Payment Service - System Design

CyberSource Integration



October 11th, 2024

Submitted to:

Icon

Description automatically generated

Prepared by TransSIGHT

**DOCUMENT VERSION CONTROL**

|  |  |  |
| --- | --- | --- |
| **DOCUMENT NAME** | **SUBMITTAL DATE** | **VERSION NO.** |
| **Payment Service – System Design -- CyberSource Integration**  - First Draft | 10/11/2024 | 1.0 |

**TABLE OF CONTENTS**

[1.0 Overview 2](#_Toc179538735)

[2.0 Existing Functionality 2](#_Toc179538736)

[Generate Client Token 3](#_Toc179538737)

[Create Customer 5](#_Toc179538738)

[Add Payment Method 8](#_Toc179538739)

[Delete Payment Method 9](#_Toc179538740)

[Find Payment Method 11](#_Toc179538741)

[Sale Transaction 12](#_Toc179538742)

[Hold Authorization Transaction 14](#_Toc179538743)

[Void Transaction 16](#_Toc179538744)

[Refund Transaction 17](#_Toc179538745)

[Transaction Search 18](#_Toc179538746)

[Find Expiring and Expired Credit Cards 20](#_Toc179538747)

[3.0 Microform User Interface 21](#_Toc179538748)

[Sequence Diagram for Tokenization and Payment Processing 24](#_Toc179538749)

[4.0 Payments Schema 25](#_Toc179538750)

[Merchant Accounts 25](#_Toc179538751)

[Users 25](#_Toc179538752)

[Payment Methods 25](#_Toc179538753)

[Sale Transactions 26](#_Toc179538754)

[Auth Transactions 26](#_Toc179538755)

[Refund Transactions 27](#_Toc179538756)

[5.0 Payments Service 28](#_Toc179538757)

[Configuration 28](#_Toc179538758)

[Proposed Implementation 28](#_Toc179538759)

[6.0 Optimizing the Settlement Process 29](#_Toc179538760)

[Braintree 29](#_Toc179538761)

[Cybersource 29](#_Toc179538762)

[Recommendation 29](#_Toc179538763)

[7.0 Issues and Questions 30](#_Toc179538764)

[Billing information required when vaulting each payment method 30](#_Toc179538765)

[8.0 Testing 31](#_Toc179538766)

[Dave’s Sandbox Account 31](#_Toc179538767)

[Test Credit Card Numbers 31](#_Toc179538768)

[9.0 Developer Notes 31](#_Toc179538769)

[Changes in latest version of cybersource-rest-client 31](#_Toc179538770)

[10.0 Summary 32](#_Toc179538771)

# Overview

The objective of this document is to outline the system design for integrating CyberSource as a payment gateway while ensuring continuity with the existing functionalities implemented using Braintree in both mobile applications and website.

**SEE:** <https://github.com/evadremlab/cybersource-demo> for my sample code used to test various features.

# Existing Functionality

These are the actions currently supported by the payment service.

|  |  |  |
| --- | --- | --- |
| **Action** | **Braintree Command** | **Comments** |
| Generate Client Token | gateway.clientToken.generate | Get client token for drop-in UI |
| Create Customer | gateway.customer.create | Called when adding payment method if users.payment\_processor\_customer\_id is null |
| Add Payment Method | gateway.paymentMethod.create |  |
| Delete Payment Method | gateway.paymentMethod.delete |  |
| Find Payment Method | gateway.paymentMethod.find | Used to validate delete payment method request |
| Sale Transaction and  Hold Authorization Transaction | gateway.transaction.sale |  |
| Refund Transaction | gateway.transaction.refund |  |
| Void Transaction | gateway.transaction.void |  |
| Transaction Search | gateway.transaction.search | Used by settlement process |
| Find Expired and Expiring Credit Cards | gateway.creditCard.expiringBetween | Used by Parking backend scheduled task notify customers of expiring or expired credit cards |

## Generate Client Token

The Flex Microform API will be used to generate the context of the customer payment information that is to be captured and tokenized. The capture context request contains all the merchant-specific parameters that tell the front-end JavaScript library how to behave within your payment experience.

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

* Merchant-specific parameters: Manages the customer payment experience for the current payment transaction.
* A one-time public key: Secures the information flow during the current payment transaction.

The capture context is signed so that the receiving party can verify it is coming from CyberSource, including automated verification by Microform.

