Secure Acceptance Flexible Token Hosted Microform 0.4.0

Integration Guide

October 2018

CyberSource®
the power of payment

CyberSource Corporation HQ | P.O. Box 8999 | San Francisco, CA 94128-8999 | Phone: 800-530-9095

CyberSource Contact Information

For general information about our company, products, and services, go to http://www.cybersource.com.

For sales questions about any CyberSource Service, email sales@cybersource.com or call 650-432-7350 or 888-330-2300 (toll free in the United States).

For support information about any CyberSource Service, visit the Support Center: http://www.cybersource.com/support

Copyright

Restricted Rights Legends

Trademarks

Contents

Recent Revisions to This Document

Release	Changes
October 2018	■ Changed title of document from “Flex Microform 0.4.0 Implementation Guide” to “Secure Acceptance Flexible Token Hosted Microform 0.4.0 Integration Guide.” ■ Provided additional information for the Token Verification process description (see "Token Verification," page 51).
July 2018	Initial release for software version 0.4.0. For information about software changes, see Appendix C, "Changelog," on page 60.

About This Guide

Audience and Purpose

This document is written for application developers who want to integrate Flex Microform into their order management system.

Conventions

Notes

A *Note* contains helpful suggestions or references to material not contained in the document.

Text and Command Conventions

Convention	Usage
`Screen text`	■ XML elements. ■ Code examples and samples. ■ Text that you enter in an API environment; for example: Set the **davService_run** field to `true`.

Related Documents

The Support Center provides different versions of Flex Microform Implementation guides, which correspond to the SDK version you are using:

https://www.cybersource.com/developers/integration_methods/hosted_flex/

Refer to the Support Center for complete CyberSource technical documentation:

http://www.cybersource.com/support_center/support_documentation

Customer Support

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

http://www.cybersource.com/support

CHAPTER 1

Introduction to Flex Microform

Flex Microform makes it easier for you to make your order management system PCI DSS compliant without requiring any compromise in user experience. The capture of card numbers is fully outsourced to CyberSource, which can qualify you for SAQ A-based assessments. With Flex API, Flex Microform provides the most secure method for tokenizing card data. Sensitive data is encrypted on the customer's device before HTTPS transmission to CyberSource. This method mitigates any compromise of the HTTPS connection by a man-in-the-middle attack.

Features

- SAQ A compliant
- Seamlessly integrates with your current checkout experience
- Advanced protection against man-in-the-middle attacks

Refer to Chapter 2, "Getting Started," on page 12 for instructions on how to get started.

How It Works

Flex Microform is simple to integrate and allows you to focus on creating the best possible checkout experience for your customers.

The provided JavaScript library enables you to replace the sensitive card number input field with a secure iframe (hosted by CyberSource), that captures data on your behalf. This embedded field will look and feel just like any other input in your checkout process, allowing you to create a smooth, user-friendly experience.

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

PCI Compliance

The least burdensome level of PCI compliance is SAQ A. To achieve this compliance, sensitive payment data must be captured securely by a validated payment provider.

To meet this requirement, Flex Microform renders a secure iframe containing the card number input. This iframe is hosted by CyberSource and payment data is submitted directly to CyberSource through the secure Flex API, thus never touching your systems.

As user experience is an integral part of any modern e-commerce application, we provide all the tools required to blend seamlessly with your checkout experience. Various event hooks and styling options are available to facilitate, maximizing customizability without compromising PCI compliance.

Browser Support

- Internet Explorer 11 or later
- Edge 13 or later
- Firefox 21 or later
- Chrome 11 or later
- Safari 6.1 or later
- Opera 15 or later

CHAPTER 2

Getting Started

Flex Microform consists of two main components:

- A server-side component that requests limited-use public keys from the Flex API.
- The Flex Microform client-side js library, which seamlessly replaces the sensitive **PAN** field in your input form.

Server-Side Setup

Currently we provide server-side Flex SDKs in Java, .Net, and Node.js.

Our serve-side SDKs provide two functions:

1 Creating a transaction specific public key: the resulting public key is sent to the client side to initiate the Microform.

Note

When creating the public key for a Flex Microform implementation, you need to specify a targetOrigin. This is the protocol, URL, and if used, port number of the page that will host the Microform. Unless using http://localhost, the protocol must be https://. For example, if serving Microform on example.com, the targetOrigin is "https://example.com."

2 Verifying the token response: after Microform creates the token, it is passed back through the cardholder's browser. The response needs to be verified because a malicious user may tamper with the token response as it passes through the cardholder device. This function can be completed using the public key that is generated at the start of the process.

Server-side Flex SDKs are available in Java, .NET, and Node.js. We also have Java-based sample integrations available on GitHub.

If an SDK is unavailable for a specific language, or if you wish to write your own integration, see Appendix A, "Custom Server-Side Integration," on page 48.

Client-Side Setup

Add the Flex Microform .js script to your page by loading directly from CyberSource:

```
<script src="https://flex.cybersource.com/cybersource/assets/microform/
0.4.0/flex-microform.min.js"></script>
```

With Subresource Integrity:

```
<script src="https://flex.cybersource.com/cybersource/assets/microform/0.4.0/flex-
microform.min.js" integrity="sha256-MmydRQWHGwi8VnO3VhE4/x7VMCaNjMKs61k6W4akmlU=
sha384-hE5NjXSa1V7SgLtMeLWMLF2whEJrG28gcwrO+LTflRqB1xr3YNsYhplspYKhsK3E sha512-
MN008vpeT03LfJPVuWW62U73Vqq6u9wwISjcROKOk6JowEo+0s13XpNFWhwGfomq1On2M/n4KhkXcB97f/
LT3Q==" crossorigin="anonymous"></script>
```

CyberSource uses Semantic Versioning for the client-side .js library. You need to update the URL above to take advantage of new features and bug fixes as they are released. For more information, see "Versioning," next.

Versioning

Version Numbering

Flex Microform follows Semantic Versioning. Details of releases, features, bug fixes, etc. are provided in Appendix C, "Changelog," on page 60.

Upgrade Paths

In adherence with our commitment to Semantic Versioning, all efforts will be made to ensure that upgrade paths for `minor` and `patch` releases are backwards-compatible and require no code change.

During initial `0.x.x` releases, if this is not possible, we will provide clear upgrade steps describing any changes required.

