Payment iFrame User Guide

Bluefin Payment Systems
8200 Roberts Drive, Suite 150
Atlanta, GA 30350 800-675-6573 www.bluefin.com

Contents

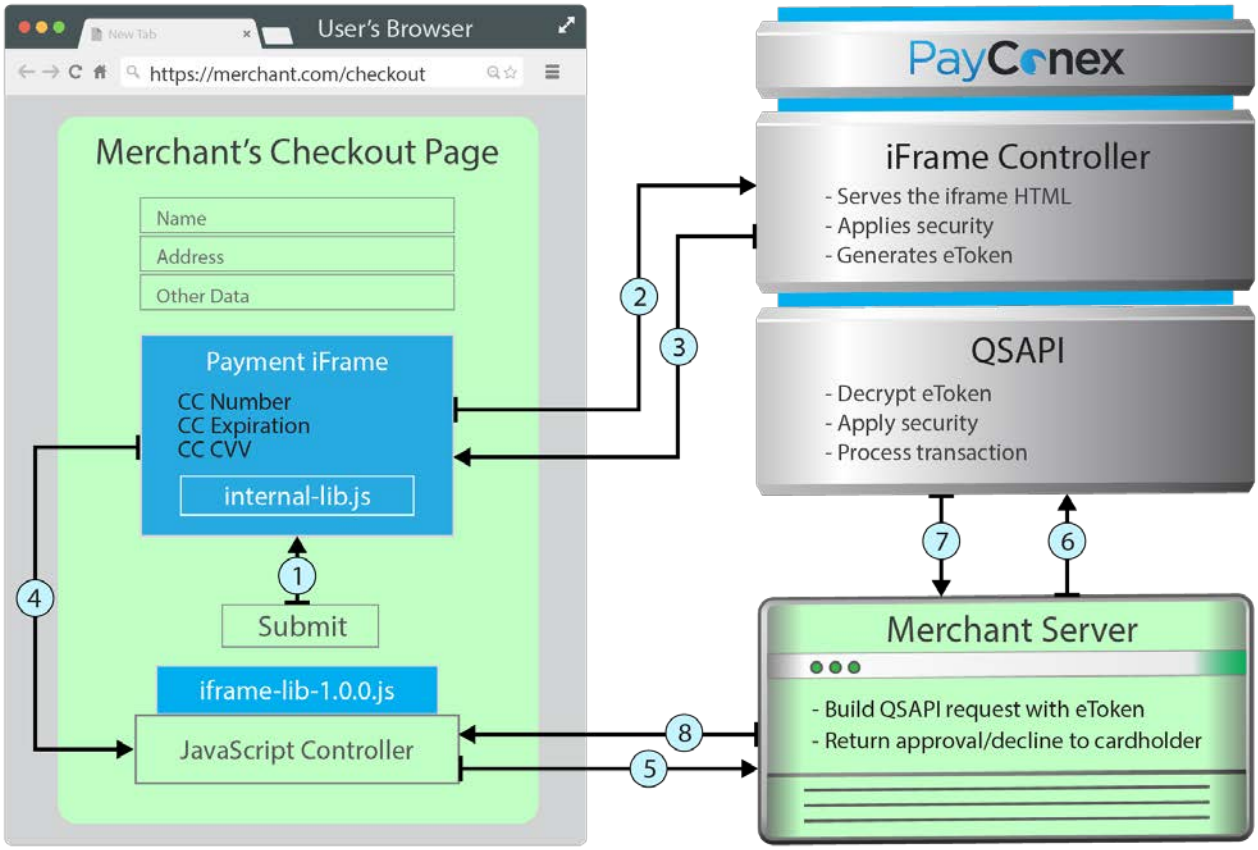
Introduction

The purpose of the Payment iFrame is to allow a merchant to embed an HTML document within their checkout page that will encrypt user-entered payment data and return the encrypted data in an "eToken" format. The eToken can be used for processing payments through Bluefin's API. The Payment iFrame and the accompanying client library allow the merchant to perform card data encryption in a PCI compliant manner while also affording them greater programmatic control over the look and feel of the embedded input form.

PCI Scope

The Payment iFrame reduces the PCI scope for the merchant by enabling them to outsource the capture of sensitive credit and debit card data to Bluefin. We control the data capture, send it to our server for encryption and release an encrypted token (eToken) to the merchant which they can use with Bluefin's API for further payment processing (SALE, AUTH, STORE). With our solution, our merchants never handle card data directly, putting their environment outside of PCI scope.

Payment iFrame Flow



This diagram outlines a typical flow of a Payment iFrame transaction.

PCI Scope

In the figure above, which outlines the typical flow of a Payment iFrame transaction, the different

environments impacted by PCI scope are represented: a) The blue areas and the PayConex iFrame Controller and QSAPI deal directly with the card data and are exclusively controlled by Bluefin, b) the green area handles eTokens and other card data and is controlled by the merchant and c) the users browser is the end users' environment which is out of the control of both the merchant and Bluefin.

Typical Flow

A typical transaction will include the following steps:

1. Cardholder enters card data on the Payment iFrame and pushes the 'Submit' button on an eCommerce site; the button calls the `encrypt` function on the iframe's' JavaScript API. The JavaScript library then passes a command to the iframe to encrypt the card data.
2. The iframe performs client side validation; if the data is valid, then it performs an AJAX call to the Bluefin server with the card data.
3. The Bluefin server then performs additional security checks; if the checks pass, it then encrypts the card data and generates an eToken. Finally, it returns the eToken to the iframe page.
4. The iframe passes the eToken to the parent page. On the parent page, the merchant's JavaScript Controller receives the eToken through a callback function (attached to the encrypt function earlier).
5. The merchant's controller passes the eToken along with other data to the merchant's server.
6. The merchant's server performs an authenticated QSAPI call to the Bluefin server with the eToken and any other transaction relevant data. The Bluefin server validates the eToken and processes the transaction based on the content of the eToken.
7. The Bluefin server sends the transaction approval/decline response to the merchant's server.
8. The merchant's server passes the response back to the JavaScript controller on their checkout page to finalize the process.

Security and Safety Measures

The following security measures and restrictions must be taken into consideration when integrating with the Bluefin Payment iFrame.

