Accelerators

Generated on: 2024-12-03 13:56:03 GMT+0000

SAP Commerce | 2205

Public

Original content: https://help.sap.com/docs/SAP_COMMERCE/4c33bf189ab9409e84e589295c36d96e?locale=en-US&state=PRODUCTION&version=2205

Warning

This document has been generated from the SAP Help Portal and is an incomplete version of the official SAP product documentation. The information included in custom documentation may not reflect the arrangement of topics in the SAP Help Portal, and may be missing important aspects and/or correlations to other topics. For this reason, it is not for productive use.

For more information, please visit the https://help.sap.com/docs/disclaimer.

Commerce B2C Accelerator

The SAP Commerce Accelerator is a ready-to-use omni-channel solution that you can use to speed implementation, boost sales, and increase growth across all of your channels.

Core Accelerator Module

The Core Accelerator Module provides Responsive Storefront functionality, as well as features such as payment services, extensions to the OCC API, and additional WCMS components beyond the defaults provided by the cms2 and cms2lib extensions.

Base Accelerator Module

The Base Accelerator module provides the majority of Accelerator functionality, from configurable checkout to Spring security, as well as link and navigation management, SEO, pick up in store, customizations, search and navigation, and much more.

B2C Accelerator Module

The B2C Accelerator module lets you set up fully-functional apparel and electronics storefronts.

B2C Accelerator AddOns Module

The B2C Accelerator AddOns module is a set of special extensions, called AddOns, that allow you to extend the functionality of your storefront without having to edit the core code base.

Core Accelerator Module

The Core Accelerator Module provides Responsive Storefront functionality, as well as features such as payment services, extensions to the OCC API, and additional WCMS components beyond the defaults provided by the cms2 and cms2lib extensions.

Features	Architecture	Implementation
		
Accelerator Payment Mock	acceleratorcms Extension	Extensible Cart Item Menu
CSV Import and Export	acceleratorfacades Extension	Responsive Storefronts
Responsive Storefronts	acceleratorservices Extension	
	acceleratorstorefrontcommons AddOn	
	acceleratorwebservicesaddon AddOn	
	addonsupport Extension	

Core Accelerator Features

The Core Accelerator features include payment mocks, adding CMS actions, and importing and exporting the cart as a CSV file.

Accelerator Payment Mock

Learn about the different checkout Payment Card Industry (PCI) strategies offered as part of SAP Commerce Accelerator that allow customers to create credit card payment subscriptions (tokens) that they can use to pay for their purchases.

CMS Actions

CMS actions add functionality to CMS components. You can use actions on a component that performs an operation, such as modifying the cart, or linking to another page. The add-to-cart button or the pick-up-in-store button are examples of such actions.

CSV Import and Export

Users can export and import their carts as a CSV file so they can view or adjust their cart outside the storefront.

Backoffice Operations

Various Backoffice operations are available for SAP Commerce Accelerator that you can use for customer support, configuring products, and setting up the online store layout.

Accelerator Payment Mock

Learn about the different checkout Payment Card Industry (PCI) strategies offered as part of SAP Commerce Accelerator that allow customers to create credit card payment subscriptions (tokens) that they can use to pay for their purchases.

i Note

The CyberSource extension has been removed from the SAP Commerce distribution.

→ Tip

The expectation is that any project implemented using SAP Commerce Accelerator will remove the ability for the customer to decide which checkout PCI strategy is used and instead adapt the Checkout PCI Strategy required. This option is provided as a development tool.

These strategies are only included in the reference B2C storefronts, but you can also do similar implementations for the B2B storefront.

Accelerator Payment Mock Overview

SAP Commerce Accelerator has a basic `Payment Gateway Mock` for handling creating customer payment information. This provides both `Hosted Order Page (HOP)` and `Silent Order Page (SOP)` style implementations, which both result in the creation of a `Subscription` or `Token`. This mocks services offered by a secure payment provider which ensure that a customer's sensitive payment data is not stored in the SAP Commerce system. In creating a subscription in this way, the SAP Commerce system holds a `subscription id` reference to a customer profile on the payment provider secure server which greatly simplifies the PCI compliance process. The SAP Commerce `cybersource` extension and its related web services are used to handle payment for items in the shopping cart. This is shown in steps 4 and 8, respectively, in the table in the Order Processing Using HOP section below.

The CyberSource Hosted Order Page service is used in SAP Commerce Accelerator as an example for the HOP integration. We have introduced an actual CyberSource HOP integration option and a Mocked CyberSource variant implementation used only for demo purposes. You can use these approaches as an example for your own payment provider HOP implementation, such as PayPal.