Basic Integration

Within your HTML checkout, replace the credit card number `<input>` tag with a simple `<div>` container. This container will be used by Flex Microform to render iframe containing secured credit card input.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <title>Sample Checkout</title>
  </head>
  <body>
    <form action="/payment/complete" id="my-sample-form" method="post">
      <label id="cardNumber-label">Card Number</label>
      <div id="cardNumber-container"></div>

      <!--
       All the other fields comprising the form post to your server
      -->

      <button type="button" id="pay-button">Pay</button>
     </form>

    <script src="https://flex.cybersource.com/cybersource/assets/microform/0.4.0/
     flex-microform.min.js"></script>
    <script>
      var jwk = { /* jwk fetched on the server side */ };

      var form = document.querySelector('#my-sample-form');
      var payButton = document.querySelector('#pay-button');

      // SETUP MICROFORM
      FLEX.microform(
      {
           keyId: jwk.kid,
           keystore: jwk,
           container: '#cardNumber-container',
           label: '#cardNumber-label',
           placeholder: 'Your custom placeholder text',
           styles: {
               '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' },
```

```
          },
          encryptionType: 'rsaoaep',
      },

        function (setupError, microformInstance) {
          if (setupError) {
            // handle error
            return;
          }
              }
          // intercept the form submission and make a tokenize request instead
          payButton.addEventListener('click', function () {

          // Send in optional parameters from other parts of your payment form
            var options = {
              // cardExpirationMonth: /* ... */,
              // cardExpirationYear:  /* ... */,

              // cardType: /* ... */
            };

            microformInstance.createToken(options, function (err, response) {
              if (err) {
                // handle error
                return;
              console.log('Token generated: ');
              console.log(JSON.stringify(response));

              // At this point the token may be added to the form
              // as hidden fields and the submission continued
              form.submit();
            });
          });

        }
      );
    </script>
  </body>
</html>
```

In the example above, when the customer submits the form, Flex Microform securely collects and attempts to tokenize the card details. If tokenization succeeds, your success callback handler receives the token. This token can then be sent to your server and used with any supporting CyberSource services in place of the PAN.

Token Verification

You can send the token to your server where its integrity can be cryptographically verified using the public key to ensure that it has not been tampered with. The CyberSource server-side SDKs include functions to make it easy, or it can be easily implemented in the language of your choice (see "Token Verification," page 51).

Using the Token

Use the token instead of card data in CyberSource services.

Connection Method	Field in Which to Use Token
Simple Order API	recurringSubscriptionInfo_subscriptionID
SCMP API	subscription_id
Secure Acceptance	payment_token

CHAPTER 3

Styling

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

General Appearance

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

Explicitly Setting Container Height

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

Managed Classes

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

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

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

```
#cardNumber {
  background: #efefef;
  -webkit-transition: background 200ms;
  transition: background 200ms;
}

#cardNumber {
  background: white;
}

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

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

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

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

Note

`*-valid` and `*-invalid` will be applied only when the card detection feature is enabled (see "Card Detection," page 22).

Input Field Text

To style the text within the iframe, 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 CSS properties is supported see "Supported Properties," page 20).

```
// ...
FLEX.microform({
  // ...
   styles: {
    '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'
    },
    '.invalid': {
      'color': 'red'
    }
   }
  // ...
});
```

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`

CHAPTER 4

Events

Overview

You can subscribe to Flex Microform 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.

Event Name	Emitted When
`enabled`	Field is enabled.
`disabled`	Field is disabled.
`focus`	Field gains focus.
`blur`	Field loses focus.
`empty`	Field transitions from having data to being empty.
`notEmpty`	Field transitions from an empty state to having data.
`cardTypeChange`	Possible change of card type has been detected.
`validationChange`	Validity of the entered card number has changed.
`inputSubmitRequest`	Customer requests submission of the field by pressing they Return key or similar.
`tokenizeLoadStart`	A tokenize request is triggered.
`autocomplete`	Customer fills the credit card number using a browser or third-party extension. This event provides a hook onto the additional information provided during the autocomplete event.
`autocompleteChange`	Customer edits the credit card field after they filled it in with a previous autocomplete action.

Note

`cardTypeChange` and `validationChange` only trigger if the card detection feature is enabled (see "Card Detection," page 22).

Subscribing

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

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

```
var secCodeLbl = document.querySelector('label[for="secCode"]');

FLEX.microform(options, function (err, microformInstance) {

  // Respond to changes in the detected card type by updating the
  // security code label to that of the detected card type
  microformInstance.on('cardTypeChange', function (data) {
    secCodeLbl.innerHTML = (data && data.card) ? data.card.code.name :
'CVN';
  });

});
```

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

Card Detection

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

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

If Flex Microform 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.


Note

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


Note

You can opt out of the card detection feature by specifying `cardDetection: false` in the initial set-up call.

Autocomplete

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

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

```
{
   name: '_____',
   code: '_____',
   cardExpirationMonth: '__',
   cardExpirationYear: '____'
}
```

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:

```
microformInstance.on('autocomplete', function(data) {
    if (data.name) document.querySelector('#myName').value = data.name;
    if (data.code) document.querySelector('#myCvn').value = data.code;
    if (data.cardExpirationMonth)
    document.querySelector('#myMonth').value = data.cardExpirationMonth;
    if (data.cardExpirationYear) document.querySelector('#myYear').value
    = data.cardExpirationYear;
});
```


Note

You can opt out of auto complete support by specifying `autocomplete: false` in the initial set-up call.

CHAPTER 5

Security Recommendations

Content Security Policy

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 Flex Microform is:

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

CHAPTER 6

PCI DSS Guidance

Self Assessment Questionnaire

Any merchant accepting payments must comply with the PCI Data Security Standards (PCI DSS). Flex Microform handles the card number input and transmission from within iframe elements served from CyberSource controlled domains. This approach can qualify merchants for SAQ-A based assessments. Other fields, such as CVV and expiry data, are not considered sensitive data when not accompanied by the PAN.

Storing Returned Data

Responses from Flex Microform 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 CVV, note that it can be used for the initial authorization but not stored for subsequent authorizations.

CHAPTER 7

API Reference

Module: FLEX v0.4.0

For detailed setup instructions, see Chapter 2, "Getting Started," on page 12.

The following example displays a basic setup:

```
<script src="https://flex.cybersource.com/cybersource/assets/microform/
0.3.0/flex-microform.min.js"></script>
<script>
   window.FLEX.microform(...);
</script>
```

Methods

(static) microform(options, callback) → {void}

This method is the main setup function used to initialize Flex Microform. Upon successful setup, the callback receives a **MicroformInstance** (see "Class: MicroformInstance v0.4.0," page 32), which is used to interact with the service and build your integration.

