Zip Payments

SiteGenesis Integration Guide

**Version <19.1.0>**





Contents

[Features Overview 4](#_Toc2866574)

[Checkout with Zip and Handle Zip Checkout Experience Outcomes 4](#_Toc2866575)

[Control Available Zip products via min/max ranges 5](#_Toc2866576)

[Handle Zip Checkout Experience cancellation. 5](#_Toc2866577)

[Use tokens for delayed captures, order cancellations and refunds. 5](#_Toc2866578)

[Checkout with New Zip Account Registration 6](#_Toc2866579)

[Checkout with Zip Saved Payment method 6](#_Toc2866580)

[Initiate Zip checkout experience as a redirect to a new page or a lightbox within the same page. 6](#_Toc2866581)

[Advertise Zip support via Zip marketing features 6](#_Toc2866582)

[System Extensions 8](#_Toc2866583)

[Payment Processors 8](#_Toc2866584)

[ZipMoney 8](#_Toc2866585)

[ZipPay 8](#_Toc2866586)

[Payment Methods 10](#_Toc2866587)

[Zip Money 10](#_Toc2866588)

[Zip Pay 10](#_Toc2866589)

[System Object Type Extensions 11](#_Toc2866590)

[SitePreferences 11](#_Toc2866591)

[CustomerPaymentInstrument 11](#_Toc2866592)

[Order 11](#_Toc2866593)

[OrderAddress 11](#_Toc2866594)

[OrderPaymentInstrument 12](#_Toc2866595)

[PaymentTransaction 12](#_Toc2866596)

[Profile 12](#_Toc2866597)

[Services 15](#_Toc2866598)

[Content Slots 16](#_Toc2866599)

[Installation 17](#_Toc2866600)

[Archive Contents 17](#_Toc2866601)

[Archive Preparation 17](#_Toc2866602)

[Cartridge Installation 17](#_Toc2866603)

[Configuration 19](#_Toc2866604)

[Configure Zip Product Features 19](#_Toc2866605)

[Configure Allowed Countries 19](#_Toc2866606)

[Configure Zip Marketing Features 19](#_Toc2866607)

[Configure Marketing Key 20](#_Toc2866608)

[Enable/Disable Zip Marketing Widget 20](#_Toc2866609)

[Configure Zip Footer Link Text 20](#_Toc2866610)

[Configure Zip Footer Logo 20](#_Toc2866611)

# General Information

## Components Overview

### Zip Client

This component is used to connect and work with Zip API.

Each request and corresponding response made with the Zip Client is logged in a custom error log file.

The cartridge uses the following API functions:

* Checkout - Create a Checkout – when a customer clicks “Continue with Zip” button.
* Checkout - Retrieve a Checkout – when a customer completes a Zip Checkout Experience with any status.
* Charge – Create a Charge – when payment is authorized on approved orders.
* Charge – Capture a Charge – when authorizing a payment (with automatic capture) or when executing a manual capture via the back-end hooks (refer to “Customizations and Extensions”).
* Charge – Cancel a Charge – when executing an order cancellation via the back-end hooks.
* Refund – Create a Refund – when executing a payment refund via the back-end hooks.
* Token – Create a Token – when handling approved order and tokenization is required (turned on by the merchant).

### Zip Processor

This component is used to handle payments including authorizations, captures and refunds. Zip allows partial captures and partial refunds as well.

### Zip Checkout

This component is used to extend and customize the SFRA checkout with additional features to allow checking out using Zip Checkout Experience.

Two types of errors are handled when attempting to checkout with Zip:

* General, technical errors – due to unsuccessful order creation, local server issues.
* Zip Checkout errors – either due to unsuccessful connection with Zip API or bad parameters passed to it.

### Zip Checkout Experience

This component is provided by Zip when the customer clicks “Continue with Zip” button on last checkout step. The Zip Checkout Experience has two modes of operation:

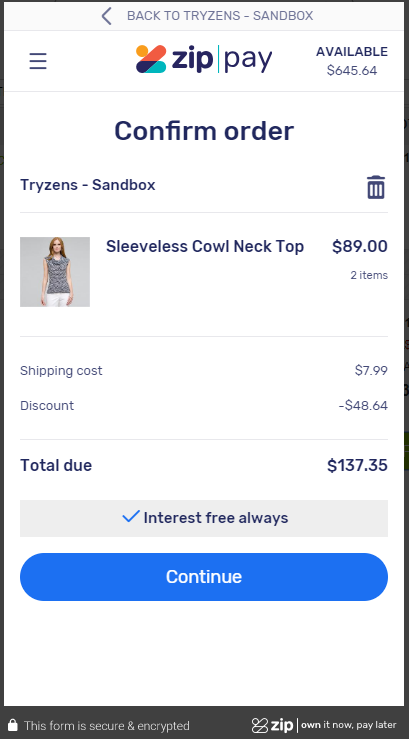
* Lightbox – the Zip Checkout Experience opens within an iframe.
* Standalone – the Zip Checkout Experience opens as a stand-alone page.

No matter of the operation mode chosen by the merchant, the customer can:

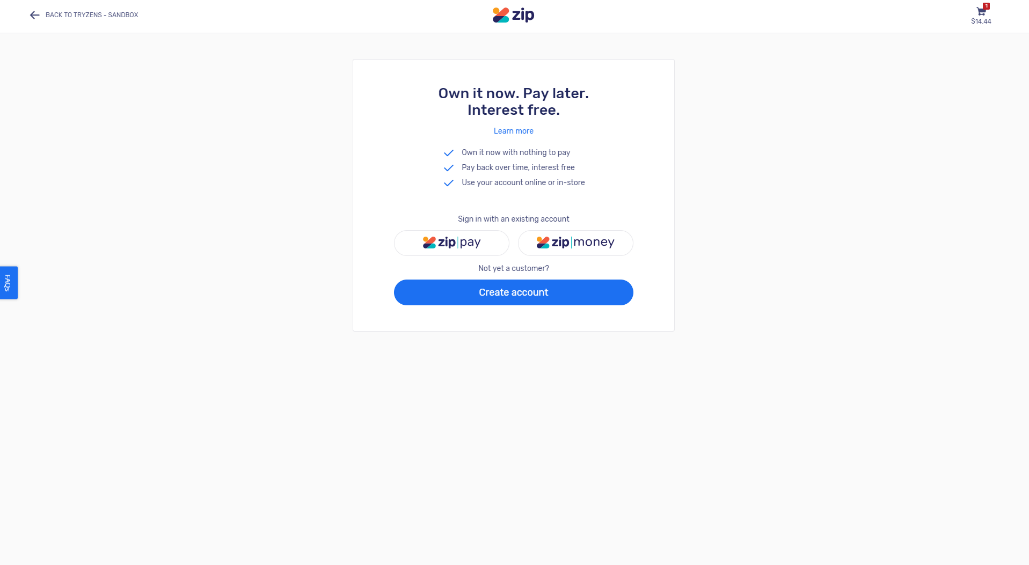
* Login with an existing ZipMoney or ZipPay account to complete the order.
* Request a new customer account with ZipMoney or ZipPay.

When the customer requests a new account, there are three possibilities:

* Immediate Approval - the account gets approved immediately. The customer is then taken back to a “Thank You” page. The customer also receives a confirmation email of their completed order.
* Immediate Rejection – the account gets rejected immediately. The customer can attempt to create a new account or cancel the order.
* Referal – the account is pending approval/rejection. In such a case, the customer gets redirected back to a “Thank You” page. An additional message appears to notify the customer that the order is pending and additional steps are required. Later on, the customer may receive either an approval in which there would be a link to complete the order (in case of approval). Refer to “Immediate Approval” above.