Account ID

When the iFrame is loaded, the merchant's account ID (embedded in src URL) is used to lookup the whitelist of domains. If the incorrect ID is used, the iframe will not load. The same account ID is embedded into the eToken data before encryption. *QSAPI will reject transaction requests if the* `account_id` *in the request does not match the value embedded in the eToken.* This means that eTokens are locked to a merchant account and also means that they cannot be used without the matching api_accesskey on QSAPI.

Domain whitelist

A whitelist of allowed domains is maintained in the merchant's settings page. The list is consulted before the iframe is loaded and when AJAX requests are received from the iframe.

Before a merchant can use the iframe, they have to add their domain to the `IFRAME OPTIONS` section on their settings page (https://secure.payconex.net/admin-site.php)[https://secure.payconex.net/admin-site.php]. The input supports a comma separated list of domains, subdomains, and wildcard subdomains. Entries must exclude the protocol. Sample entry:

```
google.com, bluefin.com, www.bluefin.com, *.example.com
```

If the iframe is loaded from a domain not included in this list, a security error message is presented.

Environment

An eToken from one Bluefin environment cannot be used in another; if crossover is attempted then the QSAPI request will fail. In Bluefin, certification http://cert.payconex.net and production (http://secure.payconex.net) are the considered environment.

Time to live

All eTokens have a timestamp embedded within them; this timestamp is used to restrict their lifespan to 5 minutes on QSAPI. If a transaction is attempted with an eToken that is older than 5 minutes, it will be rejected.

iFrame Timeout

As the Payment iFrame contains sensitive card data, it is treated with high security standards. Part of that increased security includes a process whereby the HTML page will timeout after 10 minutes of inactivity. The reasoning behind this feature is to protect end users who may enter their card data on an open page and then walk away leaving their computer unattended before submitting the page. By automatically redirecting the iFrame, we protect their card data from other users on the same computer.

If a user enters data on the form, setMasked functions are called in the API and the timeout is extended another 10 minutes. The default timeout can be overridden with a startup parameter; values between 1 and 60 minutes are allowed.

Quickstart Guide

Development

If you are integrating with the Payment iFrame for development purposes, please change the base URL of the iframe to the Bluefin development server `https://cert.payconex.net/`.

1. ***iFrame settings***

   Add your website's domain to the settings page, e.g.:

   ```
   google.com,bluefin.com, www.bluefin.com
   ```

2. ***Download the Payment iFrame JavaScript lib***

   The Payment iFrame JavaScript library must be downloaded from the Bluefin Server.

3. ***Load JavaScript lib***

   Embed the Javascript library on your HTML page.

```
<script type="text/javascript" src="/js/iframe-lib-1.0.0.js"></script>
```

4. ***Add iFrame element***

   Add an iframe element that points to the Bluefin iFrame server on the HTML page. Make sure to replace the *aid* value in the src attribute to your Bluefin account id.

```
<iframe
id="payment_iframe"
width="300px"
height="140px"
style="background-color:#FFFFFF; border-radius: 20px; padding:20px;"
frameborder="0"
src="https://secure.payconex.net/iframe?
aid=000000000001&lang=en&cvv=required&expy=single_input&layout=4&show_placeho
ders=true&label_font_family=13&label_font_size=14&input_style=1&label_font_co
or=%23000000">
</iframe>
```

5. ***Initialise the JavaScript lib***

```
var paymentiFrame = new PaymentiFrame({
    create : false,
    iframeId : "payment_iframe"
});
```

6. ***Integrate into web application***

Integrate the *encrypt*, *setMasked*, and *clear* functions into your web application. E.g. call the encrypt function when the user clicks the pay now button.

```
$("#pay_now_button").click(function(){
paymentiFrame.encrypt()
    .failure( function (err) {
        alert("Error: " + err.id + " -> " + err.message );
    })
    .invalidInput( function (data) {
        for ( var i = 0; i < data.invalidInputs.length; i++ ){ alert("Error:
            " + data.invalidInputs[i].code + " -> " + data.inv
alidInputs[i].message );
        }
    })
    .success( function (res) {
        alert( "id " + res.id + " token=>" + res.eToken );
    }) return
false; });
```

Sample code that attaches callback functions to the encrypt API call

```
$("#pay_now_button").click(function(){
    paymentiFrame.encrypt({
        failure : function (err) {
            alert("Error: " + err.id + " -> " + err.message );
        },
        invalidInput :function (data) {
            for ( var i = 0; i < data.invalidInputs.length; i++ ){
                alert("Error: " + data.invalidInputs[i].code + " -> " + data
.invalidInputs[i].message );
            }
        },
        success : function (res) {
            alert( "id " + res.id + " token=>" + res.eToken );
        }
    })
    return false;
```

The same code using the alternative method for specifying the callbacks.

Alternative Payment iFrame Initialization

As an alternative to steps 4 and 5 in the *Quickstart Guide*, the Payment iFrame can be initialized with JavaScript alone.

1. ***Add a parent HTML element***

   Add an html element to your page in the location that you want the Payment iFrame to appear.

   ```
   <div id="iframe_container"></div>
   ```

2. ***Initialise the JavaScript lib***

   Next, when we initialize the iframe JavaScript client library with the create flag set to true, we also pass a settings object that contains the URL configuration properties as well as number of configuration options for the iframe tag. Note:

   - the `parentId` in the settings object that must match up with the ID of the HTML element that is placed on the page.
   - the `devServer` value in the settings object overrides the default server from which the iframe is loaded.
   - the `onload` property in the settings object attaches a JavaScript function to be executed once the page has loaded within the Payment iFrame. This is useful when used in combination with the setMasked function.

```
new PaymentiFrame({
create: true,
iframeId: "payment_iframe",
settings: {
account          : "000000000001",
parentId         : "iframe_container",
lang             : "en",
cvv              : "required",
expy             : "single_input",
layout           : "1",
inputFontFamily  : "13",
inputStyle       : "1",
labelFontSize    : "14",
labelFontColor   : "#000000",
style            : "background-color:#FFFFFF; border-radius: 20px;
padding:20px;",
width            : "300px",
height           : "140px",
showFrame        : false,
devServer        : "https://cert.payconex.net",
onload           : function(){ alert("The Payment iFrame has loaded") }
}
});
```

Fast initialization

When using the JavaScript API to initialize the Payment iFrame, the API first looks for the parent element (parentId in the settings). If it is found, the iFrame is placed on the page and the rest of the initialization function is executed. If it is not found then the iFrame output and remainder of the initialization is placed in a JQuery ready function which is executed after the page has fully loaded. Consequently, the iFrame is not loaded until the other elements on the page load first.

