuthenticationStatus` object is included in the `metadata` and enables you to determine if the payload is fully authenticated. When `cardholderAuthenticationStatus` is set to `true`, the payload is fully authenticated. When `cardholderAuthenticationStatus` is set to `false`, the transaction is not authenticated.

```
"metadata": {
  "cardholderAuthenticationStatus": "true"
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

PAN BIN in **metadata** Object

The `cardDetails` object, including the PAN BIN, is included in the transient token `metadata` when a Click to Pay network token is used as a payment method. This allows you to display information about the card on invoices and see the BIN details that are linked to the underlying card.

```
"metadata": {
  "cardDetails": {
    "suffix": "9876",
    "prefix": "123456",
    "expirationMonth": "MM",
    "expirationYear": "YYYY"
  }
}
```

The `cardholderAuthenticationStatus` object is included in the `metadata` and enables you to determine if the payload is fully authenticated. When `cardholderAuthenticationStatus` is set to `true`, the payload is fully authenticated. When `cardholderAuthenticationStatus` is set to `false`, the transaction is not authenticated.

```
"metadata": {
  "cardholderAuthenticationStatus": "true"
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Dual-Branded Cards

Unified Checkout accepts dual-branded cards. To use this feature, you must include the card networks that have overlapping BIN ranges in the capture context request. For example:

```
"allowedCardNetworks": ["VISA", "MASTERCARD", "AMEX", "CARTESBANCAIRES"]
```

When a card number within an overlapping BIN range is entered, the network that is listed first in the value array for the **allowedCardNetworks** field is used. Based on the previous example, if the card number 403550XXXXXXXXXX is entered, the payment network for payment processing is Visa.

During the transaction, the card type is populated with the first network in the list, and the **detectedCardTypes** field returned in the transient token includes all of the detected card types in the transient token.

The **detectedCardTypes** field is returned in the transient token response only when more than one card type is detected.

If you include Cartes Bancaires as a supported dual-branded card type, Unified Checkout displays a radio button with Visa and Mastercard options at checkout. This enables the customer to select which payment scheme they want to use to process the payment. The radio button defaults to the card type that you specify in the capture context request, but the payment is processed using the option that the customer selects during checkout.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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, and they are described in the *Getting Started with REST Developer Guide*:

- Set Up a JSON Web Token Message
- Set Up HTTP Signature Message

Important:

Cybersource recommends that you dynamically parse the response for the fields that you are looking for when you integrate with Cybersource APIs. Cybersource may add additional fields in the future.

You must ensure that your integration can handle new fields that are returned in the response. Even though the underlying data structures do 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-details/{jti}`

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

Production in Saudi Arabia: `GET https://api.sa.cybersource.com/flex/v2/payment-details/{jti}`

Test in Saudi Arabia: `GET https://apitest.sa.cybersource.com/flex/v2/payment-details/{jti}`

The `{jti}` is the ID of the JWT within the transient token that is returned by Unified Checkout. The transient token is a JWT object that you retrieved as part of a successful capture of payment information from a cardholder.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Required Field for Retrieving Transient Token Payment Details

Your payment details request must include this field:

id

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

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

REST Example: Retrieving Transient Token Payment Details

Request

```
GET https://apitest.cybersource.com/flex/v2/payment-details/{jti}
```

Response to Successful Request

```
{
  "paymentInformation": {
    "card": {
      "expirationYear": "2026",
      "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"
    }
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Payment Credentials API

This section contains the information you need to retrieve the full payment credentials collected by Click to Pay Drop-In UI 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 470).

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:

Cybersource recommends that you dynamically parse the response for the fields that you are looking for when you integrate with Cybersource APIs. Cybersource may add additional fields in the future.

You must ensure that your integration can handle new fields that are returned in the response. Even though the underlying data structures do not change, you must also ensure that your integration can handle changes to the order in which the data is returned.

Returned Credentials

A payment account number (PAN) or network token is returned on your request depending on your payment method and Click to Pay account status:

Payment Credentials Returned by Card Type and Click to Pay Account Status

Click to Pay Account Status	American Express	Mastercard	Visa
New card not saved in Click to Pay	PAN	PAN	PAN
New card saved in Click to Pay	PAN	Network Token	Network Token
Existing card stored in Click to Pay	PAN	Network Token	Network Token

When you retrieve PAN information from the Payment Credentials API, the response includes the PAN, card expiration date, and the card verification value (CVV). When you retrieve network token information, the response includes the network token and network token cryptogram.

Important: Visa and Mastercard always attempt to provision a network token. A PAN is returned when a network token is not provisioned before checkout or when the cardholder did not request to enroll the card in Click to Pay.

Network tokens are generated in the wallet of the Click to Pay token requestor ID (TRID). When tokenization is successful, Visa and Mastercard can also complete authentication during the Click to Pay experience to acquire a fully authenticated response. For information on authentication, see Click to Pay Customer Authentication (on page 476).

Endpoint

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

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

Production in Saudi Arabia: `GET https://api.sa.cybersource.com/flex/v2/payment-credentials/{paymentCredentialsReference}`

Test in Saudi Arabia: `GET https://apitest.sa.cybersource.com/flex/v2/payment-credentials/{paymentCredentialsReference}`

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

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Required Field for Retrieving Payment Credentials

Your payment credentials request must include this field:

ReferenceID

The reference ID that is returned in the `paymentCredentialsReference.[organizationID]` field of the transient token data object after a successful Click to Pay interaction. This is an example `paymentCredentialsReference.[organizationID]` field in the transient token response payload:

```
"paymentCredentialsReference": {
    "na_partner_ctp2": "qlXufLdFrU_TIIye497LN"
  },...
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

REST Example: Retrieving Payment Credentials

Request

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

Encrypted Response to Successful Request

```
eyJhdWQiOiJwc3AiLCJzdWIiOiJwc19ocGEiLCJraWQiOiIyMDIzMDUxNC1kcmFmdC1wc3AtZW5jcnlwdC
IsI
mN0eSI6IkpXVCIsImVuYyI6IkEyNTZHQ00iLCJleHAiOjE2ODQxNDk2NjQsImFsZyI6IlJTQS1PQUVQLTI
1Ni
IsImp0aSI6IjA0NDUwNWNiLTM1ZDYtNDU2ZS05OTBlLWRkZjQwYzI5NzlhNCJ9.enhUfZJOjbMX-wZPIOb
1zj
8sFZiix6JSJyNw2i9QJ4k_hd7Iy_UMYvOmS-X1FJwjH0IQxMIblSV8XqMegIOm5dYBYdqouUfC8zq4Zm_d
sMo
Tp3m9T6z-A_eJ8MGaxqTHSf2vWiXB-EMrww2eCXPyVTBkI1OdmYIX-s85vsqYpW-s0ThlCKaGI7B4_rJKN
a7m
```

```
ou9VMBtBnfzhHLtnHDW8vsX8rLmTT76Ct2jMdIoQnlQRgEOi-zYu0Jm0gHERavUtq_7lDw9Ta73_TFw3KA
2fs
G13CURyR7ZXoZy9_nRifwHjwNVbaFRceAzXoVtvM8H8F-ZzIC8AdA1FRye7RqcK9Q.OlrMxOMDkVDU6goS
.TP
fBhm1eBfRjCSSvuT6SxFeZ3SGwOC6qX2Z4rlAEY9lOor2Q2E1CMqB6o-q6DNkGtASFONBzKtoB0yAgXBpx
3S7
2FltR8bd40qmRnPyTOAscXa3eWbP45EqZqHW58lwUtMwcBORcfSjxPnWUo-OGmKCtIgiUO4MTlBsl9HdCL
x7R
WpwsIo0pKQAuFrURHJyhdE1JUArgjNQMdQwPvCjoZ2RxTzECEqE1l0KmBGM-w8suowrnTNZl8cwVUZKzHQ
EJV
-twAGykQIIRCI3ydHfCupyUuA-5-Wvlk6nhcL3qND4JF-E3EIRpzm7WH8pCV5nzByUue-grHejg774c7fi
1eh
fTBUZ8v6X7rTZUBLL0V5343X3zQQy_G-vq5qcaJZ8AS2XWSi17r8UEHoU5emYu5QAuXy1AhL32nDRZuXzO
zQ1
9JsrTN2CD8qxU7tDpkUCEmY2GEMp4sd-rfu_2qBZDdr74tjYNgMsTIXSpgGDiwjLMJu4r460YencO6-Jwe
GCT
8woIySjBRYpX1_axxcO6I9RUTSopPbslZwq_zpy3UuDa9InlSexM--fatYfAehY857F7bFVXlnXeqr7X0_
Lri
bJsx6CWJU1ihjMVtnF-SxeE3IdpJxyFYBb7D1iL3ywFooxcGqarXU-3_CBuDHvnJFDC_iQPaeH7csb-EMe
NqF
TmFf8dWNQYG7IJDfEnrnRW_XtnczH-ZS67iVuGzGwJZDQfJZ-KLhnWr6FE1EnT1VLyXPM78WeocT7cnLXm
r9B
gevNmU3q_SV5nxlDLPuCqF0PmFNxaTjqfF2Qw_zOCvazwFWuBdUDdHilPqhj3gfsOesAJVA7VoTDw2U3zt
e3V
09KcJLaHygwPomopWOODinKzcZeWfJ39984pQa5cOMSEToGegkRZyvSxpf5PTht30uB3F3qC4cVLOu4quk
Ysr
jXqOtxg3icde7lXywfAtEZgf54jAP2Cl8JFmGWL5YnIY44-zj-GVz2C8iCN1CCUP3U4eVxz2GtxNNSXuwY
8OR
Udino4rF-OpqqdjX5F0Uw6J2D3uR9cWB4Ee3v8TIA3-tRkG4ScAcclEwjkwsILPgVLU57HOm0AnaEsznyH
rd9
-Qfz_p-UjbsaD3e-_sr56-x2UZVVL6TAMmJqmS2C55CHgkkhtHBCu-vb0KOmssopIvaQA5jK6ZoCftewE8
-98
816ZmoU8Sty05PSeK0yBlxFwTIeJxt-moszRawFuBrLAbOu72y_eeUtk1tHpHV2Db7T6XvaRD4NvOFZg8i
anY
Y6uHidoTl1ApjCp8VG9oTJ-uKWAEp9TU6qEHUswZZUIBeGTKjzBkRAQ20cZs5POb-qtjteoWo9QdnczipZ
8de
my-FSZwNRFPkeedl3oHLepeTgwVnmij9ovk0e5Wqq2GVUMe8sLa-4eEnjliIjAVUQ9YNJBeqLf6_wo3HF8
o2k
4ZgSJTuPHAuP41-D6sYrOcM6WvkCfKRTXw7ue5unri3M0Rpd2TEnzyw.TaLt6G8QyRykbrxb0iV9Jg
```


Decrypted Response Payment Credentials JWE - Visa PAN


```
{
  "aud": "na_partner_ctp2",
  "sub": "0404_testctpdiui001",
  "clientReferenceInformation": {
    "applicationVersion": "1.4.0",
```

```
    "applicationUser": "UC",
    "code": "1778664244584",
    "applicationName": "unifiedCheckout"
  },
  "deviceInformation": {
    "fingerprintSessionId": "ea5f708f-0a77-4623-bdf8-62bb656c4a1f"
  },
  "orderInformation": {
    "billTo": {
      "firstName": "firstName",
      "lastName": "lastName",
      "country": "GB",
      "phoneNumber": "07514573636",a
      "address2": "test address1",
      "address1": "test address1",
      "postalCode": "W26TT",
      "locality": "London",
      "buildingNumber": "",
      "administrativeArea": "",
      "email": "malachieuk@gmail.com"
    },
    "amountDetails": {
      "totalAmount": "100.00",
      "currency": "GBP"
    },
    "shipTo": {
      "country": "GB",
      "lastName": "lastName",
      "firstName": "firstName",
      "address1": "test address1",
      "postalCode": "W26TT",
      "locality": "London",
      "buildingNumber": "",
      "administrativeArea": ""
    }
  },
  "paymentInformation": {
    "card": {
      "expirationYear": "2029",
      "number": "FOURONE11111111111111",
      "securityCode": "123",
      "expirationMonth": "12",
      "type": "001",
      "typeSelectionIndicator": "1"
    }
  },
  "exp": 1778664881,
  "jti": "P6RXHH8Lc79oT1DKS1_Rs"
}
```

Decrypted Response Payment Credentials JWE - Visa Network Token


```
{
  "aud": "na_partner_ctp2",
  "sub": "0404_testctpdiui001",
  "clientReferenceInformation": {
    "applicationVersion": "1.4.0",
    "applicationUser": "UC",
    "code": "1778664940677",
    "applicationName": "unifiedCheckout"
  },
  "deviceInformation": {
    "fingerprintSessionId": "e8cc0b02-7eb3-476b-8253-055b39cdcb62"
  },
  "processingInformation": {
    "paymentSolution": "027"
  },
  "orderInformation": {
    "billTo": {
      "firstName": "David",
      "country": "GB",
      "lastName": "Pentland",
      "phoneNumber": "44 7514573636",
      "address2": "test address1",
      "address1": "17 Marius Avenue",
      "postalCode": "NE15 0EB",
      "locality": "Newcastle Upon Tyne",
      "buildingNumber": "",
      "administrativeArea": "",
      "email": "malachieuk@gmail.com"
    },
    "amountDetails": {
      "totalAmount": "100.00",
      "currency": "GBP"
    },
    "shipTo": {
      "lastName": "lastName",
      "firstName": "firstName",
      "country": "GB",
      "address1": "test address1",
      "postalCode": "W26TT",
      "locality": "London",
      "buildingNumber": "",
      "administrativeArea": ""
    }
  },
  "paymentInformation": {
    "tokenizedCard": {
      "transactionType": "1",
```

```
      "expirationYear": "2033",
      "number": "FOURTHREE23126880583278",
      "expirationMonth": "08",
      "type": "001",
      "cryptogram": "Ax0AAGQfGv6nK5QDMx6WgIUxYeg="
    },
    "card": {
      "typeSelectionIndicator": "1",
      "useAs": "D"
    }
  },
  "exp": 1778665630,
  "jti": "U3lO1Flys1fcQmAWLzchn"
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

JavaScript API Reference

This reference provides details about the JavaScript API for creating the Click to Pay Drop-In UI payment form.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

VAS.UnifiedCheckout(sessionJWT)

This is a factory function that initializes the SDK. It returns a frozen, immutable client interface.

VAS.UnifiedCheckout(sessionJWT) Parameters

Name	Type	Required?	Description
`sessionJWT`	`string`	Yes	Signed JSON Web Token (JWT) from the server-side session endpoint

Returns

`Promise<UnifiedCheckoutInterface>`

Errors

Returns `UnifiedCheckoutError` with reason `CAPTURE_CONTEXT_INVALID` if the JWT signature is invalid, or `UNUSED_TARGET_ORIGINS` if the current page origin is not in the JWT's `targetOrigins` list.

Example

