Monetra® Payment Software

PaymentFrame Guide

Revision: 2.1.0
Publication date June 18, 2023

PaymentFrame Guide

Monetra Technologies, LLC

Revision: 2.1.0

Publication date June 18, 2023

Legal Notice

Table of Contents

1 Revision History

Version	Date	Changes
v1.0	2017-08-30	• Initial document.
v1.1	2017-10-31	• Documented the ability to split cardholder name field into first and last.
v1.2	2018-02-12	• This update applies to the "HMAC Parameters" section. It enhances the description for the 'hmac-css-url' parameter and adds the three new ones below: * `hmac-include-cardholdername` * `hmac-include-street` * `hmac-include-zip`
v1.3	2018-07-18	• This update applies to the "HMAC Parameters" section. It describes how to use the new 'auto-load' feature.
v1.4	2018-10-04	• This update applies to the "HMAC Parameters" section. It describes how to use the new `'autocomplete'` feature.
v1.5	2019-10-11	• This update describes how to let Monetra/TranSafe host the javascript files.
v1.6	2019-12-09	• Support for suppressing the submit button to allow a submit button external to the iframe to submit.
v1.6.1	2020-04-02	• Explicitly note that only HTTPS is supported.
v1.7.0	2021-08-02	• Add ACH support.
v2.0.0	2022-04-27	• Authentication now uses API Keys
v2.1.0	2022-06-18	• Expand Javascript documentation • Support for 3DS 2.0

2 Monetra PaymentFrame

2.1 Introduction

PaymentFrame is a Monetra feature that allows you to embed a secure iframe containing a payment form on an HTTPS-enabled web page. This prevents credit card data from ever touching your systems, while allowing your customers to complete seamless ecommerce transactions on your site.

Note: The PaymentFrame can not be embedded into a non-secured HTTP site, even for testing and development purposes, due to content security policies and mixed active content, the browser will silently fail to load the page.

PCI Notice: This approach to Ecommerce integration follows the PCI Council's best practices as outlined in https://www.pcisecuritystandards.org/pdfs/best_practices_securing_ecommerce.pdf, and qualifies for `SAQ-A`.

2.2 How It Works

Note: In order to use the PaymentFrame, you will need to create a non-expiring merchant profile API Key on the Monetra (or TranSafe) instance you are connecting to. This API Key needs to at a minimum contain the `action_admin=TICKETREQUEST`, however if using the same API Key for the follow-up actions, it must contain any necessary permissions for the follow-up actions.

Examples for PaymentFrame implementation and styling are available https://www.monetra.com/developers with a live demo at https://iframe.test.transafe.com.

1. *Prepare your credentials:* Before rendering your payment page (the page that will host the iframe), you will need to generate a 256-bit keyed-hash message authentication code (HMAC-SHA256) See Section 2.3: HMAC Parameters.

2. *Add iframe element and components:* Your html payment page must include an empty iframe element (with no `src` attribute specified) with a unique id which will be referenced by the JavaScript where you would like the payment form to appear. The HMAC message components (along with the HMAC itself from step 1) must be included as `data-` attributes on the iframe element. Note: The order in which the data attributes are concatenated for the HMAC message must match the order of the parameters in the table in Section 2.3: HMAC Parameters. The ordering is crucial for Monetra/TranSafe to be able to properly verify the HMAC.

3. *Load script:* Your payment page will need to load `https://{{monetra_url}}/PaymentFrame/PaymentFrame.js`, which provides the client-side logic for displaying the iframe. For example, if you are using test.transafe.com as your payment server, the URL would be https://test.transafe.com/PaymentFrame/PaymentFrame.js.

4. *Load Payment Form:* From here you'll just need to write a few lines of your own Javascript to instantiate a `PaymentFrame` object and request the iframe. You will need to define a callback function to be executed after the user's payment information has been submitted. The full sequence would be to instantiate `PaymentFrame` with the iFrame id and URL for the Monetra Server, define callback using the `.setPaymentSubmittedCallback()` method, and call the `.request()` method to load the iFrame. You can find an example of this sequence in the code example below.

5. *Submit Cardholder Data:* Once the iframe has loaded, it will render a payment form into which the user can enter their payment information. When the user submits the form, their information will be sent directly to the Monetra/TranSafe server, which will return a CardShield ticket.