<https://developer.cybersource.com/docs/cybs/en-us/digital-accept-flex/developer/all/rest/digital-accept-flex/microform-integ-v2/microform-integ-getting-started-v2/creating-server-side-context-v2.html>

Example request:

|  |
| --- |
| const path = require('path');  const cyberSourceConfig = require(path.resolve('config/cybersource-config.js'));  const {  ApiClient,  GenerateCaptureContextRequest  } = require('cybersource-rest-client');  const requestData = GenerateCaptureContextRequest.constructFromObject({  clientVersion: 'v2.0',  targetOrigins: ['http://localhost:3000'], **🡨** url hosting the microform  allowedCardNetworks: ['VISA', 'MASTERCARD', 'AMEX', 'DISCOVER']  });  instance.generateCaptureContext(requestData, (err, data) => {  if (err) {  reject(err);  } else if (data) {  resolve(data);  }  }); |

Example decoded capture context from response:

|  |
| --- |
| {  "flx": {  "path": "/flex/v2/tokens",  "data": "OnNUTif50p3N6pgB51VBZBAAEKd2sZ/Vi4swG2FjLCDuNf4VpoHp47Yfede94N3yR/ehab0RHEIS5cVp/m80Otc40J+mLQXecnRxPBue9n6F3ZN6ptXRmWnX53FtWPs0TP3ClkEbte7nZdGIu8LmUaiLEQ==",  "origin": "https://testflex.cybersource.com",  "jwk": {  "kty": "RSA",  "e": "AQAB",  "use": "enc",  "n": "3fqFjmYzm-48wEtYL4lBJKDPLJGCgBJ3CBygBqqZHcXylcC7vWiUBaLINgqpVUsE8QIuPjlkntB-RHiI8J-x7vIdHNRfBe6FEjxVMCP7N9jK49\_r6m6XB1jJGdNFYiAmVqGgbuj3MLcQH701wF\_hQXCBwuxZbCbWLgdSG7eZcdOQm9Xr4ahYhZVITD0OMjn\_qWrsCWlpXt9lICpVS4cy7VatQaTPXCSFVbLa69-X2z13O6kxjR02g26\_4POMoYaGoQUsTHCimLtIeosRFGN5XZRjsEr6f9bivaez8soNvoZNThQPYmvhS89CuA8\_hqFgQ8frVTVDOduHrbkW\_JnGTw",  "kid": "089U7tM9jWI0Lcbm5IiQPmS0fDwKmGzV" **🡨** decoded to get public key for verification  }  },  "ctx": [  {  "data": {  "clientLibrary": "https://testflex.cybersource.com/microform/bundle/v2.0/flex-microform.min.js", **🡨** use instead of hardcoded path  "allowedCardNetworks": [  "VISA",  "MASTERCARD",  "AMEX",  "DISCOVER"  ],  "targetOrigins": [  "http://localhost:3000"  ],  "mfOrigin": "https://testflex.cybersource.com"  },  "type": "mf-2.0.0"  }  ],  "iss": "Flex API",  "exp": 1727991541,  "iat": 1727990641,  "jti": "gLeb73YVSfrkJs9g"  } |

## Create Customer

*“In CyberSource it is recommended that a customer be created with a zero amount Payment Authorization.”*

*“Cybersource can automatically verify that a payment card or electronic check account is valid prior to tokenization by authorizing a zero or low value amount, depending on the card type. There is no additional charge from Cybersource for this service.”*

A CyberSource customer and payment method will be created at the same time when adding a user’s first payment method. This requires a transient token (JWT) generated by the Flex API from the Microform used to enter the payment information. See the [Microform User Interface](#_Microform_User_Interface) sections for details on creating the transient token.

The payment service currently receives a “nonce” generated from the Braintree drop-in UI, and we would use this same parameter to receive the transient token from the Cybersource UI in the website or app.

Example request:

|  |
| --- |
| {  "tokenInformation": {  "transientTokenJwt": "<from microform.createToken()>",  "paymentInstrument": {  "default": true 🡨 required to be true for first payment method  }  },  "processingInformation": {  "commerceIndicator": "internet",  "actionList": [  "TOKEN\_CREATE" **🡨** create the following token types  ],  "actionTokenTypes": [ **🡨** customer and payment method at the same time  "customer",  "paymentInstrument"  ],  "capture": false **🡨** auth only  },  "orderInformation": {  "amountDetails": {  "totalAmount": "0.00",  "currency": "USD"  },  "billTo": { **🡨** all fields required  "email": "david.balmer@transsight.com",  "locality": "Alameda",  "firstName": "David",  "lastName": "Balmer",  "address1": "100 Main Street",  "country": "US",  "administrativeArea": "CA",  "postalCode": "94501"  }  },  "clientReferenceInformation": {  "code": "new-customer-order-id" **🡨** order ID  }  } |

Example response data: (not all properties included)

|  |
| --- |
| {  "status": "AUTHORIZED",  "paymentAccountInformation": {  "tokenInformation": {  "instrumentidentifierNew": false,  "customer": {  "id": "23FE2439AD65BBCFE063AF598E0A66A9" **🡨** customer id  },  "paymentInstrument": {  "id": "23FE0306218C3D37E063AF598E0A7E90" **🡨** payment token  },  "instrumentIdentifier": {  "id": "7031410000201134444",  "state": "ACTIVE"  }  }  } |

Following a successful CyberSource response, create **users** and **payment\_methods** records. See the **mapping** section in [Users](#_Users) and [Payment Methods](#_Payment_Methods) for details on populating these records from the response.

**NOTE:** When adding a payment method with a zero dollar authorization, the Cybersource dashboard shows this transaction as having a successful “Subscription Creation”:

A screenshot of a computer

Description automatically generated

According to the Cybersource docs, a subscription includes:

* Customer contact information, such as billing and shipping information.
* Customer payment information, such as card type, masked account number, and expiration date.
* Customer order information, such as the transaction reference number and merchant defined data fields.