Parameters

Name	Type	Description and Properties
`options`	Object	Flex Microform setup option properties: ■ `keyId`: • **Type**: String • **Description**: KeyId used for this integration. ■ `keystore`: • **Type**: Object • **Description**: Flex API key in JSON WebKey format. It must be generated server-side on your own infrastructure. ■ `container`: • **Type**: String • **Description**: A CSS selector to find the container where the iframe is inserted. ■ `label`: • **Type**: String • **Attributes**: <optional> • **Description**: A CSS selector to find a label element to mimic an HTML label corresponding to the **input** field. ■ `placeholder`: • **Type**: String • **Attributes**: <optional> • **Description**: Text to set on the input placeholder attribute. ■ `styles`: • **Type**: StylingOptions • **Attributes**: <optional> • **Description**: StylingOptions

(*continued on next page*)

Name	Type	Description and Properties
`options` (*continued*)		▪ `cardDetection`: • **Type**: Boolean • **Attributes**: <optional> • **Default**: true • **Description**: Turn off client-side card detection events and related styling, etc. ▪ `autocomplete`: • **Type**: Boolean • **Attributes**: <optional> • **Default**: true • **Description**: Turn off support for in-browser autocomplete tools. ▪ `timeout`: • **Type**: Number • **Attributes**: <optional> • **Default**: 120000 • **Description**: Time in milliseconds to wait for initial setup before erroring out. ▪ `encryptionType`: • **Type**: 'RsaOaep256' | 'RsaOaep' | 'None' • **Attributes**: <optional> • **Default**: 'RsaOaep256' • **Description**: Encryption type used.
`callback`	callback	Receives a MicroformInstance upon successful setup. See "callback(erropt, nullable, dataopt, nullable) → {void}," page 47, and "Class: MicroformInstance v0.4.0," page 32.

Returns

Type: void

(static) version() → {string}

The version of Flex Microform in use. For example, *0.4.0*

Returns

Type: string

Type Definitions

StylingOptions

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

The following CSS selectors are supported:

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

The following CSS properties are supported:

- 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

Note

Unsupported properties are not applied and raise a `console.warn()`.

Type

Object

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 element 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 using `MicroformInstance#disable`. See "disable()," page 36).
`valid`	Object	<optional>	Styles applied when Flex Microform detects that the input card number is valid. Relies on card detection being enabled using `module:FLEX.microform`. See "(static) microform(options, callback) → {void}," page 27.
`invalid`	Object	<optional>	Styles applied when Flex Microform detects that the input card number is invalid. Relies on card detection being enabled using `module:FLEX.microform`. See "(static) microform(options, callback) → {void}," page 27.

Example

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

Class: MicroformError v0.4.0

This class defines how error scenarios are presented by Microform, primarily as the first argument to callbacks (see "callback(erropt, nullable, dataopt, nullable) → {void}," page 47).

Members

(nullable) details :object

Additional error specific information. For example, this could be the payload of an underlying Flex API response or the details of a JavaScript error that is caught and handled.

Type

Object

message :string

A simple description of the error that has occurred.

Type

String

Class: MicroformInstance v0.4.0

An instance of this class is returned upon the creation of a Flex Microform integration using `module:FLEX.microform` [see "(static) microform(options, callback) → {void}," page 27]. This provides methods to interact with the Flex Microform iframe and to access the various integration hooks available.

Methods

clear()

Programmatically clears any entered value in the **card number** field.

Example

```
microformInstance.clear();
```

createToken(options, callback)

Requests a token using the card data that is entered on the form.

A successful token creation will receive a Flex API token response as its second callback parameter.

Parameters

Name	Type	Description
`options`	Object	Additional tokenization option properties: ■ `cardType` • **Type**: string • **Attributes:** <optional> • **Description:** three-digit CyberSource card type string. If set, it will override any automatic card detection. ■ `cardExpirationMonth` • **Type**: string • **Attributes:** <optional> • **Description:** two-digit month string. Must be padded with leading zeros if single digit. ■ `cardExpirationYear` • **Type**: string • **Attributes:** <optional> • **Description:** four-digit year string.
`callback`	callback	Receives a `createTokenResponse` upon successful token creation. See "callback(erropt, nullable, dataopt, nullable) → {void}," page 47 and "createTokenResponse," page 39).

Examples

The following minimal example omits all optional parameters:

```
microformInstance.createToken({}, function(err, response) {
  if (err) {
    console.error(err);
    return;
  }

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

The following example displays an override of the cardType parameter using a select element that is part of your checkout:

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

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

Handling Error Scenarios

```
microformInstance.createToken(options, function(err, response) {
  if (err) {
    // use the returned Flex API response
    switch (err.details.responseStatus.reason) {
      case 'VALIDATION_ERROR':
        // Occurs when any validation issues have been flagged and these
are explained by
        //   err.details.responseStatus.message
        // Issues with specific data fields are described by the details
property array
        err.details.responseStatus.details.forEach(function(detail) {
          console.log(detail.message);
        });
        console.error('Validation error');
        break;
      case 'DECRYPTION_ERROR':
        // The card was unable to be decrypted on the server side. This
was likely due to the
        // Microform 'encryptionType' parameter not matching the type
specified when initially
        // requesting the public key.
        console.error('Decryption error');
        break;
      case 'TOKENIZATION_ERROR':
        // The card was unable to be tokenized by the service which could
indicate that there
        // is some problem with the data. For further investigation use
the embedded icsReply
        // details to check the EBC transaction search.
        //   err.details.responseStatus._embedded.icsReply.requestId
        console.error('Tokenization error');
        break;
      case 'RESOURCE_QUOTA_EXCEEDED':
        // Occurs when the supplied public key has been used too
        // many times. Request a new public key and try again.
        console.error('Resource quota exceeded');
        break;
      case 'INTERNAL_ERROR':
        // Something went wrong on CyberSource's end
        console.error('Server-side error');
        break;
        default:
        console.error('Unknown error');
    }
  } else {
    console.log('Token created:', data.token);
  }
});
```

disable()

Disables the **card number** field from user input.

Example

```
microformInstance.disable();
```

enable()

Enables the **card number** field for user input.

Example

```
microformInstance.enable();
```

focus()

Programmatically sets the user focus to the **Microform** input field.

Example

```
microformInstance.focus();
```

off(name, the)

Unsubscribes a specific event handler from `MicroformInstance`.

Parameters

Name	Type	Description
`name`	String	The event to which you want to unsubscribe.
`the`	function	The handler to execute when you want to unsubscribe to an event.

Example

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

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

on(name, event)

Subscribe to events emitted by `MicroformInstance`. The following event types are supported:

- autocomplete
- autocompleteChange
- blur
- cardTypeChange
- disabled
- empty
- enabled
- focus
- inputSubmitRequest
- notEmpty
- tokenizeLoadStart
- validationChange

Some events may return data as the first parameter to the callback; otherwise, it will be `undefined`. See "Events," page 40, for information about the event details listed above.

Parameters

Name	Type	Description
`name`	String	The event to which you want to subscribe.
`event`	function	The handler to execute when event is triggered.

Example