```
const client = await VAS.UnifiedCheckout(sessionJWT);
```

UnifiedCheckoutInterface

The client object returned by `VAS.UnifiedCheckout()`. All methods throw an `Error` if called after `destroy()`.

client.createCheckout(options?)

client.createCheckout(options?) Parameters

Name	Type	Required?	Description
`options`	`CreateCheckoutOptions`	No	Configuration for the checkout

`CreateCheckoutOptions` Properties

Property	Type	Default	Description
`autoProcessing`	`boolean`	Inferred from capture context.	`false`: `mount()` returns transient token.

Returns

`Promise<Checkout>`

Example

```
const checkout = await client.createCheckout({ autoProcessing:
 false });
```

client.createTrigger(paymentType, options?)

client.createTrigger(paymentType, options?) Parameters

Name	Type	Required?	Description
`paymentType`	`AllowedPaymentType`	Yes	Support trigger of the UI from client- driven button.
`options`	`CreateTriggerOptions`	No	The configuration for the trigger.

Returns

`Trigger`

Example

```
const trigger = client.createTrigger(CLICKTOPAY);
```

client.on(event, callback)

Subscribes to a client-level event and returns an unsubscribe function.

client.on(event, callback) Parameters

Name	Type	Required?	Description
`event`	`string`	Yes	Event name. Possible values: • `*` • `created` • `destroyed` • `error`
`callback`	`function`	Yes	Handler function that receives event-specific payload.

Returns

`Unsubscribe`: A function that removes the handler when called.

Errors

Returns `Error` when `event` is not a valid event name with reason `TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED` when the payment type cannot be used with a trigger.

Example

```
const unsubscribe = client.on('error', (err) => {
  console.error(err.source, err.code, err.message);
});

// Later
unsubscribe();
```

client.off(event, callback?)

Removes an event handler. This method is permissive — calling it with an unknown event or callback does not throw.

client.off(event, callback?) Parameters

Name	Type	Required?	Description
`event`	`string`	Yes	Event name to unsubscribe from
`callback`	`function`	No	Specific handler to remove. When this is not included, all handlers for the event are removed.

client.destroy()

Permanently destroys the client. Returns a `destroyed` event, clears all event listeners, and marks the instance as destroyed.

You can call `destroy()` multiple times.

client.isDestroyed()

Returns a value of `true` if **client.destroy()** is called.

Checkout

This field is returned by `client.createCheckout()` and manages the full checkout UI lifecycle.

checkout.mount(target)

Subscribes to a client-level event and returns an unsubscribe function.

checkout.mount(target) Parameters

Name	Type	Required?	Description
`target`	`string` or `CheckoutContainers`	No	CSS selector string for sidebar mode, or an object with `paymentSelection` and `paymentScreen` for embedded mode. Omit for full sidebar

`CheckoutContainers` **Properties**

Property	Type	Default	Description
`paymentSelection`	`string`	Yes	CSS selector for the button list container
`paymentScreen`	`string`	No	CSS selector for the payment form container. If omitted, payment screens appear in sidebar mode

Returns

`Promise<string>`: This is a transient token JWT when `autoProcessing: false`.

Errors

Returns `UnifiedCheckoutError`. For information about how to handle mount error codes, see Handle Errors (on page 509).

Example

```
// Sidebar
const result = await checkout.mount('#buttons');

// Embedded
const result = await checkout.mount({
  paymentSelection: '#buttons',
  paymentScreen: '#form'
});
```

checkout.unmount()

Removes the payment UI from the page. The checkout is not destroyed — you can call `mount()` again.

checkout.isMounted()

Returns `true` when the checkout UI is mounted.

checkout.isDestroyed()

Returns `true` when `destroy()` is called.

checkout.on(event, handler)

Subscribes to a checkout-level event and returns an unsubscribe function.

Valid events:

- `mounted`
- `ready`
- `unready`
- `unmounted`
- `destroyed`
- `paymentMethodSelected`
- `paymentMethodCancelled`
- `paymentMethodUpdate"`
- `error`
- `*`

checkout.off(event, handler?)

Removes a checkout event handler.

checkout.destroy()

Permanently destroys the checkout. This field removes the payment UI, cleans up iframes, and emits a `destroyed` event.

Trigger

The trigger is returned by `client.createTrigger()` and programmatically launches a specific payment method.

trigger.mount(target?)

Launches the payment method UI.

trigger.mount(target?) Parameters

Name	Type	Required?	Description
`target`	`string`	No	CSS selector for embedded mode. Omit for sidebar mode.

Returns

`Promise<string>`: A transient token or completed payment result.

Example

```
const result = await trigger.mount('#payment-screen');
```

trigger.unmount()

Hides the payment method UI. The trigger is not destroyed.

trigger.isMounted()

Returns a boolean value.

trigger.isDestroyed()

Returns a boolean value.

trigger.on(event, handler)

Subscribes to trigger events. Same event names and payloads as checkout events.

trigger.off(event, handler?)

Removes a trigger event handler.

trigger.destroy()

Permanently destroys the trigger.

Events

Click to Pay provides a type-safe event system for monitoring the payment lifecycle. Events are emitted at the client and integration levels.

Subscribe to Events

Use `on()` to subscribe to events. this returns an unsubscribe function:

```
const unsubscribe = checkout.on('ready', (data) => {
  console.log('Ready:', data.availablePaymentMethods);
});

// Later, remove the handler
unsubscribe();
```

You can use `off()` to remove a specific handler:

```
function onReady(data) { /* ... */ }

checkout.on('ready', onReady);
checkout.off('ready', onReady);
```

Unified Checkout Configuration

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

- Upload Your Encryption Key (on page 470)
- Enable Click to Pay (on page 396)
- Manage Permissions (on page 273)

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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 can retrieve payment information from the Click to Pay Drop-In UI platform from the checkout response payloads. This information is transmitted in an encrypted format to ensure that sensitive data is secure. To retrieve and decrypt this information, you must set up message-level encryption. For information about enabling MLE, see *How to Set up REST* in the Getting Started with REST Developer Guide and follow the steps based on your integration method.

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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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. 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** > **Key Management**. The Key Management page appears.
3. Click **+Generate key**. The Create Key page appears.
4. On the Create Key page, select **Token Management MLE**.
5. Click **Generate key**.
6. Enter your MLE public key value in JWK format.

7. Click **Create key**.
8. On the Key Management page, search for your key and select it.
9. Click **Activate**. Your key is now active.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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 275).

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 274).

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Click to Pay Customer Authentication

When you enable customer authentication through Click to Pay, you give Cybersource permission to request that Visa and Mastercard provide an authenticated payload for each transaction. Authentication takes place within the authentication service of each card type. You must inspect the payload that is returned to you to determine if the transaction is authenticated.

For information about enabling customer authentication through Click to Pay, see these topics:

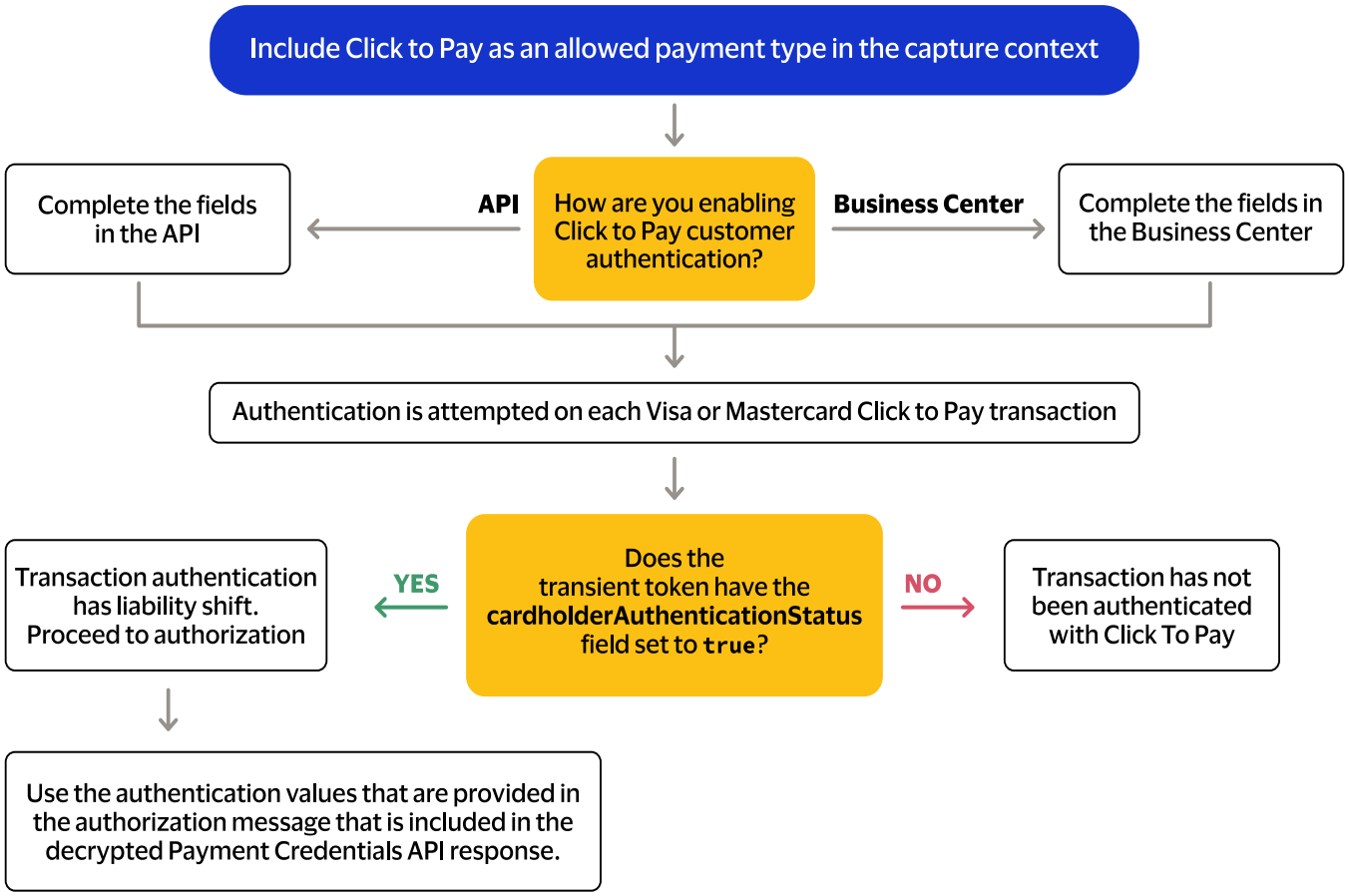
- Enable Click to Pay Customer Authentication Using the API (on page 481)
- Enable Click to Pay Customer Authentication Using the Business Center (on page 486)

When the customer completes a transaction using a Visa or Mastercard Click to Pay credentials, authentication is managed within Click to Pay. When the customer checks out using manual card entry and does not save their card to Click to Pay, the transaction is not processed through Click to Pay and you must complete authentication based on your existing authentication method.

Important: American Express cards cannot be authenticated through Click to Pay customer authentication. You must use another authentication method for this card type.

Authentication Flow

This image shows the Click to Pay authentication flow:



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

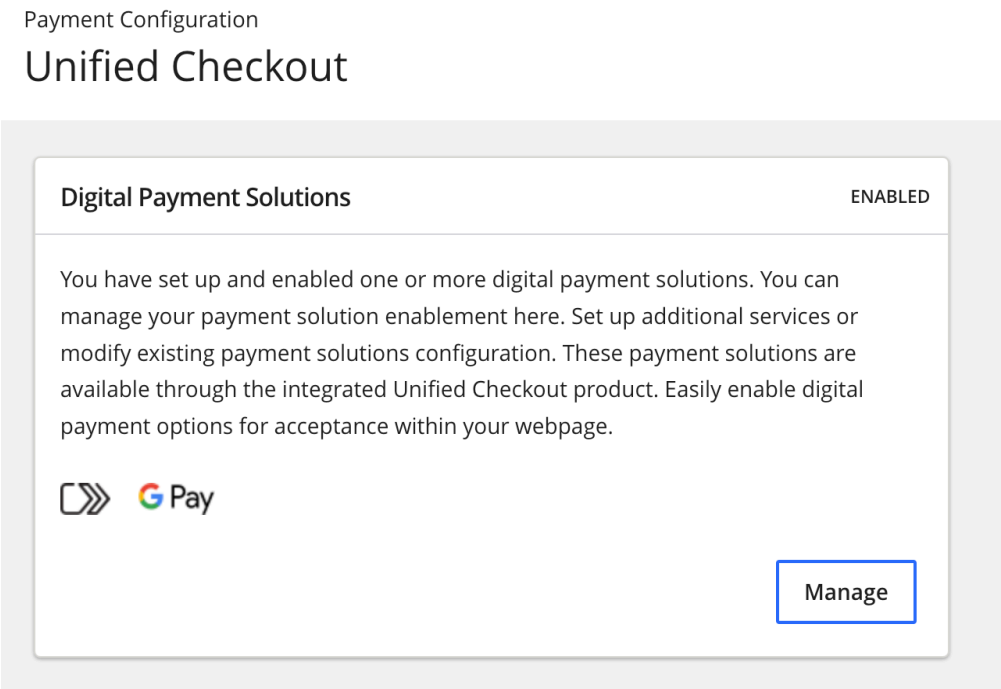
Set Up Customer Authentication for Click to Pay

Follow these steps to use the Business Center to enable customer authentication through Click to Pay. Authentication methods differ in each region and are dependent on the issuer, the cardholder device, and the Click to Pay configuration. These authentication methods are available:

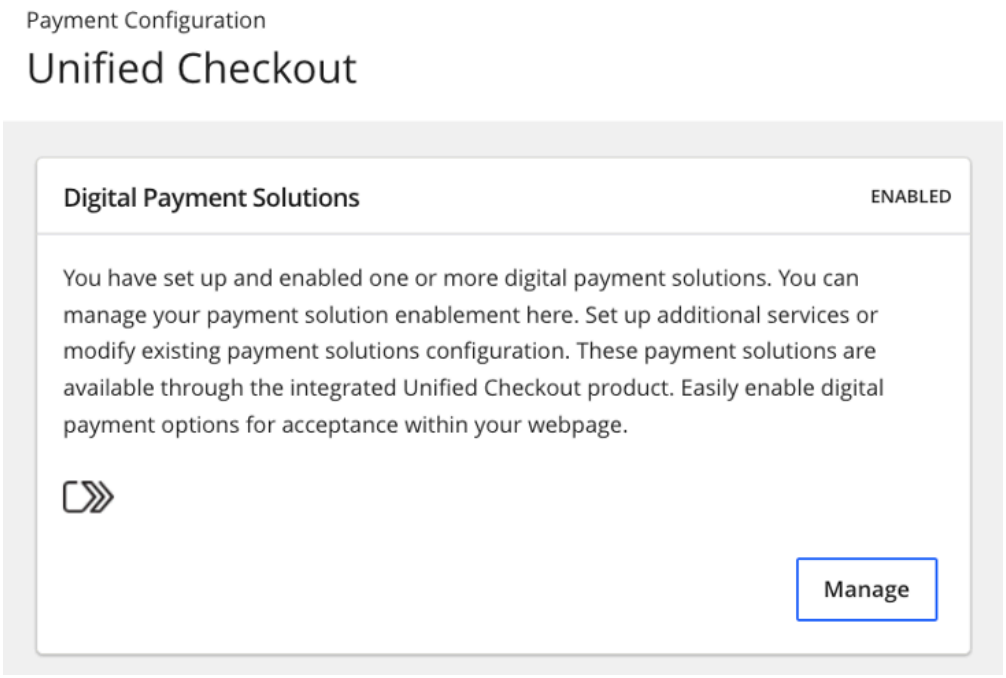
- 3-D Secure
- Payment Passkey
- Card verification value (CVV)
- One-time password (OTP)

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.

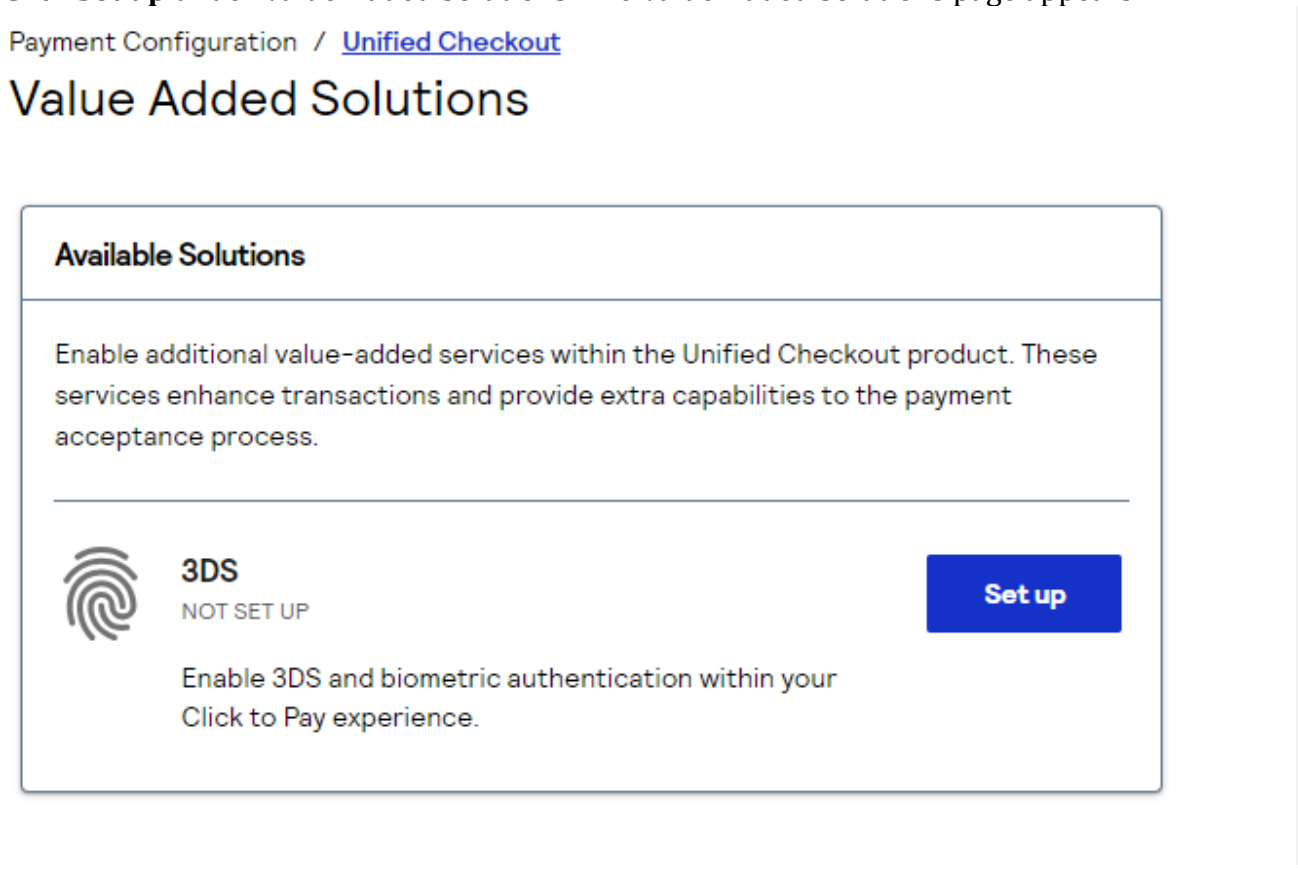
2. In the Business Center, go to the left navigation panel and choose **Payment Configuration** > **Unified Checkout**.
   You must have Click to Pay enabled as a digital payment method in order to use this method of authentication. Click **Manage** to view the digital payment methods that you have enabled.



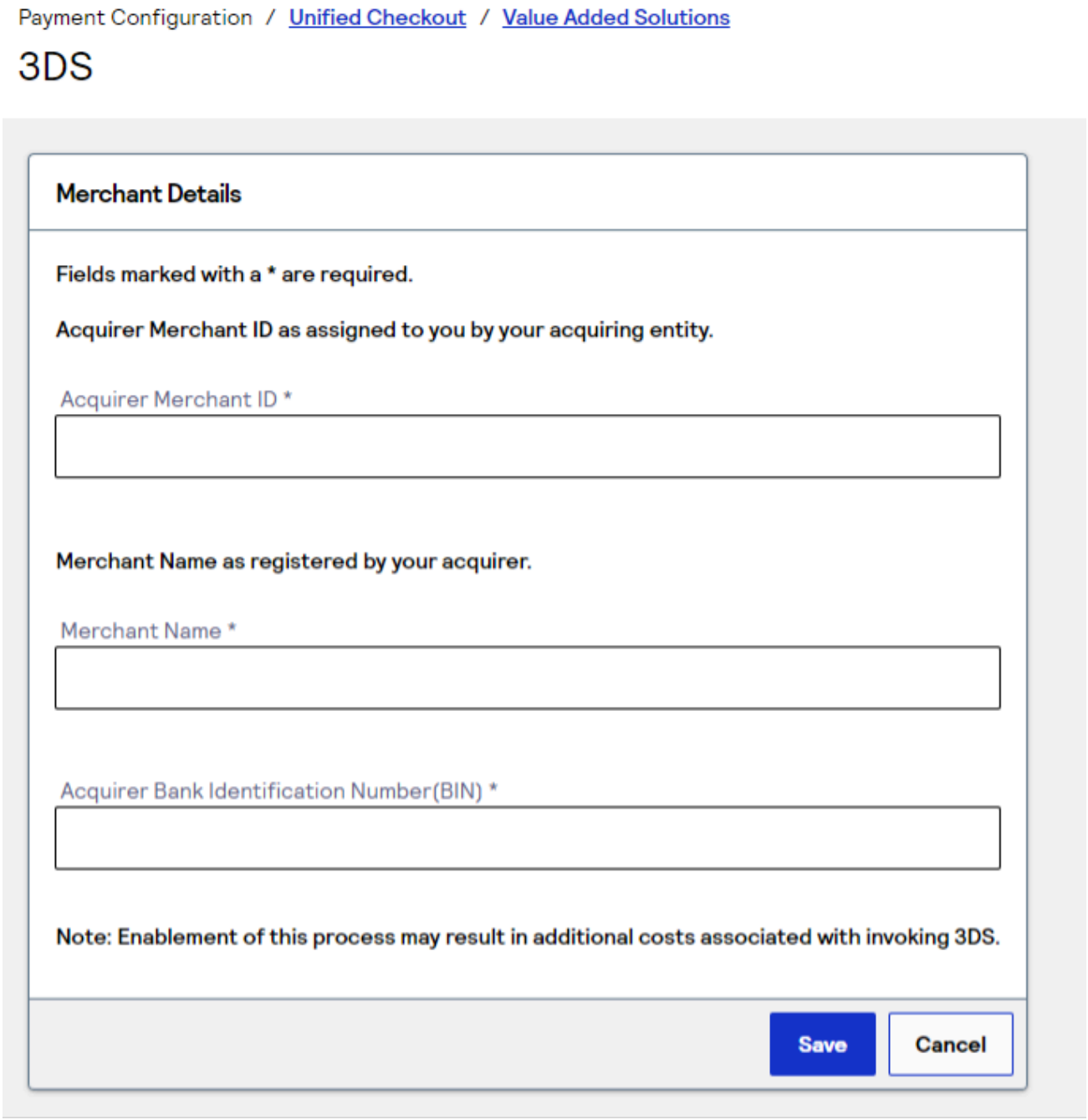
   If Click to Pay is not enabled, click **On** next to Click to Pay.



3. Click **Set up** under Value Added Solutions. The Value Added Solutions page appears.



4. Click **Set up** to set up 3-D Secure. The 3DS page appears.
5. Enter the required information in the Merchant Details section. You must enter the information that is provided to you by your acquirer or processor.



This completes the authentication setup for the entered acquirer merchant ID and BIN. If you do not know what these values are, you must contact your acquirer. Completing this information enables Cybersource to send Visa the information that is required for authentication.

Important: Charges for 3-D Secure may apply. You must speak with your acquirer for more information about the charges associated with 3-D Secure.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Enable Click to Pay Customer Authentication Using the API

This section shows you how to enable Click to Pay Authentication using the Boarding Registration Service (BRS) API.

Endpoint

Production: `POST https://api.cybersource.com/boarding/v1/registrations`

Test: `POST https://apitest.cybersource.com/boarding/v1/registrations`

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Required Fields for Enabling Click to Pay Authentication

organizationInformation.businessInformation.merchantCategoryCode

organizationInformation.businessInformation.name

organizationInformation.businessInformation.websiteUrl

organizationInformation.organizationId

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.merchantName

Required if the **enrollmentData** object is included in the request.

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.merchantURL

Required if the **enrollmentData** object is included in the request.

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.portfolioAccessofSensitiveData.merchantAccessofSensitiveData

Set to `false`.

productInformation.selectedProducts.payments.unifiedCheckout.subscriptionInformation.enabled

Required to enable Unified Checkout.

productInformation.selectedProducts.payments.unifiedCheckout.subscriptionInformation.features.clickToPay.enabled

Set to `true`.

productInformation.selectedProducts.payments.unifiedCheckout.subscriptionInformation.features.portfolioAccessofSensitiveData.enabled

Set to `true`.

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.acquirerBIN

Required for authentication with Click to Pay 3DS authentication.

productInformation.selectedProducts.payments.unifiedCheckout.configurationInformation.configurations.features.clickToPay.enrollmentData.acquirerName

Required for authentication with Click to Pay 3DS authentication.

Related Information

- API Field Reference for the REST API

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Optional Fields for Enabling Click to Pay Authentication

organizationInformation.businessInformation.address

organizationInformation.businessInformation.address.address1

organizationInformation.businessInformation.address.administrativeArea

organizationInformation.businessInformation.address.country

organizationInformation.businessInformation.address.locality

organizationInformation.businessInformation.address.postalCode

organizationInformation.configurable

organizationInformation.parentOrganizationId

This value is dependent on your organization hierarchy rules.

organizationInformation.status

organizationInformation.type

registrationInformation.boardingFlow

registrationInformation.boardingPackageId

registrationInformation.mode

Related Information

- API Field Reference for the REST API

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

REST Example: Enabling Click to Pay Authentication

Enable Click to Pay Drop-In UI

```
{
  "registrationInformation": {
    "boardingFlow": "ENTERPRISE",
    "mode": "COMPLETE",
    "boardingPackageId": "74921204027"
  },
  "organizationInformation": {
    "organizationId": "testucctp001",
    "status": "TEST",
    "businessInformation": {
      "name": "testucctp",
      "websiteUrl": "https://www.test.com",
      "merchantCategoryCode": "0742",
      "address": {
        "country": "US",
        "address1": "Test Dr",
        "postalCode": "78641",
        "administrativeArea": "TX",
        "locality": "Austin"
      }
    },
    "parentOrganizationId": "testucctp",
    "type": "TRANSACTING",
    "configurable": false
  },
  "productInformation": {
    "selectedProducts": {
      "payments": {
        "unifiedCheckout": {
          "subscriptionInformation": {
            "enabled": true,
            "features": {
              "clickToPay": {
                "enabled": true
              },
              "portfolioAccessofSensitiveData ": {
                "enabled": true
              }
            }
          },
          "configurationInformation": {
            "configurations": {
              "features": {
                "clickToPay": {
```

```
                "enrollmentData": {
                  "merchantName": "testucctp",
                  "merchantURL": "https://www.test.com",
                  "acquirerBIN": "123456",
                  "acquirerName": "ExampleAcquirer"
                }
              },
              "portfolioAccessofSensitiveData ": {
                " merchantAccessofSensitiveData ": false
              }
            }
          }
        }
      }
    }
  }
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