The CyberSource Silent Order Port service is also used as an example for SOP integration. This is less obviously a mock as the form page used to collect card information appears as a normal page. However this PCI-enabling integration will post directly to the third party (which is CyberSource, in this case) and provide a channel for receiving updates on this.

CyberSource HOP Implementation

There are two types of HOP integration options offered by CyberSource:

- Standard Implementation
- Partially Customized Implementation.

The reference solution offered by Commerce Accelerator uses the Partially Customized Implementation option. This option is more inline with other payment provider HOP services and helps in integrating other HOP integration services in your project.

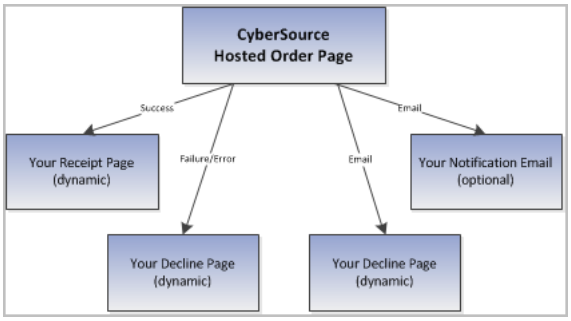


Figure: CyberSource partially customized integration option.

→ **Tip**

This option allows Commerce Accelerator storefronts to handle responses from the HOP, displaying custom transaction result information.

Mocked HOP Implementation

The mocked HOP implementation, provided with Commerce Accelerator, follows a similar process flow to that of the actual CyberSource implementation as described in the Order Processing Using HOP section below. In general, a request is sent to the mocked HOP service, a dummy page is shown and the option to fail or approve the transaction is available.

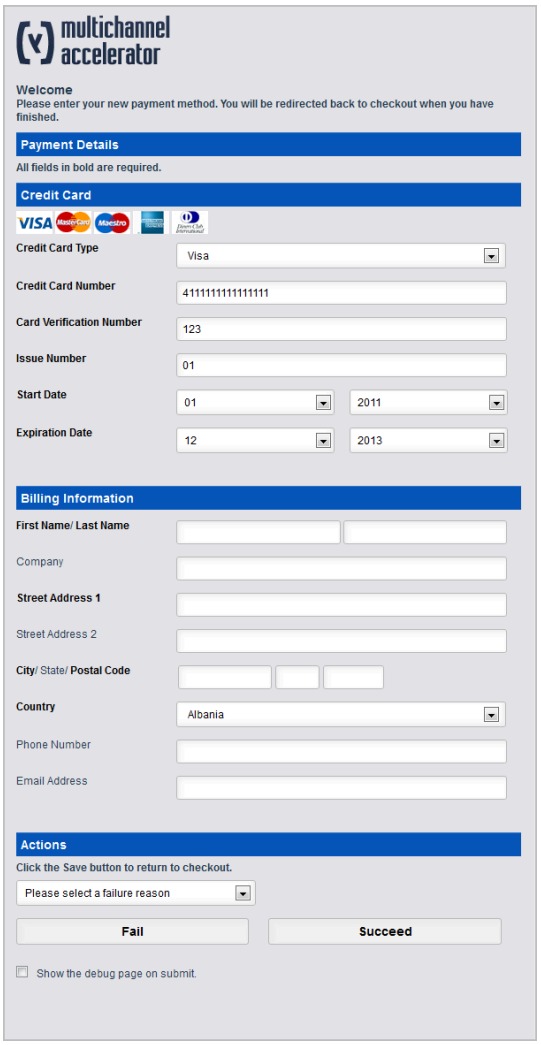


Figure: Commerce Accelerator mocked Hosted Order Page.

In the image above, you can see that the mocked Hosted Order Page is very similar to the actual CyberSource Hosted Order Page shown in the Order Processing Using HOP section below. There are two main differences:

- The form is pre-populated with credit card data and, if available, the delivery address provided during the checkout process in the storefront. The pre-populated card data is configurable. For more details, see the Mocked Configuration document.
- The new **Actions** section is added. This section contains:
  - The **Fail** button with a drop down list of reason codes for failure. This is used to simulate general HOP server errors.
  - The **Succeed** button which causes the transaction to succeed without any errors and return to the Commerce Accelerator storefront configured in the return URL.
  - The **Show the debug page on submit** check box to show a debug page listing all the parameters that are sent to the storefront return URL. You can use this option for debugging purposes during development stage.