There are two types of subscriptions but neither seems to apply to our use cases, and I did not provide the “required” information during my testing so not sure how a subscription was created…

**Installment Subscriptions** which will charge a customer’s preferred payment method on a fixed number of scheduled payments. The number of payments, the amount and frequency of each payment, and the start date for processing the payments are required.

**Recurring Subscriptions** which will charge a customer’s preferred payment method on a recurring payment basis with no specific end date. The amount and frequency of each payment and the start date for processing the payments are required.

**NOTE:** I’m assuming that no recurring payment will happen because no schedule or amount was provided.

<https://developer.cybersource.com/library/documentation/dev_guides/Token_Management/SO_API/html/Topics/Automatically_Preauthorizing_an_Account.htm>

<https://developer.cybersource.com/library/documentation/dev_guides/Recurring_Billing/UBC_Deprecated/html/Topics/Customer_Subscriptions.htm>

## Add Payment Method

This requires a transient token (JWT) generated by the Flex API from the Microform used to enter the payment information. See the [Microform User Interface](#_Microform_User_Interface) sections for details on creating the transient token.

The payment service currently receives a “nonce” generated from the Braintree drop-in UI, and we would use this same parameter to receive the transient token from the Cybersource UI.

*“You can also*[*add additional Payment Instruments to a Customer via a Payment Authorization*](https://developer.cybersource.com/api-reference-assets/index.html#payments_payments_process-a-payment_samplerequests-dropdown_authorization-with-token-create_authorization-create-default-payment-instrument-shipping-address-for-existing-customer_liveconsole-tab-request-body)*.”*

Example request data:

|  |
| --- |
| {  "paymentInformation": {  "customer": {  "id": "23FE2439AD65BBCFE063AF598E0A66A9" 🡨 existing customer  }  },  "clientReferenceInformation": {  "code": "add-new-payment-method" **🡨** order id  },  "processingInformation": {  "actionList": [  "TOKEN\_CREATE" **🡨** create the following token type  ],  "actionTokenTypes": [  "paymentInstrument"  ],  "capture": false, **🡨** auth only  "commerceIndicator": "internet"  },  "orderInformation": {  "amountDetails": {  "totalAmount": "0",  "currency": "USD"  },  "billTo": {  "email": "cybersource\_test\_002@yopmail.com",  "firstName": "David",  "lastName": "Balmer",  "address1": "3355 Geary Blvd.",  "locality": "San Francisco",  "country": "US",  "administrativeArea": "CA",  "postalCode": 94118  }  },  "tokenInformation": {  "transientTokenJwt": "<from microform.createToken()>",  "paymentInstrument": {  "default": false **🡨** don’t replace their default payment method  }  }  } |

Example response data: (not all properties included)

|  |
| --- |
| {  "status": "AUTHORIZED",  "paymentAccountInformation": {  "tokenInformation": {  "instrumentidentifierNew": false,  "customer": {  "id": "23FE2439AD65BBCFE063AF598E0A66A9"  },  "paymentInstrument": {  "id": "23FE2439B3F7BBCFE063AF598E0A66A9" **🡨** payment token  },  "instrumentIdentifier": {  "id": "7037870000055421111",  "state": "ACTIVE"  }  }  } |

Following a successful CyberSource response, create a **payment\_methods** record. See the **mapping** section in [Payment Methods](#_Payment_Methods) for details on populating this record from the response.

## Delete Payment Method

**DELETE** [https://apitest.cybersource.com/tms/v2/customers/{{customer}}/paymentinstruments/{{token}}](https://apitest.cybersource.com/tms/v2/customers/%7b%7bcustomer%7d%7d/paymentinstruments/%7b%7btoken%7d%7d)

**NOTE:** Cannot delete if it’s the customers default payment method, so would need to add a new default payment method, or update an existing one to default, before deleting.

|  |
| --- |
| {  "errors": [  {  "type": "conflict",  "message": "Action cannot be performed as the PaymentInstrument is the customers default"  }  ]  } |

Update Payment Method

<https://developer.cybersource.com/docs/cybs/en-us/tms/best-practices/all/rest/tms-best-practices/wallet-intro/9-defaults.html>

Customer 23FE2439AD65BBCFE063AF598E0A66A9 has these payment methods:

* 23FE0306218C3D37E063AF598E0A7E90 (default)
* 23FE2439B3F7BBCFE063AF598E0A66A9
* 23FE2439B056BBCFE063AF598E0A66A9 🡨 tried to make this the default, but doesn’t work

**PATCH** [https://apitest.cybersource.com/tms/v2/customers/{{customer}}/payment-instruments/{{token}}](https://apitest.cybersource.com/tms/v2/customers/%7b%7bcustomer%7d%7d/payment-instruments/%7b%7btoken%7d%7d)

|  |
| --- |
| {  "default": true  } |

returns this error:

|  |
| --- |
| {  "errors": [  {  "type": "conflict",  "message": "Action cannot be performed as the if-match header value does not match the token etag"  }  ]  } |

