**Chase Orbital Hosted Payment Form Integration Guide**

**Hosted Payment Form Basics**

The Chase Orbital Hosted Payment For, or HPF, using PayFrame technology is designed to allow you to remove all payment interaction from your application while still maintaining the look, feel and flow of your original application. Unlike other hosted offerings, the HPF allows you to maintain complete control over all user interaction with the payment gateway, including submission, error handling and completion.

The Chase Orbital HPF runs on top of the Chase Orbital Transaction Platform and compliments a number of the existing APIs. Currently, the following APIs are supported in HPF mode:

* NewOrder
* AccountUpdater
* Profile
* Level II/Level III Elements

Additional information about these API calls can be found at the [Chase Orbital Documentation Site.](http://www.chasepaymentech.com/)

Initiating a HPF transaction has two steps. The first step is a call to the **HPF Primer** function. This call extends the standard Chase Orbital API and returns a unique transaction identifier. After the identifier has been appended to a meta-tag on the application's payment page, the payment form is invoked by setting the address of an iFrame to the Merchant Form URL. The user interacts with the Chase Orbital Payment Form directly, completely removing your application from any PCI compliance scope.

As the user interacts with the payment form, notification messages will be sent to your payment page via JavaScript. These messages invoke a number of JavaScript functions that you implement to your specifications. The functions are:

* hpfComplete( message ): invoked when the user has successfully processed payment.
* hpfCancel( ): invoked when the user selects the 'Cancel' button.
* hpfError( message ): invoked when there has been a problem processing payment.
* hpfWhat( whatCode ): invoked when a user requests additional information, such as What's CVV or What's Routing.
* hpfException( message ): invoked when a system exception occurs.

After the user has completed the payment, the sanitized transaction response will be returned to your server using a standard HTTP POST . This transaction return will be sanitized and all data within the scope of PCI compliance will be removed.

**Primer Call**

The Primer function is the initial call to the HPF and a key element of PayPlum's patent pending technology. It can be accessed here:

https://chase.payplum.com/primer

The Primer accepts a standard POST URL consisting of the original Chase Orbital request, plus an additional element for HPF control. The response will contain a single QuickResp type containing a 'one-time' use GUID. The ProcStatus will be 0 and the StatusMsg field will contain the transaction GUID.

Once the Primer has been successfully invoked, the GUID cannot be reused and will be discarded by the HPF. The Primer uses the same API as the general integration call with a few small changes. The Chase Request payment element must include an additional HostedPayFrame element. This element includes four parameters that control the request:

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| ReturnURL | The URL where the sanitized payment response will be POST'd. *NOTE: If the protocol is not included, will default to https* |
| CSSUrl | The URL where the form will pull in the CSS |
| FormType | The specific type of form to render. Currently, only CREDIT is supported. |
| RefererIP | Optional. This parameter is the IP of the referer that invoked the form. If submitted, it is used in the PayPlum Main-in-theMiddle Detection algorithms. |

When sending the request to PayPlum, replace the AccountNum and Exp fields with the 0000 placeholder. The fields will be set by the PayFrame. For an example NewOrder request with the included HostedPayFrameType, [click here.](http://docs.payplum.com/chase/new_order.xml)

In addition to the above element, the Primer supports the standard Chase Orbital API for the given operation. When the Primer transaction has been successfully generated and processed, it will return a QuickResp element containing the specific GUID of the transaction:

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<Response>

<QuickResp>

<ProcStatus>0</ProcStatus>

<StatusMsg>SDLKFJ2LO4I6J3LKFNP43O8JT348JFP35498GJ34:1</StatusMsg>

</QuickResp>

</Response>

**Invoking the Form**

After successfully receiving the GUID from the Primer step, it is possible to invoke the payment form. In order to successfully inject the form into your payment page, you must execute the following steps:

1. Add a meta tag to the page containing the GUID. This can be accomplished by adding the following tag to the page BEFORE the iFrame element:

<meta name="hpfGuid" content="SDLKFJ2LO4I6J3LKFNP43O8JT348JFP35498GJ34:1" />

1. Designate an iFrame to hold the form. This form must have the id 'hpfFrame'. For the initial rendering of the page, it is necessary to set the iFrame to a page that explains that the payment form is loading. The merchant is responsible for styling the form's border and other associated elements. The HPF will inject the content of the form using the CSS specified in the Primer call. Sample frame:

<iframe class="iframeStyle" id="hpfFrame" src="loading.html"></iframe>

1. The final piece is adding a script tag that will invoke the payment form processor. This script does not take any elements and is simply a direct call to Chase Orbital's payment processing site.

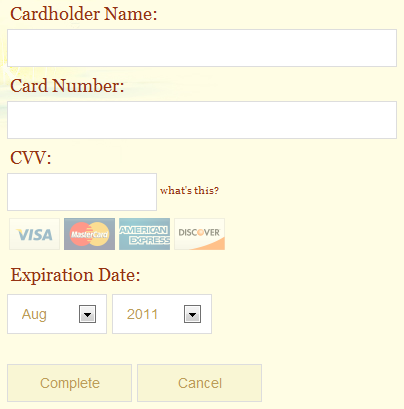
<script type="text/javascript" language="javascript" src="https://chase.payplum.com/merchant/start"></script>

The script will load, find the hpfFrame iFrame, then use the hpfGuid meta value to configure and display the form.

**NOTE:** If there are any issues with the loading of the form or with the JavaScript functions necessary to drive the payment, all errors will be logged into the JavaScript console. This is available in Chrome and Safari, and available in FireFox and IE using Firebug.