```
microformInstance.on('notEmpty', function() {
  console.log('Input field is no longer empty!');
});
```

setPlaceholder(text)

Sets the placeholder text on the **card number** field.

Parameters

Name	Type	Description
`text`	String	Applied to the *placeholder* attribute of the **input** field.

Examples

The following example displays how to set the custom placeholder text to fit your user experience:

```
microformInstance.setPlaceholder('●●●● ●●●● ●●●● ●●●●');
```

The following example displays how to provide a custom prompt for your customer using placeholder text:

```
var customerName = 'Bob';
microformInstance.setPlaceholder('Please enter your card number, ' +
customerName);
```

teardown(callback*opt*)

Completely disposes of the `MicroformInstance`.

Parameters

Name	Type	Attributes	Description
`callback`	callback	<optional>	Executed after tear down is complete. See "callback(erropt, nullable, dataopt, nullable) → {void}," page 47).

Example

```
microformInstance.teardown(function() {
  console.log('Teardown complete!');
});
```

Type Definitions

createTokenResponse

Response format of a successful createToken call (see "createToken(options, callback)," page 33).

Type

Object

Properties

Name	Type	Description
`keyId`	String	Id of the public key used in creating this token.
`token`	String	The generated token. The token replaces card data and is used as the Subscription ID in the CyberSource Simple Order API or SCMP API.
`maskedPan`	String	The masked card number. Masking settings can be determined in the creation of the Keys resource, the default displaying the first 6 digits and the last 4 digits.
`cardType`	String	The three digit CyberSource card type code of the PAN used to create this token.
`timestamp`	Number	The UTC date and time in milliseconds at which the signature was generated.
`signedFields`	String	Comma separated list indicating which fields from the response make up the data used when verifying the response signature.
`signature`	String	Digital signature that should be verified server-side, using the public key. This ensures that values have not been tampered with whilst passing through the client.
`discoverableServices`	Object	Links to follow-on CyberSource RESTful services if available.
`_embedded`	Object	Transactional metadata related to token creation.

Events

autocomplete

This event is emitted when a customer uses a browser or third-party tool to perform an `autocomplete/autofill` event on the **input** field. Flex Microform attempts to capture additional information from the autocompletion and supply this information to the callback, if available.

Possible additional values returned are:

- `name`
- `code` (credit card security code)
- `cardExpirationMonth`
- `cardExpirationYear`

If a value is not provided in `autocomplete`, it is `undefined` in the callback data. Check for the value before using this event.

Examples

The following example displays the possible format of data supplied to callback:

```
{
  name: '_____',
  code: '_____',
  cardExpirationMonth: '__',
  cardExpirationYear: '____'
}
```

The following example displays how to update the rest of your checkout process following an `autocomplete` event:

```
microformInstance.on('autocomplete', function(data) {
  if (data.name) document.querySelector('#myName').value = data.name;
  if (data.code) document.querySelector('#myCvn').value = data.code;
  if (data.cardExpirationMonth)
  document.querySelector('#myMonth').value = data.cardExpirationMonth;
  if (data.cardExpirationYear) document.querySelector('#myYear')
  .value = data.cardExpirationYear;
});
```

autocompleteChange

This event is emitted when a customer makes an edit to the **input** field subsequent to having used a browser or third-party tool to perform an `autocomplete/autofill` event.

Example

```
microformInstance.on('autocompleteChange', function() {
  console.log('Input field has been manually edited');
});

// perform an autocomplete, then focus, then delete a couple of
characters to see your handler execute
```

blur

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

Example

```
microformInstance.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
```

cardTypeChange

This event is emitted when the customer **input** field is detected as potentially being a specific card type. This event enables building dynamic user experiences that updates information as the customer is enters a card number. For example, card branding information and client-side validation rules update in real time.

Examples

The following example displays the format of data supplied to callback:

```
{
  card: [
    {
      name: "visa",
      brandedName: "Visa",
      cybsCardType: "001",
      spaces: [ 4, 8, 12 ],
      lengths: [ 13, 14, 15, 16, 17, 18, 19 ],
      code: { name: "CVV", length: 3 },
      luhn: true
    },
    ...
  ]
}
```

The following example displays how to update the UI and validation options using the card type change data:

```
var cardImage = document.querySelector('img');
var cardTypeLabel = document.querySelector('label[for=cardType]');
var cardSecurityCode = document.querySelector('input[name=csc]');
var cardSecurityCodeLabel = document.querySelector('label[for=csc]');

microformInstance.on('cardTypeChange', function(data) {
  if (data.card.length === 1) {
    // update UI with the detected card info
    cardImage.src = '/appropriate_card_art.png';
    cardTypeLabel = data.card[0].brandedName;
    cardSecurityCode.maxLength = data.card[0].code.length;
    cardSecurityCodeLabel.innerText = data.card[0].code.name;
  } else {
    // otherwise show a generic UI
    cardImage.src = '/unknown.png';
    cardTypeLabel = 'No card detected';
    cardSecurityCode.maxLength = 4;
    cardSecurityCodeLabel.innerText = '';
  }
});
```

The following example displays how to filter a card type select element in another part of your checkout process:

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

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

  // if any cards have been detected, disable the options not in that
list
  // otherwise enable all the options
  cardTypeOptions.forEach(function (o) {
    o.disabled = (detectedCardTypes.length > 0) &&
!detectedCardTypes.includes(o.value);
  });
});
```

disabled

This event is emitted following the disabling of the **input** field using `MicroformInstance#disable`. See "disable()," page 36.

Example

```
microformInstance.on('disabled', function() {
  console.log('Field has been disabled');
});

// disable to see your supplied handler execute
microformInstance.disable();
```

empty

This event is emitted when the **input** field has transitioned from a state of including data to not including data. This condition could occur when a customer deletes the field contents or the field is programmatically cleared using `MicroformInstance#clear`. See "clear()," page 32.

Example

```
microformInstance.on('empty', function() {
  console.log('Field is now empty');
});

// enter a card number into the field
// either delete card number fully or trigger a clear to see your
supplied handler execute
microformInstance.clear();
```

enabled

This event is emitted after the **input** field is enabled using `MicroformInstance#enable`. See "enable()," page 36

Example

```
microformInstance.on('enabled', function() {
  console.log('Field has been enabled');
});

// disable & re-enable the field to see your supplied handler execute
microformInstance.disable();
microformInstance.enable();
```

focus

This event is emitted when the **input** field receives focus.