Zip Checkout Experience, Lightbox Mode



Zip Checkout Experience, Stand-Alone Page

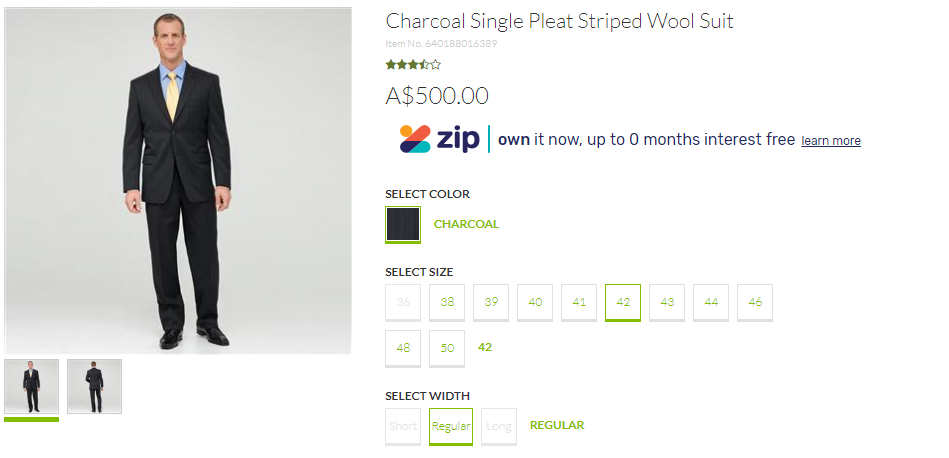
Please refer to <https://zip-online.api-docs.io/v1/checkout-and-registration/overview-use-cases> for more information on testing the Zip Checkout Experience: test credit card numbers, phone number, etc.

### Zip Payment Method

This component is used to show Zip as a payment method in checkout and allow customers to choose Zip as a payment method.

### Zip Marketing

This component is used to display marketing features including landing page, widgets, etc.



## Features Overview

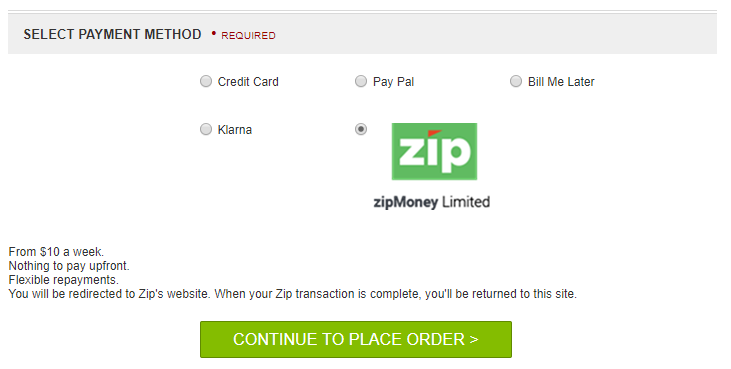
The integration described in this document supports the following features:

* Checkout with Zip and Handle Zip Checkout Experience Outcomes
* Control Available Zip products via min/max ranges.
* Use tokens for delayed captures, order cancellations and refunds.
* Checkout with New Zip Account Registration
* Handle Zip Checkout Experience cancellation.
* Handle Order Denial/Recoverable Errors.
* Checkout with Zip Saved Payment method
* Initiate Zip checkout experience as a redirect to a new page or a lightbox within the same page.
* Advertise Zip support via Zip marketing features

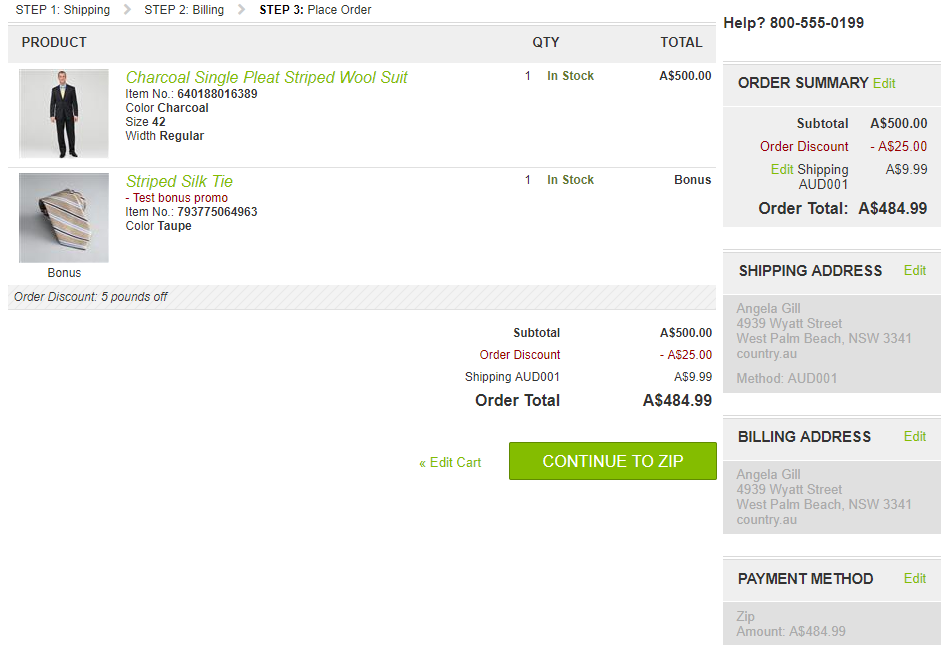
This section describes all the features listed above in detail.

### Checkout with Zip and Handle Zip Checkout Experience Outcomes

A customer visiting the merchant’s website may attempt to checkout with one of the available Zip products: ZipPay or ZipMoney by completing an enhanced SFRA checkout. On the payment checkout stage, the customer selects Zip as a payment method:



The order review stage has been enhanced with a Zip-specific button “Continue to Zip”, which when clicked will attempt to create and initiate a new Zip Checkout Experience and create a new order within the merchant’s system with status CREATED.



In case of order approval, the integration will attempt to place the order which will change the order’s status to NEW and authorize a new payment by creating a new charge via Zip API. If the merchant has enabled automatic capture, the payment amount associated with the newly created charge is also captured.

If not, the integration will redirect the customer back to the checkout’s order review stage and show a standard error message. The customer may then attempt to complete a new Zip Checkout as the “Continue to Zip” button is going to be enabled.

### Control Available Zip products via min/max ranges

The merchant can control Zip applicability via per-currency min/max range.

### Handle Zip Checkout Experience cancellation.

A customer can cancel the Zip Checkout Experience by clicking a link within. If the merchant has enabled lightbox, it will close. If not, the customer is redirected back to the SFRA checkout’s order review stage. Either way, the customer can attempt a new Zip Checkout Experience on the same page.

### Use tokens for delayed captures, order cancellations and refunds.

The merchant can activate tokenization which allows the system to keep a token within a successfully created order. Later on, charges can be created, and payment amounts may be captured or refunded without customer’s involvement using the order’s token. Note that the created token does not expire.

The basic checkout creates a new checkout experience which later, when handling checkout completion/cancellation redirect, is used to create charges. This “checkout with tokenization” is a type of checkout which uses a token to create charges. The token is available for further captures and refunds for an order. The merchant can activate tokenization for each one of the Zip products to allow this payment flow.

