Scalapay + SAP Commerce (Hybris)

EXTENSION INSTALLATION AND CONFIGURATION GUIDE

Last modified: March 2021

Table of contents

Purpose and Audience

This document is intended to serve as a guide to install, configure and integrate the Scalapay payment solution into an instance of SAP Commerce (Hybris) version 6.0 and above.
Main audience for this document are software developers familiar with the development and operation of the SAP Commerce platform.

Features

Scalapay SAP Commerce payment library provides a set of extensions allowing merchants to easily integrate Scalapay into the site's product, cart and checkout pages..

Key features of the library:

- Java wrapper for the Scalapay REST API
- Custom JSTL tag wrapper for the Scalapay Widget
- Backoffice UI for configuration (per BaseStore)
- Application components
- Initial setup data

Installation

Scalapay's payment add-on for SAP Commerce will be provided as a zip package. To install the library please follow these steps:

1. Copy the contents of the zip file into: **`${HYBRIS_BIN_DIR}/custom/`**

2. Add Scalapay extensions to **`config/localextensions.xml`**

```
<extension dir="${HYBRIS_BIN_DIR}/custom/scalapay"/>
<extension dir="${HYBRIS_BIN_DIR}/custom/scalapayBO"/>
```

3. Add Scalapay properties to **`config/local.properties`**

```
scalapay.application-context=scalapay-spring.xml
scalapay.url.staging=https://staging.api.scalapay.com/v2
scalapay.url.live=https://api.scalapay.com/v2
scalapay.redirect.confirm.url=/scala-pay/response
scalapay.redirect.cancel.url=/scala-pay/response
sop.post.url=https://YOUR-WEBSITE-URL-HERE/scala-pay/process
```

   If you already have a SOP post url set up, you will need to extend the logic in the handler to cater for ScalaPay, please refer to the Checkout section below.

4. Add the Scalapay dependency to your storefront extensioninfo.xml file

```
<requires-extension name="scalapay"/>
```

5. Re-build the platform

```
cd bin/platform
```

```
. ./setenv.sh

ant clean all
```

Your system will now begin to rebuild, updating the type-system (database) and installing the Scalapay software components.

Once complete you will be ready to complete the configuration and integration with the storefront.

