# Microform Integration Developer's Guide

Digital Accept Secure Integration

V1 May 2024

# Contents

# 1 Recent Revisions to This Document

**None.**

# 2 About This Guide

This guide is intended for online businesses who have developed their own custom shopping cart and want to enable acceptance of customer payment information.

## 2.1 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.

**WARNING!** A warning contains information or instructions, which, if not heeded, can result in a security risk, irreversible loss of data, or significant cost in time or revenue or both.

# 3 Microform Integration v2

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

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

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

## 3.1 How It Works

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

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

## 3.2 PCI Compliance

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

To meet this requirement, Microform Integration renders secure iframes for the payment card and card verification number input fields. These iframes are hosted by Bank of America and payment data is submitted directly to Bank of America, never touching your systems.

## 3.3 Browser Support

- Chrome 37 or later
- Edge 12 or later
- Firefox 34 or later
- Internet Explorer 11 or later
- Opera 24 or later
- Safari 10.1 or later

## 3.4 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.
- Client-side JavaScript library that you integrate into your digital payment acceptance web page for the secure acceptance of payment information.

Implementing Microform Integration is a three-step process:

1. Creating the Server-Side Capture Context
2. Setting Up the Client Side
3. Validating the Transient Token

**Version Numbering**

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

**Upgrade Paths**

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

### 3.4.1 Creating the Server-Side Context

The first step in integrating with Microform Integration is developing the server-side code that generates the capture context. The capture context is a digitally signed JWT that provides

authentication, one-time keys, and the target origin to the Microform Integration application. The target origin is the protocol, URL, and port number (if used) of the page on which you will host the microform. You must use the https://protocol unless you use http://localhost. For example, if you are serving Microform on example.com, the target origin is https://example.com.

You can also configure microform to filter out cards by designating the accepted card types. Sample Microform Integration projects are available for download in the Flex samples on GitHub.

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

   For example:

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

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

   For example:

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

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

   See Example: Node.js REST Code Snippet.

   **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.

### 3.4.2 Validating the Capture Context

The capture context that you generated is a JSON Web Token (JWT) data object. The JWT is digitally signed using a public key. The purpose is to ensure the validity of the JWT and confirm that it comes from Bank of America. 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 Bank of America 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-keysendpoint.

**Example**

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

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

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

**IMPORTANT!** When validating the public key, some cryptographic methods require you to convert the public key to PEM format.

**Resource**

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

- Test: https://apitest.merchant-services.bankofamerica.com/flex/v2/public-keys/{kid}
- Production: https://api.merchant-services.bankofamerica.com/flex/v2/public-keys/{kid}

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

```
eyJraWQiOiIzZyIsImFsZyI6IlJTMjU2In0.eyJmbHgiOnsicGF0aCI6Ii9mbGV4L3YyL3Rva2VucyIsIm
RhdGEiOiI2bUFLNTNPNVpGTUk5Y3RobWZmd2doQUFFRGNqNU5QYzcxelErbm8reDN6WStLOTVWQ2c5bThm
QWs4czlTRXBtT21zMmVhbEx5NkhHZ29oQ0JEWjVlN3ZUSGQ5YTR5a2tNRDlNVHhqK3ZoWXVDUmRDaDhVY1
dwVUNZWlZnbTE1UXVFMkEiLCJvcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbSIs
Imp3ayI6eyJrdHkiOiJSU0EiLCJlIjoiQVFBQiIsInVzZSI6ImVuYyIsIm4iOiJyQmZwdDRjeGlkcVZwT0
pmVTlJQXcwU1JCNUZqN0xMZjA4U0R0VmNyUjlaajA2bEYwTVc1aUpZb3F6R3ROdnBIMnFZbFN6LVRsSDdy
bVNTUEZIeTFJQ3BfZ0I3eURjQnJ0RWNEanpLeVNZSTVCVjNsNHh6Qk5CNzRJdnB2Smtqcnd3QVZvVU4wM1
RaT3FVc0pfSy1jT0xpYzVXV0ZhQTEyOUthWFZrZFd3N3c3LVBLdnMwNmpjeGwyV05STUIzTS1ZQ0xOb3FC
dkdCSk5oYy1uM1lBNU5hazB2NDdiYUswYWdHQXRfWEZ0ZGItZkphVUVUTW5WdW9fQmRhVm90d1NqUFNaOH
FMOGkzWUdmemp2MURDTUM2WURZRzlmX0tqNzJjTi1OaG9BRURWUlZyTUtiZ3QyRDlwWkJ1d2gzZlNfS3VR
clFWTVdPelRnT3AzT2s3UVFGZ1EiLCJraWQiOiIwOEJhWXMxbjdKTUhjSDh1bkcxc1NDUVdxN2VveWQ1Zy
J9fSwiY3R4IjpbeyJkYXRhIjp7InRhcmdldE9yaWdpbnMiOlsiaHR0cHM6Ly93d3cudGVzdC5jb20iXSwi
bWZPcmlnaW4iOiJodHRwczovL3Rlc3RmbGV4LmN5YmVyc291cmNlLmNvbSJ9LCJ0eXBlIjoibWYtMC4xMS
4wIn1dLCJpc3MiOiJGbGV4IEFQSSIsImV4cCI6MTYxNjc3OTA5MSwiaWF0IjoxNjE2Nzc4MTkxLCJqdGki
OiJ6SG1tZ25uaTVoN3ptdGY0In0.GvBzyw6JKl3b2PztHb9rZXawx2T817nYqu6goxpe4PsjqBY1qeTo1
9R-CP_DkJXov9hdJZgdlzlNmRY6yoiziSZnGJdpnZ-pCqIlC06qrpJVEDob3O_efR9L03Gz7F5JlLOiTXS
j6nVwC5mRlcP032ytPDEx5TMI9Y0hmBadJYnhEMwQnn_paMm3wLh2v6rfTkaBqd8n6rPvCNrWMOwoMdoTe
Fxku-d27jlA95RXqJWfhJSN1MFquKa7THemvTX2tnjZdTcrTcpgHlxi22w7MUFcnNXsbMouoaYiEdAdSlC
Z7LCXrS1Brdr_FWDp7v0uwqHm7OALsGrw8QbGTafF8w
```


**Example**

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

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

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


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

Introduction to JWT
JWT (signed)
Specification JWK
Specification

### 3.4.3 Setting Up the Client Side

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

**Web Page**

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

1. Add the Microform Integration JavaScript library to your page by loading it directly from Bank of America. See Version Numbering. You should do this dynamically per environment by using the asset path returned in the JWT from `/microform/v2/sessions`.