To avoid this behavior while speeding up the loading of the Payment iFrame, the JavaScript constructor invocation code should be placed after the parent HTML element on the page.

```
<html>
  <head>
    <title>My checkout page</title>
  </head>
  <body>
    <div id="iframe_container"></div>


    <script type="text/javascript">
 new PaymentiFrame({
   create: true,
   iframeId: "payment_iframe",
   settings: {
       account     : "000000000001",
       parentId    : "iframe_container"
   }
 });
    </script>
</body>
```

```
    </html>
```

By placing the JavaScript invocation after the parent HTML element the Payment iFrame is loaded immediately.

```
<html>
  <head>
    <title>My checkout page</title>
  </head>
  <body>
    <div id="iframe_container"></div>
    <script type="text/javascript" src="application_logic.js"></script>
  </body>
</html>
```

Similar example, but this time using an inline script tag.

Note: the lazy initialization caused by the parent not being readily available for the JavaScript API introduces a dependency on JQuery. The only other area in which JQuery is used in the API is in the matchLabelStyle * matchInputStyle feature. In both instances the dependency can be avoided, therefore eliminating the need for JQuery.

iFrame Configuration

The URL parameters are used to control the look and feel of the iFrame as well as configure some functionality.

Startup Parameters

The following table outlines the parameters, the allowed values, and the default values from a high level.

Parameter	Description	Options	Default
aid	Sets the merchant account number. This value is locked into the eToken and only that account can use the eToken with the payment API.		
cvv	Sets if CVV data must be gathered, may be gathered, or if the input field should be displayed at all.	`required` , `optional` , `omit`	`required`
lang	Sets the language to use on the input labels and the placeholders.	`en` , `es` , `de` , `it` , `fr`	`en`
number_label	Optional parameter that overrides the default text for the card number input field. See table below for a dictionary of allowed values.	`{1-3}`	

expy_label	Optional parameter that overrides the default text for the expiration date input field. See table below for a dictionary of allowed values.	`{1-3}`	
cvv_label	Optional parameter that overrides the default text for the CVV2 input field. See table below for a dictionary of allowed values.	`{1-13}`	
expy	Sets the type of HTML element to use when gathering the expiry date information.	`single_input` , `double_input` , `single_dropdown` , `double_dropdown`	`single_input`
layout	Sets the general layout for the iframe. Options include single input box, two input boxes (one for month and the other for year), single dropdown and double dropdown.	`layout_1` or `1` , `layout_2` or `2` , `layout_3` or `3` , `layout_4` or `4`	`layout_1` or `1`
input_width	Sets the width of the input boxes for layout_1 and layout_4. Dimensions can be set in pixels, ems or percentages.	`{0-1000}px` , `{0-100}%` , `{0-100}em`	`29%`

Parameter	Description	Options	Default
label_width	Sets the width of the text labels for layout_1 and layout_4. Dimensions can be set in pixels, ems or percentages.	{0-1000}px , {0-100}% , {0-100}em	70%
show_placeholders	Sets if placeholders should be displayed in the input boxes or not.	true , false	true
input_style	Sets the look & feel for the input boxes.	input_style_{1-4} or {1-4}	input_style_1 or 1
label_font_size	Sets the look & feel for font size on the label text.	font_size_{8-34} or {8-34}	
label_font_family	Sets the look & feel for font family on the label text.	font_family_{1-20} or {1-20}	
label_fony_color	Sets color for the iframe text on the label text.	Hex colour	#000000
input_font_size	Sets the look & feel for font size on the input/select elements.	font_size_{8-34} or {8-34}	
input_font_family	Sets the look & feel for font family on the input/select elements.	font_family_{1-20} or {1-20}	
input_font_color	Sets color for the iframe text on the input/select elements.	Hex colour	#000000
timeout	Sets the duration of the timeout period. If the Payment iframe is idle for longer than this period then it redirects to a timeout page. User activity on the iframe extends the timeout by the period defined. The unit is minutes.	{1-60}	10

CVV Options

Option	Description
required	The CVV input is displayed and the end user has to enter CVV data or a validation error is returned.
optional	The CVV input is displayed, however the end user does not have to enter CVV data. If the input is blank the a validation error is not returned.
omit	The CVV input is not displayed.

Language Options

	Description
en	English
es	Spanish
de	German
it	Italian
fr	French

Number Label Override Options

Option	Description
1	Card Number
2	Credit/Debit Card Number
3	Credit Card Number

Expiration Date Label Override Options

Option	Description
1	Expiration
2	EXPY
3	Expiry

Number Label Override Options

Option	Description
1	Card Security Code
2	Card Verification Value
3	Card Validation Number
4	Card Verification Data
5	Card Validation Code
6	Unique Card Code
7	Card Identification Number
8	CSC

9	CVV
10	CVN2
11	CVD
12	CVC2
13	CID

Layout Options

Option	Shorthand Option	Description	Sample
layout_1	1	The text labels and input boxes are inline, with the text left aligned. Use the `input_width` and the `label_width` options to override the default layout.	
layout_2	2	The text label and input box for the card number are full width. The expiry and CVV are split into two columns underneath the card number input.	
layout_3	3	The text labels and input boxes are the full width of the iframe and each is started on a new line.	
layout_4	4	The text labels and input boxes are inline, with the text right aligned. Use the `input_width` and the `label_width` options to override the default layout.	

Expiry Options

Option	Description	Sample
single_input	Single input box to capture expiry date in format MMYY e.g. 0419	
double_input	Double input boxes to capture expiry dates. One for month and another for year.	
single_dropdown	Single dropdown to capture expiry date. The format is MM - YYYY e.g. 05 - 2019	

Option	Description	Sample
double_dropdown	Double dropdowns to capture expiry dates. One for month and another for year.	

Input Box Style Options

Option	Shorthand Option	Description
input_style_1	1	Large Input boxes with rounded corners. Height is 2.6 times the height of font size. This is the default option.
input_style_2	2	Large Input boxes with square corners. Height is 2.6 times the height of font size.
input_style_3	3	Small Input boxes with rounded corners. Height is 1.6 times the height of font size.
input_style_4	4	Large Input boxes with square corners. Height is 1.6 times the height of font size.