6. *Process response:* At this point, the callback function that you defined in your Javascript will be executed, receiving a JSON object representing the Monetra/TranSafe server's response (including the CardShield ticket) as a parameter.

7. *Process the ticket:* Once you have a valid ticket then you can communicate directly with the Monetra/TranSafe system to run a standard transaction such as as SALE or a PRE-AUTH as defined in the Monetra Application Interface Guide.

2.3 HMAC Parameters

Note: Please ensure these iframe attributes are prefixed with `data-` when using our standard PaymentFrame javascript to display the iFrame as the javascript library uses this prefix to determine the attributes used by the PaymentFrame and sent in the request to the Payment Server.

Legend:
Req = Required or not [Y=Yes C=Conditional O=Optional]

Key	Req	Description
`hmac-hmacsha256`	Y	Generated HMAC-SHA256 using the API Key Secret as the key. The message must be a concatenation (with no delimiters) of all of the other hmac- values you are

Key	Req	Description
		providing, in the same order in which they appear in this table (with the exception of this attribute)
`hmac-timestamp`	Y	Standard Unix timestamp. Must be within 15 minutes of server's time.
`hmac-domain`	Y	The domain of the page embedding the iframe. This must match exactly what would show in the browser URL bar, including 'https://' (and port, if it is explicitly set). Please note that using 'http://' will NOT work due to Content Security Policies and Mixed Active Content.
`hmac-sequence`	Y	Merchant-specified sequence number. May be alphanumeric.
`hmac-auth_apikey_id`	Y	The API Key ID returned by the Monetra server when creating the merchant profile API Key.
`hmac-css-url`	O	URL of CSS file the iframe should load. It must be served from the same domain as `hmac-domain`. If no CSS URL is provided, the payment form will be rendered with default styling (see "Styling the Payment Form" for more information).
`hmac-include-cardholdername`	O	Indicates whether the cardholder name field should be present on the payment form. Value can be "yes" or "no". Defaults to "yes" if not provided. Only provide "no" here if cardholder name is collected elsewhere during your checkout process, and sent along with the ticket in the Monetra sale request.
`hmac-include-street`	O	Indicates whether the street address field should be present on the payment form. Value can be "yes" or "no". Defaults to "yes" if not provided. Only provide "no" here if street address is collected elsewhere during your checkout process, and sent along with the ticket in the Monetra sale request.
`hmac-include-zip`	O	Indicates whether the zip code field should be present on the payment form. Value can be "yes" or "no". Defaults to "yes" if not provided. Only provide "no" here if zip code is collected elsewhere during your checkout process, and sent along with the ticket in the Monetra sale request.
`hmac-expdate-format`	O	Format of expiration date the iframe should use. Choices: * single-text: Free-form text entry. Default if not specified. * separate-selects: Two drop downs for month and year. * coupled-selects: Two drop downs for month and year in a container.
`hmac-cardholdername-format`	O	Single, or split cardholder name. Choices: * `separate-firstlast` * `combined-firstlast`

Key	Req	Description
		`separate-firstlast` will split the card holder name field into first and last name fields. This field must be part of the hmac if present. It comes after 'expdate-format' in the hmac data.
`hmac-auto-reload`	O	Indicates whether checkout page containing payment form should auto-reload every 15 minutes in order to avoid the HMAC becoming invalid due to the time it was generated. Value can be 'yes' or 'no'. Defaults to 'no' if not provided. If 'yes', users will see a message on the checkout form (starting approximately 5 minutes before the reload) indicating that the page will be reloaded soon for security purposes. The message will include a timer that counts down the time remaining until the reload occurs.
`hmac-autocomplete`	O	Indicates whether browser `autocomplete` should be enabled on the payment form. Value can be 'yes' or 'no'. Defaults to 'yes' if not provided. If 'yes', payment form fields will allow `autocomplete` (as long as the user's browser settings are configured to allow it, which is generally the default). If 'no', `autocomplete` will be disabled on the payment form fields.
`hmac-include-submit-button`	O	Indicates whether the payment form should include a submit button within the iframe. Defaults to 'yes' if not provided. If you would like to use a submit button on your own page outside of the iframe, set this value to 'no'. You would then use Javascript to send a message to the iframe to trigger a form submit. See Section 2.6: The PaymentFrame Javascript Library.
`hmac-ach-enabled`	O	Whether to display an ACH payment form in the iframe. Value can be 'yes' or 'no'. Defaults to 'no' if not provided (meaning a credit card payment form will be displayed).
`hmac-ach-accounttype`	O	Which ACH account type options should be allowed, and the order in which they will appear. Should be provided as a comma-separated list of values. Options are "business", "personal", "checking", and "savings". Options will be displayed to user in two separate `select` elements, one for selecting "business" or "personal" and another for selecting "checking" or "savings". The order of these options (and therefore which options are the default choices) will match the order in which they are provided in this list. If ACH is enabled and this value is not provided, the allowed options will default to "personal"/"business" and "checking/savings". If ACH is not enabled, this value has no effect and should not be provided.