Example

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

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

inputSubmitRequest

This event is emitted when a customer requests the submission of the **input** field by pressing the **Return** key, or similar command. By subscribing to this event, you are able to transparently replicate the familiar UX of pressing **Enter** to submit a form.

This event is implemented as follows:

1. The `inputSubmitRequest` handler will call the `MicroformInstance#createToken`. See createToken(options, callback), page 33.
2. The result is added to a hidden input on your checkout.
3. A trigger submission of the form including the newly-created token is available for you to use server-side.

Example

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

microformInstance.on('inputSubmitRequest', function() {
  microformInstance.createToken({}, function(response) {
    hiddenInput.value = response.token;
    form.submit();
  });
});
```

notEmpty

This event is emitted when the **input** field is transitioned from a state of being empty to including data.

Example

```
microformInstance.on('notEmpty', function() {
  console.log('Field is not empty');
});

// enter a card number into the field to see your supplied handler
execute
```

tokenizeLoadStart

This event is emitted when the `MicroformInstance#createToken` is called. See "createToken(options, callback)," page 33. Using this event enables you to build a user interface that lets the customer know a request is in progress.

Example

```
microformInstance.on('tokenizeLoadStart', function() {
  console.log('Show a loading indicator');
});

// later on...
microformInstance.createToken({}, function(response) {
  console.log('Remove the loading indicator');
});
```

validationChange

This event is emitted when Flex Microform detects a change in the validation state of the entered card number. The callback receives an object including two values:

- `valid`: indicates that based on the card detected, this is a valid card.
- `couldBeValid`: indicates that while the card may not be fully valid, there is possible additional user input that could make the card valid. For example, the user is still entering their card information.

Examples

The following example displays the format of the data provided to the callback:

```
{
  isValid: false,
  couldBeValid: true
}
```

The following example displays how to update validation styles on your **form** element based on card validation events:

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

microformInstance.on('validationChange', function(data) {
  myForm.classList.toggle('cardIsValidStyle', data.isValid);
  myForm.classList.toggle('cardCouldBeValidStyle', data.couldBeValid);
});
```

Global v0.4.0

Type Definitions

callback(erropt, nullable, dataopt, nullable) → {void}

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

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

Parameters

Name	Type	Attributes	Description
`err`	MicroformError (see "Class: MicroformError v0.4.0," page 31)	<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);
});
```

APPENDIX A

Custom Server-Side Integration

Requesting a Flex Key Directly from CyberSource

This section demonstrates how to authenticate a `POST` request to `https://apitest.cybersource.com/flex/v1/keys`. The authentication schema for calls to this resource closely follows Signing HTTP Messages RFC.

For demonstration purposes, we provide Java code samples that should be easy to adapt to any platform.

Assumptions

- Your MID is `merchant`
- You created your REST API Keys through the CyberSource Business Center, not the Visa Developer Center
- Your keyId (serial number) is `01dbbc88-0736-4d31-94ed-7b84579731b2`
- Your shared secret is `SXQgaXMgc2hhcmVkIHNlY3JldA==` (Base64 encoded)
- In the following example, the system time is `Mon, 01 Jan 2018 00:00:00 GMT`

To obtain the Flex one-time-use public RSA key, the body of the request should appear as follows:

```
{
  "encryptionType": "RsaOaep",
  "targetOrigin": "https://example.com"
}
```

First, create the `Digest` header:

```
String body = "{\n  \"encryptionType\": \"RsaOaep\",\n
\"targetOrigin\": \"https://example.com\"\n}";
// NOTE: this should exactly match what you send in the request body.
// In this case we have included new lines and indentation.

MessageDigest digester = MessageDigest.getInstance("SHA-256");
byte[] digest = digester.digest(body.getBytes(StandardCharsets.UTF_8));
String value = String.format("SHA-256=%s",
Base64.getEncoder().encodeToString(digest));
System.out.println("digest: " + value);

// outputs
// digest: SHA-256=YljtibTei+du4xVIDxMr3HBsyLAEDuiYaag9TcU9jHA=
```

Second, prepare a list of HTTP headers that will be signed. The header list must have a predictable iteration order:

```
Calendar calendar = Calendar.getInstance();
SimpleDateFormat dateFormat = new SimpleDateFormat("EEE, dd MMM yyyy
HH:mm:ss z", Locale.US);
dateFormat.setTimeZone(TimeZone.getTimeZone("GMT"));
String date = dateFormat.format(calendar.getTime());

Map<String, String> signedHeaders = new LinkedHashMap<>();
signedHeaders.put("host", "apitest.cybersource.com");
signedHeaders.put("date", date);
signedHeaders.put("(request-target)", "post /flex/v1/keys");
signedHeaders.put("digest", "SHA-256=fRDzptXm4RRRD3pC/
eoIBoHShRzjRAf7Xkj18upMtI8=");
signedHeaders.put("v-c-merchant-id", "merchant");
System.out.println(signedHeaders);

// outputs
// {host=apitest.cybersource.com, date=Wed, 24 May 2017 15:30:48 GMT,
(request-target)=post /flex/v1/keys, digest=SHA-
256=YljtibTei+du4xVIDxMr3HBsyLAEDuiYaag9TcU9jHA=
```

Sign the headers and generated signature header by:

- Creating a comma-separated list of signed headers—used to construct a signature header
- Creating a string that comprises `key: value` pairs separated by a new line character—it will be digested by HMAC
- Calculating the HMAC value

- Assembling the signature header from keyId, signed header names, and calculated HMAC

```
StringBuilder signatureString = new StringBuilder();
StringBuilder headersString = new StringBuilder();
for (Map.Entry<String, String> e : signedHeaders.entrySet()) {
  signatureString.append('\n').append(e.getKey())
  .append(": ").append(e.getValue());
  headersString.append(' ').append(e.getKey());
}
signatureString.delete(0, 1);
headersString.delete(0, 1);


Mac sha256HMAC = Mac.getInstance("HmacSHA256");
SecretKeySpec secretKey = new
SecretKeySpec(Base64.getDecoder().decode("SXQgaXMgc2hhcmVkIHNlY3JldA=="
), "HmacSHA256");
sha256HMAC.init(secretKey);

StringBuilder signature = new StringBuilder();
sha256HMAC.update(signatureString.toString().getBytes(StandardCharsets.
UTF_8));
byte[] hashBytes = sha256HMAC.doFinal();
signature.append("keyid=\"").append("01dbbc88-0736-4d31-94ed-
7b84579731b2").append("\", ")
        .append("algorithm=\"HmacSHA256\", ")
        .append("headers=\"").append(headersString).append("\", ")
       .append("signature=\"").append(Base64.getEncoder()
       .encodeToString(hashBytes)).append('\"');