Font Size Options

Option	Shorthand Option	Description
size_size_8	8	Sets the text size to 8px (also effects the size of the input elements).
size_size_9	9	Sets the text size to 9px (also effects the size of the input elements).
size_size_10	10	Sets the text size to 10px (also effects the size of the input elements).
size_size_11	11	Sets the text size to 11px (also effects the size of the input elements).
size_size_12	12	Sets the text size to 12px (also effects the size of the input elements).
size_size_13	13	Sets the text size to 13px (also effects the size of the input elements).
size_size_14	14	Sets the text size to 14px (also effects the size of the input elements).
size_size_15	15	Sets the text size to 15px (also effects the size of the input elements).
size_size_16	16	Sets the text size to 16px (also effects the size of the input elements).
size_size_17	17	Sets the text size to 17px (also effects the size of the input elements).

size_size_18	18	Sets the text size to 18px (also effects the size of the input elements).
size_size_19	19	Sets the text size to 19px (also effects the size of the input elements).

Option	Shorthand Option	Description
size_size_20	20	Sets the text size to 20px (also effects the size of the input elements).
size_size_21	21	Sets the text size to 21px (also effects the size of the input elements).
size_size_22	22	Sets the text size to 22px (also effects the size of the input elements).
size_size_23	23	Sets the text size to 23px (also effects the size of the input elements).
size_size_24	24	Sets the text size to 24px (also effects the size of the input elements).
size_size_25	25	Sets the text size to 25px (also effects the size of the input elements).
size_size_26	26	Sets the text size to 26px (also effects the size of the input elements).
size_size_27	27	Sets the text size to 27px (also effects the size of the input elements).
size_size_28	28	Sets the text size to 28px (also effects the size of the input elements).
size_size_29	29	Sets the text size to 29px (also effects the size of the input elements).
size_size_30	30	Sets the text size to 30px (also effects the size of the input elements).
size_size_31	31	Sets the text size to 31px (also effects the size of the input elements).
size_size_32	32	Sets the text size to 32px (also effects the size of the input elements).
size_size_33	33	Sets the text size to 33px (also effects the size of the input elements).
size_size_34	34	Sets the text size to 34px (also effects the size of the input elements).

Font Family Options

Option	Shorthand Option	Family
font_family_1	1	'Helvetica Neue', Helvetica, Arial, sans-serif
font_family_2	2	Impact, Charcoal, sans-serif
font_family_3	3	'Palatino Linotype', 'Book Antiqua', Palatino, serif
font_family_4	4	Tahoma, Geneva, sans-serif
font_family_5	5	Century Gothic, sans-serif
font_family_6	6	'Lucida Sans Unicode', 'Lucida Grande', sans-serif
font_family_7	7	'Arial Black', Gadget, sans-serif
font_family_8	8	'Times New Roman', Times, serif
font_family_9	9	'Arial Narrow', sans-serif
font_family_10	10	Verdana, Geneva, sans-serif
font_family_11	11	'Copperplate / Copperplate Gothic Light', sans-serif

font_family_12	12	'Lucida Console', Monaco, monospace
font_family_13	13	'Gill Sans / Gill Sans MT', sans-serif
font_family_14	14	'Trebuchet MS', Helvetica, sans-serif
font_family_15	15	'Courier New', Courier, monospace
font_family_16	16	Arial, Helvetica, sans-serif
font_family_17	17	Georgia, Serif
font_family_18	18	'Comic Sans MS', cursive, sans-serif
font_family_19	19	sans-serif
font_family_20	20	serif

JavaScript Reference (Constructor)

new PaymentiFrame([settings])

Description * Created an instance of the PaymentiFrame client

lib. new PaymentiFrame([settings])

settings

Type : `Object`

The setting object contains settings required to initialize the JavaScript library. It has :

- **create** (Type: `Boolean`)
  If create is set to false then the JavaScript lib assumes that an iframe element already exists on the page and it initializes the code with the provided iframeId. If create is set to true then the JavaScript lib will build up the HTML for the iframe tag and then outputs it onto the page before initializing.
- **iframeId** (Type: `String`)
  The id of the iframe HTML tag on page.
- **settings** (Type: `Object`)
  Configuration settings used to configure and construct the iframe. This object is only required if the create is set to true. Properties include:
  - `account` the merchants account ID.
  - `parentId` the ID of the HTML element to embed the iframe element into.
  - `lang` the language pack to use. See iFrame Configuration section for a list of the allowed values.
  - `cvv` configure if CVV is required, optional or omitted. See iFrame Configuration section for a list of the allowed values.
    `expy` configures the type of expiry HTML element(s) to use on the page. See iFrame Configuration

section for a list of the allowed values.

- `layout` configures the type of layout to use. See iFrame `Configuration` section for a list of the allowed values.
- `labelFontFamily` configures the font family to use on the label text. See iFrame Configuration section for a list of the allowed values.
- `inputStyle` configures the style for the input boxes. See iFrame Configuration section for a list of the allowed values.
- `labelFontSize` configures the size of the text on the labels. See iFrame Configuration section for a list of the allowed values.

  `labelfontColor` configures the color of the text on the labels. See iFrame Configuration section for a list of the allowed values.
- `inputFontFamily` configures the font family to use on the input/select element text. SeeiFrame Configuration section for a list of the allowed values.

  `inputFontSize` configures the size of the text on the input/select elements. See iFrame Configuration section for a list of the allowed values.
- `inputfontColor` configures the color of the text on the input/select elements. See iFrame Configuration section for a list of the allowed values.
- `style` value of the CSS to set on the iframe tag. Can be used to set background color for the whole iframe element, set rounded corners, etc.
- `width` sets the width of the iframe element.
- `height` sets the height of the iframe element. See iFrame `Configuration` section for a list of the allowed values.
- `showFrame` boolean that sets if the frame should be visible or not.
- `showPlaceholders` boolean that sets if the placeholders should be set on the input boxes or not.
- `labelWidth` Overrides the default width of the text labels. See iFrame `Configuration` section for a list of the allowed values.
- `inputWidth` Overrides the default width of the input boxes. See iFrame Configuration section for a list of the allowed values.
- `timeout` Overrides the default timeout period. See iFrame Configuration section for a list of the allowed values.
- `onload` a function that is called when the Payment iFrame page has finished loading.
- `masked` an object containing masked data and eToken that will be placed in the iframe once it has loaded. The object has two properties `eToken` and `formData`. See the setting masked data section for more details and sample code.
- `matchLabelStyle` the id or JQuery selector for a HTML element to base the labelFontSize, labelFontFamily and labelFontColor values on. See the Computing style from the parent page section for more details.