Configuration

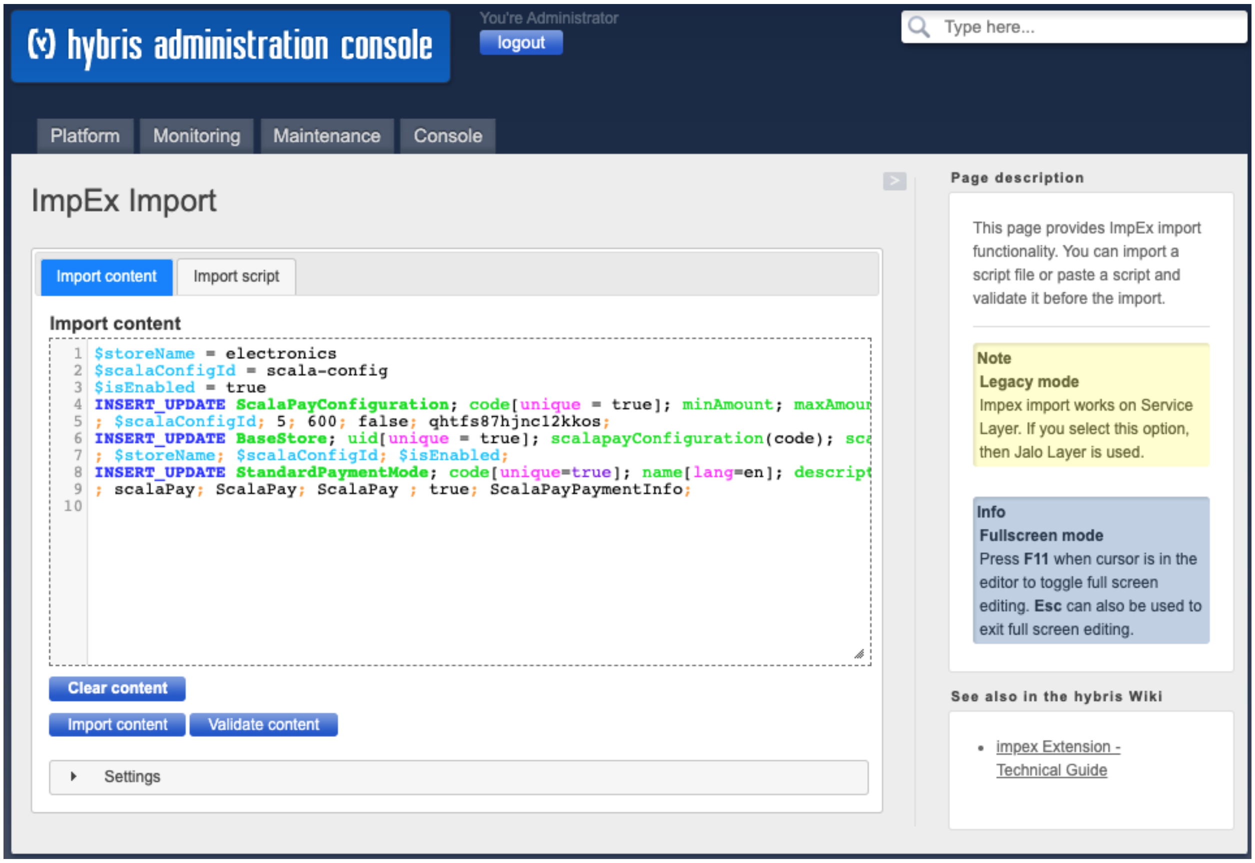
Impex

You will need to populate the system with Scalapay settings using impex. You will update the configuration using Backoffice at a later stage.

The following impex creates a standard configuration inside SAP Commerce with the staging endpoint enabled by default.
Note: you must supply the storeName that applies to your system

```
$storeName = <YOUR STORE>
$scalaConfigId = scala-config
$isEnabled = true
INSERT_UPDATE ScalaPayConfiguration; code[unique = true]; minAmount; maxAmount; liveMode;
scalapayToken_staging; styling_pdp_showWidget; styling_cart_showWidget
                                   ; $scalaConfigId     ; 5        ; 600      ; false   ;
qhtfs87hjnc12kkos    ; true                  ; true
INSERT_UPDATE BaseStore; uid[unique = true]; scalapayConfiguration(code);
scalapayEnabled;
                       ; $storeName        ; $scalaConfigId             ; $isEnabled
;
INSERT_UPDATE StandardPaymentMode; code[unique = true]; name[lang = en]; description[lang
= en]; active; paymentinfotype(code);
                                 ; Scalapay           ; Scalapay       ; Scalapay
; true  ; ScalaPayPaymentInfo  ;
```

To import the impex navigate to your Administration Console (hac) then in the menu select **Console** then **ImpEx Import**
Copy the contents of the above impex into the form and press **Import Content** - ensure you have updated the storeName to match your system



ImpEx Import

Import content

```
$storeName = electronics
$scalaConfigId = scala-config
$isEnabled = true
INSERT_UPDATE ScalaPayConfiguration; code[unique = true]; minAmount; maxAmou
; $scalaConfigId; 5; 600; false; qhtfs87hjncl2kkos;
INSERT_UPDATE BaseStore; uid[unique = true]; scalapayConfiguration(code); sca
; $storeName; $scalaConfigId; $isEnabled;
INSERT_UPDATE StandardPaymentMode; code[unique=true]; name[lang=en]; descript
; scalaPay; ScalaPay; ScalaPay ; true; ScalaPayPaymentInfo;
```

Page description

This page provides ImpEx import functionality. You can import a script file or paste a script and validate it before the import.

Note

Legacy mode

Impex import works on Service Layer. If you select this option, then Jalo Layer is used.

Info

Fullscreen mode

Press **F11** when cursor is in the editor to toggle full screen editing. **Esc** can also be used to exit full screen editing.