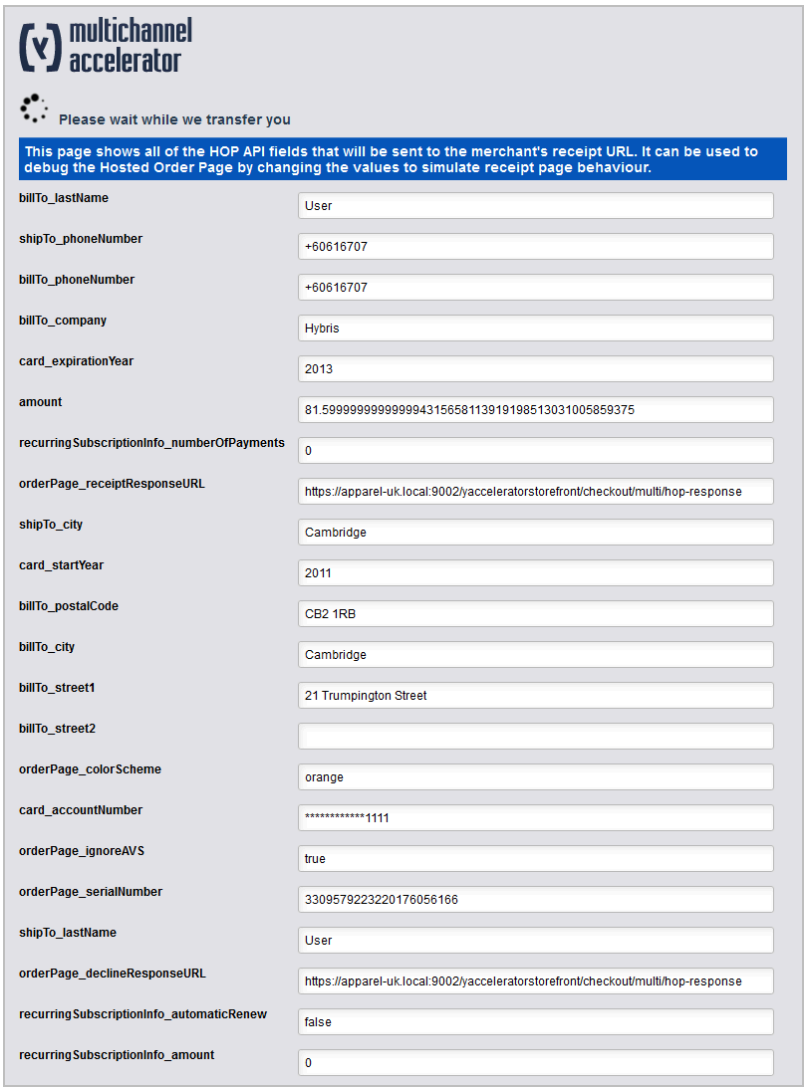


Figure: An excerpt from the mocked hosted order debug page.

ℹ Note

You **cannot** use the subscriptions created using the mocked Hosted Order Page with a live CyberSource payment authorization provider to purchase items. This is because the subscription `Id` created using the mock implementation does not exist in the live CyberSource implementation. It is therefore recommended to use the mock Hosted Order Page only for demo or development purposes.

For additional information, see Mocked Configuration.

Difference Between HOP and SOP

The Hosted Order Page (HOP) and Silent Order Post (SOP) are very similar to each other except for the following differences:

HOP	SOP
Hosted on a secure CyberSource server.	Hosted on the merchant's server.
Does not require the merchant to have an SSL secured server.	Requires the merchant to have an SSL secured server.
Limited customization of the form layout.	Form layout completely controlled by the merchant.

Order Process Using HOP

The diagram below provides an overview of the Commerce Accelerator implementation of the order process using the HOP:

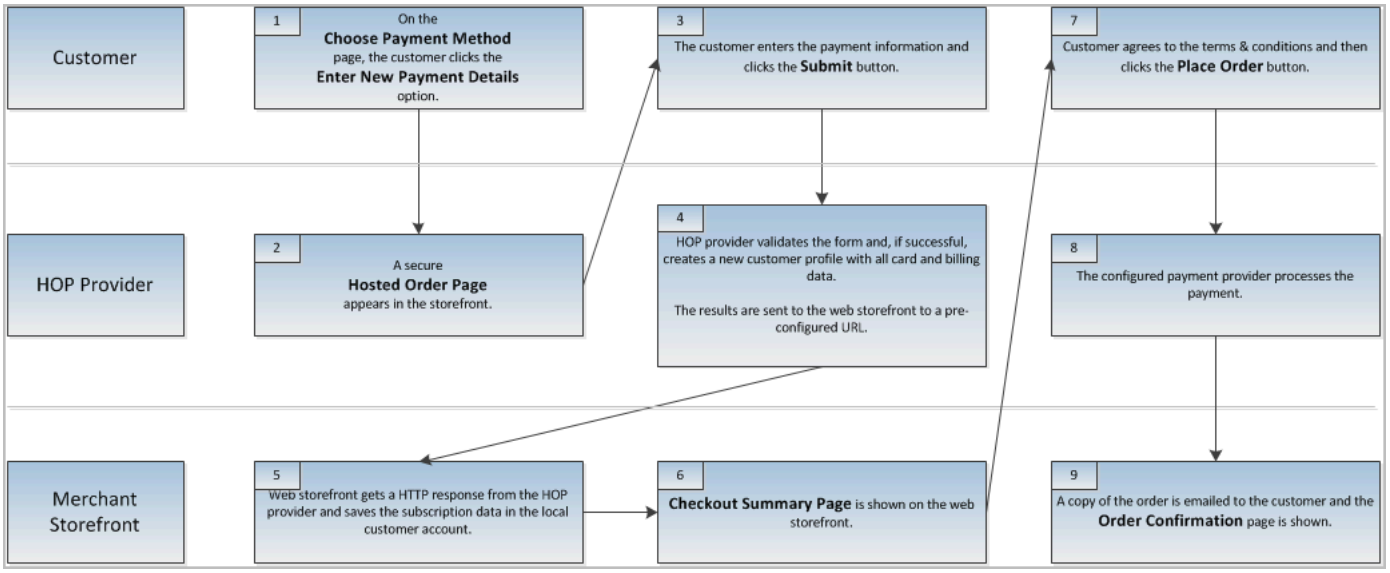
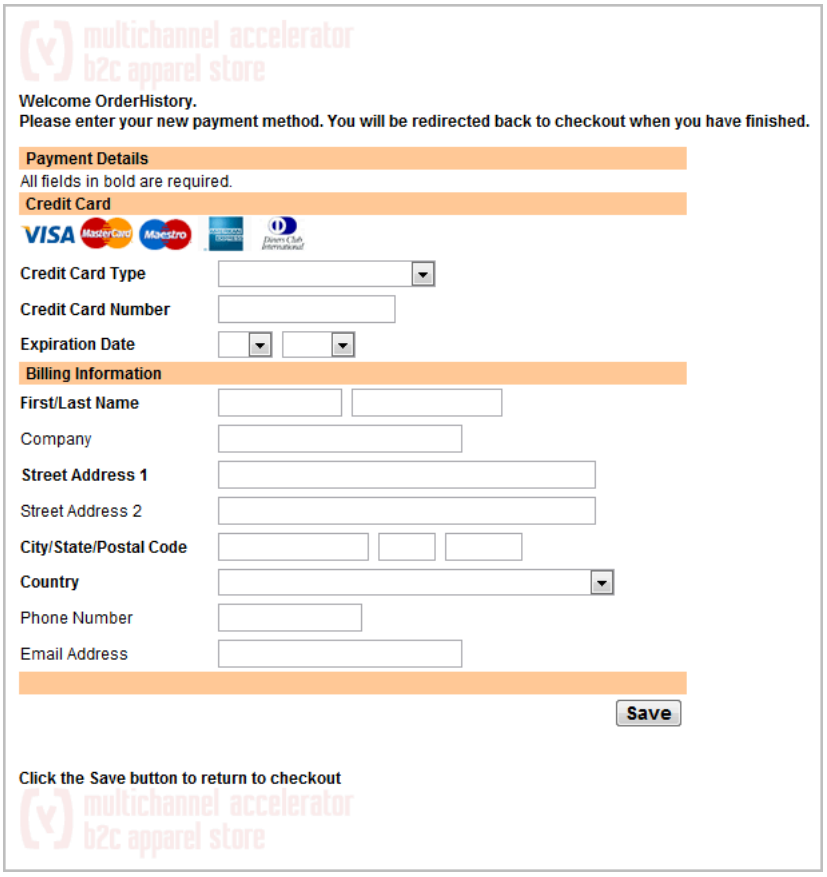


Figure: Hosted Order Page overview diagram.