  `matchInputStyle` the id or JQuery selector for a HTML element to base the inputFontSize, inputFontFamily and inputFontColor values on. See the Computing style from the parent page

section for more details.

`devServer` The base URL to load the iframe from an environment other than production. e.g. https://cert.payconex.net.

Returns

Type : `Object`

`PaymentiFrame` The PaymentiFrame object is returned, the Payment iFrame has three core functions `encrypt` , `setMasked` and `clear` .

Exceptions

There are certain conditions where the PaymentiFrame library will throw an exception during construction;these exceptions are only thrown in scenarios where the library cannot continue with stability e.g. iframe cannot be found. All exceptions are of type `BluefinException` .

BluefinException

Type : `Object`

An object that contains information about the circumstances that caused the exception to be thrown. It has :

- **name** (Type: `String`)
  Object name, this is always set to BluefinException.
- **message** (Type: `String`)
  Human readable message that describes the exception.
- **code** (Type: `Number`)
  The code identifies the exceptional circumstances that caused the exception to be thrown. The codes are as follows:
  - `100` an element with the ID provided in the iframeId cannot be found on the page.
  - `101` an element with the iframeId ID was found on the page but it was not an iframe.
  - `102` an iframe with the iframeId ID was found but its source is not pointing to a valid Bluefin domain.
    `103` an iframe with the iframeId ID was found but it doesn't point to the correct path.

JavaScript Reference (Functions)

paymentiFrame.encrypt([settings])

Description *Encrypts card data on the iframe and passes eToken to success callback. Internally, the encrypt function communicates with the encryption iframe and instructs it to capture the users input, validate the input, encrypt the card data and send the resulting eToken back to the parent page. The function takes a `settings` object as a parameter and returns the `EncryptResponse` object.*

paymentiFrame.encrypt([settings])

settings

Type : `Object`

A set of key/value pairs that configure the encryption function call. All properties are optional.

success

Type: `Function(Object)`

A function to be called when the asynchronous encrypt function successfully finishes. The function gets passed one argument, an Object with a three properties:

- **ID** (Type: `String`)
  Unique GUID that identifies the API/function call. The same id is returned from the encrypt function call.
- **eToken** (Type: `String`)
  Which contains a unique identifier for the API call.
- **masked** (Type: `Object`)
  An object containing the masked card data. The object has the following properties: **number** (Type: `String`) a masked payment card number in the following format 4xxxxxxxxxxxx1234. **expy** (Type: `String`) the unmasked payment expiry value in the clear e.g. 1219. **cvv** (Type: `String`) the masked CVV value e.g. xxx or xxxx.

---

failure

Type: `Function(Object)`

A function to be called when the asynchronous encrypt function fails. The function gets passed one argument, an Object with a two properties:

- **ID** (Type: `String`)
  Unique GUID that identifies the API/function call. The same id is returned from the encrypt function call.
- **message** (Type: `String`)
  A human readable error message.

---

invalidInput

Type: `Function(Object)`

A function that is called if a user inputs invalid data on the iframe. An array of objects will be passed to this function so that the parent page can provide error feedback to the users. The function gets passed

one argument, an Object with a two properties one of which is an array of error objects.

- **ID** (Type: `String`)
  Unique GUID that identifies the API/function call. The same ID is returned from the encrypt function call.
- **invalidInputs** (Type: `Array`)
  An array of *invalidInput* objects. The *invalidInput* object has the following properties: **field** (Type: `String`) identifies the field with invalid input, must be one of*number*, *expy*, or *cvv*. **message** (Type: `String`) a human readable error message for the invalid input and **code** (Type: `Number`) a code to identify the reason that the user input value was rejected. For a full list of Codes refer to the Invalid Input Codes section.

Returns

Type : `Object`

`EncryptResponse` The EncryptResponse object is returned from the `paymentiFrame.encrypt( )` function. It contains the `success()`, `failure()`, and `invalidInput()` methods which attach functions to the internal success, failure and invalidInput events. The callback functions passed to these methods have the same signature as the corresponding properties in the `settings` object.

paymentiFrame.setMasked(maskedData, eToken, [settings])

Description *This function populates an iframe with an eToken and a masked card data object. If the user does not modify the values in the input box then the iframe outputs the same masked data and eToken when the `encrypt()` function is called. However, if the user modifies the data in the input fields then the encrypt function will use the new values when creating an eToken. The function takes a `settings` object as a parameter and returns the `MaskedResponse` object.*

paymentiFrame.setMasked(maskedData, eToken, [settings])

maskedData

Type : `Object`

An object containing the masked card data. The object has the following properties:

- **number** (Type: `String`) a masked payment card number in the following format 4xxxxxxxxxxxx1234.
- **expy** (Type: `String`) the unmasked payment expiry value in the clear e.g. 1219.
- **cvv** (Type: `String`) the masked CVV value e.g. xxx or xxxx.

---

eToken

Type : `String`

An eToken obtained through the `paymentiFrame.encrypt()` function.

---

settings

Type : `Object`

A set of key/value pairs that configure the encryption function call. All properties are optional.

success

Type: `Function(Object)`

A function to be called when the asynchronous encrypt function successfully finishes. The function gets passed one argument, an Object with a three properties:

- **ID** (Type: `String`)
  Unique GUID that identifies the API/function call. The same id is returned from the encrypt function call.
- **eToken** (Type: `String`)
  Which contains a unique identifier for the API call.
- **masked** (Type: `Object`)
  An object containing the masked card data. The object has the following properties: **number** (Type: `String`) a masked payment card number in the following format 4xxxxxxxxxxxx1234. **expy** (Type: `String`) the unmasked payment expiry value in the clear e.g. 1219. **cvv** (Type: `String`) the masked CVV value e.g. xxx or xxxx.

---

failure

Type: `Function(Object)`

A function to be called when the asynchronous setMasked function fails. The function gets passed one argument, an Object with a two properties:

- **ID** (Type: `String`)
  Unique GUID that identifies the API/function call. The same ID is returned from the encrypt function call.
- **message** (Type: `String`)
  A human readable error message.