Note: Response Parameters are the same as returned by the Monetra POST Protocol ticket request. The hmac- timestamp and hmac-sequence will be used as part of the POST response to allow the hmac-hmacsha256 response from the ticket request to be validated.

2.4 3D Secure

If you account has 3DS enabled, and some or all of the 3DS field values are already known prior to rendering the payment form, you specify these values by setting `data-` attributes on the `<iframe>` that will contain the payment form. Once the payment form is requested, these fields will be passed along with the request and included in the payment form as hidden `<input>` tags.

Note: 3D Secure data fields are not part of the HMAC process

For example, if the customer's shipping address is collected in an earlier checkout step, it can be passed into the iframe like so (note the `data-3ds` attributes following the required `data-hmac` attributes):

```
<iframe id="my-iframe-id"
 data-hmac-hmacsha256="2f46a6407581b4153b97f589fbb321cc13c2fc84c122226991ea95e2aac08c3e"
 data-hmac-domain="https://merchantsite.com"
 data-hmac-timestamp="1685470601"
 data-hmac-sequence="64763d89e3aa2"
 data-hmac-auth_apikey_id="P0052319E456B379E"
 data-3ds-ship_addr_line_1="123 Main St"
 data-3ds-ship_addr_line_2="Apt 1B"
 data-3ds-ship_addr_city="Miami"
 data-3ds-ship_addr_state="FL"
 data-3ds-ship_addr_post_code="33132"
 data-3ds-ship_addr_country="USA">
</iframe>
```

Note: If 3DSecure fields are collected on the same page, however, you can submit these into the iframe using Section 2.6.7.1: `add3dsData(Object data)`

The fields below are the 3DSecure-specific fields. When passed into the iframe as attributes, prefix with `data-3ds-`:

- `cardholdername`: Cardholder name (if not included in iframe payment form)
- `bill_addr_city`: Billing address city
- `bill_addr_country`: ISO 3166-1 numeric three-digit country code of the billing address
- `bill_addr_line_1`: Line 1 of billing street address (if not included in iframe payment form)
- `bill_addr_line_2`: Line 2 of billing street address (if not included in iframe payment form)
- `bill_addr_line_3`: Line 3 of billing street address (if not included in iframe payment form)
- `bill_addr_post_code`: Billing address zip/postal code (if not included in iframe payment form)

- `bill_addr_state`: Billing address state
- `email`: Email address
- `home_phone_countrycode`: Home phone country code
- `home_phone_number`: Home phone number
- `mobile_phone_countrycode`: Mobile phone country code
- `mobile_phone_number`: Mobile phone number
- `ship_addr_city`: Shipping address city
- `ship_addr_country`: ISO 3166-1 numeric three-digit country code of the shipping address
- `ship_addr_line_1`: Line 1 of shipping street address
- `ship_addr_line_2`: Line 2 of shipping street address
- `ship_addr_line_3`: Line 3 of shipping street address
- `ship_addr_post_code`: Shipping address zip/postal code
- `ship_addr_state`: Shipping address state
- `work_phone_countrycode`: Work phone country code
- `work_phone_number`: Work phone number
- `purchase_amount`: Total amount of the purchase
- `purchase_currency`: 3-digit ISO 4217 currency code in which purchase amount is expressed

2.5 Styling the Payment Form

The `iframe` content can be styled using custom CSS. You can use the classes and IDs documented below in your CSS to style the payment form and the elements it contains.