### Checkout with New Zip Account Registration

A customer’s registration request may be approved within the same customer session (in a matter of seconds) or, it may require a manual approval. In case of manual approval, a Zip Agent will review the registration request and either approve or deny it. If the request has been approved, an email is sent to the customer with a link to finish the checkout process.

### Checkout with Zip Saved Payment method

This use case is applicable only for logged in customers. An additional checkbox is available when a Zip payment method is selected “Save payment method for future use”. The customer continues with checkout as usual. If the order is approved as a result of completing the Zip checkout experience, a new token is going to be requested via the ZIP API and saved to the customer’s profile.

Later on, the token is used anytime the customer attempts to place an order, bypassing the need for a new Zip checkout experience altogether. On the final checkout stage, the button will read “Place Order”.

The token will be available in “My Profile” section where customers can remove the token should they want to discontinue using it.

### Initiate Zip checkout experience as a redirect to a new page or a lightbox within the same page.

The merchant can choose if the Zip Checkout Experience is to be loaded within an in-page lightbox. If the lightbox is not enabled, the Zip Checkout Experience is loaded via a page redirect instead.

### Advertise Zip support via Zip marketing features

Merchant can enable/disable the following Zip marketing features:

* Product Description Page marketing link which, when clicked, opens the Zip Marketing Widget Popup.
* Cart Page marketing link which, when clicked, opens the Zip Marketing Widget Popup.
* Footer link which leads to an external Zip Landing Page loaded inline.
* Footer icon signifying website Zip payments support.

### Hook capture, refund and cancel

Merchant can programmatically hook to capture, refund and cancel charge. To do that, the merchant needs to disable automatic capture. When automatic capture is disabled, the charges are just authorized. Later, the merchant can use the hooks to trigger multiple captures of a single charge (partial capturing), refunds and charge cancellation.

# System Extensions

## Payment Processors

The system has been extended with a new payment processor: **Zip.**

**Settings are accessible here: “**Merchant Tools – Ordering – Payment Processors – Zip – Settings**”.**

### Zip

|  |  |  |
| --- | --- | --- |
| **Setting** | **Value** | **Description** |
| Lightbox  (zipLightbox) | On/Off | If turned on, ZIP will open in lightbox; false - ZIP will simply redirect to URL in the same page.  Default: Off |
| Tokenization  (zipPayTokenization) | On/Off | Enable/disable tokenization for this processor.  Default: On |
| API Key  (zipPayApiKey) | String | API Key for this payment processor. |
| Automatic Capture  (zipPayAutoCapture) | On/Off | Enable/disable automatic captures of charges for this processor.  Default: On |

## Payment Methods

A payment method has been added: **Zip**, linked to the payment processor described in previous section.

### Zip

|  |  |  |
| --- | --- | --- |
| **Setting** | **Value** | **Description** |
| Payment Processor | Zip |  |
| Countries | AU, NZ | This payment method is applicable for Australia and New Zealand by default.  Selected Countries: AU, NZ. |
| Min/Max Payment Ranges |  | The merchant can set a specific range for which Zip is applicable as a payment method. |

## System Object Type Extensions

### SitePreferences

Custom preferences have been added to a new “**Zip**” group. Those preferences apply to the zip integration (they are not per-processor).

|  |  |
| --- | --- |
| **ID** | **Description** |
| Zip Marketing Environment (zipMarketingEnvironment) | Enumeration of names of environments to be passed on when loading the zip marketing widget.  Values: sandbox (selected by default), production. |
| Zip Marketing Key  (zipMarketingKey) |  |

### CustomerPaymentInstrument

|  |  |
| --- | --- |
| **ID** | **Description** |
| Zip Token  (zipToken) | Used when tokenization has been turned on and an order has been approved. Before placing the order, a new token is created and saved in this attribute for the specific order. Later, the token may be used to create charges for this specific order. |

### Order

The following custom attributes have been added to a “Zip” attribute group:

|  |  |
| --- | --- |
| **ID** | **Description** |
| Zip Token  (zipToken) | Used when tokenization has been turned on and an order has been approved. Before placing the order, a new token is created and saved in this attribute for the specific order. Later, the token may be used to create charges for this specific order. |
| Zip Require Approval  (zipRequireApproval) | Flag to indicate if the Zip order is linked to a customer account which requires approval. |

### OrderAddress

AU and NZ are added as allowed countries in this system object type so they appear as valid values for selection within the payment method’s configuration settings.

|  |  |
| --- | --- |
| **ID** | **Description** |
| Country (countryCode) | Allowed countries for billing address (see the Countries setting within the ZIP Payments payment method).  Values:   |  |  | | --- | --- | | United States | US | | France | FR | | Italy | IT | | Japan | JP | | China | CN | | United Kingdom | GB | | Australia | AU | | New Zealand | NZ | |

### OrderPaymentInstrument

The following custom attributes have been added to a “Zip” attribute group:

|  |  |
| --- | --- |
| **ID** | **Description** |
| Zip Email  (zipEmail) | This field holds the Zip email, which a customer entered when checking out with one of the available Zip payment methods. The field is shown in the order’s payment tab in BM. |
| Zip Phone  (zipPhone) | This field holds the Zip phone, which a customer entered when checking out with one of the available Zip payment methods. The field is shown in the order’s payment tab in BM. |

### PaymentTransaction

The following custom attributes have been added to a “Zip” attribute group:

|  |  |
| --- | --- |
| **ID** | **Description** |
| Zip Capture Id  (zipCaptureId) | Holding Zip captured id. |
| Zip Refund Result  (zipRefundResult) | List of refund response results after refund request. |
| zipCaptureResult | Holding capture response result after capture request. |
| ZipCancelResult | Holding cancel response result after cancel request. |
| isZipAutoCapture | Holds the state of AutoCapture param in the capture request. |

### Profile

The following custom attributes have been added to a “Zip” attribute group:

|  |  |
| --- | --- |
| **ID** | **Description** |
| Zip Token  (zipToken) | The token is saved to the customer’s profile when an order has been approved and the customer has opted to save zip payment method for future transactions. The token is used anytime the customer attempts to place an order, bypassing the need for a new Zip checkout experience altogether. The token will be available in “My Profile” section where customers can remove the token should they want to discontinue using it. |
| Zip Save Token  (zipSaveToken) | A flag programmatically set if the customer wants to save a Zip token. |

## Services

An HTTP service has been added: “zip.http.defaultendpoint” with a profile “zip.http.service” and two credentials: “zip.http.credentials.production” and “zip.http.credentials.sandbox” for each API mode respectively.

## Content Slots

Two new slots have been added:

|  |  |
| --- | --- |
| **Page** | **Slot** |
| Cart Page | zip-cart |
| PDP Page | zip-pdp |

Each one has a slot configuration added, linked to an HTML template with code that loads the Zip marketing widget.

## Content Assets

Two content assets have been added to “*SFRASharedLibrary*”.

|  |  |
| --- | --- |
| zip-declined | This message appears when a Zip payment has been declined. |
| zip-footer-link-text | Text of the footer link leading to the landing page. |

# Installation

This section describes the contents of the archive you downloaded from Salesforce as well as the way to prepare and install the archive to support payments with Zip.