## Find Payment Method

GET [https://apitest.cybersource.com/tms/v2/customers/{{customer}}/payment-instruments/{{token}}](https://apitest.cybersource.com/tms/v2/customers/%7b%7bcustomer%7d%7d/payment-instruments/%7b%7btoken%7d%7d)

Example response: (not all properties included)

|  |
| --- |
| {  "id": "23FE0306218C3D37E063AF598E0A7E90", **🡨** token  "default": true, **🡨** this is their default payment method  "state": "ACTIVE",  "card": {  "expirationMonth": "12",  "expirationYear": "2024",  "type": "002"  },  "buyerInformation": {  "currency": "USD"  },  "billTo": {  "firstName": "DAVID",  "lastName": "BALMER",  "address1": "3355 Geary Blvd.",  "locality": "San Francisco",  "administrativeArea": "CA",  "postalCode": "94118",  "country": "US",  "email": "cybersource\_test\_002@yopmail.com"  },  "processingInformation": {  "billPaymentProgramEnabled": false  },  "instrumentIdentifier": {  "id": "7031410000201134444"  }  } |

## Sale Transaction

In Braintree we’re passing **submitForSettlement = true**, but in CyberSource we need to pass **capture=true** to initiate the settlement at the same time as the hold authorization.

A capture is a follow-on transaction to an authorization. It is used to transfer the authorized funds from the customer's account to the merchant account.

Example request data:

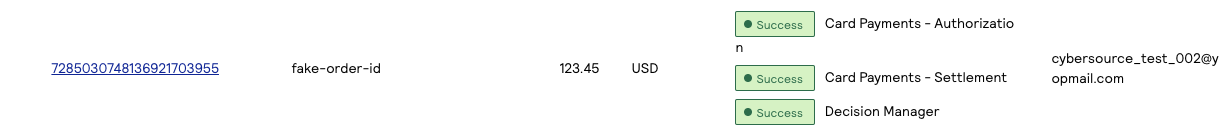
|  |
| --- |
| {  "paymentInformation": {  "paymentInstrument": {  "id": "23FE0306218C3D37E063AF598E0A7E90" **🡨** payment token  }  },  "orderInformation": {  "amountDetails": {  "totalAmount": "105.00",  "currency": "USD"  }  },  "processingInformation": {  "capture": true, **🡨** request auth and submit for settlement  "commerceIndicator": "internet"  },  "clientReferenceInformation": {  "code": "fake-order-id" **🡨** order id  }  } |

Example response data: (not all properties included)

|  |
| --- |
| {  "id": "7285030748136921703955", **🡨** transaction id  "submitTimeUtc": "2024-10-09T19:44:35Z",  "status": "AUTHORIZED",  "reconciliationId": "79518956",  "clientReferenceInformation": {  "code": "fake-order-id"  },  "processorInformation": {  "authIndicator": "1",  "approvalCode": "831000",  "transactionId": "0602MCC603474", **🡨** what is this? cannot search in dashboard  "networkTransactionId": "0602MCC603474",  "responseCode": "00",  "avs": {  "code": "Y",  "codeRaw": "Y"  }  } |

Following either a failed or successful CyberSource response, create either a **direct\_payment\_transactions** or **wallet\_refill\_transactions** record. See the **mapping** section in [Sale Transactions](#_Sale_Transactions) for details on populating these records from the response.

The Cybersource dashboard shows this transaction as having both an Authorization and Settlement:



## Hold Authorization Transaction

In Braintree we’re passing **submitForSettlement = false**, but in CyberSource omitting the **capture** property will have the same effect and only perform the hold authorization.

A hold authorization transaction can later be submitted for settlement using a Capture transaction, but these are typically not performed in real time. They are placed in a batch file and sent to the processor and the processor settles all the captures at one time.

Example request data:

|  |
| --- |
| {  "paymentInformation": {  "paymentInstrument": {  "id": "23FE0306218C3D37E063AF598E0A7E90" **🡨** payment token  }  },  "orderInformation": {  "amountDetails": {  "totalAmount": "105.00",  "currency": "USD"  }  },  "processingInformation": {  "capture": false, **🡨** request auth only (can also omit this property)  "commerceIndicator": "internet"  },  "clientReferenceInformation": {  "code": "fake-order-id" **🡨** order id  }  } |

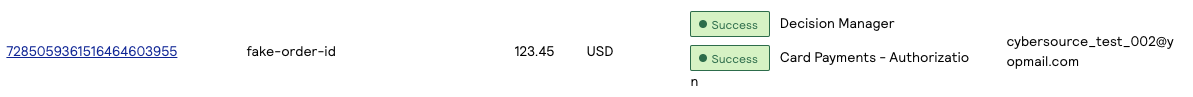
Example success response data: (not all properties included)

|  |
| --- |
| {  "id": "7285059361516464603955", **🡨** transaction id  "submitTimeUtc": "2024-10-09T20:32:17Z",  "status": "AUTHORIZED",  "reconciliationId": "W6SFV21YR3RA",  "clientReferenceInformation": {  "code": "fake-order-id"  },  "processorInformation": {  "authIndicator": "1",  "approvalCode": "831000",  "transactionId": "0602MCC603474",  "networkTransactionId": "0602MCC603474",  "responseCode": "00",  "avs": {  "code": "Y",  "codeRaw": "Y"  }  }  } |

Following either a failed or successful CyberSource response, create an **authorization\_transactions** record. See the **mapping** section in [Auth Transactions](#_Auth_Transactions) for details on populating this record from the response.