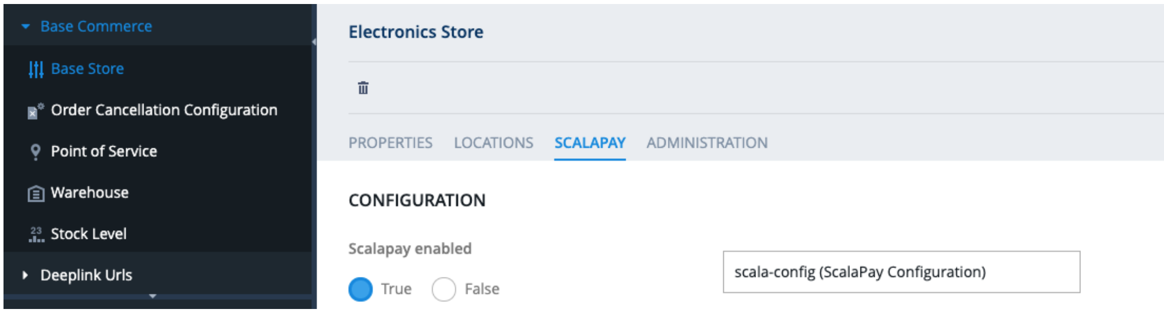
See also in the hybris Wiki

- impex Extension - Technical Guide

Backoffice

Changes to the configuration can be made via the Backoffice:

1. Login to you Backoffice and navigate to **Base Commerce > Base Store**



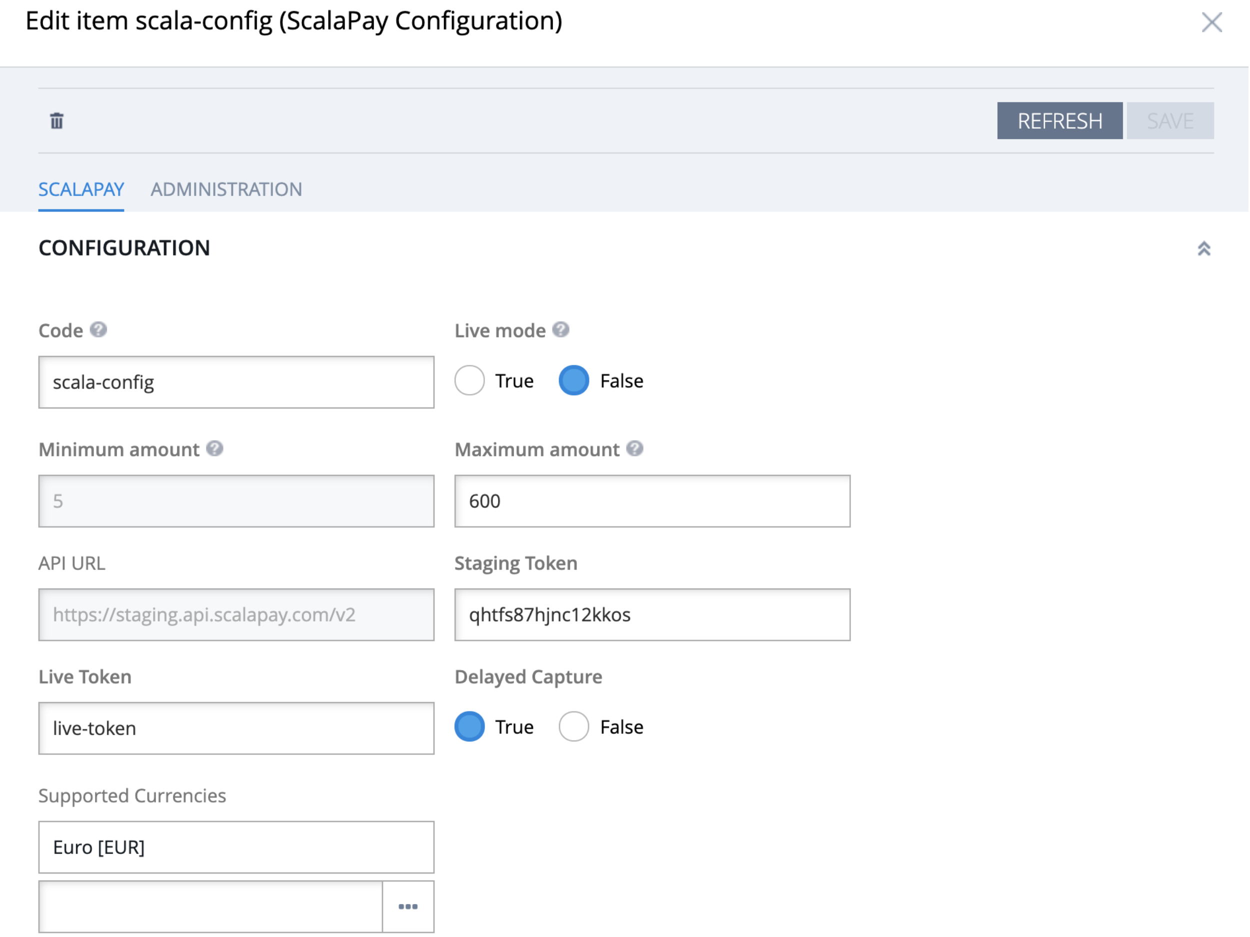
CONFIGURATION

Scalapay enabled

● True ○ False

scala-config (ScalaPay Configuration)

2. Double click the scala-config (Scalapay Configuration) entry to reveal the configuration overlay



Edit item scala-config (ScalaPay Configuration)

CONFIGURATION

Code: scala-config

Live mode: ○ True ● False

Minimum amount: 5

Maximum amount: 600

API URL: https://staging.api.scalapay.com/v2

Staging Token: qhtfs87hjnc12kkos

Live Token: live-token

Delayed Capture: ● True ○ False

Supported Currencies: Euro [EUR]