## Archive Contents

The archive contains the following folders within:

* “build” – contains front-end build tools and configuration scripts. Note that, building the cartridge front-end resources is unnecessary as they have been added to the archive as well.
* “cartridges” – contains the Zip integration cartridges.
* “cartridges/int\_zippay\_sfra” – contains SFRA and system code extensions.
* “cartridges/bm\_zippay” – contains Business Manager UI BM-related system extensions to help with QA testing and validating delayed payment flows (partial/full refund and capture of charges).
* “documentation” – contains this “SFRAIntegrationGuide.docx” document which describes integration of Zip Payments with the SFRA.
* “metadata” – contains meta-data to be imported when installing the cartridge.

## Archive Preparation

Before following the installation process in this section, please copy-paste the metadata folder in a temporary directory if your choosing. Then, follow these steps to ensure the archive is prepared for site import:

1. Within the site-template/sites directory, there is the SFRA folder. Rename the folder so its name corresponds to the name of your SFRA site onto which you want the cartridge installed. For example, if you have a site called RefArch, you have to rename the SFRA folder into a new folder called RefArch in the same directory.
2. Within the site-template/Libraries directory, there is the SFRASharedLibrary folder. Copy-paste the folder into a new one whose name corresponds to the <your-library-id> content library assigned to your site.
3. Then, within the site-template/Libraries/<your-library-id> directory, open library.xml file and look to find library-id="SFRASharedLibrary" at the first opening tag. Replace that with the name of your library <your-library-id>.

Note that if you have more than one sites onto which you want the cartridge installed, you have to follow the same steps and just copy-paste the folders.

## Cartridge Installation

Please, make sure the archive contents you received corresponds to the one described in “Archive Contents” and that you have followed the steps in “Archive Preparation” first.

Cartridge installation is done via following the steps laid out in the installation plan shown below:

1. Add currencies (AUD, NZD).
2. Add Locales (AU, NZ) – the AU and NZ locales need to be configured.
3. Import cartridge metadata via site import – the system integration expert is required to import metadata via “Site Import/Export” section in the business manager. Beforehand, the “sites/SFRA” folder may need to be renamed or duplicated so configuration is applied to the right site(s), based on SFRA, within the site structure of the merchant’s system.
4. Update cartridge path – for each SFRA-enabled site which is going to support zip payments, the system integration expert needs to prepend zip cartridge name to the cartridge path. In other words, “int\_zippay\_sfra” needs to come before “app\_storefront\_base” in the path.
5. Update SiteGenesis – the base cartridges must be updated with the custom code from section “Custom Code”.

# Custom Code

## Updates on your storefont core cartridge

Please follow steps as below:

* {your\_storefront\_core}/cartridge/templates/default/components/footer/footer\_UI.isml
* {your\_storefront\_core}/cartridge/templates/default/checkout/cart/cart.isml
* {your\_storefront\_core}/cartridge/templates/default/checkout/billing/billing.isml
* {your\_storefront\_core}/cartridge/templates/default/product/components/pricing.isml
* {your\_storefront\_core}/cartridge/templates/default/checkout/billing/paymentmethods.isml
* {your\_storefront\_core}/cartridge/templates/default/checkout/summary/summary.isml
* {your\_storefront\_core}/cartridge/templates/default/checkout/confirmation/confirmation.isml
* {your\_storefront\_core}/cartridge/templates/default/components/order/orderdetailsemail.isml

### {your\_storefront\_core}/cartridge/templates/default/components/footer/footer\_UI.isml



Code updated:

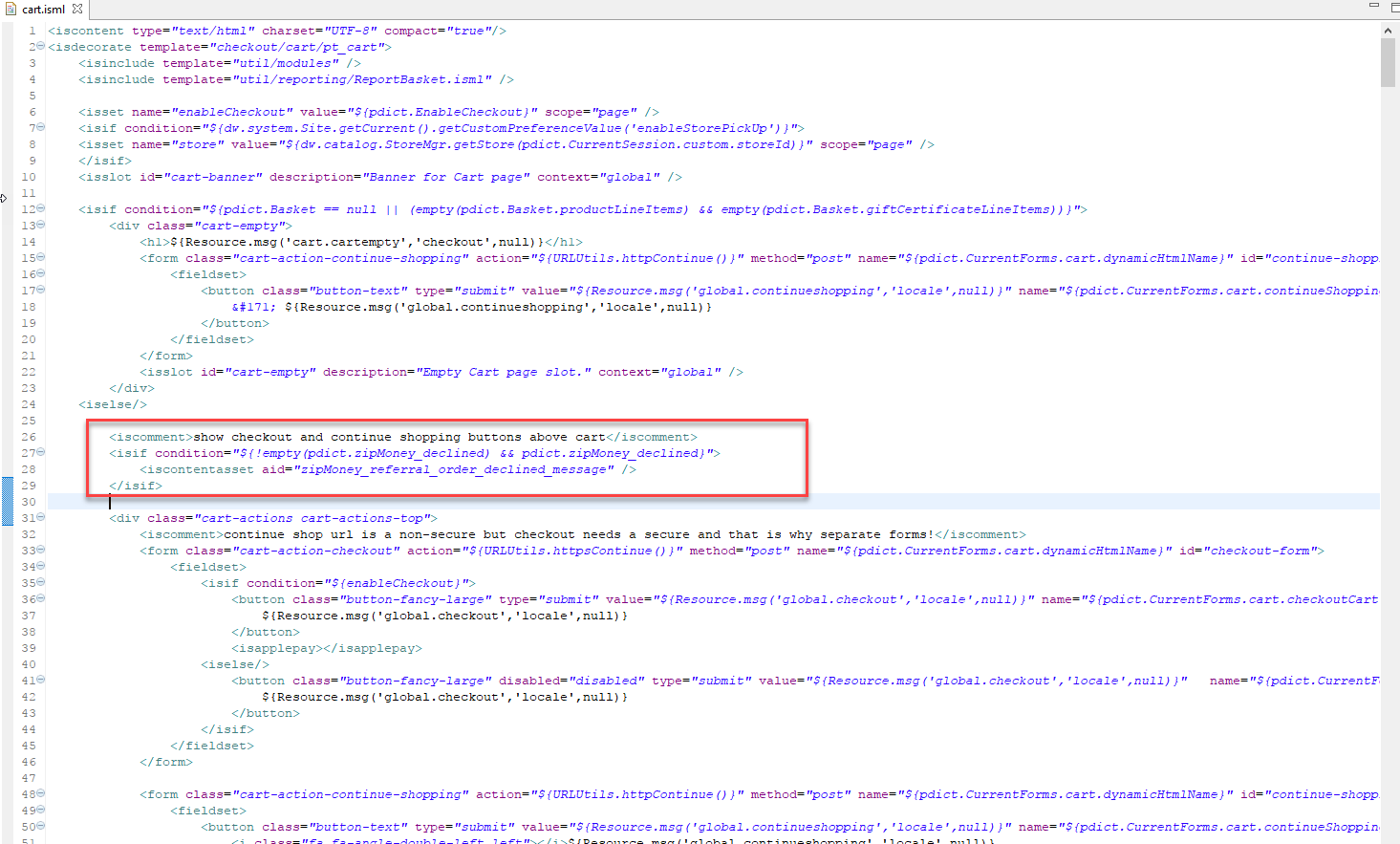
<!-- zipMoney Checkout Scripts -->

<isinclude template = "zipMoney/checkout/zipMoney\_checkout" />

<!-- zipMoney Widget API -->

<isinclude template = "zipMoney/widgets/zipMoney\_widget\_api"/>

### {your\_storefront\_core}/cartridge/templates/default/checkout/cart/cart.isml



Code updated:

<iscomment>show checkout and continue shopping buttons above cart</iscomment>

<isif condition="${!empty(pdict.zipMoney\_declined) && pdict.zipMoney\_declined}">

<iscontentasset aid="zipMoney\_referral\_order\_declined\_message" />

</isif>

### {your\_storefront\_core}/cartridge/templates/default/checkout/billing/billing.isml



Code updated:

var Transaction = require('dw/system/Transaction');

Transaction.wrap(function(){

session.privacy.zipMoneyDeclined = null;

});

### {your\_storefront\_core}/cartridge/templates/default/product/components/pricing.isml

****

**Code updated:**

<isset name="CurrencyCode" value="${session.getCurrency().getCurrencyCode()}" scope="pdict" />

<isif condition="${pdict.ProductMinPrice > 0 && pdict.ProductMaxPrice > 0}">

<div class="pricerange"><isprint value="${dw.util.StringUtils.formatMoney(dw.value.Money(pdict.ProductMinPrice, pdict.CurrencyCode))}" /> - <isprint value="${dw.util.StringUtils.formatMoney(dw.value.Money(pdict.ProductMaxPrice, pdict.CurrencyCode))}" /></div>

<iselseif condition="${pdict.ProductSetSalesPrice > pdict.ProductSetStandardPrice}">

<div class="standardprice"><isprint value="${dw.util.StringUtils.formatMoney(dw.value.Money(pdict.ProductSetStandardPrice, pdict.CurrencyCode))}" /></div>

<isset name="zipMoney\_amount" value="${pdict.ProductSetStandardPrice.value}" scope="page"/>

<iselseif condition="${pdict.ProductSetStandardPrice >= pdict.ProductSetSalesPrice}">

<div class="salesprice"><isprint value="${dw.util.StringUtils.formatMoney(dw.value.Money(pdict.ProductSetSalesPrice, pdict.CurrencyCode))}" /></div>

<isset name="zipMoney\_amount" value="${pdict.ProductSetSalesPrice.value}" scope="page"/>

</isif>

<isif condition="${ShowStandardPrice}">

<span class="price-standard">

<isif condition="${StandardPrice.valueOrNull != null && StandardPrice.valueOrNull > 0}">

<isprint value="${StandardPrice}"/><iselse/>${Resource.msg('product.pricing.noprice','product',null)}

<isset name="zipMoney\_amount" value="${StandardPrice}" scope="page"/>

</isif>

</span>

</isif>

<iscomment>Let promotional price of 0 display as is, in this case isPromoPrice is true</iscomment>

<span class="price-sales">

<isif condition="${(SalesPrice.valueOrNull != null && SalesPrice.valueOrNull > 0) || isPromoPrice}">

<isprint value="${SalesPrice}"/>

<isset name="zipMoney\_amount" value="${SalesPrice}" scope="page"/>

<iselse/>

${Resource.msg('product.pricing.noprice','product',null)}</isif>

</span>

<isscript>

var TemplateUtils = require('~/../int\_zipmoney/cartridge/scripts/zipMoney/util/template');

</isscript>

<isif condition=*"${TemplateUtils.isShowZipWidgets()}"*>

<isslot id=*"zip-pdp"* description=*"Zip PDP"* context=*"global"* />

</isif>

### {your\_storefront\_core}/cartridge/templates/default/checkout/billing/paymentmethods.isml

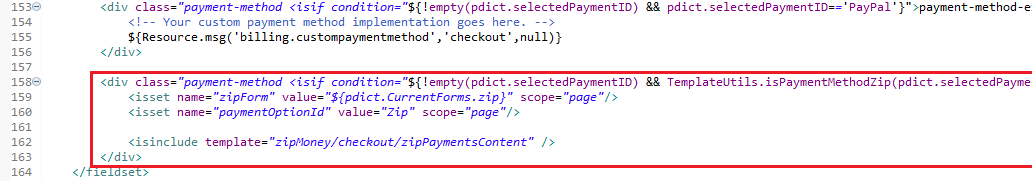


First, place the <isscript> code snippet as shown in the screenshot above:

|  |
| --- |
| <isscript>  var TemplateUtils = require('~/../int\_zipmoney/cartridge/scripts/zipMoney/util/template');  </isscript> |

Then, replace the label’s inner HTML with the following code:

|  |
| --- |
| <isif condition="${TemplateUtils.isPaymentMethodZip(paymentMethodType.label)}">  <img src="${TemplateUtils.getPaymentMethodLogoUrl(paymentMethodType.value)}" alt="${Resource.msg('setting.paymentmethid.zip', 'zipMoney', null)}">  <iselse>  <isprint value="${Resource.msg(paymentMethodType.label,'forms',null)}"/>  </isif> |



Last, the following code snipper must be inserted after the last *div* with class payment-method within the fieldset:

<div class=*"payment-method <isif condition="*${!empty(pdict.selectedPaymentID) && TemplateUtils.isPaymentMethodZip(pdict.selectedPaymentID)}">payment-method-expanded</isif>" data-method="Zip">

<isset name=*"zipForm"* value=*"${pdict.CurrentForms.zip}"* scope=*"page"*/>

<isset name=*"paymentOptionId"* value=*"Zip"* scope=*"page"*/>

<isinclude template=*"zipMoney/checkout/zipPaymentsContent"* />

</div>

### {your\_storefront\_core}/cartridge/templates/default/checkout/summary/summary.isml



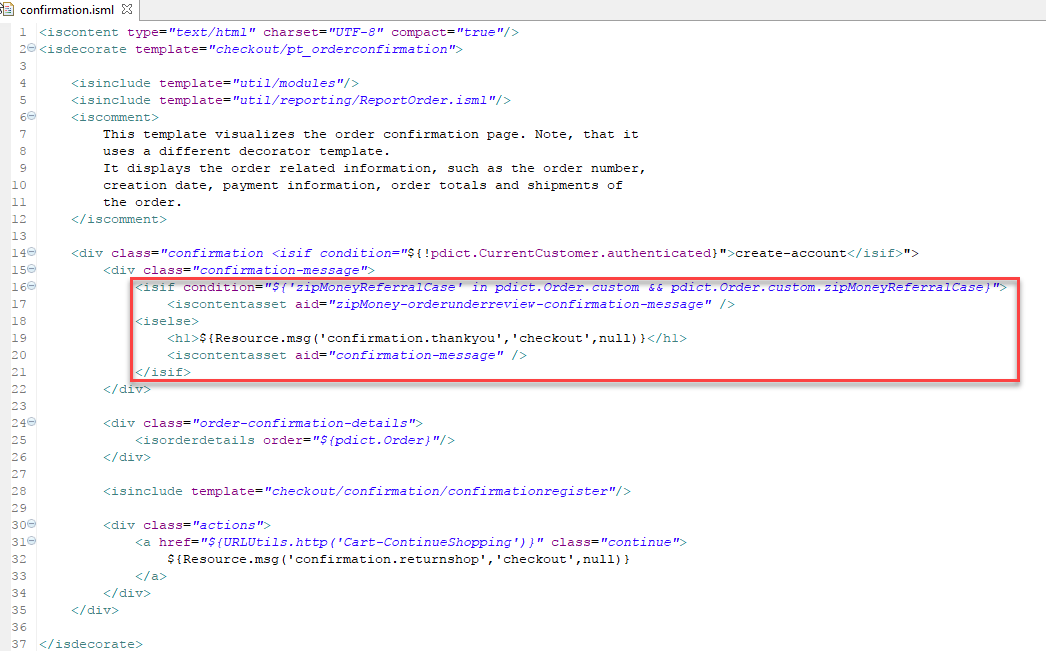
Replace the “submit-order” form opening tag with the following code:

|  |
| --- |
| <isscript>  var TemplateUtils = require("~/../int\_zipmoney/cartridge/scripts/zipMoney/util/template");  </isscript>  <form action="${TemplateUtils.getPlaceOrderFormUrl()}" method="post" class="submit-order"> |