You can also provide custom text for the form's labels by specifying appropriate CSS rules. Each label contains an empty span element, so that you can use the ::before selector in conjunction with the content property to insert your desired text into the label. For example, in order to provide custom text for the ZIP code label, you would use a CSS rule similar to the following:

```
#payment-zip-label span::before {
      content: "Custom text here";
}
```

Note: If you do not provide custom text for any label, it will default to the value referenced in the "IDs" table below.

CLASSES	
Class	**Description**
`payment-form-label`	Applied to all label elements in the payment form. Each of these elements contains an input or select element into which payment data will be entered.
`payment-expdate-label`	If hmac-expdate-format is set to separate-selects or coupled-selects, this will be applied to the two label elements that contain the select drop-downs for expiration date month and year.

CLASSES	
Class	**Description**
`payment-cardholdername-label`	If hmac-cardholdername-format is set to separate-firstlast, this will be applied to the two label elements that contain the first and last name text entry fields.
`payment-ach-accounttype-container`	`div` element containing a `label` and `select` for selecting ACH account type

ID's	
ID	**Description**
`monetra-payment-form`	The payment `form` element.
`payment-account-label`	The label containing the account (credit card number) input element. Label text defaults to 'Card Number'.
`payment-expdate-container`	If hmac-expdate-format is set to coupled-selects, this is the div that contains the two select elements.
`payment-expmonth-label`	If hmac-expdate-format is set to separate-selects or coupled-selects, this is the label that contains the expiration month select element. Label text defaults to 'Expiration Month'.
`payment-expyear-label`	If hmac-expdate-format is set to separate-selects or coupled-selects, this is the label that contains the expiration year select element. Label text defaults to 'Expiration Year'.
`payment-expdate-label`	If hmac-expdate-format is set to single-text, this is the label that contains the 'expdate' (expiration date) input element. Label text defaults to 'Expiration Date'.
`payment-cardholdername-label`	The label containing the `'cardholdername'` input element. Label text defaults to 'Cardholder Name'.
`payment-cardholdernamefirst-label`	If hmac-cardholdername-format is set to separate-firstlast, this is the label that contains the first name field. Label text defaults to Cardholder First Name.
`payment-cardholdernamelast-label`	If hmac-cardholdername-format is set to separate-firstlast, this is the label that contains the last name field. Label text defaults to Cardholder Last Name.
`payment-street-label`	The label containing the 'street' (street address) input element. Label text defaults to 'Street Address'.
`payment-zip-label`	The label containing the 'zip' (ZIP code) input element. Label text defaults to 'ZIP Code'.
`payment-cv-label`	The label containing the 'cv' (card verification value) input element. Label text defaults to 'CV'.
`payment-submit-button`	The button element used to submit the form.
`payment-ach-account-label`	`label` element containing ACH account number input
`payment-ach-account`	ACH account number input

ID's	
ID	**Description**
`payment-ach-account-confirm-label`	`label` element containing ACH account number confirmation input
`payment-ach-account-confirm`	ACH account number confirmation input
`payment-ach-routing-label`	`label` element containing ACH routing number input
`payment-ach-routing`	ACH routing number input
`payment-ach-cardholdername-label`	`label` element containing ACH account holder name input
`payment-ach-cardholdername`	ACH account holder name input
`payment-ach-accounttype-owner-label`	`label` element containing `select` element for selecting Business or Personal ACH account type
`payment-ach-accounttype-owner-select`	`select` element for selecting Business or Personal ACH account type
`payment-ach-accounttype-purpose-label`	`label` element containing `select` element for selecting Checking or Savings ACH account type
`payment-ach-accounttype-purpose-select`	`select` element for selecting Checking or Savings ACH account type

2.6 The PaymentFrame Javascript Library

The PaymentFrame Javascript library provides a `PaymentFrame` object that can be used to set up and configure your PaymentFrame instance.

The object can be instantiated as follows:

```
1  paymentFrame = new PaymentFrame(
2   "my-iframe-id",
3   "https://test.transafe.com"
4  );
```

The first argument is the ID of the `<iframe>` element on your checkout page that will contain the PaymentFrame payment form.

The second argument is the domain of the payment server you are using to generate the PaymentFrame.

In the above example, the `<iframe>` element has an ID of `"my-iframe-id"`, and the payment server is `https://test.transafe.com`.