3. Update the relevant attributes and pres SAVE - your configuration will be active immediately

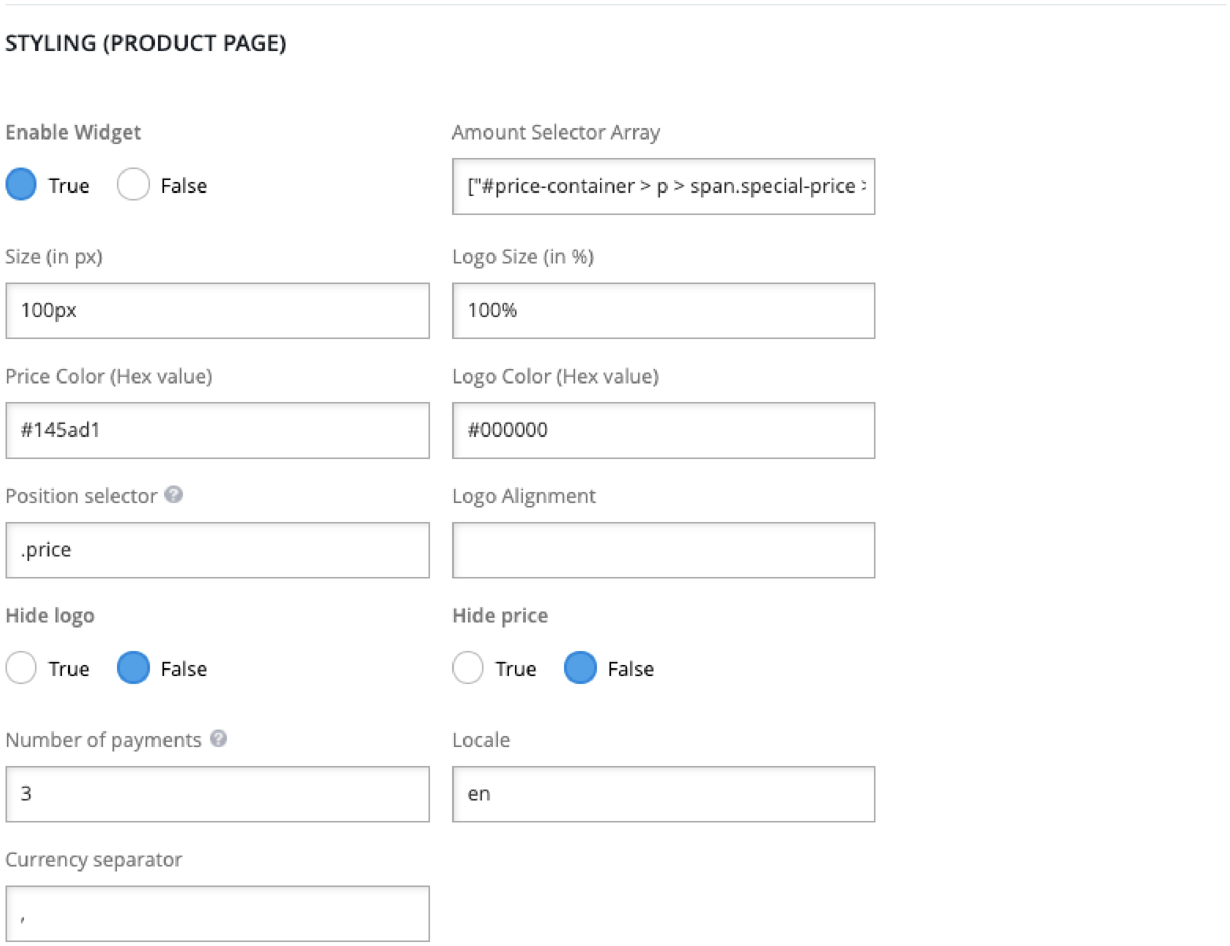
Core Configuration

Core configuration allows you to manage the key attributes related to the integration with Scalapay system

Live mode	Controls if Live mode is on When set to FALSE [default] the Staging token is used When set to TRUE the Live Token is used
API URL	URL of the Scalapay API. This cannot be changed, it will be automatically switched based on the Live mode toggle selection
Minimum amount	Set as a system property and cannot be changed using the Backoffice
Maximum amount	This must be set to the value agreed in the commercial contract
Staging Token	Token value provided by Scalapay for testing & certification
Live Token	Token value provided by Scalapay for production use
Supported Currencies	List of currencies for which Scalapay will be enabled for the selected store
Delayed Capture	Merchant informs Scalapay it is requesting a delayed capture by calling /v2/payments/{token}/delay. The payment schedule for the customer will be created however no settlement of funds will occur until a capture call is made.