Replace the code for the Place Order button with:

|  |
| --- |
| <isif condition="${TemplateUtils.isZipPaymentMethodSelected(pdict.Basket) && !TemplateUtils.canTokenize(pdict.Basket)}">  <button id="zip-checkout" class="button-fancy-large continue-with-zip" type="button" name="submit" value="">  ${Resource.msg('continueToZip', 'zip', null)}  </button>  <iselse>  <button class="button-fancy-large" type="submit" name="submit" value="${Resource.msg('global.submitorder','locale',null)}">  ${Resource.msg('global.submitorder','locale',null)}  </button>  </isif> |

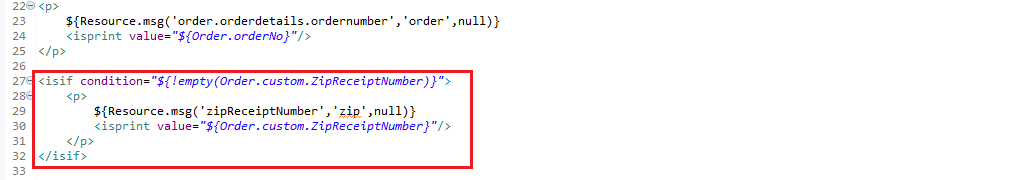
### {your\_storefront\_core}/cartridge/templates/default/checkout/confirmation/confirmation.isml



Code updated:

|  |
| --- |
| <h1>${Resource.msg('confirmation.thankyou','checkout',null)}</h1>  <isif condition="${pdict.zipRequireApproval}">  <isprint value="${Resource.msgf('info.pending.order.confirmation', 'zip', null, pdict.Order.customerEmail)}" encoding="htmlcontent" />  <iselse>  <iscontentasset aid="confirmation-message" />  </isif> |

### {your\_storefront\_core}/cartridge/templates/default/components/order/orderdetailsemail.isml



Add the following code to this template:

<isif condition="${!empty(Order.custom.ZipReceiptNumber)}">

<p>

${Resource.msg('zipReceiptNumber','zip',null)}

<isprint value="${Order.custom.ZipReceiptNumber}"/>

</p>

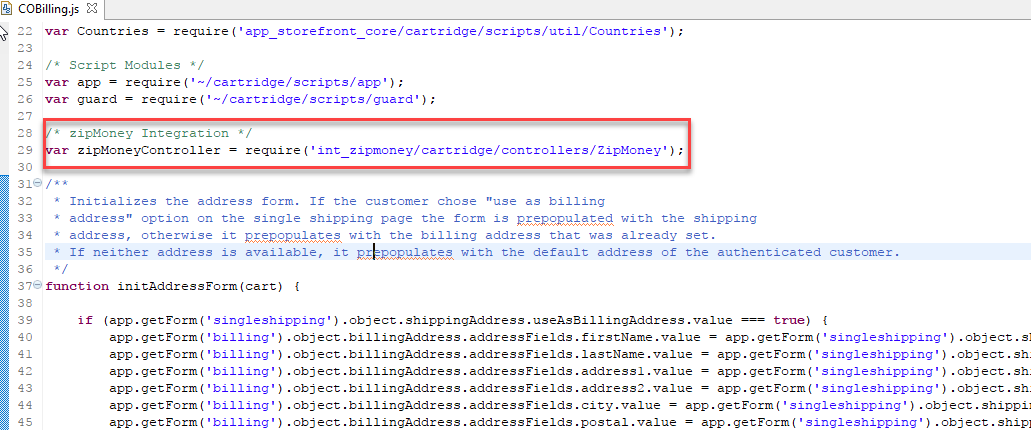
</isif>

## Updates on your storefront controller cartridge

* **{your-controller-cartridge}/cartridge/controllers/COBilling.js**
* **{your-controller-cartridge}/cartridge/controllers/Cart.js**

### ****{your-controller-cartridge}/cartridge/controllers/COBilling.js****

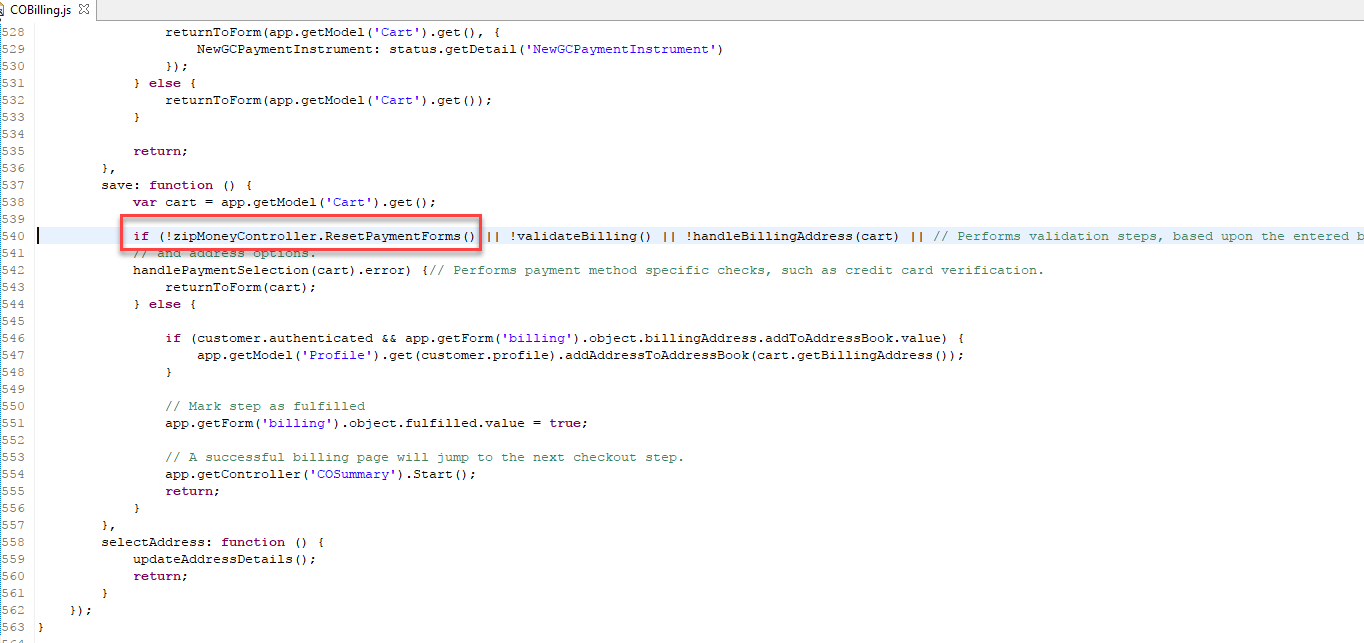
First, please add a reference of ZipMoney controller to the top of the **COBilling** controller as below.