Once you have instantiated the PaymentFrame object, you can utilize the following methods to set callback functions, request the payment form from the payment server, and trigger events within the iframe when necessary.

2.6.1 setPaymentFormLoadedCallback(Function callback)

Set a function that will be triggered once the payment form has loaded within the iframe. Useful if any action should be taken on the host checkout page after the iframe has finished loading.

```
paymentFrame.setPaymentFormLoadedCallback(function() {
 console.log("Payment form has loaded");
});
```

2.6.2 setFormSubmissionInvalidCallback(Function callback)

Set a function that will be triggered if the user submits the payment form, and the form's data is determined to be invalid. This could be due to required fields that are missing, or fields that are formatted incorrectly. The iframe will highlight any erroneous fields within the payment form, but this function allows the host checkout page to take additional action as needed.

```
paymentFrame.setFormSubmissionInvalidCallback(function() {
 console.log("Form submission was invalid");
});
```

2.6.3 setPaymentSubmittedCallback(Function callback)

Set a function that has been triggered after the payment form has been submitted and a response has been received from the payment server. The callback will receive an object containing the response data returned by the payment server. This response will indicate whether a ticket was successfully generated, and if so, the response will also contain the ticket.

```
paymentFrame.setPaymentSubmittedCallback(function(response) {
 if (response.code === "AUTH") {
  console.log("Ticket received:", response.ticket);
 } else {
  console.log("Error:", response.verbiage);
 }
});
```

2.6.4 request()

This method is used to request the iframe payment form from the payment server. Once you have instantiated the PaymentFrame object and set any desired callback functions, you must call this method to render the payment form.

```
paymentFrame.request();
```

2.6.5 submitPaymentData()

Depending on the layout of your payment page, you might wish to use your own submit button for the payment form rather than use the one shown in the iframe by default. If you have configured the iframe to hide its default submit button, use this method to trigger a submit of the iframe payment form.

```
var submitButton = document.getElementId("my-submit-button");
submitButton.addEventListener("click", function() {
 paymentFrame.submitPaymentData();
});
```

2.6.6 enableSubmitButton()

By default, once the iframe-generated payment form is submitted, its submit button is disabled. This is to prevent users from clicking the submit button multiple times and potentially generating multiple payment requests. Some integrations may require the submit button to be re-enabled in certain circumstances, such as a user's card declining, in which case the user would need to enter a different card and try again.

Use this method to re-enable the iframe's submit button after submission.

```
paymentFrame.enableSubmitButton();
```

2.6.7 3DSecure Methods

If you have 3DSecure (3DS) enabled on your account, there are additional methods you can use that are specific to 3DS-enabled integrations.

2.6.7.1 add3dsData(Object data)

This method can be used to pass 3DS field data to the iframe. From there it will be added to the payment form and used for 3DS authorization. The method accepts a plain object consisting of 3DS field names and values.

```
var customerEmailField = document.getElementById("customer-email-field");
customerEmailField.addEventListener("change", function() {
 paymentFrame.add3dsData({ email: customerEmailField.value });
});
```

2.6.7.2 set3dsDataResultsCallback(Function callback)

When the iframe receives 3DS data from the host page, it will send a response indicating whether it was able to successfully add the 3DS data to the payment form. This method sets a callback function that will be triggered whenever one of these responses is received. The callback will receive results data as an object with the following keys:

- `valid` - Boolean, whether or not the message was structured correctly.
- `fieldResults` - Array of objects with results for each field included in the message. Object keys:

- `fieldName` - The provided field name
- `added` - Boolean, whether the field was added to the form
- `message` - "Field added" if the field was added; otherwise, the reason why it wasn't added

```
paymentFrame.set3dsDataResultsCallback(function(response) {
 if (response.valid) {
  for (let result of response.fieldResults) {
   if (result.added) {
    console.log("Field", result.fieldName, "was added.");
   }
  }
 } else {
  console.log("3DS data was structured incorrectly");
 }
});
```