Returns

Type : `Object`

`MaskedResponse` The MaskedResponse object is returned from the `paymentiFrame.setMasked()` function. It contains the `success()` and `failure()` methods which attach functions to the internal success and failure events. The callback functions passed to these methods have the same signature as the corresponding properties in the `settings` object.

paymentiFrame.clear([settings])

Description *Clears the input fields within the iframe*

paymentiFrame.clear([settings])

settings

Type : `Object`

A set of key/value pairs that configure the clear function call. All properties are optional.

success

Type: `Function(Object)`

A function to be called when the asynchronous clear function successfully finishes. The function gets passed one argument, an Object with a single property `id` which contains a unique identifier for the API call.

---

failure

Type: `Function(Object)`

A function to be called when the asynchronous clear function fails. The function gets passed one argument, an Object with a two properties: an `id` which contains a unique identifier for the API call and `message` a human readable error message.

Returns

Type : `Object`

`ClearResponse` The ClearResponse object is returned from the `paymentiFrame.clear()` function. It contains the `success()` and `failure()` methods which attach functions to the internal success and failure events. The callback functions passed to these methods have the same signature as the corresponding properties in the `settings` object.

QSAPI Transaction with eToken

To use the eToken with Bluefin's API (`QSAPI`) you need to use the `eToken` parameter. Note: when you include the eToken parameter in a QSAPI request it will override the values set in the `number` , `expy` , and `cvv` values. For further details on the other supported QSAPI parameters, please refer to the QSAPI document `Bluefin PayConex API Library` .

The following is an example of a QSAPI request; to replicate the transaction make sure to replace the placeholders with the appropriate values for your account `<replace_account_id>`, `<place_api_accesskey_here>` and `<place_token_here>`.

```
curl "https://secure.payconex.net/api/qsapi/3.8/" \
```

```
    -X POST \
    --header "Content-Type: application/x-www-form-urlencoded" \
    --header "Cache-Control: no-cache" \
    -d response_format=JSON \
    -d account_id=<replace_account_id> \
    -d api_accesskey=<place_api_accesskey_here> \
    -d tender_type=CARD \
    -d transaction_type=SALE \
    -d transaction_amount=99.00 \
    -d first_name=Joseph \
    -d etoken=<place_token_here> \
    -d custom_id=f04f4fee-91a7-4329-9d65-c228dbd8319a
```

Sample QSAPI request with etoken.

```
<?php
    header('Content-type: text/plain');

    require_once('includes/qsapi-post-3.8.class.php');

    $qs = new PayConexPost();

    $qs->setParams(array (
        'account_id'              => '<replace_account_id>',
        'api_accesskey'           => '<place_api_accesskey_here>',
        'tender_type'             => 'CARD',
        'transaction_type'        => 'SALE',
        'transaction_amount'      => 176.58,
        'transaction_description' => 'Test transaction',
        'eToken'                  => '<place_token_here>',
        'first_name'              => 'John',
        'last_name'               => 'Smith',
        'street_address1'         => '123 Main St',
        'city'                    => 'Anytown',
        'state'                   => 'NY',
        'zip'                     => '10101',
        'phone'                   => '212-555-1212',
    ));

    $qs->process();
    $response = $qs->getResponse();
```

The same request with PHP client lib.

Setting Masked Data

The purpose of the set masked data feature is to reset the Payment iFrame to a `process complete` state without getting the end user to re-enter their card data. In the `process complete` state the iFrame has the eToken stored in its internal cache. Once the cache is primed with the eToken, it is output in subsequent `encrypt` function calls. The purpose of this feature is to allow merchants to repopulate their checkout page without having to store sensitive card holder data. It is particularly useful if the merchant has to reload the page after data from another field fails validation; allowing the end user to fix the incorrect data and avoid card data re-entry.

To use this feature the JavaScript API exposes a `setMasked` function that takes three arguments: 1) an

eToken, 2) a masked data object, and 3) a setting object for success and failure callback function. The masked data object must contain three properties `number`, `cvv` , and `expy` . The expy can be entered in plain text, however the cvv must be masked with the numbers replaced with x's. The number must mask out all the number except for the first digit and the last four e.g. 5xxxxxxxxxx5454.

For convenience the `encrypt` function outputs the masked data object along with the eToken to the success function.

When the `setMasked` function is called the input boxes within the Payment iFrame are populated with the values in the masked data object. If the user alters the masked data in any way then the whole form is cleared and the eToken is cleared from the internal cache. At this point the card data has to be reentered and processed/encrypted.

Sample API call. The following example outlines how a merchant can set the masked data for a Payment iFrame that is **not** created by the JavaScript API.

```
var paymentiFrame = new PaymentiFrame({ create : false, iframeId : "pay_if"});
var masked_data = {
            number : "5xxxxxxxxxx5454",
            cvv    : "xxx",
            expy   : "1219"
        };

var eToken = "xRyzz2-9eREX_7Zu2Ix1zamDV2cMkqNKb4P05-CU-
KXGhAfmfz67AVj0qkj3N278Khw80rxjzKfZEYdtnKPOnIzcgPlJRZ6weC4AVc5WQR00yhgT-
Hcg0dAF7faZXSYd-J1-u94cTfsQwJmo5i5lGYIPGGa7Kex3w9-
OO9VTFjdhFFVmAsuNx1ugJnGESy4zdeyR8N53H8D_nCOJ5jdwPSzA3dbw_TvY";

paymentiFrame.setMasked( masked_data, eToken )
            .success(function(res){
                alert(res);
            })
            .failure(function(err){
                alert(err);
            });
```

In instances where the Payment iFrame is loaded through the init function in the JavaScript SDK, the masked data can be set through the `masked` property of the `settings` object.

```
new PaymentiFrame( {
    create    : true,
    iframeId  : "pay_if",
    settings  : {
        account          : 000000000001,
        parentId         : "pay_if_parent_div",
        lang             : "en",
        cvv              : "required",
        expy             : "single_input",
        layout           : "3",
        labelFontFamily  : "13",
        inputStyle       : "1",
        labelFontSize    : "14",
        labelFontColor   : "#0000FF",
        style            : "background-color:red; border-radius: 25px; padding:
x;",
        showFrame        : false,
        masked           : {
            formData: {
                number:
                "5xxxxxxxxxxx5454
                ", cvv: "xxx",
                expy: "1219"
            },
            eToken: "xRyzz2-9eREX_7Zu2Ix1zamDV2cMkqNKb4P05-CU-
KXGhAfmfz67AVj0qkj3N 278Khw80rxjzKfZEYdtnKPOnIzcgPlJRZ6weC4AVc5WQR00yhgT-
Hcg0dAF7faZXSYd-J1-u94cTfsQwJmo5i5lGYIPGGa7Kex3w9-
OO9VTFjdhFFVmAsuNx1ugJnGESy4zdeyR8N53H8D_nCOJ5jdwPSzA3dbw_TvY"
        }
    }
});
```

Computing style from the parent page

The JavaScript API's constructor has a feature that computes the values of the Payment iFrame's fontSize, fontFamily and fontColor for both the text labels and the input/select elements based on a HTML element on the parent page. To enable the feature use the `matchLabelStyle` and `matchInputStyle` properties of the `settings` object (Note: this feature only works in `create` mode. The value for the `matchLabelStyle` and `matchInputStyle` properties can either be a) the ID of the HTML element or b) a JQuery selector for the HTML element.