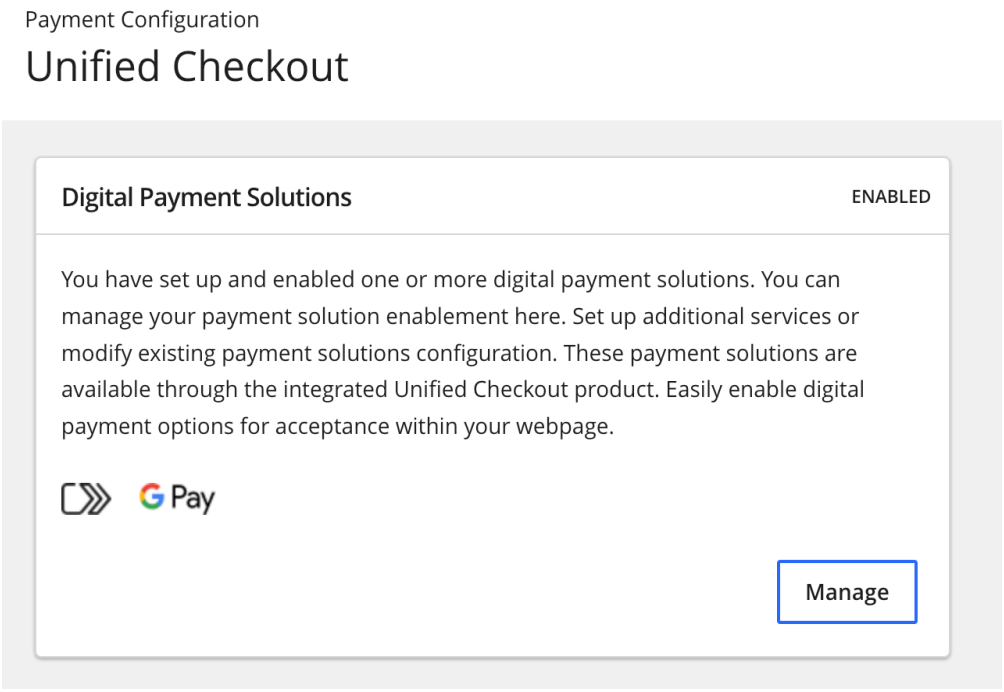
Enable Click to Pay Customer Authentication Using the Business Center

Follow these steps to use the Business Center to enable customer authentication through Click to Pay. Authentication methods differ in each region and are dependent on the issuer, the cardholder device, and the Click to Pay configuration. These authentication methods are available:

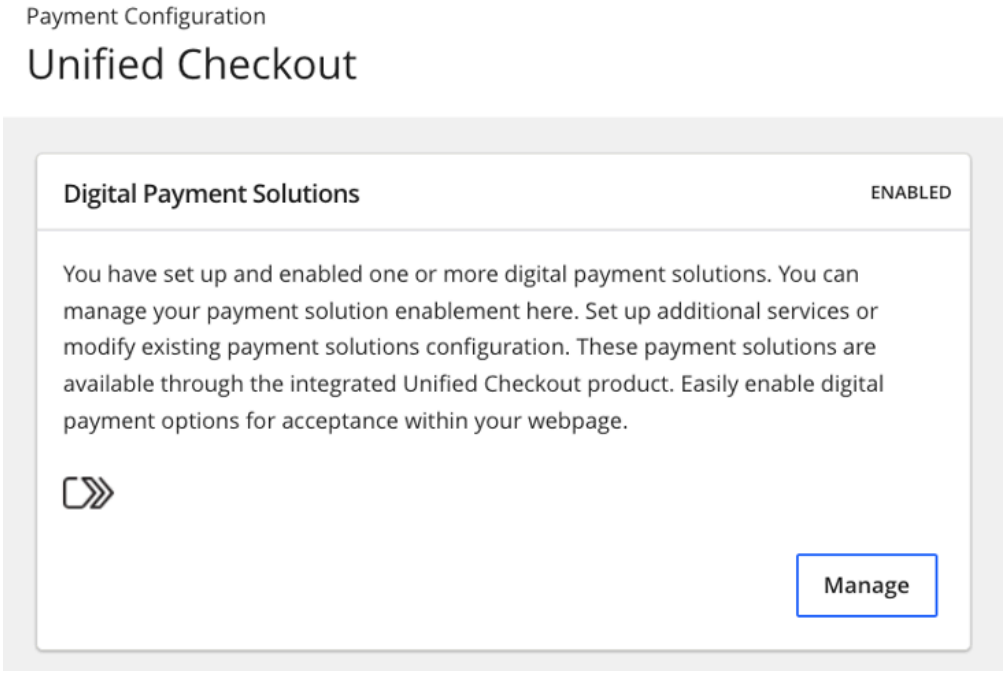
- 3-D Secure
- Payment Passkey
- Card verification value (CVV)
- One-time password (OTP)

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.

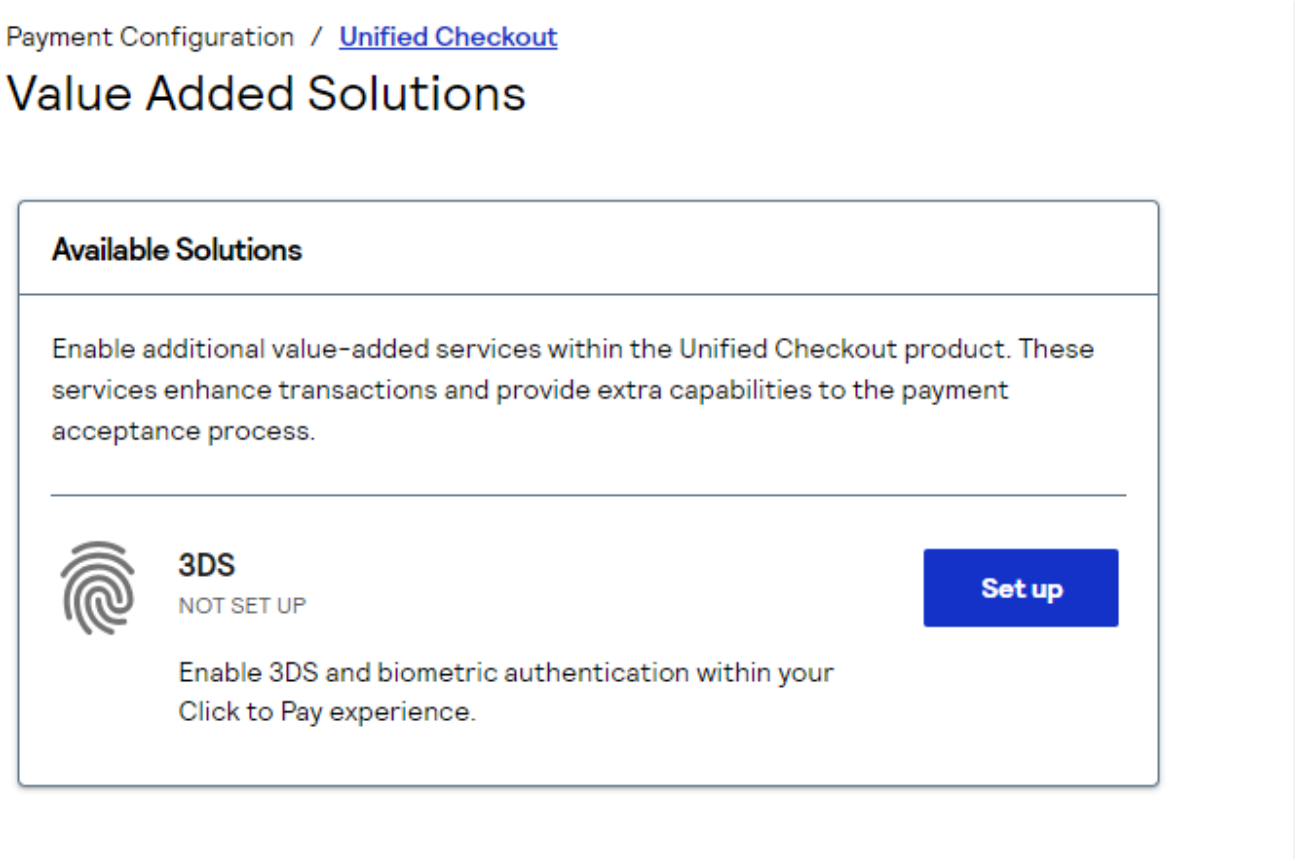
2. In the Business Center, go to the left navigation panel and choose **Payment Configuration** > **Unified Checkout**.
   You must have Click to Pay enabled as a digital payment method in order to use this method of authentication. Click **Manage** to view the digital payment methods that you have enabled.



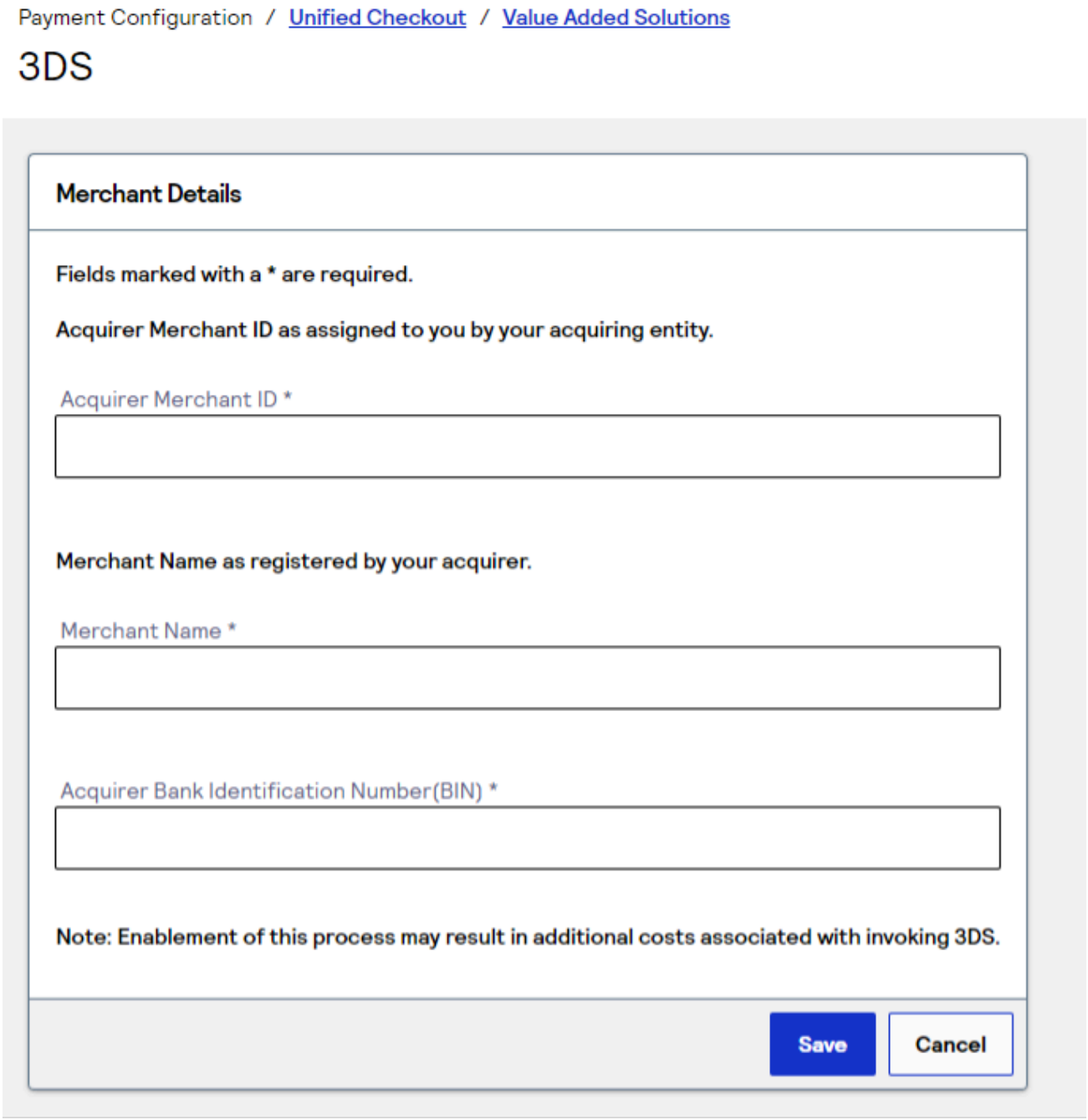
   If Click to Pay is not enabled, click **On** next to Click to Pay.



3. Click **Set up** under Value Added Solutions. The Value Added Solutions page appears.



4. Click **Set up** to set up 3-D Secure. The 3DS page appears.
5. Enter the required information in the Merchant Details section. You must enter the information that is provided to you by your acquirer or processor.



This completes the authentication setup for the entered acquirer merchant ID and BIN. If you do not know what these values are, you must contact your acquirer. Completing this information enables Cybersource to send Visa and Mastercard the information that is required for authentication.

Important: Charges for 3-D Secure may apply. You must speak with your acquirer for more information about the charges associated with 3-D Secure.

Related information

Getting Started with REST

Response Codes

API Field Reference Guide

API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Update to Click to Pay Drop-In UI Version 1

The version 1 (v1) SDK simplifies your integration with fewer lines of code, a streamlined API, and enhancements such as auto-processing and a full event system. The core flow is the same in v1, and migrating to v1 involves only straightforward method renames.

Summary of Changes

Aspect	v0	v1
Initialization	`new Accept(session).unifiedPayments()`	`VAS.UnifiedCheckout(session)`
Display the payment UI	`up.show(options)`	`checkout.mount(target)`
Events	None	Full event system on client and checkout
Cleanup	`up.dispose()`	`checkout.destroy()` + `client.destroy()`
Hide UI	`up.hide()`	`checkout.unmount()`
Target Origin	Multiple non-usable URLs can be included in the request.	If any origins are absent or mismatched, for example, they are not presented in Click to Pay Drop-In UI, the system prevents Click to Pay Drop-In UI from loading and displays a client-side error message.

Summary of v0 and v1 Changes

Feature	Pre V1 Support	V1 Support	Description
Status	✓	Business Center	
Capture context endpoint	`/up/v1/capture-contexts`	`/up/v1/sessions`	

Summary of v0 and v1 Changes (continued)

Feature	Pre V1 Support	V1 Support	Description
Capture context management	API only	API only or API and Business Center	Business Center configuration is at the merchant level.
Unified Checkout Look and Feel in Business Center	✗	✓	Business Center configuration is at the merchant level.
Unified Checkout Look and Feel Using the API	✗	✓	Configure the look and feel in a Sessions API request.
Click to Pay Configuration	API only	API or API and Business Center	Business Center configuration is at the merchant level.
Real-time preview in Business Center	✗	✓	Business Center configuration is at the merchant level.
Three-decimal currency support	✗	✓	
SDK	Legacy Unified Payments SDK supported	New UC SDK	
Payment Details API	`/up/v1/payment-details/{id}`	JTI used in place of transient token	JTI is located in the transient token
Future enhancements	Manual opt-in is required.	Automatic when the **clientVersion** is not included in the Sessions API request.	Legacy versions receive critical updates only.