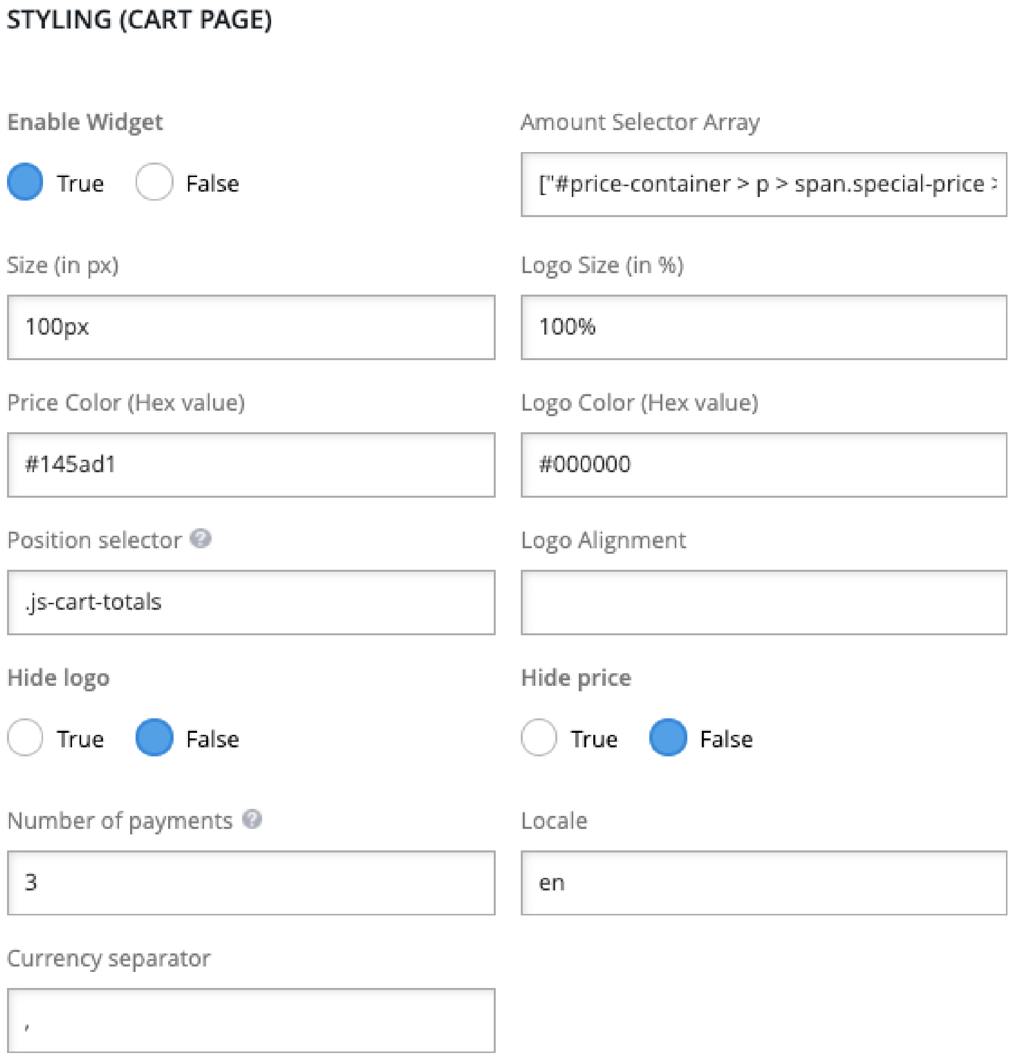
Product Page and Cart Page configuration

You can configure and customize some aspects of where and how the Scalapay widget appears on the product page and the cart page. Following attributes can be changed, see Scalapay Widget for further details.



STYLING (PRODUCT PAGE)

Enable Widget ● True ○ False	Amount Selector Array ["#price-container > p > span.special-price
Size (in px) 100px	Logo Size (in %) 100%
Price Color (Hex value) #145ad1	Logo Color (Hex value) #000000
Position selector .price	Logo Alignment
Hide logo ○ True ● False	Hide price ○ True ● False
Number of payments 3	Locale en
Currency separator ,	



STYLING (CART PAGE)

Enable Widget
● True ○ False

Amount Selector Array: ["#price-container > p > span.special-price

Size (in px): 100px

Logo Size (in %): 100%

Price Color (Hex value): #145ad1

Logo Color (Hex value): #000000

Position selector: .js-cart-totals

Logo Alignment:

Hide logo
○ True ● False

Hide price
○ True ● False

Number of payments: 3

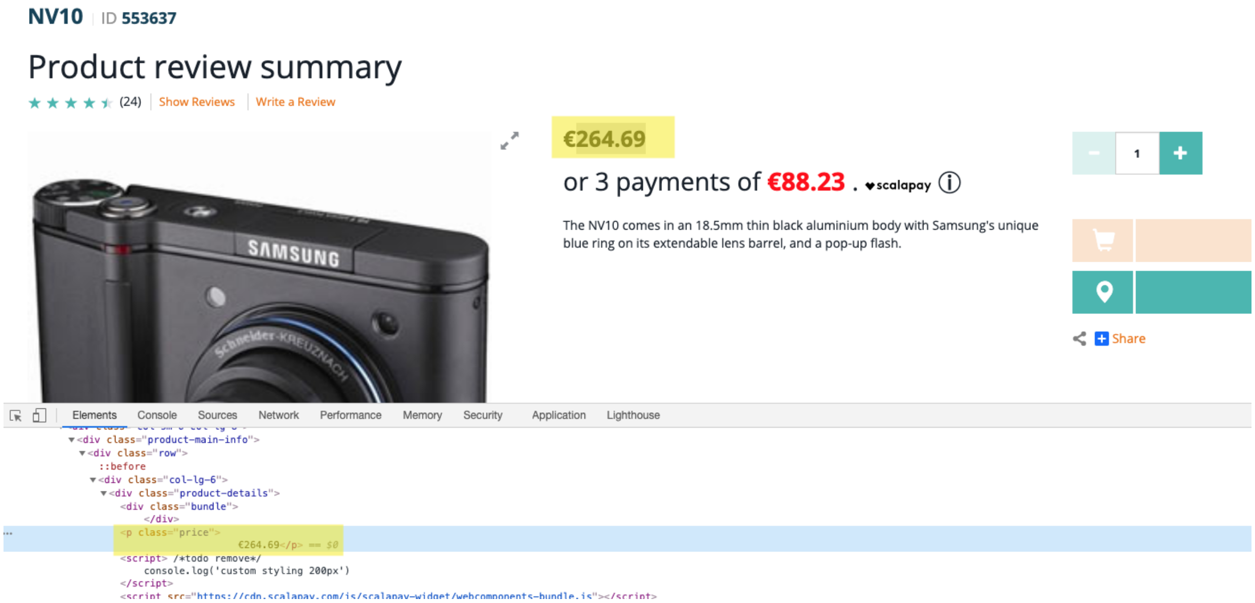
Locale: en

Currency separator: ,

Amount Selector Array	CSS selectors of elements that contains the price (See below)
Size	Width of the widget in px
Logo Size	% of the actual logo image
Price Color	Price text color (not recommended according to Scalapay brand identity)
Logo Color	Logo color (not recommended according to Scalapay brand identity)
Position selector	CSS selector of the element to which the Scalapay widget

	will be appended
Logo Alignment	How the logo should be aligned
Hide Logo	If the logo should be hidden
Hide Price	If the price should be hidden
Number of Payments	Number of payments. Strictly for display purposes, it does not change the actual configuration on the Scalapay side
Locale	The locale to be used
Currency Separator	How currencies are being separated

Note: You can find the CSS selectors by using Web developer tools available with modern browsers (e.g. Chrome, Firefox, Safari) - see example below



The NV10 comes in an 18.5mm thin black aluminium body with Samsung's unique blue ring on its extendable lens barrel, and a pop-up flash.

```
▼<div class="product-main-info">
 ▼<div class="row">
    ::before
  ▼<div class="col-lg-6">
   ▼<div class="product-details">
     <div class="bundle">
         </div>
     <p class="price">
                       €264.69</p> == $0
     <script> /*todo remove*/
         console.log('custom styling 200px')
     </script>
     <script src="https://cdn.scalapay.com/js/scalapay-widget/webcomponents-bundle.js"></script>
```

Page Integration

Tag Library

Copy the below Scalapay tags and tag library into your storefront

scalapay/web/webroot/WEB-INF/common/tld
scalapay/web/webroot/WEB-INF/common/tags

Product Page

Scalapay widget can easily be added to the page using the supplied JSTL tag. Open the correct JSP template/fragment, add the taglib definition then add the scalapay tag at the bottom of the page - the actual target location is being controlled by the **Price selector** property in the backoffice configuration. Ensure to place the tag ***AFTER*** the dom element specified in the Price selector property.
Consider placing in: productDetailsPanel.tag

```
...
<%@ taglib prefix="scala" tagdir="/WEB-INF/tags/responsive/scala"
%>
...
<scala:scala_pdp amount="${product.price.value}"
currencyIso="${product.price.currencyIso}"/>
...
```

Cart Page

Similarly the widget can be added to the Cart page template/fragment
Consider placing in: cartTotals..tag

```
...
<%@ taglib prefix="scala" tagdir="/WEB-INF/tags/responsive/scala"
%>
...
<scala:scala_cartPage amount="${cartData.totalPrice.value}"
currencyIso="${cartData.totalPrice.currencyIso}"/>
...
```

Checkout

All essential functionality can be found in the ScalapayFacade.