Code updated:

|  |
| --- |
| /\* zipMoney Integration \*/  var zipMoneyController = require('int\_zipmoney/cartridge/controllers/Zip'); |

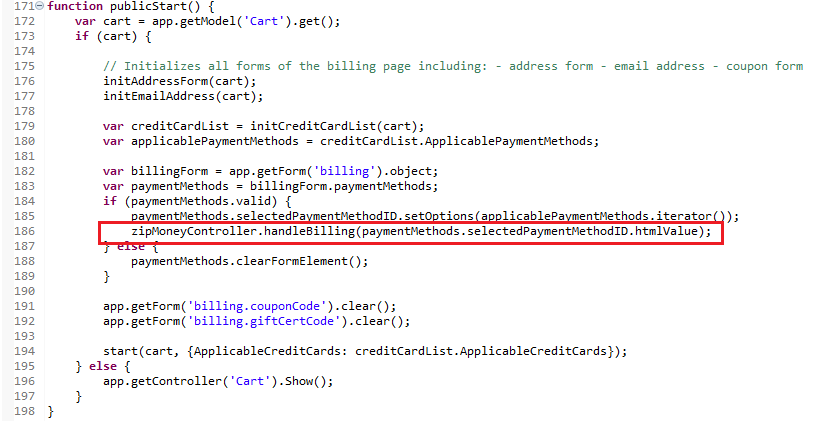
Then, update the logic code as below screenshot:



Code updated:

|  |
| --- |
| !zipMoneyController.ResetPaymentForms |

Then, update the logic of publicStart as below:

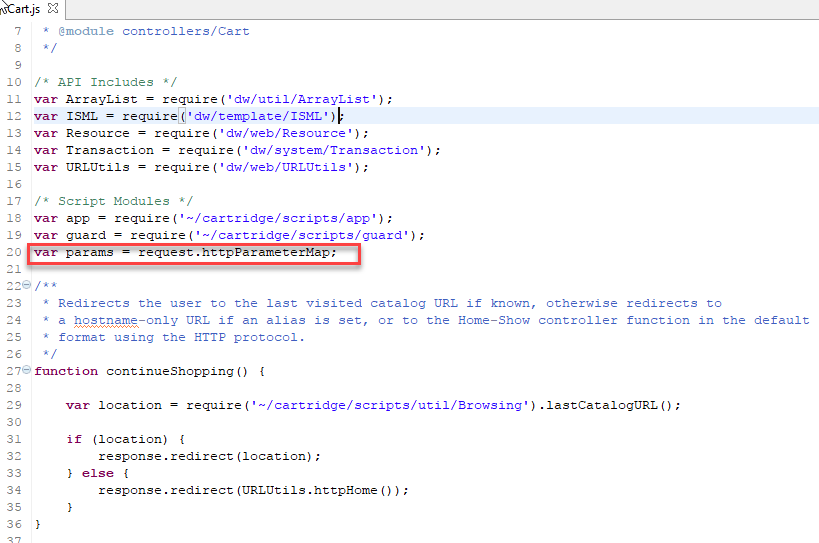


Code Updated:

|  |
| --- |
| zipMoneyController.handleBilling(paymentMethods.selectedPaymentMethodID.htmlValue); |

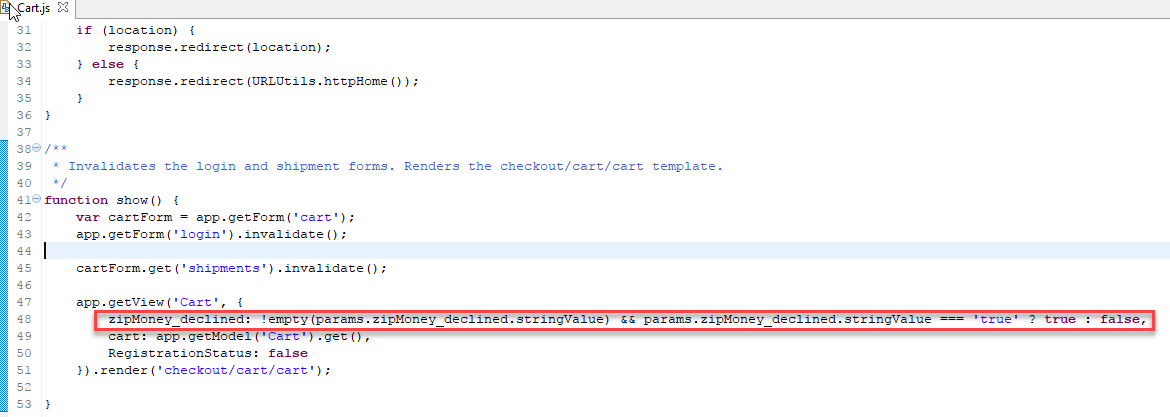
### ****{your-controller-cartridge}/cartridge/controllers/Cart.js****

First, please define a HTTP Parameter Map as below:



Code updated:

|  |
| --- |
| **var params = request.httpParameterMap;** |



Add the zipMoney\_declined parameter:

zipMoney\_declined: !empty(params.zipMoney\_declined.stringValue) && params.zipMoney\_declined.stringValue === 'true' ? true : false,

# Configuration

This section of the integration guide describes how merchants can configure the features of this integration.

## Configure Zip Payments Processor

### Enable/Disable Lightbox

To configure this feature, the merchant needs to use the setting located in “Merchant Tools – Custom Preferences – Zip – Lightbox”.

### Activate Production Mode

To configure this feature, the merchant needs to use the setting located in “Merchant Tools – Custom Preferences – Zip – Zip API Mode Production”.

(when the merchant is ready for production, the API Key also needs to be updated.)

### Set API Key

To configure this feature, the merchant needs to use the setting located in “Merchant Tools – Custom Preferences – Zip – Zip API Mode Production”.

### Enable/Disable Automatic Capture

To configure this feature, the merchant needs to use the setting located in “Merchant Tools – Custom Preferences – Zip – Automatic Capture”.

### Enable/Disable Zip Tokenization

To configure this feature, the merchant needs to use the setting located in “Merchant Tools – Custom Preferences – Zip – Zip Tokenization”.

## Configure Zip Payment Method

### Update Payment Method Logo

The Zip payment method logo can be updated.

The setting is in “Merchant Tools – Ordering – Payment Methods”. On the payment method screen, choose “Zip”. Then, use the “Image” setting and upload a new logo.

### Allowed Countries

The setting is in “Merchant Tools – Ordering – Payment Methods”. On the payment method screen, choose “Zip” and then use the “Countries” setting to configure allowed countries. Zip payment method will only be available for those countries you select.

### Min/Max range for each currency

The setting is in “Merchant Tools – Ordering – Payment Methods”. On the payment method screen, choose “Zip” and then use the “Min/Max Payment Ranges” setting for each currency.