**Styling the Chase Orbital Form**

Almost every attribute of the form can be styled using CSS. To find a complete list of the available styles and their descriptions, please refer to the full documentation suite. A sample CSS file, which shows the basic format of the class names and is used to format the demo HPF form, can be found [here](http://docs.payplum.com/samples/hpf.css).

**  
Sample HPF Form using** [**hpf.css**](http://docs.payplum.com/samples/hpf.css)

**NOTE:** It is important to remember that this CSS file will be pulled into an external iFrame and be completely isolated from any other assets on your site. Make sure to change all relative links to absolute URLs. Also, remember that this form will be rendered under SSL. In order to create a seamless experience for the customer, it's **strongly** recommend that the CSS and associated elements be accessed under SSL as well.

**JavaScript Callback Functions**

As the user interacts with the form, the HPF will report the status of the transaction back to the merchant's page using a number of JavaScript callback functions. These functions are implemented by the merchant and can be customized to do whatever action the merchant would normally do when processing payments natively. The three payment functions are passed a "message" parameter. This parameter is a piece of JSON that has a number of details about the status of the transaction. The message looks like:

Example Message:

{

defaultMessage : "The card was declined.",

errorCode : 23,

subCode : 0,

traceMessage : "Credit Card Number not specified",

type : 1

}

The parameters are described below. For more information, please refer to the complete documentation.

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| defaultMessage | This is the default message that is returned from the HPF and is presentable to the user. It will be well formed and designed to inform the customer. |
| errorCode | A numeric code assigned to this particular error. Identical to the error codes returned from the standard integration |
| subCode | Depending on the message, this numeric code might signify some other sort of condition, typically 0 |
| traceMessage | The technical explanation returned from the standard integration. Typically, it is not user presentable. |
| type | Message type. Not typically needed by merchant functions. |

While onComplete() , onError() and onException() are straightforward in their function, the doWhat() function needs more explanation. On the payment forms, there are number of elements that the user can click if there is a question about the element. Such elements are things like CVV fields on credit card forms, routing numbers on ACH forms, etc.

In order to make the form feel as native as possible, the merchant needs to display the explanation about those elements. The doWhat() function is called with a numeric code. The code describes the element that the user clicked. The codes are as follows:

|  |  |
| --- | --- |
| **Code** | **Clicked Element** |
| 0 | Account Number on an ACH Form |
| 1 | CVV "what's this?" link on a credit card form. |
| 2 | Routing box/symbol on an ACH Form |

Sample implementations of these functions are included [here](http://docs.payplum.com/samples/hpf.js). Please note that in order for the functions to work, the merchant will need to adapt various pieces of the code to match the implementation on the Merchant's site.

**NOTE:** if onException() is not implemented, the system will log to the JavaScript console.

**Server Side Return**

After completing a payment, the Chase Orbital HPF will send a sanitized version of the response to the location that was sent in the primer as the return\_url parameter. Here, the merchant can execute whatever server-side functionality necessary, such as updating a database, marking an order as shipped and sending confirmation emails. The return will not contain information subject to PCI compliance scope and it is safe for the merchant to use the resulting information however they see fit.

**NOTE: It is very important** that the merchant have this service up and running when using the HPF. If the HPF attempts to send the response and receives an error, the HPF will assume that there was a terminal failure on the merchant's server and **void** the transaction, stopping the customer's execution and sending an exception message to the customer.

**One Page Checkout**

One of the unique features of PayPlum's PayFrame is the ability for the form to be triggered from an external source. This behavior is desirable in a number of scenarios, such as when the payment form is not the last step of the process, AVS verification is requested or if the merchant wishes to dramatically change the look of the buttons. Any element on the merchant's site can be used to trigger the form through a function call. When triggering the form, there are two calls that can be used. The **first** is an execution call while the **second** programmatically pushes details from the merchant's site into the transaction flow itself.

The first call is hpfSetParameter( key, value ) . Calling this function will push a single key/value pair into the transaction flow. For Chase Orbital, the possible keys and values appear below.

**NOTE:** It is important to note that setting these values programmatically will overwrite the values sent in the primer call.

**AVS Details Block**

The first group of values is the name block. If any one of the name fields are submitted, the values will overwrite the name block in the CardInfo.BillingAddress node as well as the TransactionInfo.CustomerInfo node.

|  |  |
| --- | --- |
| **Key** | **Value** |
| AVSName | The customer's name. |
| AVSzip | The customer's zip code for AVS checking. |
| AVSAddress1 | The AVS Address 1 field. |
| AVSAddress2 | The AVS Address 2 field. |
| AVSCity | City used for AVS checking. |
| AVSState | The AVS State Field |
| AVSPhoneNum | Customer's AVS Phone Number |
| AVSCountryCode | The AVS Country Code |

**Form Submission**

The second call is hpfExecuteSubmission() with no arguments. This will push the values into the transaction and trigger the submission of the form. When using these functions, it is important that the PayPlum Complete and Cancel buttons be hidden. Only when the hpfExecuteSubmission() function is executed will the parameters be pushed into the transaction.

The name field, as well as the submission and cancel buttons can be hidden on the form by setting the display value of the CSS rules to none.

**Customer Profiles and Recurring Card Notes**

The profile methods of the Chase Orbital API generally work as expected with the only necessary change being the inclusion of the HostedPayFrame element as above. For a sample of the XML for a profile transaction, please [click here.](http://docs.payplum.com/chase/profile_add.xml)

**NOTE:** One important characteristic to note is that the PayFrame is **only** used for the initial capture and transport of the credit card processing in order to facilitate passing the sensitive card data to the payment processor's vault. Once a card has been tokenized and added to a customer profile, it is not necessary to run any follow-on or subsequent recurring transactions through PayPlum. You should continue to use your current integration using customer profiles for follow-on transactions exactly as you do today, as described in the Chase Orbital API documentation.