**NOTE:** Braintree returns the expiration date for the authorization, but Cybersource does not. According to the Cybersource Developer Center, most authorizations expire within 5 to 7 days, but the issuing bank sets the length of time.

The Cybersource dashboard shows this transaction as having only an Authorization:

****

## Void Transaction

*“A void cancels a capture or credit request that was submitted but not yet processed by the processor.*

*Capture and credit requests are usually submitted once a day. A void request is declined when the capture or credit request has already been sent to the processor.”*

*“After a void is processed, you cannot credit or capture the funds. You must perform a new transaction to capture or credit the funds. Further, when you void a capture, a hold remains on the authorized funds. If you are not going to re-capture the authorization, and if your processor supports authorization reversal after void (ARAV), you should request an authorization reversal to release the hold on the unused funds.”*

<https://developer.cybersource.com/docs/cybs/en-us/payments/developer/vital/sa/payments/payments-intro/payments-services-intro/payments-intro-processing-void.html>

Example request data:

|  |
| --- |
| {  "clientReferenceInformation": {  "code": "some-order-id" **🡨** order id recommended but not required  }  } |

Example success response data: (not all properties included)

|  |
| --- |
| {  "id": "7285186975596936503954", **🡨** transaction id being voided  "submitTimeUtc": "2024-10-10T00:04:57Z",  "status": "VOIDED", **🡨** success  "clientReferenceInformation": {  "code": "fake-order-id"  },  "voidAmountDetails": {  "voidAmount": "12.34",  "currency": "usd"  }  } |

Following a successful CyberSource, update either the **direct\_payment\_transactions**, **wallet\_refill\_transactions** or **authorization\_transactions** record. See the **void transaction mapping** section in [Sale Transactions](#_Sale_and_Auth) or [Auth Transactions](#_Auth_Transactions) for details on populating these records from the response.

## Refund Transaction

Cybersource supports multiple refunds against the same payment transaction, the same as Braintree.

Example request data:

|  |
| --- |
| {  clientReferenceInformation: {  code: "refund-order-id" **🡨** order id of refund transaction  },  orderInformation: {  amountDetails: {  currency: 'USD',  totalAmount: '1.23'  }  }  } |

Example success response data: (not all properties included)

|  |
| --- |
| {  "id": "7286681477456003604953",  "submitTimeUtc": "2024-10-11T17:35:48Z",  "status": "PENDING",  "reconciliationId": "79615119",  "clientReferenceInformation": {  "code": "refund-order-id" **🡨** order id of refund transaction  },  "refundAmountDetails": {  "refundAmount": "1.23",  "currency": "USD"  },  "orderInformation": {  "amountDetails": {}  }  } |

Following either a failed or successful CyberSource response, create a **refund\_transactions** record. See the **mapping** section in [Refund Transactions](#_Refund_Transactions) for details on populating this record from the response.

**NOTE:** I was able to submit a refund request for more than the original transaction amount, and still got a successful response with status = “PENDING”, so need more investigation into how this could be rejected…

According to the Cybersource docs:

*“When the status is pending, wait 15 minutes and request the check status service again.”*

## Transaction Search

*“To search for settled transactions in Cybersource, you can use the Transaction Search API, and the request body should include the search name, time zone, and search parameters. A single search can return up to 2,000 transactions.”*

<https://developer.cybersource.com/docs/cybs/en-us/txn-search/developer/all/rest/txn-search/txn-search-intro.html>

Example request data (for settled transactions):

|  |
| --- |
| {  "save": false, **🡨** could save a named search if required  "name": "test",  "timezone": "America/Los\_Angeles",  "searchType": "pendingSettlement", **🡨** Settlements Pending Batch  "query": "submitTimeUtc:[NOW/DAY-1DAYS TO NOW/DAY+1DAY}", **🡨** date range  "offset": 0,  "limit" 2000, **🡨** got a 404 error when trying 2600, but 2500 was ok  "sort": "submitTimeUtc:asc"  } |

To get around this 2000 record limit, we would need to make multiple transaction search requests with a different “offset” until we get an empty result set.