Initialization

Initialization with Click to Pay Drop-In UI v1 is done in a single asynchronous factory call. There is no intermediate `Accept` object:

v0 Initialization

```
const accept = new Accept(sessionJWT);
const up = accept.unifiedPayments();
```

v1 Initialization

```
const client = await VAS.UnifiedCheckout(sessionJWT);
```

Unified Checkout v1 validates the JWT signature and target origins during initialization.

Display the Payment UI

Unified Checkout v1 passes your UI payment selectors directly to `mount()`.

v0 Display Payment UI with `show()`

```
// Sidebar
const token = await up.show({
  containers: {
    paymentSelection: '#buttons'
  }
});

// Embedded
const token = await up.show({
  containers: {
    paymentSelection: '#buttons',
    paymentScreen: '#form'
  }
});
```

v1 Display Payment UI with `mount()`

```
// Sidebar
const result = await checkout.mount('#buttons');

// Embedded
const result = await checkout.mount({
  paymentSelection: '#buttons',
  paymentScreen: '#form'
});
```

Events

Unified Checkout v0 does not include an event system, as the integration resolution or rejection from `show()` and `complete()`. v1 includes a full event system as the client and integration levels.

v1 Full Event System

```
// Client-level — centralized error tracking
client.on('error', (err) => {
  console.error(`[${err.source}] ${err.code}: ${err.message}`);
});

// Checkout-level — granular lifecycle events
checkout.on('ready', (data) => {
  console.log('Available methods:', data.availablePaymentMethods);
});

checkout.on('paymentMethodSelected', (data) => {
  console.log('Selected:', data.type);
});

checkout.on('error', (err) => {
  console.error('Checkout error:', err.code);
});
```

Cleanup

Unified Checkout v1 distinguishes between `unmount()`, which is reversible, and `destroy()`, which is permanent. Before a cleanup, `client.destroy()` sends a `destroyed` event.

v0 Cleanup

```
up.hide();     // Hide UI
up.dispose();  // Clean up resources
```

v1 Cleanup

```
checkout.unmount();   // Remove UI from page (can remount later)
checkout.destroy();   // Permanent cleanup

client.destroy();     // Destroy client and clear all event listeners
```

Handle Errors

The `UnifiedCheckoutError` class and its reason codes are the same in v0 and v1:

v0 Error Handling

```
try {
  const token = await up.show({
    containers: {
      paymentSelection: '#buttons'
    }
  });
} catch (err) {
  console.error(err.reason, err.message);
}
```

v1 Error Handling

```
// Same error class, same properties
try {
  const result = await checkout.mount('#buttons');
} catch (err) {
  console.error(err.reason, err.message);
}
```

Migrate Triggers

If your Unified Checkout v0 integration uses triggers, the migration is similar to checkout. In v1, triggers are created from the `client.createTrigger`, not from `UnifiedPayments` as in v0. In v1, `show()` is renamed to `mount()`.

v0 Triggers

```
const trigger = up.createTrigger('CLICKTOPAY', {
  containers: { paymentScreen: '#screen' }
});
const token = await trigger.show();
```