   For example:

```
ctx": [
    {
      "data": {
        "clientLibrary":
 https://testflex.merchant-
services.bankofamerica.com/microform/bundle/v2/fl ex-microform.min.js,
      ...
```

   - **Test**: `<scriptsrc="https://testflex.merchant- services.bankofamerica.com/microform/ bundle/v2/flex-microform.min.js"></script>`
   - **Production**: `<scriptsrc="https://flex.merchant-services.bankofamerica.com/ microform/bundle/v2/flex-microform.min.js"></script>`

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

   Microform Integration attaches the microform fields to containers within your HTML. Within your HTML checkout, replace the payment card and CVN tag with a simple container.

   Microform Integration uses the container to render an iframe for secured credit card input. The following example contains simple div tags to define where to place the PAN and CVN fields within the payment acceptance page: `<div id="number-container" class="form-control"></ div>`. See Example: Checkout Payment Form.

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

   `var flex = new Flex(captureContext);`

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

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

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

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

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

**Mobile Application**

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

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

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

- iOS sample: https://github.com/
- Android sample: https://github.com/

**Transient Token Time Limit**

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

See Example: Creating the Pay Button with Event Listener.

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

See Example: Customer-Submitted Form.

**Transient Token Response Format**

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

- Header
- Payload
- Signature

JWT example: `xxxxx.yyyyy.zzzzz`

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

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

See Example: Token Payload.

**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 Bank of America issued the token and that no data tampering occurred during transit. See Example: Capture Context Public Key.

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

See Example: Validating the Transient Token.

**Using the Transient Token**

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

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 jtivalue 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": {<br><br>"transientTokenJwt":<br>"eyJraWQiOiIwNzRsM3p5M2xCRWN5d1gxcnhXNFFoUmJFNXJLN1NmQiIsImFsZyI6IlJTMjU2In0.eyJkYXRhIjp7ImV4cGlyYXRpb25ZZWFyIjoiMjAyMSIsIm51bWJlciI6IjQxMTExMVhYWFhYWDExMTEiLCJleHBpcmF0aW9uTW9udGgiOiIwNSIsInR5cGUiOiIwMDEifSwiaXNzIjoiRmxleC8wOCIsImV4cCI6MTU4ODcwMjkxNSwidHlwZSI6Im1mLTAuMTEuMCIsImlhdCI6MTU4ODcwMjAxNSwianRpIjoiMUU0Q0NMSUw4NFFXM1RPSTFBM0pUU1RGMTZGQUNVNkUwNU9VRVNGWlRQNUhIVkJDWTQwUTVFQjFBRUMzNDZBMCJ9.FB3b2r8mjtvqo3_k05sRIPGmCZ_5dRSZp8AIJ4u7NKb8E0-6ZOHDwEpxtOMFzfozwXMTJ3C6yBK9vFIPTIG6kydcrWNheE2Pfort8KbxyUxG-PYONY-xFnRDF841EFhCMC4nRFvXEIvlcLnSK6opUUe7myKPjpZI1ijWpF0N-DzZiVT8JX-9ZIarJq2OI0S61Y3912xLJUKi5c2VpRPQOS54hRr5GHdGJ2fV8JZ1gTuup_qLyyK7uE1VxI0aucsyH7yeF5vTdjgSd76ZJ1OUFi-3Ij5kSLsiX 4j-D0T8ENT1DbB_hPTaK9o6qqtGJs7QEeW8abtnKFsTwVGrT32G2w"<br>} |
| REST API with JSON Web Token ID | "tokenInformation": {<br><br>"jti":<br>"1E3GQY1RNKBG6IBD2EP93C43PIZ2NQ6SQLUIM3S16BGLHTY4IIEK5EB1AE5D73A4",<br>} |

See Example: Authorization with a Transient Token Using the REST API.

### 3.4.4 Getting Started Examples

**Example: Node.js REST Code Snippet**

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

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

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

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

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

Back to Creating the Server-Side Context.

**Example: Checkout Payment Form**

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

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

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

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

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

Back to Setting Up the Client Side.

**Example: Creating the Pay Button with Event Listener**

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

      // Compiling MM & YY into optional parameters
      var options = {
      expirationMonth: document.querySelector('#expMonth').value,
      expirationYear: document.querySelector('#expYear').value
      };
      //
      microform.createToken(options, function (err, token) {
        if (err) {
          // handle error
          console.error(err);
          errorsOutput.textContent = err.message;
        } else {
          // At this point you may pass the token back to your server as you wish.

          // In this example we append a hidden input to the form and submit it.

          console.log(JSON.stringify(token));
          flexResponse.value = JSON.stringify(token);
          form.submit();
         }
       });
     });
```

Back to Transient Token Time Limit.

**Example: Customer-Submitted Form**

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

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

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

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

  // Compiling MM & YY into optional paramiters
    var options = {
      expirationMonth: document.querySelector('#expMonth').value,
      expirationYear: document.querySelector('#expYear').value
    };
  //
    microform.createToken(options, function (err, token) {
      if (err) {
        // handle error
        console.error(err);
        errorsOutput.textContent = err.message;
      } else {
        // At this point you may pass the token back to your server as you
wish.

        // In this example we append a hidden input to the form and submit it.

        console.log(JSON.stringify(token));
        flexResponse.value = JSON.stringify(token);
        form.submit();
       }
     });
   });
</script>
```

Back to Transient Token Time Limit.