System.out.println("signature: " + signature);

// outputs
// signature: keyid="01dbbc88-0736-4d31-94ed-7b84579731b2",
algorithm="HmacSHA256", headers="host date (request-target) digest v-c-
merchant-id", digest:
signature="u1bAYVoTneRlIckQAJtFSMxoav7hw3Uegxoc4b/+APM="
```

Last, append the calculated signature as a header in your request along with the content type. The complete HTTP headers should appear similar to the following example:

```
POST /flex/v1/keys
host: apitest.cybersource.com
date: Wed, 24 May 2017 15:30:48 GMT
digest: SHA-256=YljtibTei+du4xVIDxMr3HBsyLAEDuiYaag9TcU9jHA=
signature: keyid="01dbbc88-0736-4d31-94ed-7b84579731b2",
algorithm="HmacSHA256", headers="host date (request-target) digest v-c-
merchant-id", signature="u1bAYVoTneRlIckQAJtFSMxoav7hw3Uegxoc4b/+APM="
content-type: application/json
v-c-merchant-id: merchant
```

Note

`(request-target)` is not set as a header in the actual HTTP request.

Token Verification

After a successful token creation, the response should be sent to your server. The response is signed by CyberSource, to highlight any tampering attempts as it passes from Microform through the browser to your servers. You should verify the contents against the digital signature, using the public key generated at the beginning of the transaction. If the signature does not match, the results may have been tampered with and you should reject the token. Example Java code for this follows:

```
// example token structure
{
  "keyId": "00GaWD0Dr3zodGup0BhIuyLs8i0M9wFr",
  "token": "9905000003693378",
  "maskedPan": "424242XXXXXX4242",
  "cardType": "001",
  "timestamp": 1506437341961,
  "signedFields": "token,cardType,maskedPan,timestamp",
  "signature": "dFoMulkzIdkjkSgh53HCuxOsS/47wuhbKcnywLcdujieeDGBY/
   vtj3NrAFgUxutbctAF2gSXqYNAPhx5zc0bPU6ENf4/
   IUdYlG9jyeOQYqbD8EJxXkqpBWSUe0udvz8W9f+GxJ2FURiTRqnAg0aJE6cmBF29k6WL
   Aq3PG1bRU2UBCrbUR+/rFB/OztWDblR/t8iMBVINxDko2hOav6pLY0uubwWd8WEi/
   bxJYBefwB8VY9ECA1XeIvVafbpPPQDniEDv4f1Km39AETump7jnXqfG7c0i9zJHh6AcU
   tdFDmS+1cEpwzB0P7wOPGdFYpLoeQAiLC5+BjQgL/awHQWfZQ==",
  "discoverableServices": { /* */ },
  "_embedded": { /* */ }
}
```

The following code example assumes that the token has been parsed into a `Map` and that a `PublicKey` is initialized:

```
public boolean verify(PublicKey publicKey, Map tokenFields) throws
Exception {
  String signedFields = (String) tokenFields.get("signedFields");

  // Use the signed fields list to recreate the data to be verified
  StringBuilder sb = new StringBuilder();
  for (String fieldValue : signedFields.split(",")) {
    sb.append(',');
    sb.append(tokenFields.get("" + fieldValue));
  }
  String dataToVerify = sb.substring(1);
  String signatureString = (String) tokenFields.get("signature");