***“Settlements Pending Batch:*** *Use this filter to find transactions that were settled by the merchant in the past 48 hours, but that are not yet forwarded by Business Center to the backend processors to fulfill a transaction.”*

<https://developer.cybersource.com/docs/cybs/en-us/apple-pay/developer/six/rest/applepay/applepay-txn-reports/transactionby_phase.html>

Example response data:

|  |
| --- |
| {  "save": false,  "name": "test",  "timezone": "America/Los\_Angeles",  "query": "submitTimeUtc:[NOW/DAY-1DAYS TO NOW/DAY+1DAY}",  "offset": 0,  "limit": 100,  "sort": "submitTimeUtc:asc",  "count": 12,  "totalCount": 12,  "\_embedded": {  "transactionSummaries": [  {  "id": "7285030748136921703955", **🡨** transaction id  "submitTimeUtc": "2024-10-09T19:44:34Z",  "applicationInformation": {  "reasonCode": "100",  "rCode": "1",  "rFlag": "SOK",  "applications": [  {  "name": "ics\_auth", **🡨** card authorization  "reasonCode": "100", **🡨** successful transaction  "rCode": "1",  "rFlag": "SOK",  "reconciliationId": "W1HKS21YLSOR",  "rMessage": "Request was processed successfully.",  "returnCode": 1010000  },  {  "name": "ics\_bill", **🡨** card settlement  "reasonCode": "100", **🡨** successful transaction  "rCode": "1",  "rFlag": "SOK",  "reconciliationId": "79518956",  "rMessage": "Request was processed successfully.",  "returnCode": 1260000  }  ]  }  }  ]  }  } |

See <https://developer.cybersource.com/docs/cybs/en-us/apple-pay/developer/amexbrighton/rest/applepay/reason_code_descriptions.html> for the list of “reasonCode” values.

## Find Expiring and Expired Credit Cards

CyberSource doesn’t appear to have this feature, so we would just use the expirationYear and expirationMonth returned from [Find Payment Method](#_Find_Payment_Method) response when creating the payment method record, and use that to select records to be processed.

# Microform User Interface

Microform Integration replaces the card number and security code fields of a client application with Cybersource-hosted fields that accept payment information securely and replaces it with a short-term transient token with a duration of 15 minutes that can be used for payment or saving the payment method to create a permanent token (similar to vaulting a Braintree nonce).

For web applications, these fields can be styled to match the current UI. For mobile applications, these fields must be hosted on a web page and the native application must load the hosted card entry form web page in a web view.

* iOS sample: <https://github.com/cybersource/flex-v2-ios-sample>
* Android sample: <https://github.com/cybersource/flex-v2-android-sample>

<https://developer.cybersource.com/docs/cybs/en-us/digital-accept-flex/developer/all/rest/digital-accept-flex/ctp-intro/ctp-tokens-intro.html>

The hosted fields have their own validation rules based on the configured “allowedCardNetworks”, and the form provides events that can be used to detect validation errors such as entering a MasterCard number when only Visa is configured, which could also trigger input field style changes.

|  |  |
| --- | --- |
| **Braintree form** | **Cybersource Microform** |
|  | A screenshot of a login form  Description automatically generated |

The first step in integrating with Microform Integration is developing the server-side code that generates the capture context which is a digitally signed JWT that provides authentication, one-time keys, and the target origin to the Microform Integration application. See [Generate Client Token](#_Generate_Client_Token) for sample code used to generate the capture context.

Express route to render the Microform:

|  |
| --- |
| // Render microform using captureContext  app.get('/checkout', async function (req, res) {  try {  var captureContext = await generateCaptureContext();  await validateCaptureContext(captureContext); // will throw error if invalid  res.render('checkout', { captureContext });  } catch (err) {  console.log(err);  }  }); |

Example code for the checkout page: (see views/checkout.ejs in my sample repo)

|  |
| --- |
| <form action="/payment" method="post"  <div id="cardNumber-container" class="form-control"></div>  <div id="securityCode-container" class="form-control"></div>  <input type="hidden" id="transientToken" name="transientToken">  </form>  <script src="flex-microform.min.js"></script>  <script>  <-- initialize microform -->  var flex = new Flex(captureContext);  var microform = flex.microform({ styles: {…}});  var cardNumber = microform.createField('number', { placeholder: '…' });  var securityCode = microform.createField('securityCode', { placeholder: '…' });  <-- replace input fields with microform fields -->  cardNumber.load('#cardNumber-container');  securityCode.load('#securityCode-container');  <-- pay button click handler -->  microform.createToken(options, function (err, token) {  if (err) {  for (let details of err.details) {  if (details.location === 'number') {  // valid number is required  } else if (details.location === 'securityCode') {  // valid security code is required  }  }  } 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.  transientToken.value = JSON.stringify(token);  form.submit();  }  });  </script> |

Sample decoded transient token (JWT):

|  |
| --- |
| {  "iss": "Flex/07",  "exp": 1728068879,  "type": "mf-2.0.0",  "iat": 1728067979,  "jti": "1E36E8UME8403QX532YNKVO56QZQ6JZV46WSNFLIESC7S0XHUHJC67003D0F523A",  "content": {  "paymentInformation": {  "card": {  "expirationYear": {  "value": "2024"  },  "number": {  "maskedValue": "XXXXXXXXXXXX1111",  "bin": "411111"  },  "securityCode": {},  "expirationMonth": {  "value": "12"  },  "type": {  "value": "001"  }  }  }  }  } |

## Sequence Diagram for Tokenization and Payment Processing

A document with text and arrows

Description automatically generated

Figure 1 Sequence diagram for tokenization and payment processing

# Payments Schema

## Merchant Accounts

***Braintree:*** *“a business can have multiple merchant accounts to process transactions for different currencies or businesses. A merchant account ID is a unique identifier for each merchant account within a Braintree gateway.”*