**Example: Token Payload**

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

Back to Transient Token Response Format.

**Example: Token Payload with Multiple Card Types**

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

```
        }
      }
    }
  }
}
```

Back to Transient Token Response Format.

**Example: Capture Context Public Key**

```
"jwk": {
                "kty": "RSA",
                "e": "AQAB",
                "use": "enc",
                "n":
 "3DhDtIHLxsbsSygEAG1hcFqnw64khTIZ6w9W9mZNl83gIyj1FVk-H5GDMa85e8RZFxUwgU_zQ0kHLtON
o8SB52Z0hsJVE9wqHNIRoloiNPGPQYVXQZw2S1BSPxBtCEjA5x_-bcG6aeJdsz_cAE7OrIYkJa5Fphg9_p
xgYRod6JCFjgdHj0iDSQxtBsmtxagAGHjDhW7UoiIig71SN-f-
gggaCpITem4zlb5kkRVvmKMUANe4B36v4XSSSpwdP_H5kv4JDz_cVlp_Vy8T3AfAbCtROyRyH9iH1Z-4Yy
6T5hb-9y3IPD8vlc8E3JQ4qt6U46EeiKPH4KtcdokMPjqiuQ",
                "kid": "00UaBe20jy9VkwZUQPZwNNoKFPJA4Qhc"    }
```

Back to Validating the Transient Token.

**Example: Validating the Transient Token**

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

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

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

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

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

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

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

Back to Validating the Transient Token.

**Example: Authorization with a Transient Token using the REST API**

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

     "transientTokenJwt":
"eyJraWQiOiIwN0JwSE9abkhJM3c3UVAycmhNZkhuWE9XQlhwa1ZHTiIsImFsZyI6Il
```

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

Back to Using the Transient Token.

## 3.5 Styling

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

### 3.5.1 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.

### 3.5.2 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.

### 3.5.3 Managed Classes

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

| Class | Description |
|---|---|
| .flex-microform | Base class added to any element in which a field has been loaded. |
| .flex-microform-disabled | The field has been disabled. |
| .flex-microform-focused | The field has user focus. |
| .flex-microform-valid | The input card number is valid. |
| .flex-microform-invalid | The input card number invalid. |
| .flex-microform-autocomplete | The field has been filled using an autocomplete/autofillevent. |

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 {
  eight: 20px;
  background: #ffffff;
  -webkit-transition: background 200ms;
  transition: background 200ms;
}

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

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

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

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

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

### 3.5.4 Input Field Text

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

```
var customStyles = {
  'input': {
    'font-size': '16px',
    'color': '#3A3A3A'
  },
  '::placeholder': {
    'color': 'blue'
  },
  ':focus': {
    'color': 'blue'
  },
  ':hover': {
    'font-style': 'italic'
  },
  ':disabled': {
    'cursor': 'not-allowed',
  },
  'valid': {
    'color': 'green'
  },
  'invalid': {
    'color': 'red'
  }
};

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

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

### 3.5.5 Supported Properties

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

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

## 3.6 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 autocompleteevent. |
| 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.

### 3.6.1 Subscribing to Events

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

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

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

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

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

The dataobject supplied to the event listener's callback includes any information specific to the triggered event.

### 3.6.2 Card Detection

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

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

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

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

### 3.6.3 Autocomplete

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

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

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

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

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

## 3.7 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.merchant-services.bankofamerica.com/ | https://flex.merchant-services.bankofamerica.com/ |
| child-src | https://testflex.merchant-services.bankofamerica.com/ | https://flex.merchant-services.bankofamerica.com/ |
| script-src | https://testflex.merchant-services.bankofamerica.com/ | https://flex.merchant-services.bankofamerica.com/ |

## 3.8 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.

### 3.8.1 Self-Assessment Questionnaire

Microform Integration handles the card number input and transmission from within iframe elements served from Bank of America 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.

### 3.8.2 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.

## 3.9 API Reference

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

### 3.9.1 Class: Field

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

**Methods**

clear()

Programmatically clear any entered value within the field.

**Example**

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

dispose()

Permanently remove this field from your Microform integration.

**Example**

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

### focus()

Programmatically set user focus to the Microform input field.

**Example**

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

### load(container)

Load this field into a container element on your page. Successful loading of this field will trigger a load event. **Parameters**

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

**Examples**

Using a CSS selector

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

Using an HTML element

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

### off(type, listener)

Unsubscribe an event handler from a Microform Field.

**Parameters**

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

**Example**

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

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

`on(type, listener)`

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

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

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

**Parameters**

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

**Example**

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

### unload()

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

**Example**

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

### update(options)

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

**Parameter**

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

**Example**

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

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

**Events**

### Autocomplete

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

- name
- expirationMonth
- expirationYear

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

**Examples:**

Possible format of data supplied to callback

```
{
     name: '_______',
     expirationMonth: '__',
     expirationYear: '___'
}
```

Updating the rest of your checkout after an autocomplete event

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

### blur

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

**Example:**

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

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

### change

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

**Type:** object

**Properties**

| Name | Type |
|---|---|
| card | object |
| valid | boolean |
| couldBeValid | boolean |
| empty | boolean |

**Examples:**

Minimal example:

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

Use the card detection result to update your UI.

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

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

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

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

```
    // 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';
  }
});
```

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

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

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

Updating validation styles on your form element.

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

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

`focus`

Emitted when the input field has received focus.

**Example:**

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

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

### inputSubmitRequest

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

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

**Example:**

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

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

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

### load

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

**Example:**

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

### unload

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

**Example:**

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

update

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

**Type:** object

**Example:**

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

### 3.9.2 Module: FLEX

**Flex(captureContext)**

new Flex(captureContext)

For detailed setup instructions, see Getting Started.