v1 Triggers

```
const trigger = client.createTrigger('CLICKTOPAY');
const result = await trigger.mount('#screen');
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Update Reason Codes

Some reason codes were renamed in v1. This table shows the v0 reason code name and the corresponding name in the v1 client-side SDK:

v0 Reason Code	v1 Reason Code
SHOW_LOAD_CONTAINER_SELECTOR	MOUNT_CONTAINER_SELECTOR
SHOW_LOAD_ERROR	MOUNT_ERROR
SHOW_LOAD_INVALID_CONTAINER	MOUNT_INVALID_CONTAINER
SHOW_LOAD_SIDEBAR_OPTIONS	MOUNT_SIDEBAR_OPTIONS
SHOW_PAYMENT_TIMEOUT	MOUNT_PAYMENT_TIMEOUT
SHOW_PAYMENT_UNAVAILABLE	MOUNT_PAYMENT_UNAVAILABLE
SHOW_TOKEN_TIMEOUT	MOUNT_TOKEN_TIMEOUT
SHOW_TOKEN_XHR_ERROR	MOUNT_TOKEN_XHR_ERROR
UNIFIED_PAYMENTS_ALREADY_SHOWN	CHECKOUT_ALREADY_MOUNTED
UNIFIED_PAYMENTS_PAYMENT_PARAMETERS	CHECKOUT_PAYMENT_PARAMETERS
UNIFIED_PAYMENTS_VALIDATION_PARAMS	CHECKOUT_VALIDATION_PARAMS

Important: The server-side API continues to return these v0 reason codes. The v1 reason codes listed here are used only in the client-side SDK. For all v1 client-side error codes, see Handle Errors (on page 509).

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Version 1 Update Checklist

You must complete these tasks before you can complete your migration from Unified Checkout v0 to v1:

- Replace `new Accept(session).unifiedPayments()` with `await VAS.UnifiedCheckout(session)`.
- Replace `up.show(options)` with `checkout = await client.createCheckout(); checkout.mount(target)`.
- Update container options: `{ containers: { paymentSelection, paymentScreen } }` becomes direct arguments to `mount()`.
- Replace `up.complete(token)` with `checkout.complete(token)` or use `autoProcessing: true` to complete transactions automatically
- Replace `up.hide()` with `checkout.unmount()`.
- Replace `up.dispose()` with `checkout.destroy()` and `client.destroy()`.
- Add event listeners for observability. For example, `client.on('error')` and `checkout.on('ready')`.

Related information
Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Click to Pay UI Guidelines

The UI that is built in Unified Checkout for Click to Pay is built based on the EMV Click to Pay XC Guidelines V1.1. Unified Checkout has simplified the integration of the UI. The only UI work that you must complete is the placement of the payment option.

Important: You must include Click to Pay as one of the presented payment methods and not as a separate payment method.

Unified Checkout captures all card details that are manually entered by the cardholder. This enables the cardholder to enroll in Click to Pay and removes the requirement for the cardholder to manually enter their card details the next time they check out.

Unified Checkout provides a standard payment label in the Unified Checkout JavaScript that is loaded in your checkout page. One of these scenarios occurs when the cardholder selects the button:

- The cardholder is recognized.
- The cardholder is not recognized but has a Click to Pay account.
- The cardholder does not have a Click to Pay account.

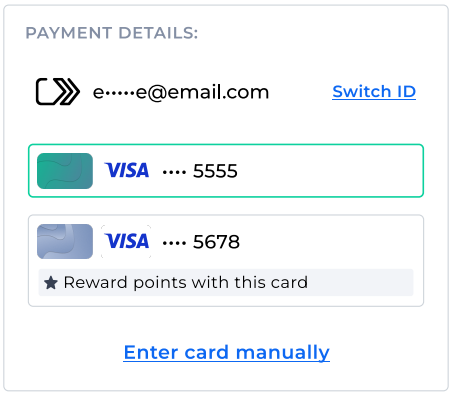
You can also trigger the Unified Checkout flow using a custom button. If you are using your own custom button, your payment button or widget must display the Click to Pay image for the cardholder. For information about a custom button, see JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry (on page 410).

Important: Your implementation consultant will ask you for a mock-up of your payment flow for confirmation that it is compliant with the Click to Pay UI design standards.

Recognized Click to Pay Customer

The cardholder is presented with their stored Click to Pay cards in the UI when they are on a recognized device:

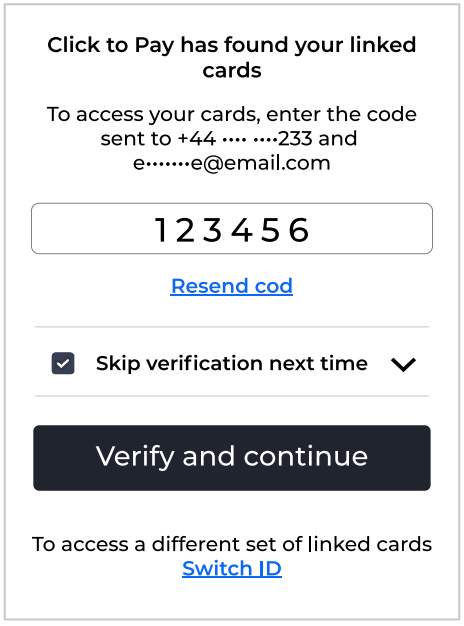
Recognized Click to Pay Customer UI



Unrecognized Click to Pay Customer

When the cardholder has a Click to Pay account but is not on a registered device, they receive a one-time password to their registered email address and phone number to authenticate their identity before their stored Click to Pay credentials are shown:

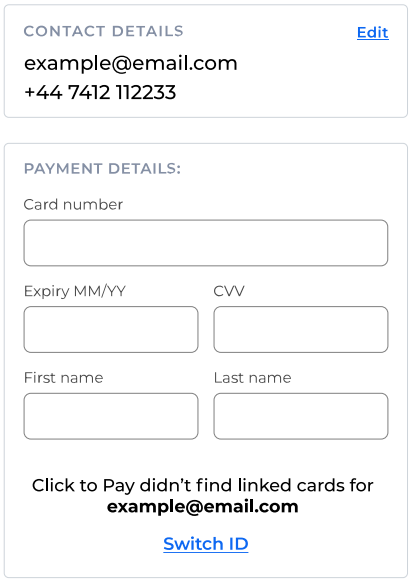
Unrecognized Click to Pay Customer on a Recognized Device UI



No Click to Pay Account

When the cardholder does not have a Click to Pay account, they can provide a new email address to perform a new lookup or they can choose to enter their card details manually. The cardholder can make a one-time payment or complete the payment and choose to create a Click to Pay account for future use:

No Click to Pay Account UI



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

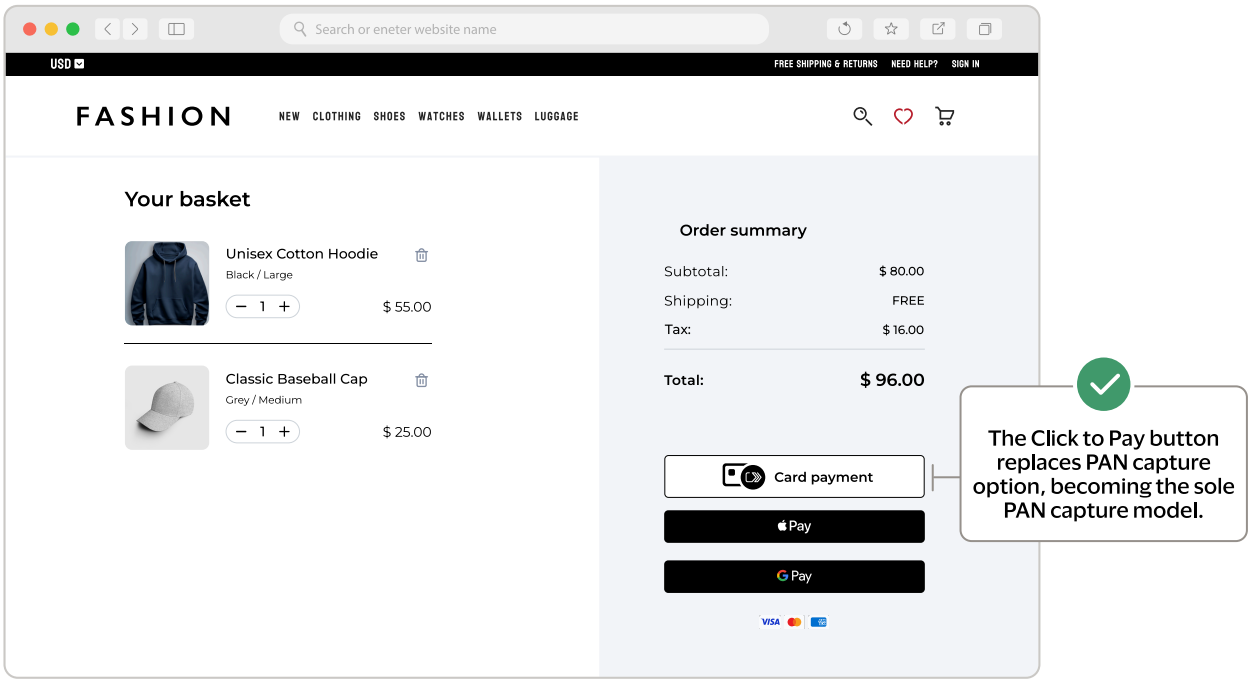
Click to Pay UI Examples

This section contains UI examples of how you should display Click to Pay on your payment page. For information about how to display the UI, see JavaScript API Reference (on page 462).

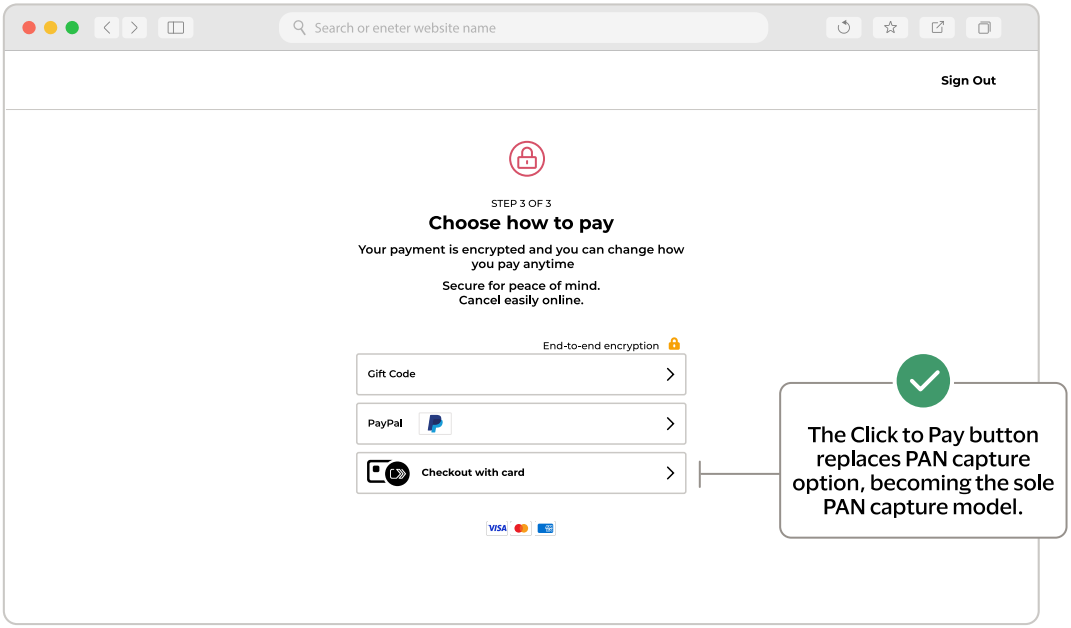
Click to Pay Replaces PAN Capture

Click to Pay is the card entry payment option within your payment page.

Click to Pay Replaces PAN Capture UI Example 1



Click to Pay Replaces PAN Capture UI Example 2

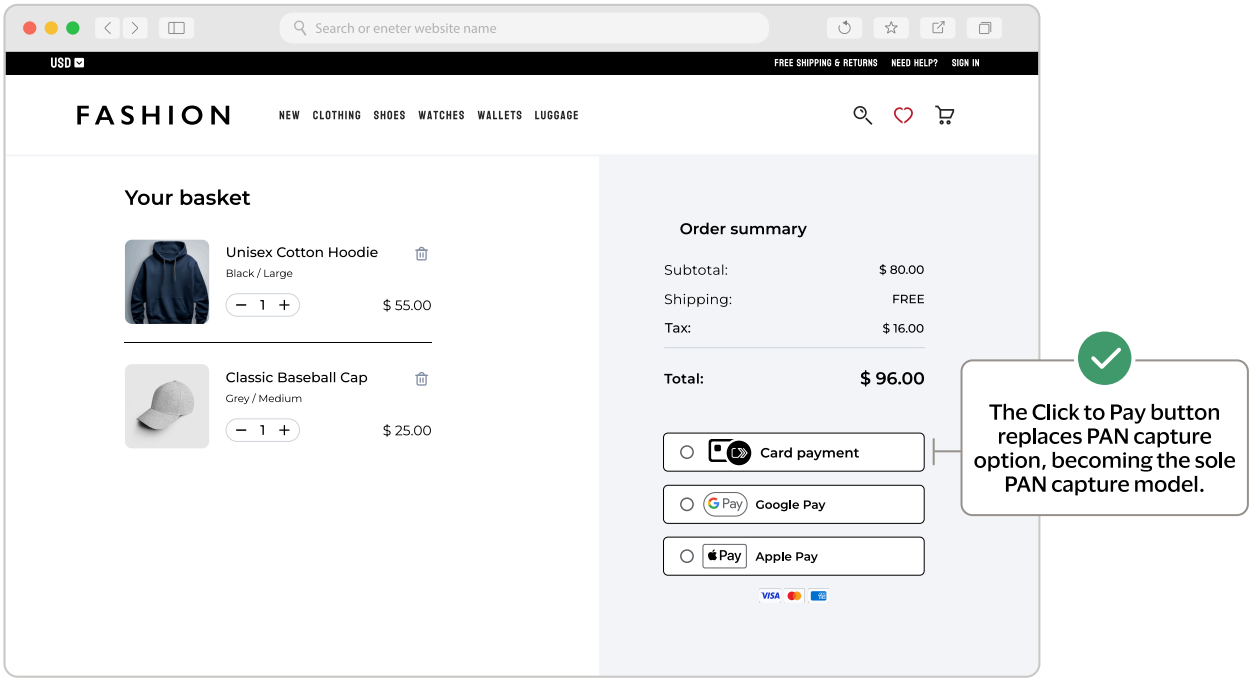


For information about how to configure this UI, see Loading the JavaScript Library and Invoking the Accept Function (on page 407).

Click to Pay as Radio Button

Click to Pay is a radio button for the card entry payment option within your payment page. When the cardholder selects this option, the Click to Pay payment flow is loaded.

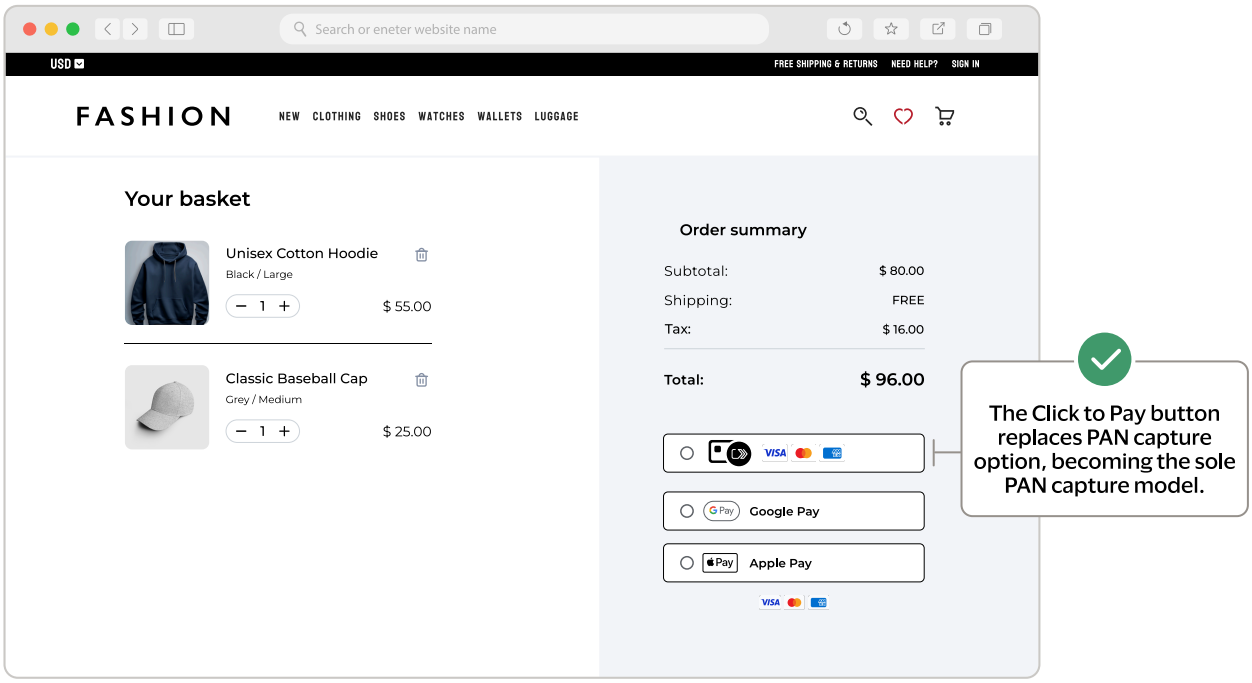
Click to Pay Radio Button Example UI



Click to Pay Icon on Radio Button

You can host the radio selection option for card payment with the Click to Pay icon displayed on the payment label. The Unified Checkout flow loads when the cardholder selects this option. For information about customizing how to trigger Unified Checkout, see JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry (on page 410).

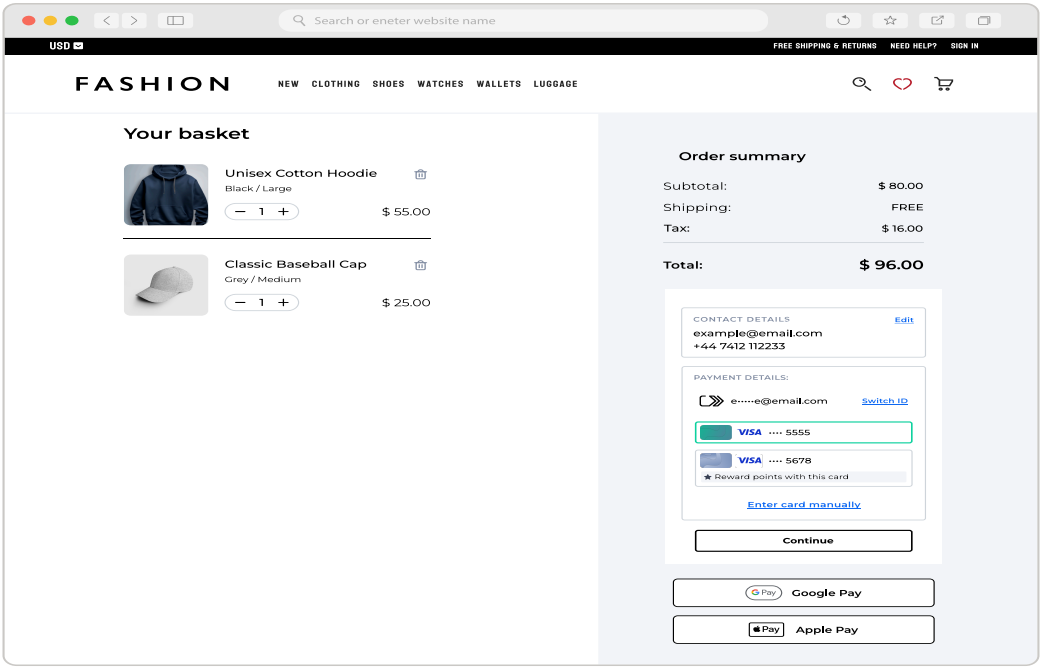
Click to Pay Icon on Radio Button Example UI



Load Click to Pay Automatically From Trigger

You can load the Unified Checkout JavaScript flow within your own payment button without requiring the cardholder to select a card payment option. This example shows a recognized user payment flow where the cardholder's information is shown automatically next to the other payment methods hosted within your payment page. For information about customizing how to trigger Unified Checkout, see JavaScript Example: Client-Defined Trigger for Click to Pay or PAN Entry (on page 410).

Click to Pay Loaded Automatically From Trigger UI

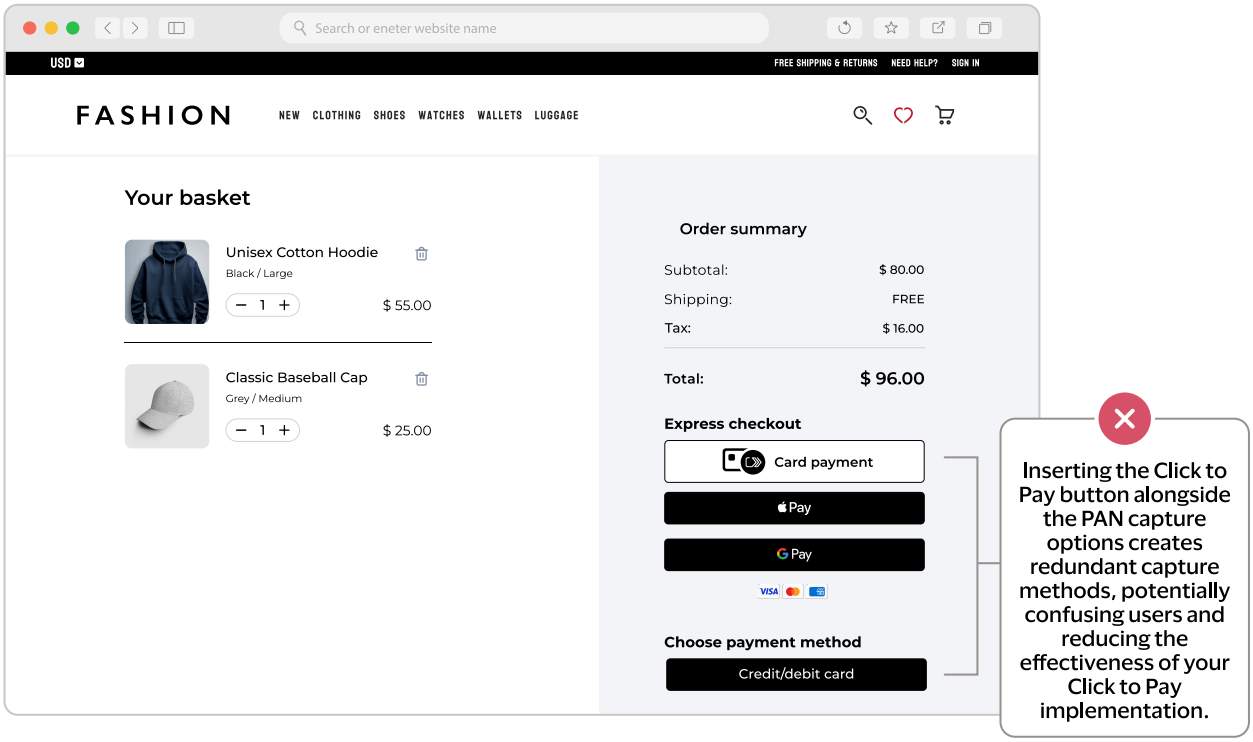


Card Payment Options with Click to Pay in UI

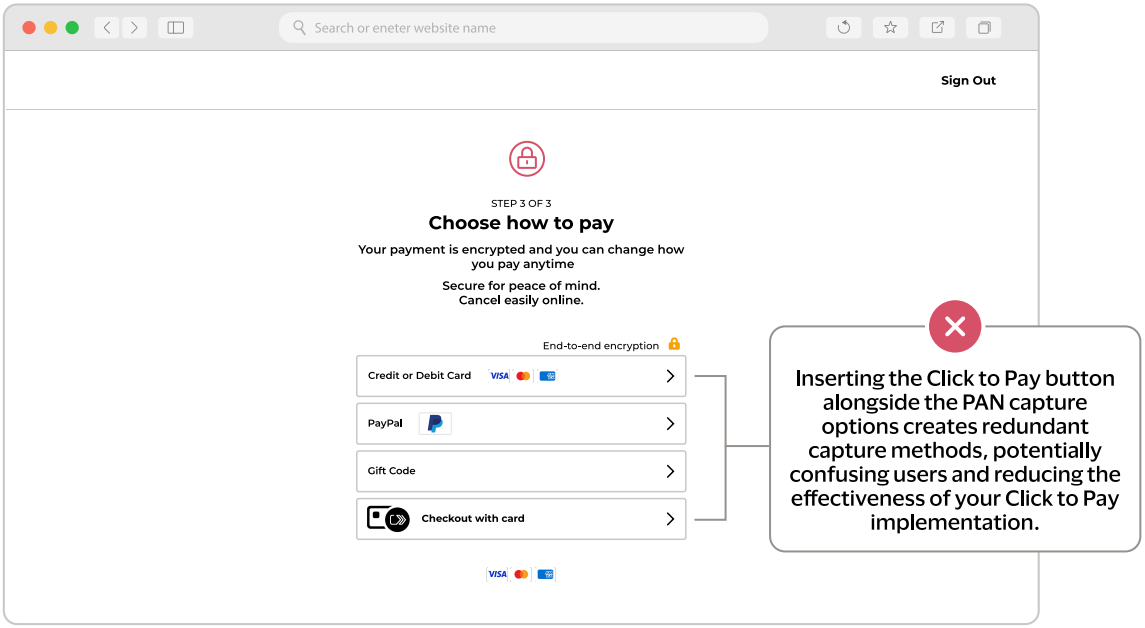
Do not present the Unified Checkout payment button as a separate payment method from the card payment button. If you do this, the cardholder is not prompted with their Click to Pay cards and must manually enter their payment details. They will also not have the option to store their card within Click to Pay for future use.

These examples show multiple card payment options and Click to Pay in a UI:

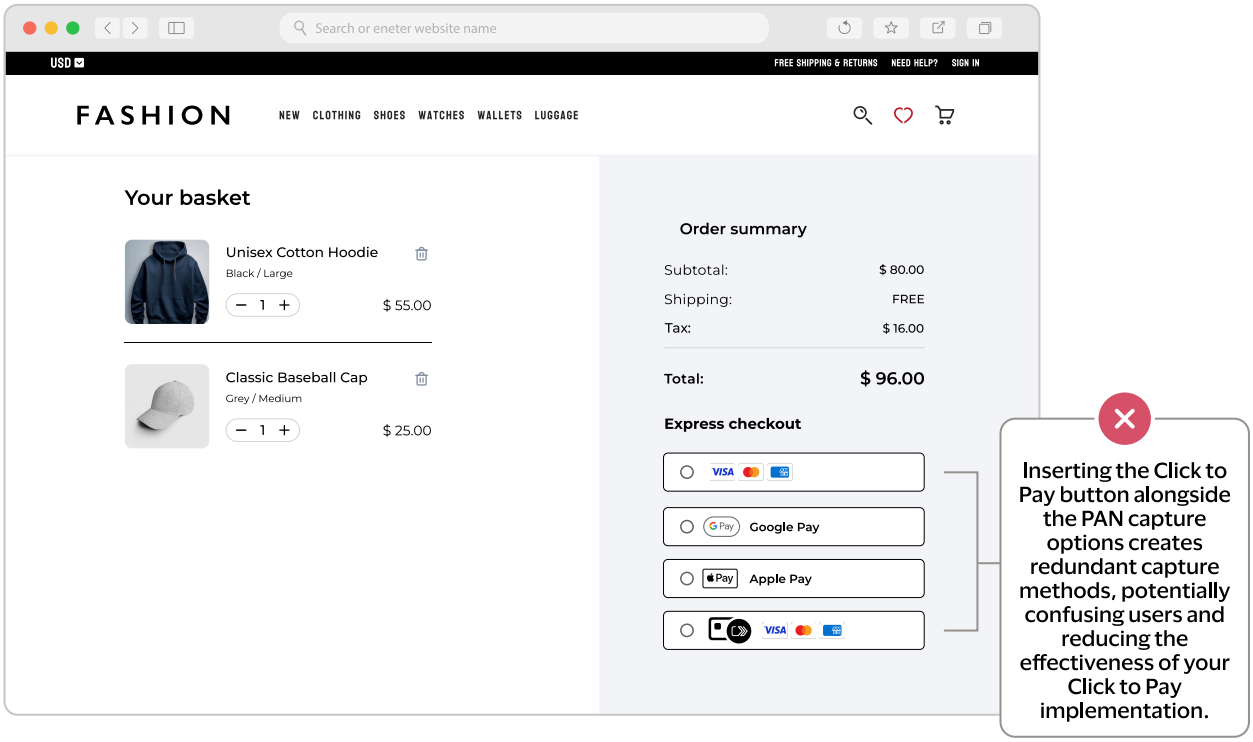
Multiple Card Payment Options in UI Example 1



Multiple Card Payment Options in UI Example 2



Multiple Card Payment Options in UI Example 3



Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Test Your Click to Pay Configuration

This section contains information about testing your Click to Pay configuration.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Test Payment Details

Use these test card numbers to test your Click to Pay configuration.

Combine the BIN with the card number when sending to Click to Pay.

Visa Click to Pay Test Cards

These Visa test cards can be added to your Click to Pay wallet.

Replace the X in the card number with 4.

You can manage your Visa Click to Pay test cards and account here:

- **Production:** https://src.visa.com/login
- **Test:** https://sandbox.src.visa.com/login

To manage Visa test cards for customer authentication, contact your implementation consultant or technical account manager.

Important: These test cards are not valid for testing in production. To test in production, you must leverage production credentials.

Visa Test Card Numbers

Card Number	Expiration Date	CVV
x6229x3123123755	12/2029	728
x6229x3123123763	12/2029	605
x6229x3123123771	12/2029	694
x6229x3123123789	12/2029	881
x6229x3123123797	12/2029	678
x6229x3123123805	12/2029	084
x6229x3123123813	12/2029	127
x6229x3123123821	12/2029	218
x6229x3123123839	12/2029	114
x6229x31231238x7	12/2029	867
x6229x312312385x	12/2029	301

To manage Visa test cards for customer authentication, contact your implementation consultant or technical account manager.

Important: These test cards are not valid for testing in production. To test in production, you must leverage production credentials.

These Visa test card numbers can be used to test ECI05 frictionless authentication. Replace the X in the card number with 4.

ECI05 Authentication Test Card Numbers

Card Number	Expiration Date	CVV
X6229X3123113723	12/2027	929
X6229X3123113731	12/2027	217

These Visa test card numbers can be used to enroll in Passkey Service. Replace the X in the card number with 4.

Passkey Enrollment Test Card Numbers

Card Number	Expiration Date	CVV
X3958X0327800110	12/2027	832
X3958X0328300110	12/2027	474

Mastercard Test Cards

Mastercard test cards can be added to your Click to Pay wallet. You must retrieve Mastercard test cards from their Click to Pay test page: https://developer.mastercard.com/mastercard-checkout-solutions/documentation/testing/test_cases/click_to_pay_case/#test-cards

Mastercard has different test cards for retrieving tokenized and non-tokenized data. Cybersource recommends that you use these test cards as follows:

- Test cards to retrieve PAN data: Use these cards when the customer is completing checkout as a one-time guest and does not have a Click to Pay account or want to create one.
- Test cards to retrieve token data: Use these cards for tokenized Click to Pay transactions.

You can manage your Mastercard Click to Pay test cards and account here:

- **Live:** https://src.mastercard.com/profile/enroll
- **SBX:** https://sandbox.src.mastercard.com/profile/enroll

Mastercard authentication test cards are available on the *Mastercard Checkout Solutions* page in the Mastercard Developer Center.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Handle Errors

The Unified Checkout SDK uses a structured error object for all error scenarios. Errors are returned as exceptions from asynchronous methods and are also returned as events for centralized handling.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

UnifiedCheckoutError

All SDK errors are instances of `UnifiedCheckoutError` with these properties:

`UnifiedCheckoutError` Properties

Property	Type	Description
`correlationId`	`string?`	The correlation ID from an underlying API call, when applicable.
`details`	`unknown?`	Additional error-specific information. This is often an array of objects.
`informationLink`	`string?`	The URL linked to the online documentation for this error.
`message`	`string`	This property is a human-readable description of the error.
`name`	`string`	The value is always `"UnifiedCheckoutError"`.
`reason`	`string`	This property is a machine-readable error code, such as `"CAPTURE_CONTEXT_INVALID"`.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Detect Errors

Errors may be serialized through `postMessage`. Cybersource recommends that you use the `name` property instead of `instanceof`:

```
try {
  const result = await checkout.mount('#buttons');
} catch (error) {
  if (error.name === 'UnifiedCheckoutError') {
    // Access error.reason, error.message, error.details
  }
}
```

You can also write a helper function that can be reused:

```
function isUnifiedCheckoutError(obj) {
  return (
    obj !== null &&
    typeof obj === 'object' &&
    obj.name === 'UnifiedCheckoutError' &&
    typeof obj.reason === 'string' &&
    typeof obj.message === 'string'
  );
}
```

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Error Handling Patterns

Try/Catch

```
try {
  const client = await VAS.UnifiedCheckout(sessionJWT);
  const checkout = await client.createCheckout();
  const result = await checkout.mount('#buttons');
```

```
} catch (error) {
  console.error(error.reason, error.message);
}
```

Promise .catch()

```
VAS.UnifiedCheckout(sessionJWT)
  .then(client => client.createCheckout())
  .then(checkout => checkout.mount('#buttons'))
  .catch(error => console.error(error.reason, error.message));
```

Centralized Error Logging via Events

Errors from all integrations are applicable up to the client level. Use `client.on('error')` for centralized logging:

```
const client = await VAS.UnifiedCheckout(sessionJWT);

client.on('error', (err) => {
  errorReporter.send({
    source: err.source,   // "checkout", "trigger", "button", or "client"
    code: err.code,
    message: err.message
  });
});
```

Cybersource recommends that you do this as it catches errors from all checkouts, triggers, and buttons created from this client instance.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Error Codes

Initialization Errors

These errors are returned during `VAS.UnifiedCheckout(sessionJWT)`:

Initialization Reason Values

Reason	Description
`CAPTURE_CONTEXT_EXPIRED`	The supplied JWT has expired. Generate a new session.
`CAPTURE_CONTEXT_INVALID`	The session JWT is not valid. For example, it has a bad signature or is malformed.
`UNUSED_TARGET_ORIGINS`	One or more `targetOrigins` in the session do not match the current page origin. The `details` array lists the unused origins.

Mount Errors

These errors are returned during `checkout.mount()` or `trigger.mount()`:

Mount Reason Values

Reason	Description
`CHECKOUT_ALREADY_MOUNTED`	The checkout or trigger is already mounted. Call `unmount()` first, or create a new instance.
`MOUNT_CONTAINER_SELECTOR`	The CSS selector does not match any Document Object Model (DOM) element. Check that the container exists before calling `mount()`.
`MOUNT_ERROR`	A problem occurred loading the payment iframe.
`MOUNT_INVALID_CONTAINER`	The supplied container parameter is not a valid CSS selector string or `HTMLElement`.
`MOUNT_PAYMENT_TIMEOUT`	A payment method timed out during initialization.
`MOUNT_PAYMENT_UNAVAILABLE`	No payment types could be presented to the customer. This may be due to browser or device support, or errors during checkout initialization.
`MOUNT_SIDEBAR_OPTIONS`	The supplied container parameter is invalid for sidebar mode.
`MOUNT_TOKEN_TIMEOUT`	Token creation timed out during mount. This may indicate a network issue.
`MOUNT_TOKEN_XHR_ERROR`	A network error occurred during token creation. Check the customer's connectivity.

Checkout Errors

Checkout Reason Values

Reason	Description
`CHECKOUT_ERROR`	A general checkout error occurred.
`CHECKOUT_PAYMENT_PARAMETERS`	One or more payment parameters have a validation error.
`CHECKOUT_VALIDATION_PARAMS`	One or more checkout parameters have a validation error.

Trigger Errors

Trigger Reason Values

Reason	Description
`TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED`	The specified payment type cannot be used with a trigger. Only `PANENTRY` and `CLICKTOPAY` values are supported.

Payment-Specific Errors

Payment-Specific Reason Values

Reason	Description
`CLICK_TO_PAY_SDK_LOAD_ERROR`	The Click to PaySDK failed to load.
`ENCRYPT_CARD_FOR_SRC_ENROLMENT_ERROR`	Card encryption for Click to Pay enrollment failed.
`GOOGLEPAY_CHECKOUT_ERROR`	A Google Pay checkout error occurred.
`LAUNCH_SRC_CHECKOUT_ERROR`	Launching the Click to Pay checkout failed.
`TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED`	The payment type is not supported for triggers.

General Errors

Reason Code	Description
`UNKNOWN_ERROR`	An unknown error has occurred.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Appendix

Client Version History

Below is a list of client versions and the features that are included in each version.

Important: Cybersource recommends that you use the most recent client version in your integration.

0.23

Accepts these card networks in the **allowedCardNetworks** field for manual card entry:

- Carnet
- Cartes Bancaires
- China UnionPay with card verification value (CVV)
- EFTPOS
- ELO
- mada
- Meeza

Ordering controls for the **allowedPaymentTypes** button.

De-coupling of PANENTRY from other payment types in the **allowedPaymentTypes** field.

0.24

Support for enabling combo cards in the capture context.

Support for eight-digit BINs.

Support for enabling card save in the capture context.

0.25

Addition of **Skip Verification next time** in the Click to Pay payment flow.

Support for CPF in the capture context.

0.26

Support for auto-lookup in Click to Pay when an email is included in the capture context.

Inclusion of the **cardDetails** field object in the transient token response.

0.28

Support for PayPak as an **allowedCardNetwork**.

Auto-enrollment for Click to Pay in supported markets.

Removal of the confirm or continue screen for specific use cases.

Static button for Click to Pay flows.

0.30

Support for Pakistan locales (en_PK and ur_PK).

New look and feel of Unified Checkout in line with EMVCO best practices.

0.31

Addition of the **data** object of the **orderInformation** field object and pass-through fields.

Support for Jaywan as an **allowedCardNetwork**.

Updated the payment details response to return detected card types. Multiple card types are shown when more than one card type is detected.

0.32

Support for KCP and UATP in the **allowedCardNetwork** field.

A radio button in the UI for Cartes Bancaires dual-branded cards.

0.33

Support for Mobile as Identity Click to Pay lookup.

0.34

Additional BIN range for Jaywan card types.

1.0

Configure payment options.

Configure customer data and payment flow.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Customization Matrix

Top-Level Appearance

Top-Level Appearance

Field Name	Data Type	Description
appearance	Object	Control checkout UI appearance
theme	Enum (String)	Theme selection (LIGHT, DARK, seasonal)
buttonType	Enum (String)	Button label/type
variables	Object	UI customisation variables container

Base

Base

Field Name	Data Type	Description
backgroundColor	Hex Colour	Main background colour
textColor	Hex Colour	Main text colour

Header

Header

Field Name	Data Type	Description
headerBackground	Hex Colour	Header background colour
headerForeground	Hex Colour	Header text/icon colour
headerAvatarBackgroundColor	Hex Colour	Header avatar background
headerAvatarForegroundColor	Hex Colour	Header avatar foreground

Input Default

Input Default

Field Name	Data Type	Description
inputBackground	Hex Colour	Input background
inputColor	Hex Colour	Input text colour
inputPlaceholderColor	Hex Colour	Placeholder text colour
inputBorderColor	Hex Colour	Border colour
inputBorderStyle	String	Border style
inputBorderRadius	CSS Size	Border radius

Input Hover State

Input Hover State

Field Name	Data Type	Description
inputHoverBackground	Hex Colour	Background when hover
inputHoverColor	Hex Colour	Text colour when hover
inputHoverPlaceholderColor	Hex Colour	Placeholder colour when hover
inputHoverBorderColor	Hex Colour	Border colour when hover
inputHoverBorderStyle	String	Border style when hover

Input Focused State

Input Focused State

Field Name	Data Type	Description
inputFocusedBackground	Hex Colour	Background when focused
inputFocusedColor	Hex Colour	Text colour when focused
inputFocusedPlaceholderColor	Hex Colour	Placeholder colour when focused
inputFocusedBorderColor	Hex Colour	Border colour when focused
inputFocusedBorderStyle	String	Border style when focused

Input Active State

Input Active State

Field Name	Data Type	Description
inputActiveBackground	Hex Colour	Background when active
inputActiveColor	Hex Colour	Text colour when active
inputActivePlaceholderColor	Hex Colour	Placeholder colour when active
inputActiveBorderColor	Hex Colour	Border colour when active
inputActiveBorderStyle	String	Border style when active

Input Pressed State

Input Pressed State

Field Name	Data Type	Description
inputPressedBackground	Hex Colour	Background when pressed
inputPressedColor	Hex Colour	Text colour when pressed
inputPressedPlaceholderColor	Hex Colour	Placeholder colour when pressed
inputPressedBorderColor	Hex Colour	Border colour when pressed
inputPressedBorderStyle	String	Border style when pressed

Input Error State

Input Error State

Field Name	Data Type	Description
inputErrorBackground	Hex Colour	Background when error
inputErrorColor	Hex Colour	Text colour when error
inputErrorPlaceholderColor	Hex Colour	Placeholder colour when error
inputErrorBorderColor	Hex Colour	Border colour when error
inputErrorBorderStyle	String	Border style when error

Input Valid State

Input Valid State

Field Name	Data Type	Description
inputValidBackground	Hex Colour	Background when valid
inputValidColor	Hex Colour	Text colour when valid
inputValidPlaceholderColor	Hex Colour	Placeholder colour when valid
inputValidBorderColor	Hex Colour	Border colour when valid
inputValidBorderStyle	String	Border style when valid

Button Default

Button Default

Field Name	Data Type	Description
buttonBackground	Hex Colour	Button background
buttonForeground	Hex Colour	Button text/icon colour
buttonShape	String	Button shape
buttonBorderColor	Hex Colour	Border colour
buttonBorderStyle	String	Border style
buttonBorderRadius	CSS Size	Border radius

Button Hover State

Button Hover State

Field Name	Data Type	Description
buttonHoverBackground	Hex Colour	Background when hover
buttonHoverForeground	Hex Colour	Text colour when hover
buttonHoverBorderColor	Hex Colour	Border colour when hover
buttonHoverBorderStyle	String	Border style when hover

Button Focus State

Button Focus State

Field Name	Data Type	Description
buttonFocusBackground	Hex Colour	Background when focus
buttonFocusForeground	Hex Colour	Text colour when focus
buttonFocusBorderColor	Hex Colour	Border colour when focus

Button Active State

Button Active State

Field Name	Data Type	Description
buttonActiveBackground	Hex Colour	Background when active
buttonActiveForeground	Hex Colour	Text colour when active
buttonActiveBorderColor	Hex Colour	Border colour when active
buttonActiveBorderStyle	String	Border style when active

Button Disabled State

Button Disabled State

Field Name	Data Type	Description
buttonDisabledBackground	Hex Colour	Background when disabled
buttonDisabledForeground	Hex Colour	Text colour when disabled

Button Disabled State (continued)

Field Name	Data Type	Description
buttonDisabledBorderColor	Hex Colour	Border colour when disabled

Typography & Other

Typography & Other

Field Name	Data Type	Description
fontFamily	String	Font family
borderRadius	CSS Size	Global border radius
paymentSelectionBackground	Hex Colour	Payment list background

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.

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