Here are the details for each step shown in the diagram:

1. Before reaching this page, a user:
    a. Adds items to the Cart.
    b. Enters the Checkout process.
    c. Supplies Delivery Address.
    d. Chooses Delivery Method.
    e. Supplies Payment Method using one of the options:
        - Chooses previously saved payment details.
        - Selects the Create new payment details option.
2. Storefront displays the Hosted Order Page:



This page is not an storefront CMS page as it is delivered by CyberSource. The following parameters need to be passed to the HOP when posting the request. Note, that the code snippet below shows mock values:

→ **Tip**

Values presented are split into two categories:

- Merchant signature parameters
- Customer subscription signature parameters

Required parameters:

```
merchantID=examplemerchant
                            currency=usd
                            amount=0
                            orderPage_serialNumber=2411069296000167904065
                            orderPage_version=4
                            orderPage_signaturePublic=girbstzE+iYrhedSb8yzDzTfAEY=
                            orderPage_timestamp=1260915756616
                            orderPage_transactionType=subscription


recurringSubscriptionInfo_signaturePublic=KHyI4uRymi6M1gP38UbOGMi24=
                                recurringSubscriptionInfo_startDate=20091216
                                recurringSubscriptionInfo_automaticRenew=false
                                recurringSubscriptionInfo_frequency=on-demand
                                recurringSubscriptionInfo_numberOfPayments=0
                                recurringSubscriptionInfo_amount=5
```

3. The customer enters all the information requested in the form and clicks the Submit button.
4.

   a. The CyberSource server validates the data entry fields, showing validation messages where necessary. The user is not able to pass this page unless the form is validated or the Back browser button is clicked.

   b. Once the form is validated, the customer profile is created on the provider server. A unique identifier, that is `subscription ID`, is assigned to the customer profile. By default, the subscription ID is a 22-digit non-format-preserving subscription ID.

   c. The setting for the size of the subscription ID is applied to the merchant account, so it affects all customer profiles created.

   Note

   To change the length of the `subscription id` CyberSource Customer Support needs to be contacted. The ID can be configured to a 16 or 22 digit subscription ID.

   The result of the customer profile creation process is sent in a response back to a pre-configured or parameter-passed URL, as shown in the sample reponse parameters from CyberSource.

```
decision=ACCEPT
                                        decision_publicSignature=zyWxMC4PrPybLAaRE3M/loKQG64=
                                        reasonCode=100
                                        requestID=2609159276880172034075
                                        merchantID=examplemerchant
                                        ccAuthReply_reasonCode=100
                                        ccAuthReply_authorizationCode=831000
                                        ccAuthReply_cvCode=3
                                        ccAuthReply_avsCodeRaw=Y
                                        ccAuthReply_avsCode=Y
                                        ccAuthReply_amount=0.00
                                        ccAuthReply_processorResponse=000
                                        ccAuthReply_authorizedDateTime=2009-12-15T222527Z
                                        billTo_firstName=John
                                        billTo_lastName=Doe
                                        billTo_street1=1295 Charleston Rd.
                                        billTo_city=Mountain View
                                        billTo_state=CA
                                        billTo_postalCode=94043
                                        billTo_country=us
                                        card_cardType=001
                                        card_accountNumber=############1111
                                        card_expirationMonth=01
                                        card_expirationYear=2011
                                        paymentOption=card
                                        orderAmount=0
                                        orderAmount_publicSignature=UCEUdG1W/48Ffuq2ZHJxDIgznWI=
                                        orderCurrency_publicSignature=Gah02OEPox8Iblx4W7DMQl8oQ5Y=
                                        orderCurrency=usd
                                        orderPage_serialNumber=2411069296000167904065
                                        orderPage_requestToken=Ahj77wSRGuK3tM7zdFA2VJ7PJsCQ6Ap7
                                        orderPage_transactionType=subscription
                                        paySubscriptionCreateReply_subscriptionID=2609159276880172034075
                                        paySubscriptionCreateReply_subscriptionIDPublicSignature=Jv6mDivpD3J7oiS8=
                                        recurringSubscriptionInfo_frequencyPublicSignature=TRZKQk3blDAULiU47zrkik=
                                        recurringSubscriptionInfo_amount=5
                                        recurringSubscriptionInfo_numberOfPayments=0
                                        recurringSubscriptionInfo_frequency=on-demand
                                        recurringSubscriptionInfo_automaticRenewPublicSignature=dF5qj0NHU9Mi0txLg=
                                        recurringSubscriptionInfo_startDate=20091216
                                        recurringSubscriptionInfo_numberOfPaymentsPublicSignature=U63QhJoBBQAE0U=
                                        recurringSubscriptionInfo_amountPublicSignature=kz6+vmBEOr4WIPciSHwbNNBVs=
```

```
recurringSubscriptionInfo_startDatePublicSignature=UHwljFNBlCGLiUsSpjAgrs=
recurringSubscriptionInfo_automaticRenew=false
transactionSignature=VvPXdt6q9RP75zH8x+NZ2Fo2JdU=
signedFields=<comma-separated list of all the data fields in the order>
```