**Parameters:**

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

**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.

**Parameter**

| Name | Type | Description |
|---|---|---|
| options | Object | |

**Property**

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

**Returns:**

Type: Microform

**Examples:**

Minimal Setup

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

Custom Styling

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

### 3.9.3 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.

**Methods**

createField(fieldType, optionsopt) > {Field}

Create a field for this Microform integration.

**Parameters**

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

**Properties**

| Name | Type | Attributes | Default | Description |
|---|---|---|---|---|
| placeholder | string | <optional> | | Sets the placeholderattribute on the input. |
| title | string | <optional> | | Sets the title attribute on the input. Typically used to display tooltip text on hover. |
| description | string | <optional> | | Sets the input's description foruse by assistive technologies using the aria-describedbyattribute. |
| disabled | Boolean | <optional> | false | Sets the disabledattribute on the input. |
| autoformat | Boolean | <optional> | true | Enable or disable automatic formatting of the input field. This is only supported for number fields and will automatically insert spaces based on the detected card type. |
| 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 3or 4. |
| styles | stylingOptions | <optional> | | Apply custom styling to this field |

**Returns** Type: Field

**Examples**

Minimal Setup

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

Providing Custom Styles

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

Setting the length of a security code field

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

### createToken(options, callback)

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

**Parameters**

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

**Properties**

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

**Examples**

Minimal example omitting all optional parameters.

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

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

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

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

var options = {
  type: document.querySelector('#cardTypeOverride').value
};
microform.createToken(options, function(err, token) {
  // handle errors & token response
});
```

Handling error scenarios

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

### 3.9.4 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} .

**Members**

(static, readonly) Reason Codes - Field Load Errors

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

**Properties**

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

(static, readonly)Reason Codes - Field object Creation

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

**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_UNSUPPORTED_FIELD_TYPE | string | Occurs when the iframe is attempting to load with an invalid field type. |
| IFRAME_JWT_VALIDATION_FAILED | string | Occurs when the iframe cannot validate the JWT passed. |

(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 createTokencall was unable to proceed. |
| CREATE_TOKEN_XHR_ERROR | string | Occurs when there is a network error when attempting to create a token. |
| CREATE_TOKEN_NO_FIELDS | string | Occurs when the data fields are unavailable for collection. |

| Name | Type | Description |
| --- | --- | --- |
| CREATE_TOKEN_VALIDATION_PARAMS | string | Occurs when there's an issue with parameters supplied to createToken. |
| CREATE_TOKEN_VALIDATION_FIELDS | string | Occurs when there's a validation issue with data in your loaded fields. |
| CREATE_TOKEN_VALIDATION_SERVERSIDE | string | Occurs when server-side validation rejects the createTokenrequest. |
| CREATE_TOKEN_UNABLE_TO_START | string | Occurs when no loaded field was able to handle the createTokenrequest. |

(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

### 3.9.5 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 autocompleteevent. |
| 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 microformInstanceobject, you can easily subscribe to any of the supported events.

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

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

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

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

The dataobject supplied to the event listener's callback includes any information specific to the triggered event.

**Card Detection**

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

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

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

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

**Autocomplete**

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

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

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

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

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

### 3.9.6 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 errargument of the callback. If no error occurs, errhas 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 59). | <optional> <nullable> | An Object detailing occurred errors, otherwise null. |
| data | * | <optional> <nullable> | In success scenarios, this is whatever data has been returned by the asynchronous function call, if any. |

**Returns**

Type: void

**Example**

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

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

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

StylingOptions

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

Supported CSS selectors:

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

Supported CSS properties:

- color
- cursor
- font
- font-family

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

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

**Properties**

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

**Example**

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

## 3.10 Using Microform with the Checkout API

Use the Digital Accept Checkout API in conjunction with Microform technologies to provide a cohesive PCI SAQ A embedded payment application within your merchant e-commerce page. The Digital Accept Checkout API provides access to payment processing and additional value-added services directly from the browser.

This approach lets the integrator manage the entire consumer experience with the exception of two Microform fields which are embedded within the page to capture the PAN and/or CVV data in a secure fashion. Microform technology embeds invisible iFrames within a merchant's payment page for the secure capture of sensitive payment information.

**Basic Flow**

1. Call the /sessionsendpoint to generate a server-to-server capture context.
    a. Define the targetOrigin of the Microform webpage.
    b. Define the signed fields for the Checkout API.
    c. Define the unsigned fields of the Checkout API.
2. Within the browser:
    a. Invoke the microform using the capture context.
    b. Capture the response transient token.
    c. Invoke the Checkout API via HTTP POST.

### 3.10.1 Requesting a Capture Context

In order to support Microform transient tokens through the Checkout API, we created a new endpoint: POST/microform/v2/sessions. This new endpoint produces a capture context that is compatible with both Microform and the Checkout API.

This endpoint replaces the need for a HMAC-SHA256 signature in Checkout API initialization.

**Microform Integration 0.11 Setup**

Follow the Setting Up the Client Side (on page 25) to initialize and trigger tokenization. (createToken).

Also, see this example Checkout Payment Form (on page 29).

**Resource**

Send an authenticated POST request to the /sessionsAPI:

- Test: https://apitest.merchant-services.bankofamerica.com/microform/v2/sessions
- Production: https://api.merchant-services.bankofamerica.com/microform/v2/sessions

Authenticate to the API using HTTP Signature or JSON Web Token (JWT) authentication. See the Getting Started with REST API developer guide for more information.

**Required Fields**

Always include the following fields:

`targetOrigins`

The merchant origin(s). For example, `https://example.com`.
Required to comply with CORS and CSP standards.

`checkoutApiInitialization`

This field contains Checkout API request fields.

Always include the following fields, which the Checkout API requires:

**access_key**

**profile_id**

**preference_number**

**transaction_type**

**transaction_uuid**

The following fields are not required, but if you do pass them, pass them inside the capture context:

**amount**

**currency**

**ignore_avs**

**ignore_cvn**

**payment_token**

**override_custom_receipt_page**

**unsigned_field_names**

If you wish to supply unsigned fields, then you must include this field in the capture context. This field is a comma-separated list of field names.

If you pass a field to the endpoint without listing it in this field, it will not result in an error. Instead, the field is ignored.

**IMPORTANT!** To use a transient token with the Checkout API, you must, at a minimum, include the transient_token field inside this field.

**Signed fields**

Signed fields refer to those fields included in the capture context, and which are thus signed by the Microform Integration 0.11.

Some reasons why fields are signed:

1. To prevent data tampering.
2. If they have already been collected.
3. They do not fall under PCI scope. For example, the field that captures the card number falls under the PCI scope.

If you have an existing integration with the Checkout API, this is similar to how the signed_field_namesare used.

**Unsigned fields**

Unsigned fields refer to those fields not included in the capture context, but which are supplied to the Checkout API.

These include fields which have not yet been collected, such as the billing address, the transient token, or may include fields which fall under PCI scope e.g., **card_number**.

Unsigned fields are not signed by the Microform Integration 0.11 and so are subject to tampering.

**Examples**

Include the fields in the request as follows:

```
{
  "targetOrigins": [
    "
            https://www.bofa-merchant.com"

             "
   ],
   "clientVersion": "v2.0",
   "checkoutApiInitialization": {
     "field_a": "value_a",
     …
   }
}
```

**An authorization using a transient token with unsigned billing details**

```
{
  "targetOrigins": [
    "https://www.my-merchant-website.com"
  ],
  "clientVersion": "v2.0",
  "checkoutApiInitialization": {
    "profile_id": "12341234-1234-1234-1234-123412341234",
    "access_key": "acce55acce55acce55acce55acce5500",
    "reference_number": "1611305732",
    "transaction_uuid": "1611305732-001",
    "transaction_type": "authorization",
    "currency": "USD",
    "amount": "100.00",
    "locale": "en-us",
    "unsigned_field_names":
 "transient_token,bill_to_forename,bill_to_surname,bill_to_phone,

 bill_to_email,bill_to_address_line1,bill_to_address_line2,bill_to_address_city,
     bill_to_address_state,bill_to_address_postal_code,bill_to_address_country"
  }
}
```

**An authorization using a transient token with signed billing details**

```
{

    "targetOrigins": [
        "
             https://www.bofa-merchant.com"

              "
    ],
    "clientVersion": "v2.0",
    "checkoutApiInitialization": {
        "profile_id": "12341234-1234-1234-1234-123412341234",
        "access_key": " acce55acce55acce55acce55acce5500",
        "reference_number": "1611305732",
        "transaction_uuid": "1611305732-001",
        "transaction_type": "authorization", "currency":
        "USD",
        "amount": "100.00",
```

```
        "locale": "en-us",
        "bill_to_forename": "Joe",
        "bill_to_surname": "Soap",
        "bill_to_phone": "07788888888",
        "bill_to_email":
 "payer_auth_vi_2.1.0_su@merchant-services.bankofamerica.com",
        "bill_to_address_line1": "1 My Apartment",
        "bill_to_address_line2": "20 My Street",
        "bill_to_address_city": "San Francisco",
        "bill_to_address_state": "CA",
        "bill_to_address_postal_code": "94043",
        "bill_to_address_country": "US",
        "unsigned_field_names": "transient_token"
    }
}
```

**An authorization using a transient token with a payment token (Secure Storage or TMS)**

```
 {

    "targetOrigins": [
        "
            https://www.bofa-merchant.com"

              "
    ],
    "clientVersion": "v2.0",
    "checkoutApiInitialization": {
        "profile_id": "12341234-1234-1234-1234-123412341234",
        "access_key": " acce55acce55acce55acce55acce5500",
        "reference_number": "1611305732",
        "transaction_uuid": "1611305732-001",
        "transaction_type": "authorization", "currency": "USD",
        "amount": "100.00",
        "locale": "en-us",
        "payment_token": "0000000000000000", "unsigned_field_names":
        "transient_token"
    }

 }
```

**An authorization using a transient token with unsigned card type and expiry date fields**

```
{

   "targetOrigins": [
       "
           https://www.bofa-merchant.com"

            "
   ],
   "clientVersion": "v2.0",
   "checkoutApiInitialization": {
       "profile_id": "12341234-1234-1234-1234-123412341234",
       "access_key": " acce55acce55acce55acce55acce5500",
       "reference_number": "1611305732",
       "transaction_uuid": "1611305732-001",
       "transaction_type": "authorization",
       "currency": "USD",
       "amount": "100.00",
       "locale": "en-us",
       "unsigned_field_names": "transient_token,card_type,card_expiry_date"
   }

}
```

### 3.10.2 Invoking the Checkout API

Once you have the transient token provided, the next step is to pass it to the Checkout API.

Make the request to the Checkout API from the customer's browser, using a standard form post (application/x-www-form-urlencoded) request.

If you are using the Checkout API inside an iframe, to avoid issues with third-party cookies not being supported, ensure that you use an iframe endpoint.

**New Checkout API Request Fields**

**capture_context**

The same capture context used with Microform Integration 0.11. This field is not supported with Hosted Checkout.

Capture contexts are valid for 15 minutes only. The Checkout API will not accept expired capture contexts.

Format: String

Required if you want to supply a transient token.

**transient_token**

The transient token JWT provided by Microform Integration 0.11. If you pass this field, you must also pass the corresponding capture context (capture_context) must also be supplied.

You do not need to validate the transient token signature. The Checkout API will do this for you.

**Example**

The following example shows a request that calls the Secure Acceptance Checkout API and creates a token.

(See example next page.)