If you already provide multiple payment options in your store, you can leverage `ScalapayFacade.validForSessionCart()` to determine if ScalaPay could be a possible payment type for the current cart.
All you need to add to your storefront is the **Controller** to receive ScalaPay responses (endpoints as defined in `scalapay.redirect.confirm.url` and `scalapay.redirect.cancel.url` in local.properties file). Ensure that the URLs are whitelisted in `DefaultGuestCheckoutCartCleanStrategy.skipPatterns`
Example:

```
import com.scalapay.data.OrderStatus;
import com.scalapay.facade.ScalapayFacade;
import
com.scalastore.store.controllers.pages.checkout.steps.PaymentMethodCheckoutStepController;
import de.hybris.platform.acceleratorstorefrontcommons.controllers.util.GlobalMessages;
import de.hybris.platform.core.model.user.CustomerModel;
import org.apache.commons.lang.StringUtils;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import java.util.Map;

@Controller
@RequestMapping("/scala-pay")
public class ScalaPayController extends PaymentMethodCheckoutStepController {

    @Resource(name = "scalapayFacade")
    private ScalapayFacade scalapayFacade;


    @RequestMapping(value = "/process", method = RequestMethod.POST)
    public String doValidateAndPost(RedirectAttributes redirectAttributes) {

        boolean valid = scalapayFacade.validForSessionCart();
        if (!valid) {
            GlobalMessages.addFlashMessage(redirectAttributes,
GlobalMessages.INFO_MESSAGES_HOLDER,
                    "checkout.scala.not.supported");
            return getCheckoutStep().currentStep();
        }
        Map<String, String> orderTokenResponse = scalapayFacade.getOrderToken();
        if (orderTokenResponse == null) {
            GlobalMessages.addFlashMessage(redirectAttributes,
GlobalMessages.INFO_MESSAGES_HOLDER,
                    "checkout.scala.error");
            return getCheckoutStep().currentStep();
        }
        String orderToken = orderTokenResponse.keySet().stream().findFirst().get();
        String redirectUrl = orderTokenResponse.get(orderToken);
        scalapayFacade.setSessionCheckoutToken(orderToken);
        return "redirect:" + redirectUrl;

    }

    @RequestMapping(value = "/response", method = RequestMethod.GET)
    public String doProcessScalaPayResponse(final HttpServletRequest request,
                                            RedirectAttributes redirectAttributes) {

        final Map<String, String> params = getRequestParameterMap(request);
        String orderToken = params.get("orderToken");
        String status = params.get("status");
        boolean validToken =
StringUtils.equalsIgnoreCase(scalapayFacade.getSessionCheckoutToken(), orderToken);
        OrderStatus orderStatus = status == null ? OrderStatus.ERROR :
OrderStatus.valueOf(StringUtils.upperCase(status));
        if (!validToken || orderStatus != OrderStatus.SUCCESS) {
            GlobalMessages.addFlashMessage(redirectAttributes,
GlobalMessages.INFO_MESSAGES_HOLDER,
                    "checkout.scala.not.successful");
            return getCheckoutStep().currentStep();
        }
        CustomerModel currentUser =
getCheckoutCustomerStrategy().getCurrentUserForCheckout();
        scalapayFacade.createAndStorePayment(orderToken, currentUser);
        getCheckoutFacade().setDeliveryModeIfAvailable(); // to force setting 'calculated'
param on cart to true
        return getCheckoutStep().nextStep();

    }

}
```

Combined Auth and Capture

If your store does not make use of the Order Review step (like the Hybris OOTB default store does), you can call `ScalapayFacade.capture()` to call the payment capture immediately after receiving the order token from ScalaPay.

If you use the review screen before placing the order, make sure to use the `ScalaPayPlaceOrderMethodHookImpl` class, which will update ScalaPay with the final order number.