## Configure Zip Marketing

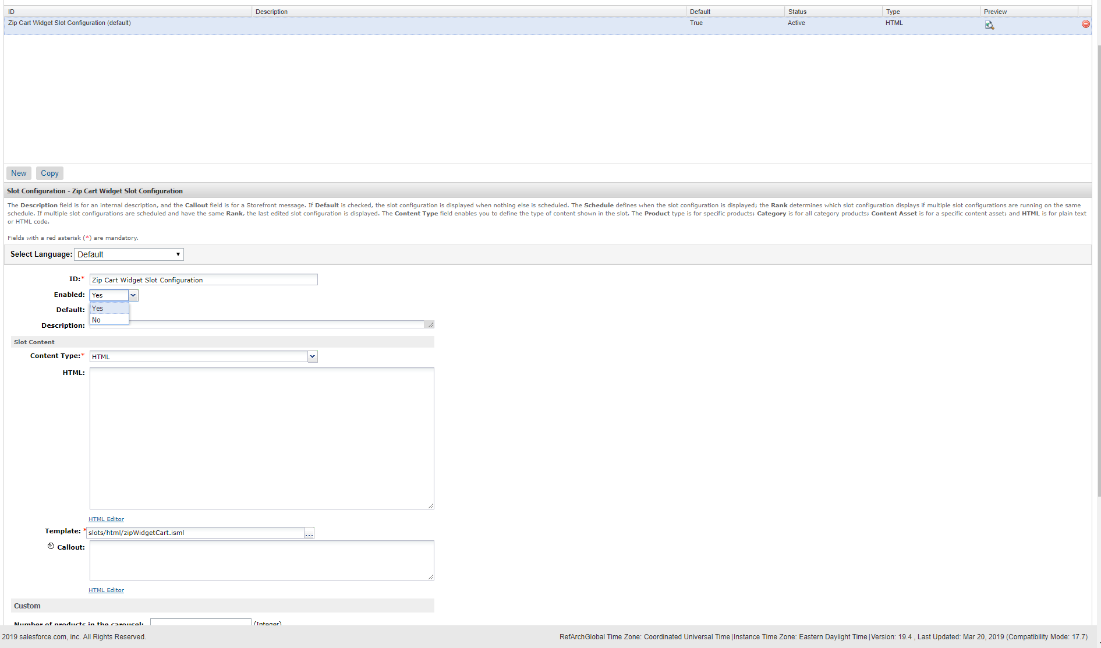
For Zip Marketing features to be available to customers, couple of things should be kept in mind:

* A marketing key should be configured (read below).
* At least one Zip payment method should be enabled from the Business Manager.

### Configure Marketing Key

The setting is within the Site’s custom preferences: “Merchant Tools – Custom Preferences”. Choose “Zip” group and within the setting is called “Zip Marketing Key”.

### Enable/Disable Zip Marketing Widget on Cart Page



The widget is implemented as a content slot “zip-cart” with one slot configuration called “Zip Cart Widget Slot Configuration”. In order to enable the widget, please use the “Enabled” setting and set to “Yes”.

### Enable/Disable Zip Marketing Widget on Product Detail Page

The widget is implemented as a content slot “zip-pdp” with one slot configuration called “Zip Product Widget Slot Configuration”. In order to enable the widget, please use the “Enabled” setting and set to “Yes”.

### Configure Zip Footer Link Text

The Zip Footer Link has been implemented as a content asset called “zip-footer-link”. The merchant may enable/disable it as well as change its appearance and text for different locales via the Business Manager. Make sure you have added the content asset following the steps in the Installation section.

### Configure Zip Footer Logo

The Zip Logo has been implemented as a static asset placed in the same folder as this integration guide named zip-footer-icon.png file.

## Configure Zip Checkout

### Configure Zip Payment Declined Message

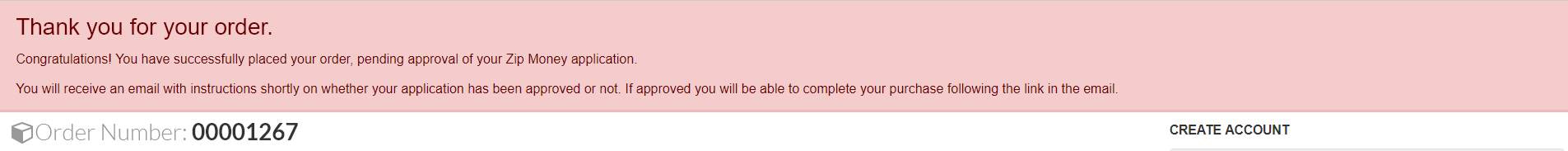
The message appearing on payment stage when Zip payment is declined is configurable via a content asset: “zip-declined”.

## Configure Thank You Page (Order Confirmation)

### Configure Zip Pending Message Custom Styling

The styling of the Zip Pending Application Message appearing on Thank You Page can be configured via the Site Preferences, Zip Preference group. The preference is called “Zip Pending Message Styles”.

The default configuration for this preference generates the following styling:

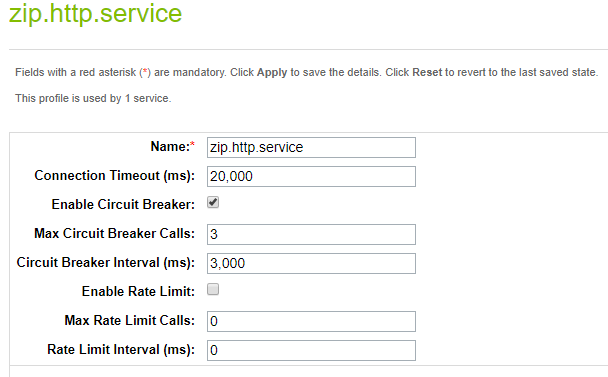


You can customize the appearance of this message box by changing the custom preference mentioned above. To change the colors, You can use an online color picker.

## Configure Zip API Client

### Configure Retries in case of General Errors

In case of general errors, the Zip API Client will retry requests according to configuration values specified in the Zip Service Profile. To access those configuration values, navigate to “Administration – Operations – Services” and then select “Profiles” tab. Choose the “zip.http.service” profile.



The **Max Circuit Breaker Calls** determine the number of attempts the Service is going to make in case of general errors. The **Circuit Breaker Interval** determine the interval in which those attempts will be made.

General errors are:

* Issues at initial connection.
  + Unknown host
  + Connection timeout
  + Connection refused
* Protocol-specific errors, such as HTTP 500 - internal server error or HTTP 503 - service unavailable.
* Exceptions explicitly thrown as part of the service call

# Customizations and Extensions

## Use Back-End Hooks To Capture and Refund Amounts

Define a hooks.json file in a cartridge with the following contents:

{

"hooks": [

{

"name": "zippay.order.processor",

"script": "./../int\_zippay\_sfra/cartridge/scripts/hooks/zip/orderprocess"

}

]

}

To implement a full capture, please use this line of code:

HookMgr.callHook('zippay.order.processor', 'CaptureFull', orderNo);

(if a partial amount has already been captured, this functionality will capture the amount not captured yet)

For partial order captures, please use this line of code:

HookMgr.callHook('zippay.order.processor', 'Capture', orderNo, amount);

(passing an amount equal to the amount not captured yet for this order will result in a full capture)

To implement a full refund, please use this line of code:

HookMgr.callHook('zippay.order.processor', 'RefundFull', orderNo, reason);

(if a partial amount has already been refunded, this functionality will refund the amount not refunded yet)

Order refunds can be implemented by using the following line of code:

HookMgr.callHook('zippay.order.processor', 'Refund', orderNo, amount, reason);

(passing an amount equal to the amount not refunded yet for this order will result in a full refund)

# Operations, Maintenance

## Availability

In case of problems with the connection to ZipMoney please contact the ZipMoney support department.

Please supply as much information as possible (Merchant account, time, order number, ZipMoney reference etc.). Also, please check the log file mentioned above.

## Support

In case of problems with the integration, missing features, etc. please contact the ZipMoney support department or your ZipMoney account manager.

## Known Issues

None.

## Release History

No releases yet.