```
<form id="sa-form" action="">
 <input type="hidden" id="capture_context" name="capture_context"
 value="eyJraWQiOi...HHWuACdnLQ" />
 <input type="hidden" id="transient_token" name="transient_token" value="" />

 <-- Optional unsigned fields -->
 <input type="text" name="bill_to_forename value="" />
 <input type="text" name="bill_to_surname value="" />
 <input type="text" name="bill_to_phone value="" />
 <input type="text" name="bill_to_email value="" />
 <input type="text" name="bill_to_address_line1 value="" />
 <input type="text" name="bill_to_address_line2 value="" />
 <input type="text" name="bill_to_address_city value="" />
 <input type="text" name="bill_to_address_state value="" />
 <input type="text" name="bill_to_address_postal_code value="" />
 <input type="text" name="bill_to_address_country" value="" />
</form>

<script type="text/javascript">
var captureContext = document.getElementById('capture_context').value;

var flex = new Flex(captureContext);

// Initialize Flex Microform ...

payButton.addEventListener('click', function() {
    // Compiling MM & YY into optional parameters
    var options = {
        expirationMonth: document.querySelector('#expMonth').value,
        expirationYear: document.querySelector('#expYear').value
    };
    microform.createToken(options, function(err, token) {
        if (err) {
            // handle error
            console.error(err);
            errorsOutput.textContent = err.message;
        } else {
            document.getElementById('transient_token').value = token;
            // No need to verify JWS
           document.getElementById('sa-form').submit();
        }
    });
});
</script>
```

### 3.10.3 FAQ

Frequently Asked Questions about using the Microform Integration 0.11 with the Secure Acceptance Checkout API.

**Can I supply both a secure storage (TMS) token and a transient token?**

Yes. A secure storage (TMS) token can be supplied in the payment_tokenfield which must be inside the capture context. The transient token is then supplied as an unsigned field (transient_token).

The transient token data will take precedence over the secure storage (TMS) token data.

**Can I use Microform to capture only the security code?**

Yes. You must ensure that the card_typeand card_expiry_dateare supplied via one of the following:

1. Through the payment token
2. Inside the capture context
3. As unsigned fields

**Can I override a transient token field, for example, the card_type field?**

Yes. Fields inside the capture context and unsigned fields both override transient token data.

**Can I use Microform to capture only the card number?**

Yes. You must ensure that the card_typeand card_expiry_dateare supplied either:

1. Inside the capture context
2. As unsigned fields

## 3.11 Using Microform with the Checkout API

Use the Digital Accept Checkout API in conjunction with Microform technologies to provide a cohesive PCI SAQ A embedded payment application within your merchant e-commerce page. The Digital Accept Checkout API provides access to payment processing and additional value-added services directly from the browser.

This approach lets the integrator manage the entire consumer experience with the exception of two Microform fields which are embedded within the page to capture the PAN and/or CVV data in a secure fashion. Microform technology embeds invisible iFrames within a merchant's payment page for the secure capture of sensitive payment information.

**Basic Flow**

1. Call the /sessionsendpoint to generate a server-to-server capture context.
    a. Define the targetOrigin of the Microform webpage.
    b. Define the signed fields for the Checkout API.
    c. Define the unsigned fields of the Checkout API.
2. Within the browser:
    a. Invoke the microform using the capture context.
    b. Capture the response transient token.
    c. Invoke the Checkout API via HTTP POST.

### 3.11.1 Requesting a Capture Context

In order to support Microform transient tokens through the Checkout API, we created a new endpoint: POST/microform/v1/sessions. This new endpoint produces a capture context that is compatible with both Microform and the Checkout API.

This endpoint:

- Replaces POST /flex/v1/keys?format=JWTfor Microform.
- Replaces the need for a HMAC-SHA256 signature in Checkout API initialization.

**Microform Integration 0.11 Setup**

Follow the Setting Up the Client Side to initialize and trigger tokenization. (createToken).

Also, see this example checkout payment form.

**Resource**

Send an authenticated POST request to the /sessionsAPI:

- Test: https://apitest.merchant-services.bankofamerica.com/microform/v1/sessions
- Production: https://api.merchant-services.bankofamerica.com/microform/v1/sessions

Authenticate to the API using HTTP Signature or JSON Web Token (JWT) authentication. See the Getting Started with REST API developer guide for more information.

**Required Fields**

Always include the following fields:

targetOrigins

The merchant origin(s). For example, https://example.com.
Required to comply with CORS and CSP standards.

checkoutApiInitialization

This field contains Checkout API request fields.

Always include these fields, which the Checkout API requires:

**access_key**

**profile_id**

**preference_number**

**transaction_type**

**transaction_uuid**

These fields are not required, but if you do pass them, pass them inside the capture context:

**amount currency**

**ignore_avs**

**ignore_cvn**

**payment_token**

**override_custom_receipt_page unsigned_field_names**

If you wish to supply unsigned fields, then you must include this field in the capture context. This field is a comma-separated list of field names.

If you pass a field to the endpoint without listing it in this field, it will not result in an error. Instead, the field is ignored.

**IMPORTANT!** To use a transient token with the Checkout API, you must, at a minimum, include the `transient_token`field inside this field.

**Signed fields**

Signed fields refer to those fields included in the capture context, and which are thus signed by the Microform Integration 0.11.

Some reasons why fields are signed:

1. To prevent data tampering.
2. If they have already been collected.
3. They do not fall under PCI scope. For example, the field that captures the card number falls under the PCI scope.

If you have an existing integration with the Checkout API, this is similar to how the

`signed_field_names` are used.

**Unsigned fields**

Unsigned fields refer to those fields not included in the capture context, but which are supplied to the Checkout API.

These include fields which have not yet been collected, such as the billing address, the transient token, or may include fields which fall under PCI scope e.g., **card_number**.

Unsigned fields are not signed by the Microform Integration 0.11 and so are subject to tampering.

**Examples**

Include the fields in the request as follows:

```
{
  "targetOrigins": [
    "
            https://www.bofa-merchant.com"
```

```
            "
  ],
  "checkoutApiInitialization": {
    "field_a": "value_a",
    …
  }
}
```

**An authorization using a transient token with unsigned billing details**