***Cybersource:*** *“allows businesses to use multiple accounts for processing functions.”, but t*he only reference I can find to passing this in a payment request is from <https://developer.cybersource.com/api-reference-assets/index.html#payments> using the acquirerInformation.merchantId, but that is defined as the *“Username for the visa directory server that is created when your acquirer sets up your account. This ID might be the same as your merchant ID. the username can be 15 or 23 characters.”*, so no idea if this is the same thing…

If merchant accounts are required and supported by Cybersource, we would need to configure the merchant\_accounts in the “payments” schema like we do for Braintree. The payment service requires that an agency/program have a configured merchant account record, but the merchant\_account\_id value is optional and not currently required for PayPal.

## Users

No change is required to the schema, because a customer id eg: “23AC2720514EA950E063AF598E0AF5C1” will be stored in the “payment\_processor\_customer\_id” field which is defined as varchar(50).

Mapping from a create customer response

|  |  |
| --- | --- |
| **users** | **createUser response** |
| payment\_processor\_customer\_id | id |

## Payment Methods

No change is required to the schema, because a payment token eg: “23FE0306218C3D37E063AF598E0A7E90” will be stored in the “token” field which is defined as varchar(64).

Mapping from a create customer and payment method, or add payment method response

|  |  |
| --- | --- |
| **payment\_methods** | **createCustomer or addPayment response** |
| token | tokenInformation.paymentInstrument.id |
|  | [**Find Payment Method**](#_Find_Payment_Method) **response** |
| source | from card.type eg: “visa” |
| nickname | from card.type and \_embedded instrumentIdentifier.card.number |
| credit\_card\_expiration\_date | from card.expirationMonth and card.expirationYear |
| is\_default | true (when creating new customer) |

## Sale Transactions

No change is required to the schema, because a transaction ID eg: “7285030748136921703955” will be stored in the “payment\_processor\_transaction\_id” field of direct\_payment\_transactions or wallet\_refill\_transactions which are defined as varchar(50).

Mapping from a sale transaction response

|  |  |
| --- | --- |
| **transaction record** | **createUser response** |
| payment\_processor\_transaction\_id | id |
| payment\_status | from status |
| payment\_failure\_code | TBD |
| payment\_failure\_message | TBD |

Mapping from a void transaction response

|  |  |
| --- | --- |
| **transaction record** | **createUser response** |
| payment\_processor\_transaction\_id | id |
| payment\_status | Void |
| payment\_failure\_code | TBD |
| payment\_failure\_message | TBD |

## Auth Transactions

No change is required to the schema, because a transaction ID eg: “7285030748136921703955” will be stored in the “payment\_processor\_transaction\_id” field which is defined as varchar(50).

Mapping from an auth transaction response

|  |  |
| --- | --- |
| **transaction record** | **createUser response** |
| payment\_processor\_transaction\_id | id |
| payment\_status | from status |
| payment\_failure\_code | TBD |
| payment\_failure\_message | TBD |
| expiration\_date | TBD |

Mapping from a void transaction response

|  |  |
| --- | --- |
| **transaction record** | **createUser response** |
| payment\_processor\_transaction\_id | id |
| payment\_status | Void |
| payment\_failure\_code | TBD |
| payment\_failure\_message | TBD |

## Refund Transactions

No change is required to the schema, because the transaction ID eg: “7286681477456003604953” will be stored in the “payment\_processor\_transaction\_id” field which is defined as varchar(50).

Mapping from a refund transaction response

|  |  |
| --- | --- |
| **transaction record** | **createUser response** |
| payment\_processor\_transaction\_id | Id |
| refund\_status | from status |
| refund\_failure\_code | TBD |
| refund\_failure\_message | TBD |

# Payments Service

## Configuration

The Cybersource payments API has similar configuration values as Braintree, so we would also get these values from the .env file.

Braintree config:

|  |
| --- |
| const gateway = new braintree.BraintreeGateway({ // v3  environment: braintree.Environment[process.env.BRAINTREE\_ENVIRONMENT],  merchantId: process.env.BRAINTREE\_MERCHANT\_ID,  publicKey: process.env.BRAINTREE\_PUBLIC\_KEY,  privateKey: process.env.BRAINTREE\_PRIVATE\_KEY  }); |

Cybersource config:

|  |
| --- |
| {  runEnvironment: 'apitest.cybersource.com',  merchantID: 'transsightdev\_1718140723',  merchantKeyId: 'ad81163c-aa36-471d-be21-9ac7c7ebbe99',  merchantsecretKey: 'H/P7ehPswD5Y2xr/kUTIWG2mk37G63rsI5tvfCtZ34g='  }; |

## Proposed Implementation

* Separate out the Braintree specific code into a new module.
* Create a new Cybersource module having the same interface.
* Create a payment provider module with this interface that creates an instance of either module depending on the configured payment provider.
* All calls will be changed to use the new payment provider module.