  // Verify the signature using the public key
  Signature signer = Signature.getInstance("SHA512withRSA");
  signer.initVerify(publicKey);
  signer.update(dataToVerify.getBytes());
  return signer.verify(Base64.decode(signatureString));
}
```

APPENDIX B

License

This Software Development Kit (“SDK”) License Agreement (“Agreement”) is between you (both the individual downloading the SDK and any legal entity on behalf of which such individual is acting) (“You” or “Your”) and CyberSource Corporation (“CyberSource”).

IT IS IMPORTANT THAT YOU READ CAREFULLY AND UNDERSTAND THIS AGREEMENT. BY CLICKING THE “I ACCEPT” BUTTON OR AN EQUIVALENT INDICATOR OR BY DOWNLOADING, INSTALLING OR USING THE SDK OR THE DOCUMENTATION, YOU AGREE TO BE BOUND BY THIS AGREEMENT. IF YOU DO NOT AGREE TO BE BOUND BY THIS AGREEMENT, YOU WILL NOT BE PERMITTED TO (AND YOU WILL HAVE NO RIGHT TO) DOWNLOAD, INSTALL OR USE THE SDK OR THE DOCUMENTATION.

1. Definitions

1.1. “Application(s)” means software programs that You develop to operate with the Gateway using components of the Software.

1.2. “Documentation” means the materials made available to You in connection with the Software by or on behalf of CyberSource pursuant to this Agreement.

1.3. “Gateway” means any electronic payment platform maintained and operated by CyberSource and any of its affiliates.

1.4. “Software” means all of the software included in the software development kit made available to You by or on behalf of CyberSource pursuant to this Agreement, including but not limited to sample source code, code snippets, software tools, code libraries, sample applications, Documentation and any upgrades, modified versions, updates, and/or additions thereto, if any, made available to You by or on behalf of CyberSource pursuant to this Agreement.

2. Grant of License; Restrictions

2.1 Limited License. Subject to and conditioned upon Your compliance with the terms of this Agreement, CyberSource hereby grants to You a limited, revocable, non-exclusive, non-transferable, royalty-free license during the term of this Agreement to: (a) in any country worldwide, use, reproduce, modify, and create derivative works of the components of the Software solely for the purpose of developing, testing and manufacturing Applications; (b) distribute, sell or otherwise provide Your Applications that include components of the Software to Your end users in all countries worldwide; and © use the Documentation in connection with the foregoing activities. The license to distribute Applications that include components of the Software as set forth in subsection (b) above includes the right to grant sublicenses to Your end users to use such components of the Software as incorporated into such Applications, subject to the limitations and restrictions set forth in this Agreement.

2.2 Restrictions. You shall not (and shall have no right to): (a) make or distribute copies of the Software or the Documentation, in whole or in part, except as expressly permitted pursuant to Section 2.1; (b) alter or remove any copyright, trademark, trade name or other proprietary notices, legends, symbols or labels appearing on or in the Software or Documentation; © sublicense (or purport to sublicense) the Software or the Documentation, in whole or in part, to any third party except as expressly permitted pursuant to Section 2.1; (d) engage in any activity with the Software, including the development or distribution of an Application, that interferes with, disrupts, damages, or accesses in an unauthorized manner the Gateway or platform, servers, or systems of CyberSource, any of its affiliates, or any third party; (e) make any statements that Your Application is “certified” or otherwise endorsed, or that its performance is guaranteed, by CyberSource or any of its affiliates; or (f) otherwise use or exploit the Software or the Documentation for any purpose other than to develop and distribute Applications as expressly permitted by this Agreement.

2.3 Ownership. You shall retain ownership of Your Applications developed in accordance with this Agreement, subject to CyberSource’s ownership of the Software and Documentation (including CyberSource’s ownership of any portion of the Software or Documentation incorporated in Your Applications). You acknowledge and agree that all right, title and interest in and to the Software and Documentation shall, at all times, be and remain the exclusive property of CyberSource and that You do not have or acquire any rights, express or implied, in the Software or Documentation except those rights expressly granted under this Agreement.

2.4 No Support. CyberSource has no obligation to provide support, maintenance, upgrades, modifications or new releases of the Software.

2.5 Open Source Software. You hereby acknowledge that the Software may contain software that is distributed under “open source” license terms (“Open Source Software”). You shall review the Documentation in order to determine which portions of the Software are Open Source Software and are licensed under such Open Source Software license terms. To the extent any such license requires that CyberSource provide You any rights with respect to such Open Source Software that are inconsistent with the limited rights

granted to You in this Agreement, then such rights in the applicable Open Source Software license shall take precedence over the rights and restrictions granted in this Agreement, but solely with respect to such Open Source Software. You acknowledge that the Open Source Software license is solely between You and the applicable licensor of the Open Source Software and that Your use, reproduction and distribution of Open Source Software shall be in compliance with applicable Open Source Software license. You understand and agree that CyberSource is not liable for any loss or damage that You may experience as a result of Your use of Open Source Software and that You will look solely to the licensor of the Open Source Software in the event of any such loss or damage.

2.6 License to CyberSource. In the event You choose to submit any suggestions, feedback or other information or materials related to the Software or Documentation or Your use thereof (collectively, “Feedback”) to CyberSource, You hereby grant to CyberSource a worldwide, non-exclusive, royalty-free, transferable, sublicensable, perpetual and irrevocable license to use and otherwise exploit such Feedback in connection with the Software, Documentation, and other products and services.

2.7 Use.

(a) You represent, warrant and agree to use the Software and write Applications only for purposes permitted by (i) this Agreement; (ii) applicable law and regulation, including, without limitation, the Payment Card Industry Data Security Standard (PCI DSS); and (iii) generally accepted practices or guidelines in the relevant jurisdictions.

(b) You represent, warrant and agree that if You use the Software to develop Applications for general public end users, that You will protect the privacy and legal rights of those users. If the Application receives or stores personal or sensitive information provided by end users, it must do so securely and in compliance with all applicable laws and regulations, including card association regulations. If the Application receives CyberSource account information, the Application may only use that information to access the end user’s CyberSource account.

© You represent, warrant and agree that You are solely responsible for (and that neither CyberSource nor its affiliates have any responsibility to You or to any third party for): (i) any data, content, or resources that You obtain, transmit or display through the Application; and (ii) any breach of Your obligations under this Agreement, any applicable third party license, or any applicable law or regulation, and for the consequences of any such breach.

3. Warranty Disclaimer; Limitation of Liability

3.1 Disclaimer. THE SOFTWARE AND THE DOCUMENTATION ARE PROVIDED ON AN “AS IS” AND “AS AVAILABLE” BASIS WITH NO WARRANTY. YOU AGREE THAT YOUR USE OF THE SOFTWARE AND THE DOCUMENTATION IS AT YOUR SOLE RISK AND YOU ARE SOLELY RESPONSIBLE FOR ANY DAMAGE TO YOUR COMPUTER SYSTEM OR OTHER DEVICE OR LOSS OF DATA THAT RESULTS FROM SUCH USE. TO THE FULLEST EXTENT PERMISSIBLE UNDER APPLICABLE LAW, CYBERSOURCE AND ITS AFFILIATES EXPRESSLY DISCLAIM ALL WARRANTIES OF ANY KIND, EXPRESS OR IMPLIED, WITH RESPECT TO THE SOFTWARE AND THE DOCUMENTATION, INCLUDING ALL WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, ACCURACY, TITLE AND NON-INFRINGEMENT, AND ANY WARRANTIES THAT MAY ARISE OUT OF COURSE OF PERFORMANCE, COURSE OF DEALING OR USAGE OF TRADE. NEITHER CYBERSOURCE NOR ITS AFFILIATES WARRANT THAT THE FUNCTIONS OR INFORMATION CONTAINED IN THE SOFTWARE OR THE DOCUMENTATION WILL MEET ANY REQUIREMENTS OR NEEDS YOU MAY HAVE, OR THAT THE SOFTWARE OR DOCUMENTATION WILL OPERATE ERROR FREE, OR THAT THE SOFTWARE OR DOCUMENTATION IS COMPATIBLE WITH ANY PARTICULAR OPERATING SYSTEM.