Internally the API matches the computed style of the HTML element to the matching settings for the fontSize, fontFamily and fontColor. It doesn't matter if the style for the element is inherited from a parent element the computed value will account for that style also.

Note: it is possible to use the matchLabelStyle and matchInputStyle features and to override the inferred values by specifying the labelFontSize, labelFontFamily, inputFontColor, inputFontSize, inputFontFamily, or inputFontColor options in the settings object.

```
<body style="color:red; fontSize : 14px ">
```

```
    <div id="elm_to_copy" style="font-family: 'Lucida Sans Unicode', 'Lucida Grand
e', sans-serif;">Lorem ipsum</div>
    <input id="input_elm_to _copy" style="color:#0000FF; font-family: 'Lucida Sans
Unicode', 'Lucida Grande', sans-serif;" value="My Text"/>
    <div id="pay_if_parent_div" ></div>

</body>
```

Add an ID to the HTML element whose style you want to copy; in this case elm_to_copy and input_elm_to_copy

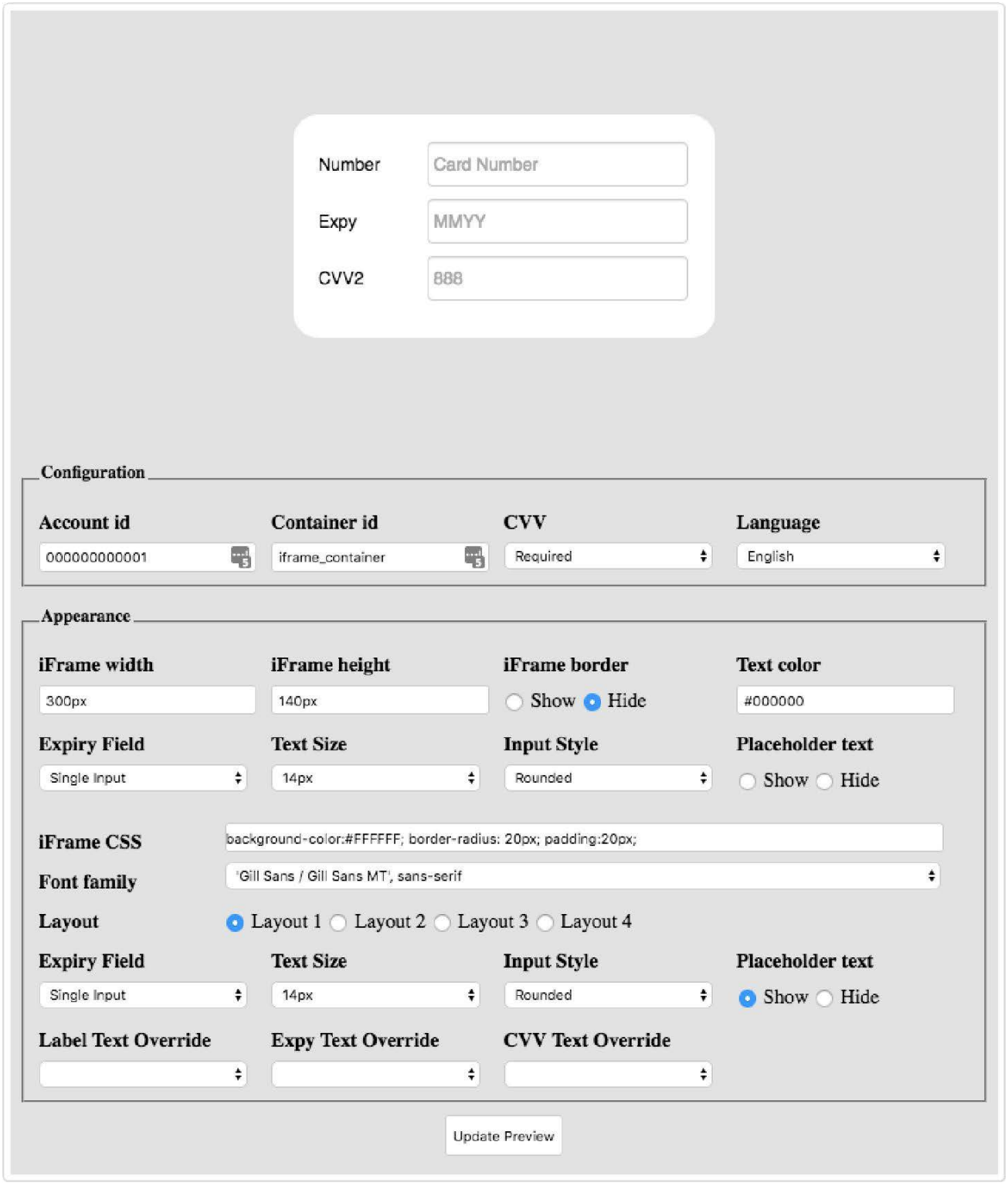
```
new PaymentiFrame( {
    create    : true,
    iframeId  : "pay_if",
    settings  : {
        account          : 000000000001,
        parentId         : "pay_if_parent_div",
        lang             : "en",
        cvv              : "required",
        expy             : "single_input",
        layout           : "3",
        inputStyle       : "1",
        labelFontColor   : "#0000FF",
        style            : "background-color:red; border-radius: 25px; padding: 25p
x;",
        showFrame        : false,
        matchLabelStyle : "elm_to_copy",
        matchInputStyle : "input_elm_to_copy"
    }
});
```