```
{
  "targetOrigins": [
    "https://www.my-merchant-website.com"
  ],
  "checkoutApiInitialization": {
    "profile_id": "12341234-1234-1234-1234-123412341234",
    "access_key":
    "acce55acce55acce55acce55acce5500",
    "reference_number": "1611305732",
    "transaction_uuid":
    "1611305732-001",
    "transaction_type":
    "authorization",
    "currency": "USD",
    "amount": "100.00",
    "locale": "en-us",
    "unsigned_field_names": "transient_token,bill_to_forename,bill_to_surname,

 bill_to_phone,bill_to_email,bill_to_address_line1,bill_to_address_line2,bill_to_a
ddress_city,
     bill_to_address_state,bill_to_address_postal_code,bill_to_address_country"
  }
}
```

**An authorization using a transient token with signed billing details**

```
{
    "targetOrigins": [
        "
            https://www.bofa-merchant.com"

             "
    ],
    "checkoutApiInitialization": {
        "profile_id": "12341234-1234-1234-1234-123412341234",
        "access_key": " acce55acce55acce55acce55acce5500",
        "reference_number": "1611305732",
        "transaction_uuid": "1611305732-001",

        "transaction_type": "authorization",
        "currency": "USD",
        "amount": "100.00",
        "locale": "en-us",
        "bill_to_forename": "Joe",
        "bill_to_surname": "Soap",
        "bill_to_phone": "07788888888",
        "bill_to_email": "",
        "bill_to_address_line1": "1 My Apartment",
        "bill_to_address_line2": "20 My Street",
        "bill_to_address_city": "San Francisco",
        "bill_to_address_state": "CA",
        "bill_to_address_postal_code": "94043",
        "bill_to_address_country": "US",
        "unsigned_field_names": "transient_token"
    }
}
```

**An authorization using a transient token with a payment token (Secure Storage or TMS)**

```
{
    "targetOrigins": [
        "
            https://www.bofa-merchant.com"

             "
    ],
    "checkoutApiInitialization": {
        "profile_id": "12341234-1234-1234-1234-123412341234",
        "access_key": " acce55acce55acce55acce55acce5500",
        "reference_number": "1611305732",
        "transaction_uuid": "1611305732-001",
        "transaction_type": "authorization",
        "currency": "USD",
        "amount": "100.00",
        "locale": "en-us",
        "payment_token": "0000000000000000",
        "unsigned_field_names": "transient_token"
    }
}
```

**An authorization using a transient token with unsigned card type and expiry date fields**

```
{
    "targetOrigins": [
        "
            https://www.bofa-merchant.com"

             "
    ],
    "checkoutApiInitialization": {
        "profile_id": "12341234-1234-1234-1234-123412341234",
        "access_key": " acce55acce55acce55acce55acce5500",
        "reference_number": "1611305732",
        "transaction_uuid": "1611305732-001",
        "transaction_type": "authorization",
        "currency": "USD",
        "amount": "100.00",
        "locale": "en-us",
        "unsigned_field_names": "transient_token,card_type,card_expiry_date"
    }
}
```

### 3.11.2 Invoking the Checkout API

Once you have the transient token provided, the next step is to pass it to the Checkout API.

Make the request to the Checkout API from the customer's browser, using a standard form post (application/x-www-form-urlencoded) request.

If you are using the Checkout API inside an iframe, to avoid issues with third-party cookies not being supported, ensure that you use an iframe endpoint.

**New Checkout API Request Fields**

**capture_context**

The same capture context used with Microform Integration 0.11. This field is not supported with Hosted Checkout.

Capture contexts are valid for 15 minutes only. The Checkout API will not accept expired capture contexts.

Format: String

Required if you want to supply a transient token.

**transient_token**

The transient token JWT provided by Microform Integration 0.11. If you pass this field, you must also pass the corresponding capture context (capture_context) must also be supplied.

You do not need to validate the transient token signature. The Checkout API will do this for you.

**Example**

The following example shows a request that calls the Secure Acceptance Checkout API and creates a token.

```
 <form id="sa-form" action="">
  <input type="hidden" id="capture_context" name="capture_context"
  value="eyJraWQiOi...HHWuACdnLQ" />
  <input type="hidden" id="transient_token" name="transient_token" value="" />

  <-- Optional unsigned fields -->
  <input type="text" name="bill_to_forename value="" />
  <input type="text" name="bill_to_surname value="" />
  <input type="text" name="bill_to_phone value="" />
  <input type="text" name="bill_to_email value="" />
  <input type="text" name="bill_to_address_line1 value="" />
  <input type="text" name="bill_to_address_line2 value="" />
  <input type="text" name="bill_to_address_city value="" />
  <input type="text" name="bill_to_address_state value="" />
  <input type="text" name="bill_to_address_postal_code value="" />
  <input type="text" name="bill_to_address_country" value="" />
</form>

<script type="text/javascript">
var captureContext = document.getElementById('capture_context').value;

var flex = new Flex(captureContext);

// Initialize Flex Microform ...

payButton.addEventListener('click', function() {
    // Compiling MM & YY into optional parameters
    var options = {
        expirationMonth: document.querySelector('#expMonth').value,
        expirationYear: document.querySelector('#expYear').value
  };
 microform.createToken(options, function(err, token) {
     if (err) {
         // handle error
         console.error(err);
         errorsOutput.textContent = err.message;
     } else {
         document.getElementById('transient_token').value = token;
         // No need to verify JWS
```