3.2 Limitation of Liability. IN NO EVENT SHALL CYBERSOURCE AND ITS AFFILIATES BE LIABLE FOR ANY INDIRECT, INCIDENTAL, SPECIAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, OR DAMAGES FOR LOSS OF PROFITS, REVENUE, BUSINESS, SAVINGS, DATA, USE OR COST OF SUBSTITUTE PROCUREMENT, INCURRED BY YOU OR ANY THIRD PARTY, WHETHER IN AN ACTION IN CONTRACT OR TORT, EVEN IF CYBERSOURCE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES OR IF SUCH DAMAGES ARE FORESEEABLE. IN NO EVENT SHALL THE ENTIRE LIABILITY OF CYBERSOURCE AND AFFILIATES ARISING FROM OR RELATING TO THIS AGREEMENT OR THE SUBJECT MATTER HEREOF EXCEED ONE HUNDRED U.S. DOLLARS ($100). THE PARTIES ACKNOWLEDGE THAT THE LIMITATIONS OF LIABILITY IN THIS SECTION 3.2 AND IN THE OTHER PROVISIONS OF THIS AGREEMENT AND THE ALLOCATION OF RISK HEREIN ARE AN ESSENTIAL ELEMENT OF THE BARGAIN BETWEEN THE PARTIES, WITHOUT WHICH CYBERSOURCE WOULD NOT HAVE ENTERED INTO THIS AGREEMENT.

4. Indemnification

You shall indemnify, hold harmless and, at CyberSource's request, defend CyberSource and its affiliates and their officers, directors, employees, and agents from and against any claim, suit or proceeding, and any associated liabilities, costs, damages and expenses, including reasonable attorneys' fees, that arise out of relate to: (i) Your Applications or the use or distribution thereof and Your use or distribution of the Software or the Documentation (or any portion thereof including Open Source Software), including, but not limited to, any allegation that any such Application or any such use or distribution infringes, misappropriates or otherwise violates any intellectual property (including, without limitation, copyright, patent, and trademark), privacy, publicity or other rights of any third party, or has caused the death or injury of any person or damage to any property; (ii) Your alleged or actual breach of this Agreement; (iii) the alleged or actual breach of this Agreement by any party to whom you have provided Your Applications, the Software or the Documentation or (iii) Your alleged or actual violation of or non-compliance with any applicable laws, legislation, policies, rules, regulations or governmental requirements (including, without limitation, any laws, legislation, policies, rules, regulations or governmental requirements related to privacy and data collection).

5. Termination

This Agreement and the licenses granted to you herein are effective until terminated. CyberSource may terminate this Agreement and the licenses granted to You at any time. Upon termination of this Agreement, You shall cease all use of the Software and the Documentation, return to CyberSource or destroy all copies of the Software and Documentation and related materials in Your possession, and so certify to CyberSource. Except for the license to You granted herein, the terms of this Agreement shall survive termination.

6. Confidential Information

a. You hereby agree (i) to hold CyberSource's Confidential Information in strict confidence and to take reasonable precautions to protect such Confidential Information (including, without limitation, all precautions You employ with respect to Your own confidential materials), (ii) not to divulge any such Confidential Information to any third person; (iii) not to make any use whatsoever at any time of such Confidential Information except as strictly licensed hereunder, (iv) not to remove or export from the United States or re-export any such Confidential Information or any direct product thereof, except in compliance with, and with all licenses and approvals required under applicable U.S. and foreign export laws and regulations, including, without limitation, those of the U.S. Department of Commerce.

b. "Confidential Information" shall mean any data or information, oral or written, treated as confidential that relates to CyberSource's past, present, or future research, development or business activities, including without limitation any unannounced products and services, any information relating to services, developments, inventions, processes, plans, financial information, customer data, revenue, transaction volume, forecasts, projections, application programming interfaces, Software and Documentation.

7. General Terms

7.1 Law. This Agreement and all matters arising out of or relating to this Agreement shall be governed by the internal laws of the State of California without giving effect to any choice of law rule. This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sales of Goods, the application of which is expressly excluded. In the event of any controversy, claim or dispute between the parties arising out of or relating to this Agreement, such controversy, claim or dispute shall be resolved in the state or federal courts in Santa Clara County, California, and the parties hereby irrevocably consent to the jurisdiction and venue of such courts.

7.2 Logo License. CyberSource hereby grants to You the right to use, reproduce, publish, perform and display CyberSource logo solely in accordance with the current CyberSource brand guidelines.

7.3 Severability and Waiver. If any provision of this Agreement is held to be illegal, invalid or otherwise unenforceable, such provision shall be enforced to the extent possible consistent with the stated intention of the parties, or, if incapable of such enforcement, shall be deemed to be severed and deleted from this Agreement, while the remainder of this Agreement shall continue in full force and effect. The waiver by either party of any default or breach of this Agreement shall not constitute a waiver of any other or subsequent default or breach.

7.4 No Assignment. You may not assign, sell, transfer, delegate or otherwise dispose of, whether voluntarily or involuntarily, by operation of law or otherwise, this Agreement or any rights or obligations under this Agreement without the prior written consent of CyberSource, which may be withheld in CyberSource's sole discretion. Any purported

assignment, transfer or delegation by You shall be null and void. Subject to the foregoing, this Agreement shall be binding upon and shall inure to the benefit of the parties and their respective successors and assigns.

7.5 Government Rights. If You (or any person or entity to whom you provide the Software or Documentation) are an agency or instrumentality of the United States Government, the Software and Documentation are “commercial computer software” and “commercial computer software documentation,” and pursuant to FAR 12.212 or DFARS 227.7202, and their successors, as applicable, use, reproduction and disclosure of the Software and Documentation are governed by the terms of this Agreement.

7.6 Export Administration. You shall comply fully with all relevant export laws and regulations of the United States, including, without limitation, the U.S. Export Administration Regulations (collectively “Export Controls”). Without limiting the generality of the foregoing, You shall not, and You shall require Your representatives not to, export, direct or transfer the Software or the Documentation, or any direct product thereof, to any destination, person or entity restricted or prohibited by the Export Controls.

7.7 Privacy. In order to continually innovate and improve the Software, Licensee understands and agrees that CyberSource may collect certain usage statistics including but not limited to a unique identifier, associated IP address, version number of software, and information on which tools and/or services in the Software are being used and how they are being used.

7.8 Headings. The headings to the Sections and Subsections of this Agreement are included merely for convenience of reference and shall not affect the meaning of the language included therein.

7.9 Entire Agreement; Amendments. This Agreement constitutes the entire agreement between the parties and supersedes all prior or contemporaneous agreements or representations, written or oral, concerning the subject matter of this Agreement. CyberSource may make changes to this Agreement, Software or Documentation in its sole discretion. When these changes are made, CyberSource will make a new version of the Agreement, Software or Documentation available on the website where the Software is available. This Agreement may not be modified or amended by You except in a writing signed by a duly authorized representative of each party. You acknowledge and agree that CyberSource has not made any representations, warranties or agreements of any kind, except as expressly set forth herein.

BY CLICKING “I ACCEPT,” “I AGREE” OR AN EQUIVALENT INDICATOR OR BY DOWNLOADING, INSTALLING OR USING THE SOFTWARE OR THE DOCUMENTATION, YOU ACKNOWLEDGE AND AGREE THAT (1) YOU HAVE READ AND REVIEWED THIS AGREEMENT IN ITS ENTIRETY, (2) YOU AGREE TO BE BOUND BY THIS AGREEMENT, (3) YOU HAVE THE POWER, AUTHORITY AND LEGAL RIGHT TO ENTER INTO THIS AGREEMENT ON BEHALF OF YOU AND, (4) THIS AGREEMENT CONSTITUTES BINDING AND ENFORCEABLE OBLIGATIONS OF YOU.

APPENDIX C

Changelog

All notable changes for Flex Microform are documented in this section. The format of the information in this section is based on Keep a Changelog recommendations. Flex Microform also adheres to Semantic Versioning.

Version 0.4.0

The following updates are incorporated in Flex Microform 0.4.0:

Added

You can now programmatically set user focus to the Microform input field using `microformInstance.focus()`.

Corrected

- Corrected keyboard navigation when using **TAB or SHIFT + TAB** in Internet Explorer and Firefox.
- Corrected keyboard navigation to correctly skip hidden fields within the iframe.
- Prevent browser from attempting to focus on input field if it has previously been disabled using `microformInstance.disable()`.