Payment Service API instance supports only one payment provider

Assuming that a payment service API instance such as <https://parkingapi4.transsight.com/modpayment> would only support Braintree **OR** Cybersource, we would need to configure a PAYMENT\_PROCESSOR setting in the .env file that would either be “BRAINTREE” or “CYBERSOURCE”, and use that to select the payment provider implementation.

Payment Service API instance supports multiple payment providers

If a payment service API instance needs to support both Braintree **AND** Cybersource, the client calling the API would need to specify the payment provider in the request, and we would need to capture this in the payment\_methods, and transaction tables.

# Optimizing the Settlement Process

Customers are having to wait until at least the next day for a payment transaction to settle before it can be refunded. Below are the settlement schedules and recommendations.

## Braintree

<https://developer.paypal.com/braintree/articles/get-started/transaction-lifecycle>

<https://developer.paypal.com/braintree/articles/au/transactions/settlement-funding-timeline>

<https://xplus3.net/2020/02/24/instant-braintree-transaction-settlement/>

According to the Braintree docs:

Transactions are submitted for settlement nightly, and the exact time depends on the processor.

The time it takes for a Braintree transaction to settle after it's submitted depends on the payment method and the type of card used:

* Visa, Mastercard, Discover, JCB, Diner's Club, and UnionPay: Usually deposited into your bank account in 2-3 business days
* American Express: Typically deposited in 3-5 days or less

## Cybersource

Aside from this generic information, I was not able to find any specific details on how long a settlement can take: *“Cybersource transfers funds from a business bank account to the acquirer for daily settlement.”*

## Recommendation

Check for settled transactions more frequently and keep track of the time to settle based on the payment source so that we can fine-tune the schedule.

We could also allow a transaction to be voided if it has not been settled, but this would only apply to a cancellation where we are refunding the entire purchase amount.

# Issues and Questions

## Billing information required when vaulting each payment method

|  |
| --- |
| billTo: { // all are required  email: 'david.balmer@transsight.com',  firstName: 'David',  lastName: 'Balmer',  address1: '3355 Geary Blvd.',  locality: 'San Francisco', // city  country: 'US', // two character ISO code  administrativeArea: 'CA', // state code  postalCode: '94118' // zipcode  } |

If these required properties are missing, the request is rejected with a validation error:

|  |
| --- |
| {  "status": "INVALID\_REQUEST",  "reason": "MISSING\_FIELD",  "message": "Declined - The request is missing one or more fields",  "details": [  {  "field": "orderInformation.billTo.locality",  "reason": "MISSING\_FIELD"  },  {  "field": "orderInformation.billTo.lastName",  "reason": "MISSING\_FIELD"  },  {  "field": "orderInformation.billTo.email",  "reason": "MISSING\_FIELD"  },  {  "field": "orderInformation.billTo.address1",  "reason": "MISSING\_FIELD"  },  {  "field": "orderInformation.billTo.country",  "reason": "MISSING\_FIELD"  }  ]  } |

**TODO:** need to find out from Cybersource if there is any way to specify which fields are required, because

Braintree only requires the card number, expiration date and zipcode, **but not security code**.

# Testing

## Dave’s Sandbox Account

<https://ebc2test.cybersource.com/ebc2>

Account ID: TransSightDev\_1718140723\_nt

Organization ID: transsightdev\_1718140723

username: evadremlab

password: SYq9dAdi43!!

|  |
| --- |
| MerchantId = 'transsightdev\_1718140723';  MerchantKeyId = 'ad81163c-aa36-471d-be21-9ac7c7ebbe99';  MerchantSecretKey = 'H/P7ehPswD5Y2xr/kUTIWG2mk37G63rsI5tvfCtZ34g='; |

## Test Credit Card Numbers

<https://developer.cybersource.com/hello-world/testing-guide.html>

# Developer Notes

## Changes in latest version of cybersource-rest-client

The <https://github.com/cybersource/cybersource-rest-samples-node> repo does have the latest usage, but

<https://github.com/cybersource/cybersource-flex-samples-node> has outdated code, so have a look at this doc for some of the changes:

<https://stackoverflow.com/questions/77962470/cybersource-rest-client-webpack-imported-module-0-keygenerationapi-is-not-a-c>

# Summary

No schema changes required to store the Cybersource customer ID, payment method token, or transaction ID.

Cybersource does not support saved payment methods for PayPal or Venmo.

Cybersource supports multiple refunds against the same payment transaction, the same as Braintree.

The search results for settled transactions does not return a settled date, so would need to use now().

Adding a Cybersource payment method requires more information than Braintree (see [Billing information required when vaulting each payment method](#_Billing_information_required)):

* security code (CCV)
* billing information
  + email
  + first and last name
  + address line 1
  + city
  + state and country codes

The Click to Pay Drop-In UI supports Visa, Mastercard, American Express, and Discover, but is much more complex than our current Braintree drop-in UI and it’s not clear from the documentation whether it can be used to generate a transient token for vaulting a payment method.

<https://developer.cybersource.com/docs/cybs/en-us/click-to-pay/developer/all/rest/click-to-pay/ctp-intro.html>