```
<Header>.<Payload>.<Signature>
```

The header and payload is **Base64-encoded JSON** and contains these claims:

- **Header**: The algorithm and token type. For example:

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

- **Payload**: The claims of what the token represents. For example:

  ```
  {
    "sub": "1234567890",
    "name": "John Doe",
    "iat": 1516239022
  }
  ```

- **Signature**: The signature is computed from the header and payload using a secret or private key.

Important: When working with JWTs, Cybersource recommends that you use a well-maintained JWT library to ensure proper decoding and parsing of the JWT.

Important: When parsing the JWT's JSON payload, you must ensure that you implement a robust solution for transversing JSON. Additional elements can be added to the JSON in future releases. Follow JSON parsing best practices to ensure that you can handle the addition of new data elements in the future.

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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` - Capture Context API	Bad request. Possible **reason** values:	
	`CAPTURE_CONTEXT_EXPIRED`	This reason is returned when the capture context JWT has passed its expiration time of 900 seconds (15 minutes). Example decrypted JWT fields include `"exp": "1762894371"` and `"iat": "1762893471"`.
	`CAPTURE_CONTEXT_INVALID`	The Unified Checkout configuration rejected the request due to invalid values. This reason is returned when the minimum required fields are missing or invalid or the capture context contradicts which products are enabled.
	`CHECKOUT_ERROR`	Checkout failed. This reason is returned when a general, non-payment-method-specific error occurs during the **UnifiedCheckout** checkout flow. When the checkout failure is specifically related to tokenization, Click to Pay SDK, SRC launch, Google Pay, etc., the SDK returns a more specific error
	`CLICK_TO_PAY_SDK_LOAD_ERROR`	This reason is returned when the UI cannot be successfully rendered. For example:

Reason Codes (continued)

Reason Code	Description	
		• Network failures (CDN unavailable, blocked, or timed out) • Browser or device restrictions are preventing the SDK from loading. • Incorrect or missing configuration causes Unified Checkout not to request the SDK asset. • The merchant site CSP is blocking the SDK. • Any runtime error that prevents Click to Pay JS initialization.
	`CREATE_TOKEN_TIMEOUT`	The token creation timed out. This reason is returned when the Unified Checkout JavaScript SDK cannot generate the transient token within the expected time-frame.
	`CREATE_TOKEN_XHR_ERROR`	This reason is returned when the system attempts to create a token, but a network / XHR-level failure occurs before the token can be created. This is a client-side SDK network failure, not a timeout or back-end validation error.
	`ENCRYPT_CARD_FOR_SRC_ENROLMENT_ERROR`	Encrypt card for SRC enrollment failed. This reason is returned when Unified Checkout attempts to encrypt a card to enroll it in the SRC / Click to Pay system and the encryption step fails. This causes the SRC enrolment to abort.
	`INVALID_APIKEY`	Returned when the API key that is used in the server-side capture context request is invalid.
	`LAUNCH_SRC_CHECKOUT_ERROR`	The launch SRC checkout failed. This reason is returned by the Unified Checkout JavaScript SDK when it cannot initialize or open the SRC checkout flow.

Reason Codes (continued)

Reason Code	Description	
	SDK_XHR_ERROR	SDK failed to load. This reason is returned when the JavaScript SDK fails to load due to an XHR/network error during Unified Checkout initialization.
	SHOW_LOAD_CONTAINER_SELECTOR	The specified DOM element cannot be found. Returned when the DOM element specified in the `show()` configuration cannot be found. This is a client-side JavaScript SDK error thrown during rendering of the payment selection UI.
	SHOW_LOAD_ERROR	There was a problem encountered when loading the payment screen. Returned when the Unified Payments UI fails to load the payment selection screen (iframe/UI) during the `.show()` step
	SHOW_LOAD_INVALID_CONTAINER	The supplied container parameter is invalid. Returned when the container provided to `up.show()` exists but is invalid—wrong type, not suitable to host UC UI, unsupported context, or malformed in configuration
	SHOW_LOAD_SIDEBAR_OPTIONS	The supplied container parameter is invalid when sidebar is selected. Returned when `sidebar = true` and the `containers` supplied to `up.show()` are not valid for the sidebar layout (wrong type, unsupported container, or structurally incompatible).
	SHOW_PAYMENT_TIMEOUT	Occurs when an error is encountered during the handling of a payment option. Returned when UC cannot progress the user's selected payment option in time:
	SHOW_PAYMENT_UNAVAILABLE	No payment types could be presented to the customer. This could be due to browser/device support or errors encountered during the checkout. Returned when *zero* payment methods can be presented in the `.show()` phase — typically due to browser/device incompatibility, disabled payment types, or internal errors while loading payment options.

Reason Codes (continued)

Reason Code	Description	
	SHOW_TOKEN_TIMEOUT	Occurs when the createToken call was unable to proceed. Returned when the createToken call cannot proceed within the expected time while rendering the payment selection UI
	SHOW_TOKEN_XHR_ERROR	Occurs when a network error is encountered while attempting to create a token. Returned when the createToken step within .show() fails due to an actual network/XHR error (blocked request, CORS/CSP violation, extension interference, unreachable endpoint).
	TOKENIZATION_ERROR	Tokenization failed. Returned when tokenization of the selected payment method fails — due to invalid payment data, a failed internal tokenization call, network issues, or an unsupported/blocked payment environment.
	TRIGGER_PAYMENT_TYPE_NOT_SUPPORTED	Trigger is not supported for this payment type. Returned when up.trigger(paymentType) is called with a payment method that does not support trigger mode, is not enabled, not available on the device/browser, or not recognized by UC.
	UNIFIED_PAYMENTS_PAYMENT_PARAMETERS	Occurs when no valid payment parameters exist when initializing button. Returned when the merchant calls VAS.UnifiedCheckout(sessionJWT) without providing valid payment parameters — meaning the SDK cannot initialize the payment buttons because the supplied configuration is missing, empty, or malformed.
	UNIFIED_PAYMENTS_VALIDATION_FIELDS	A validation error occurred. Missing or invalid values in required fields
	UNIFIED_PAYMENTS_VALIDATION_PARAMS	Trigger is not supported for this payment type. Returned when up.trigger(paymentType) is called with a payment method that does not support

Reason Codes (continued)

Reason Code	Description	
		trigger mode, is not enabled, not available on the device/browser, or not recognized by UC.
`404`	The specified resource not found in the system.	
`500`	Unexpected server error.	

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

Supported Countries for Click to Pay

Click to Pay is supported in these countries:

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

- Dominican Republic
- Ecuador
- El Salvador
- Finland
- France
- Germany
- Greece
- Honduras
- Hong Kong
- Hungary
- Indonesia
- Ireland
- Italy
- Japan
- Jordan
- Kuwait
- Malaysia
- Mexico
- Netherlands
- New Zealand
- Nicaragua
- Norway
- Panama
- Paraguay
- Peru
- Poland
- Qatar

- Romania
- Saudi Arabia
- Singapore
- Slovakia
- Slovenia
- South Africa
- Spain
- Sweden
- Switzerland
- Thailand
- Ukraine
- United Arab Emirates
- United Kingdom
- United States
- Uruguay
- Vietnam

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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
- bg_BG
- ca_ES
- cs_CZ
- da_DK
- de_AT
- de_DE
- el_GR
- en_AU
- en_CA
- en_GB
- en_IE
- en_NZ
- en_PK
- 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
- ro_RO
- ru_RU
- sk_SK
- sl_SI
- sv_SE
- th_TH
- tl_PH
- tr_TR
- uk_UA
- ur_PK
- vi_VN
- zh_CN

- zh_HK
- zh_MO
- zh_SG
- zh_TW

Related information

Getting Started with REST
Response Codes
API Field Reference Guide
API Reference Sandbox
Business Center Test
Business Center Production
Customer Support

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. The transient token can be used multiple times within the 15-minute period.

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.

Important: Each request that you send to Cybersource requires header information. For information about constructing the headers for your request, see the *Getting Started with REST Developer Guide*.

Endpoint

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

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

Production in Saudi Arabia: `POST https://api.sa.cybersource.com/pts/v2/payments`

Test in Saudi Arabia: `POST https://apitest.sa.cybersource.com/pts/v2/payments`

Required Field for an Authorization with a Transient Token

tokenInformation.transientTokenJwt

REST Interactive Example: Authorization with a Transient Token

```
Payment with Flex 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

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":
 "eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFsZyI6IlJTMjU2In0.eyJp
c3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTAuMS4wIiwiaWF0IjoxNjE0Nz
kxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPRUhJVldBSURPkhLNjJJSFQxU
VE1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4TRIVZN-jpNbamcluAKVZ1zfd
hbkrB6aNVWECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFfzTuq41sDhlIqB-UatveU_2uk
PxLYl87EX9ytpx4zCJVmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85tl2OHaajEc25tS1FwH3jDOfj
AC8mu2MEk-Ew0-ukZ70Ce7Zaq4cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bpvcKLr9k_8b9YnddIzp0p0JOC
jXC_nuofQT7_x_-CQayx2czE0kD53HeNYC5hQ"
  }
}
```

Response to Successful Request

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

```
            "href": "/pts/v2/payments/682622572509671870395 5"
        },
        "capture": {
            "method": "POST",
            "href": "/pts/v2/payments/682622572509671870395 5/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.

Important: Each request that you send to Cybersource requires header information. For information about constructing the headers for your request, see the *Getting Started with REST Developer Guide*.

Endpoint

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

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

Production in Saudi Arabia: `POST https://api.sa.cybersource.com/pts/v2/payments`

Test in Saudi Arabia: `POST https://apitest.sa.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 these values:

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

tokenInformation.transientTokenJwttokenInformation.transientTokenJwt

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

```
Payment with Flex Token(Create Permanent TMS 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

Request

```
{
  "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":
 "eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFsZyI6IlJTMjU2In0.eyJp
c3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTAuMS4wIiwiaWF0IjoxNjE0Nz
kxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPRUhJVldBSURPkhLNjJJSFQxU
VE1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4TRIVZN-jpNbamcluAKVZ1zfd
hbkrB6aNVWECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFfzTuq41sDhlIqB-UatveU_2uk
PxLYl87EX9ytpx4zCJVmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85tl2OHaajEc25tS1FwH3jDOfj
AC8mu2MEk-Ew0-ukZ70Ce7Zaq4cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bpvcKLr9k_8b9YnddIzp0p0JOC
jXC_nuofQT7_x_-CQayx2czE0kD53HeNYC5hQ"
  }
}
```

Response

```
{
    "_links": {
        "authReversal": {
            "method": "POST",
            "href": "/pts/v2/payments/682622044293611960 3954/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"
```

```
        }
    }
}
```