Set the ID in the matchLabelStyle and matchInputStyle settings

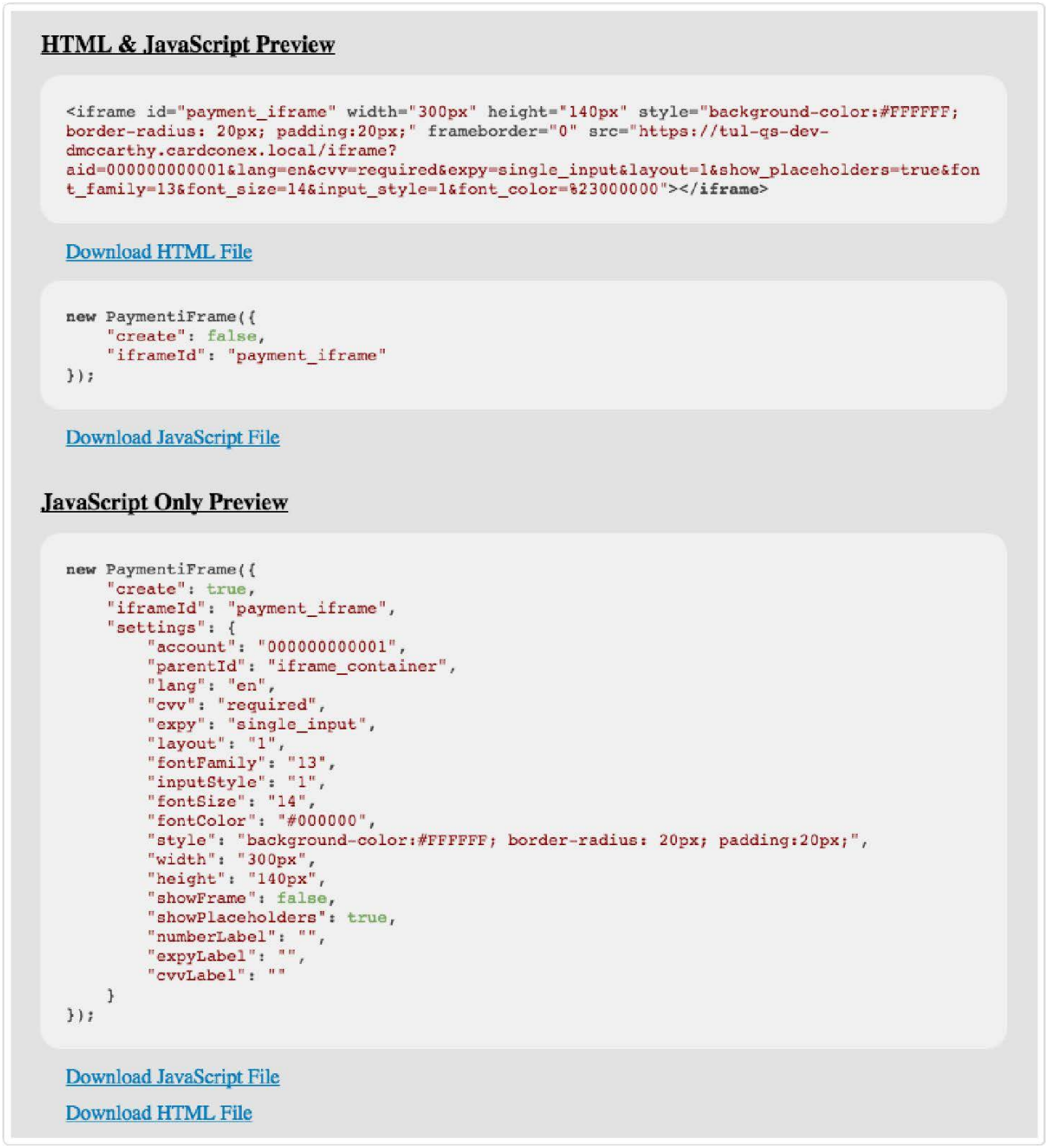
iFrame Builder

As there are many configuration options available to merchants and deciding on the best ones may be difficult, we created a builder page which simplifies the process. The builder also includes settings for the iframe html element that exist outside of the control of the Bluefin Payment iFrame code but have a large impact on the look and feel of the iFrame.

To enable rapid development the builder includes a download code feature that auto-generates the HTML and JavaScript code based on configuration options.



The top half of the builder page. This section includes a number of input options that control the behavour and the UI of the Payment iFrame. At the top of the page there is a preview area that contains an instance of the iFrame based on the options selected.



HTML & JavaScript Preview

```
<iframe id="payment_iframe" width="300px" height="140px" style="background-color:#FFFFFF;
border-radius: 20px; padding:20px;" frameborder="0" src="https://tul-qs-dev-
dmccarthy.cardconex.local/iframe?
aid=000000000001&lang=en&cvv=required&expy=single_input&layout=1&show_placeholders=true&fon
t_family=13&font_size=14&input_style=1&font_color=%23000000"></iframe>
```

Download HTML File

```
new PaymentiFrame({
    "create": false,
    "iframeId": "payment_iframe"
});
```

Download JavaScript File

JavaScript Only Preview

```
new PaymentiFrame({
    "create": true,
    "iframeId": "payment_iframe",
    "settings": {
        "account": "000000000001",
        "parentId": "iframe_container",
        "lang": "en",
        "cvv": "required",
        "expy": "single_input",
        "layout": "1",
        "fontFamily": "13",
        "inputStyle": "1",
        "fontSize": "14",
        "fontColor": "#000000",
        "style": "background-color:#FFFFFF; border-radius: 20px; padding:20px;",
        "width": "300px",
        "height": "140px",
        "showFrame": false,
        "showPlaceholders": true,
        "numberLabel": "",
        "expyLabel": "",
        "cvvLabel": ""
    }
});
```

Download JavaScript File

Download HTML File

The bottom half of the builder page. This contains code preview areas that display the HTML and JavaScript code required to display the Payment iFrame based on the option selected

The Payment iFrame builder can be found at the following location https://secure.payconex.net/iframe/builder.php.

Appendix

Appendix I: Invalid Input Codes

The error objects passed to the invalidInput callback function include an error code, these codes map to the following descriptions:

- `1000` : blank field.
- `1001` : input too short.
- `1002` : input too long.
- `1003` : input not numeric.
- `1004` : invalid card number.
- `1005` : invalid expiry date.

Appendix II: Browser Support

The Payment iFrame JavaScript API is supported by the following browsers: by the following browsers:

- Internet Explorer 8+
- Edge 13+
- Firefox 46+
- Chrome 29+
- Safari 9.1+

  Opera 38+