2.7 Code Example (PHP)

```
<?php

/* Values that will be needed for generating the HMAC.
 * DO NOT USE THE BELOW, JUST EXAMPLES. NOT REAL. Replace with proper values. */
$host_domain = "https://your.website.com";
$auth_apikey_id = "P004E346922321910";
$auth_apikey_secret = "WrRMpJBU6hGZss45Fv7GJwIFhFo7gApD2L1AblqF63A=";

$hmac_fields = [];

/* "timestamp", "domain", "sequence", and "auth_apikey_id" are the required HMAC fields. */

/* Current Unix timestamp */
$hmac_fields["timestamp"] = time();

/* Domain of the website that will host the iframe */
$hmac_fields["domain"] = $host_domain;

/* Merchant-specified alphanumeric value for tracking/verification purposes.
 * In production this should be dynamically generated.
 */
$hmac_fields["sequence"] = "abc123";

/* Merchant Profile API Key ID that will be used to request the iframe
 * and generate the ticket
 */
$hmac_fields["auth_apikey_id"] = $auth_apikey_id;

/* Optional field. This is the URL of the CSS file that will be used to style the
   iframe's contents. */
$hmac_fields["css-url"] = $host_domain . "/css/iframe.css";

/* Optional field. This will direct Monetra to generate the form with separate
 * select elements for the expiration date month and year, rather using than
 * a single text element for the expiration date.
 */
$hmac_fields["expdate-format"] = "separate-selects";
```

```php

/* Concatenate all of the defined HMAC fields into a string with no delimiters */
$data_to_hash = implode("", $hmac_fields);

/* Generate the HMAC, using the Monetra Merchant User's password as the key */
$hmac = hash_hmac('sha256', $data_to_hash, $auth_apikey_secret);

/* Assemble a string containing the "data-" attributes for the iframe element.
 * This will consist of the HMAC itself and all of the fields included in the HMAC.
 */
$iframe_attributes = [
 'data-hmac-hmacsha256="' . $hmac . '"'
];
foreach ($hmac_fields as $key => $value) {
 $iframe_attributes[] = 'data-hmac-' . $key . '="' . $value . '"';
}
$iframe_attribute_string = implode(" ", $iframe_attributes);

/* Render the payment page HTML. */
?>
<!DOCTYPE html>
<html>
 <head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Example Shopping Site</title>
  <link rel="stylesheet" type="text/css" href="./css/host.css" />
 </head>
 <body>
  <main>
   <h1>Checkout Page</h1>
   <p>
    Please fill out your payment information below.
   </p>
   <iframe id="myPaymentFrameId" <?php echo $iframe_attribute_string; ?>></iframe>
  </main>
  <!-- Load the Javascript file containing the PaymentFrame helper object -->
  <script src="https://test.transafe.com/PaymentFrame/PaymentFrame.js"></script>
  <script>

   /* Instantiate the PaymentFrame object. The constructor accepts two parameters:
    * (iframeElementId) The ID of the iframe element on your page that will
    * contain the PaymentFrame (iframeURL) The URL of the payment server you are
    * using to generate the PaymentFrame In this case, the iframe element has
    * an ID of "myPaymentFrameId", and our payment server is https://test.transafe.com.
    */
   var paymentFrame = new PaymentFrame(
    "myPaymentFrameId",
    "https://test.transafe.com"
   );

   /* You can use the "setPaymentSubmittedCallback" method of the PaymentFrame
    * object to set a callback function that will be executed once the payment
    * form has been submitted. This function will receive a "response" object
    * containing details about the payment form submission. This won't include
    * any sensitive data.
    */
   paymentFrame.setPaymentSubmittedCallback(function(response) {
    if (response.code === 'AUTH') {
```

```
 96       /* If the response code is "AUTH" (meaning the ticket request was
 97        * successful), the response object will contain the CardShield ticket,
 98        * which can be used in place of card data for the payment
 99        * transaction. At this point, you would use the ticket to continue
100        * your checkout/payment flow.
101        */
102       console.log("The CardShield ticket is " + response.ticket);
103      } else {
104       /* If the response code is "DENY", there was a problem generating the
105        * ticket. In this case, the response object will contain a "verbiage"
106        * property with a brief error message.
107        */
108       console.error(response.verbiage)
109      }
110     });
111
112     /* The "request" method requests a payment form to be loaded into your
113      * iframe from the payment server. Calling this method is the last step to
114      * rendering the PaymentFrame.
115      */
116     paymentFrame.request();
117
118    </script>
119   </body>
120  </html>
121
```