5. The related controller for the return URL decodes the response and either:
    - Saves the data to the customer local account, shows a message indicating that the profile was successfully created.
    - Shows an error message based on the `reasonCode` returned.
6. Once the profile creation process is deemed successful, the system displays the Checkout Summary page. This page allows the customer to:
    - Cancel the order
    - Change the delivery address
    - Change the delivery method
7. Once the customer is satisfied with his order, he accepts the Terms and Conditions and clicks the Place Order button.
8. The order is then processed by the payment provider configured for use.

> **Note**
> If CyberSource HOP is used to create a customer subscription then it is necessary cot configure CyberSource as the payment provider. Also ensure the same CyberSource Merchant ID is used for both HOP and payment.

9. A copy of the order is emailed to the customer and the system displays the Order Confirmation page.

Order Process Using SOP

The diagram below gives an overview of the Commerce Accelerator implementation of the order process using the SOP:

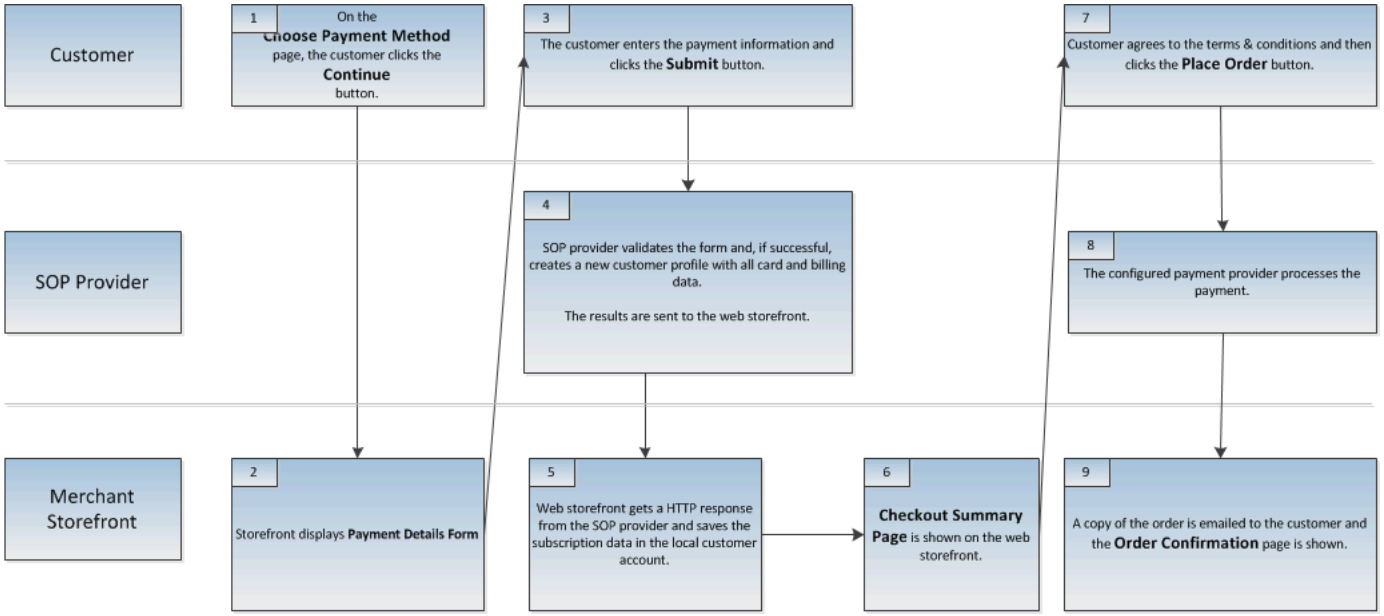
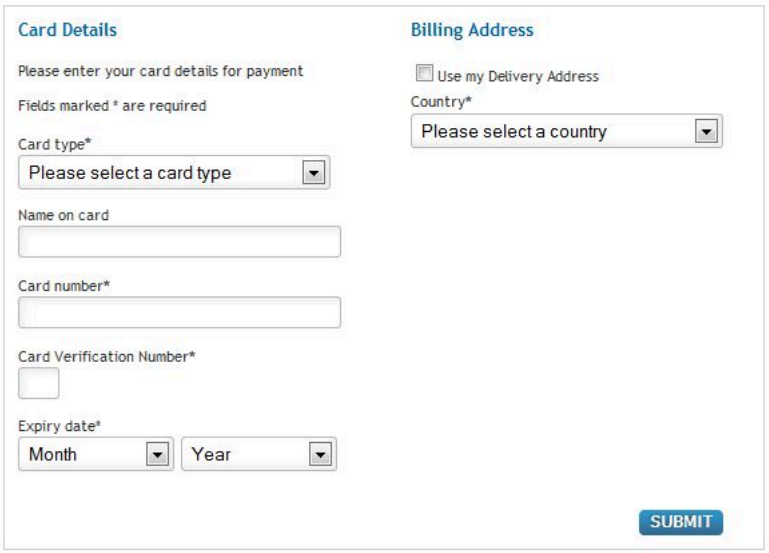