```
            document.getElementById('sa-form').submit();
        }
    });
});
</script>
```

### 3.11.3 FAQ

Frequently Asked Questions about using the Microform Integration 0.11 with the Secure Acceptance Checkout API.

**Can I supply both a secure storage (TMS) token and a transient token?**

Yes. A secure storage (TMS) token can be supplied in the payment_tokenfield which must be inside the capture context. The transient token is then supplied as an unsigned field (transient_token).

The transient token data will take precedence over the secure storage (TMS) token data.

**Can I use Microform to capture only the security code?**

Yes. You must ensure that the card_typeand card_expiry_dateare supplied via one of the following:

1. Through the payment token
2. Inside the capture context
3. As unsigned fields

**Can I override a transient token field, for example, the card_type field?**

Yes. Fields inside the capture context and unsigned fields both override transient token data.

**Can I use Microform to capture only the card number?**

Yes. You must ensure that the card_typeand card_expiry_dateare supplied either:

1. Inside the capture context
2. As unsigned fields

# 4 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.

## 4.1 Authorization with a Transient Token

This section provides the minimal set of information required to perform a successful authorization with a transient token.

**Endpoint**

**Production:** POST https://api.merchant-services.bankofamerica.com/pts/v2/payments

**Test:** POST https://apitest.merchant-services.bankofamerica.com/pts/v2/payments

### 4.1.1 Required Fields for an Authorization 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.locality**

**orderInformation.shipTo.postalCode**

**tokenInformation.transientTokenJwt**

### 4.1.2 REST Example: Authorization with a Transient Token

**Endpoint:** POST https://api.merchant-services.bankofamerica.com/pts/v2/payments


```
{
  "clientReferenceInformation":
    { "code": "TC50171_3"
  },
  "orderInformation": {
    "amountDetails": {
      "totalAmount": "102.21",
      "currency": "USD"
    },
    "billTo": {
      "firstName": "RTS",
      "lastName": "VDP",
      "address1": "201 S. Division St.",
      "locality": "Ann Arbor",
      "administrativeArea": "MI",
      "postalCode": "48104-2201",
      "country": "US",
      "district": "MI",
      "buildingNumber": "123",
      "email": "test@bankofamerica.com",
      "phoneNumber": "999999999"
    }
},
"tokenInformation": {

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

**Successful Response**


```
{
    "_links": {
        "authReversal": {
            "method":
            "POST",
            "href": "/pts/v2/payments/682622572509671870395 5/reversals"
        },
        "self": {
            "method": "GET",
            "href": "/pts/v2/payments/6826225725096718703955"
        },
        "capture": {
            "method": "POST",
            "href": "/pts/v2/payments/6826225725096718703955/captures"
        }
    },
    "clientReferenceInformation": {
        "code": "TC50171_3"
    },
    "id": "6826225725096718703955",
    "orderInformation": {
        "amountDetails": {
            "authorizedAmount": "102.21",
            "currency": "USD"
        }
    },
    "paymentAccountInformation": {
        "card": {
            "type": "001"
        }
    },
    "paymentInformation": {
        "tokenizedCard": {
            "type": "001"
        },
        "card": {
            "type": "001"
        },
        "customer": {
            "id": "AAE3DD3DED844001E05341588E0AD0D6"
```

```
        }
    },
    "pointOfSaleInformation": {
        "terminalId": "111111"
    },
    "processorInformation": {
        "approvalCode": "888888",
        "networkTransactionId": "123456789619999",
        "transactionId": "123456789619999",
        "responseCode": "100",
        "avs": {
            "code": "X",
            "codeRaw": "I1"
        }
    },
    "reconciliationId": "68450467YGMSJY18",
    "status": "AUTHORIZED",
    "submitTimeUtc": "2023-04-27T19:09:32Z"
    }
}
```


## 4.2 Authorization and Creating TMS Tokens with a Transient Token

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

**Endpoint**

**Production:** POST https://api.merchant-services.bankofamerica.com/pts/v2/payments

**Test:** POST https://apitest.merchant-services.bankofamerica.com/pts/v2/payments

### 4.2.1 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.locality**

**orderInformation.shipTo.postalCode**

**processingInformation.actionList**

Set this field to TOKEN_CREATE.

**processingInformation.actionTokenTypes**

Set to one of the following values:

- customer
- paymentInstrument
- shippingAddress

**tokenInformation.transientTokenJwt**

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

**Endpoint:** POST https://api.merchant-services.bankofamerica.com/pts/v2/payments

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

  "transientTokenJwt":"eyJraWQiOiIwMFN2SWFHSWZ5YXc4OTdyRGVHOWVGZE9ES2FDS2MxcSIsImFsZyI6Il
JTMjU2In0.eyJpc3MiOiJGbGV4LzAwIiwiZXhwIjoxNjE0NzkyNTQ0LCJ0eXBlIjoiYXBpLTAuMS4wIiwiaWF0Ijox
NjE0NzkxNjQ0LCJqdGkiOiIxRDBWMzFQMUtMRTNXN1NWSkJZVE04VUcxWE0yS0lPRUhJVldBSURPkhLNjJJSFQxUVE
1NjAzRkM3NjA2MDlDIn0.FrN1ytYcpQkn8TtafyFZnJ3dV3uu1XecDJ4TRIVZN-jpNbamcluAKVZ1zfdhbkrB6aNVW
ECSvjZrbEhDKCkHCG8IjChzl7Kg642RWteLkWz3oiofgQqFfzTuq41sDhlIqB-UatveU_2ukPxLYl87EX9ytpx4zCJ
```

```
Vmj6zGqdNP3q35Q5y59cuLQYxhRLk7WVx9BUgW85tl2OHaajEc25tS1FwH3jDOfjAC8mu2MEk-Ew0-ukZ70Ce7Zaq4
cibg_UTRx7_S2c4IUmRFS3wikS1Vm5bpvcKLr9k_8b9YnddIzp0p0JOCjXC_nuofQT7_x_-CQayx2czE0kD53HeNYC
5hQ"
  }
}
```

**Successful Response**


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

```
  },
  "processorInformation": {
      "approvalCode": "888888",
      "networkTransactionId": "123456789619999",
      "transactionId": "123456789619999",
      "responseCode": "100",
      "avs": {
```

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

}
```