Figure: Silent Order POST overview diagram.

Here are the details for each step shown in the diagram:

1. Before reaching this page, a user:
    a. Adds items to the Cart.
    b. Enters the Checkout process.
    c. Provides the Delivery Address.
    d. Chooses the Delivery Method.
2. The storefront displays the Payment Details Form defined in our storefront:



3. The customer enters all the information requested in the form and clicks the Submit button.
4. The CyberSource server validates the data entry fields and sends a response. If the entered data is correct, the customer profile is created on the provider server. A unique identifier, `subscription ID`, is assigned to the customer profile.
5. The related controller for the return URL decodes the response and either:
    - Saves the data to the customer local account and shows a message indicating that the profile was successfully created.
    - Shows an error message based on the `reasonCode` returned.
6. Once the profile creation process is deemed successful, the system displays the Checkout Summary page. This page allows the customer to: cancel the order or change the:
    - Cancel the order
    - Change the delivery address
    - Change the delivery method
7. Once the customer is satisfied with his order, he accepts the Terms and Conditions and clicks the Place Order button.
8. The order is then processed by the payment provider configured for use.

    Note

    If CyberSource SOP is used to create a customer subscription then it is necessary to configure CyberSource as the payment provider. Also ensure the same CyberSource Merchant ID is used for both SOP and payment.

9. A copy of the order is emailed to the customer and the system displays the Order Confirmation page.

Related Information

Managing Multi Step Checkout Strategies

Generic Configuration

Learn about the generic configuration options for the payment implementation in SAP Commerce Accelerator. The document is split according to extensions you have to configure in order to have running HOP or SOP implementation.

Note

The CyberSource extension has been removed from the SAP Commerce distribution.

Implementation of payment requires the following options to be configured:

- Generic configuration, described in this document.
- Mocked Configuration, described in the Mocked Configuration document.

yacceleratorstorefront Extension Configuration

Although the code for the HOP or SOP implementation resides in the different extension, the `acceleratorservices`, the generic settings are placed in the `yacceleratorstorefront` extension because this is where the functionality is used.

Note

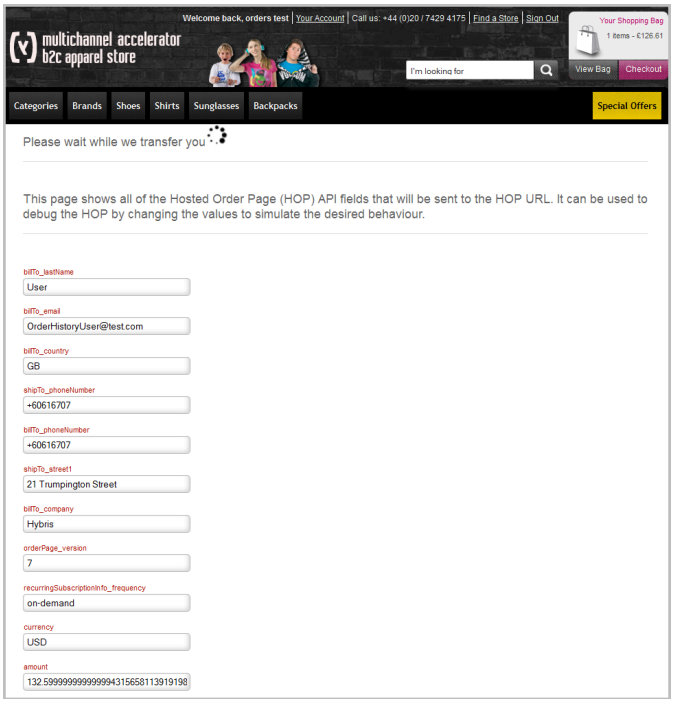
It is recommended to have these settings in the actual extension that uses the HOP or SOP functionality. This approach allows multiple extensions to use it and have separate configurations per usage.

The settings are defined in the *<${HYBRIS_BIN_DIR}>*`/ext-accelerator/yacceleratorstorefront/project.properties` file. The file allows the following configuration options:

```
############ Hosted Order Page settings ###############
                            ####################################################
                            ##### Common Properties
                            #hop.post.url=https://orderpagetest.ic3.com/hop/orderform.jsp
                            hop.post.url=/acceleratorservices/hop-mock
```

```
#sop.post.url=https://orderpagetest.ic3.com/hop/ProcessOrder.do
sop.post.url=/acceleratorservices/sop-mock/process

#These keys are valid only for MultiStep Checkout.You can use it to set SOP or HOP option.
site.pci.strategy=SOP
site.pci.strategy.apparel-uk=HOP
site.pci.strategy.apparel-de=HOP
site.pci.strategy.electronics=HOP

### Remove the comment below to show the hosted order pre-post debug page (can be done on a per-site basis)
#hop.debug.mode.apparel-uk=true
```

The table below provides description of the properties used:

Property Key	Description
<hop.post.url> / <sop.post.url>	This is the URL for the Hosted Order Page/Silent Order Post. All parameters containing order details are sent to this URL for processing. It is recommended that this uses a secure connection for data security. In the example above there are actual CyberSource HOP/SOP URL and the Mocked HOP/SOP URL. You can be comment or comment them out, depending on your requirements. **ℹ Note** Remember to use the Mocked HOP/SOP for demo/development purposes. Also if the Mocked HOP/SOP is used then you must use the Mocked Payment provider that comes with the SAP Commerce Accelerator since mocked subscription `ID`s cannot be used against a production payment provider.
<site.pci.strategy>	This property is used to determine which PCI option should be used. You can set one of the PCI options defined in `CheckoutPciOptionEnum` for example SOP or HOP. If your project has multiple storefronts then having a separate key with the CMS site name appended to that and provides site specific settings as shown in the example above.
<hop.debug.mode>	**⚠ Caution** This property should be used during development **only**. You should **never** use this page on production systems as it exposes security sensitive data. It displays a debug page with all of the parameters that are sent to the HOP. This can be useful when testing your application. If your project has multiple storefronts then having a separate key with the CMS site name appended to then end provides site-specific settings as shown in the example above.  Figure: An excerpt from the pre-HOP POST debug page.

→ Tip

For site-specific setting in a project that has multiple storefronts, append the name of your CMS site to the end of the key as in the example above.

Affected Controllers

The `MultiStepCheckoutController` handles the multi-step checkout process. The `MultistepCheckoutController` is defined in the `yacceleratorstorefront` extension.

This controller is an annotation-configured controller that contains the `paymentFacade`, see the yacceleratorfacades Extension section below for more information on this, as denoted by the `@Resource` annotation in the code extract from the `de.hybris.platform.storefront.controllers.pages.checkout` file shown below:

```
@Controller
                                @RequestMapping(value = "/checkout/multi")
                                public class MultiStepCheckoutController extends AbstractCheckoutController
                                {
                                ...   ...   ...
                                @Resource(name = "paymentFacade")
                                private PaymentFacade paymentFacade;
                                ...   ...   ...
                                }
```

acceleratorservices Extension

This extension contains the main payment service bean definitions required to integrate the HOP or SOP functionality into SAP Commerce Accelerator.

→ Tip

Development Hint

The class assignment for `acceleratorPaymentService` defined below, can be used as a base for extending functionality required by the HOP or SOP provider used in your project,

Configuration File	Description
<${HYBRIS_BIN_DIR}>/ext-accelerator/acceleratorservices/resources/acceleratorservices-spring.xml	The file contains the payment service definition and all converter and populator properties required. The code extract below shows the existing bean configuration.

```
<!-- Accelerator Payment Service -->
                                <alias name="defaultCyberSourceAcceleratorPaymentService" alias="acceleratorPaymentService"/>
                                <bean id="defaultCyberSourceAcceleratorPaymentService" class="de.hybris.platform.acceleratorservices.payment.cyberso
                                <property name="cartService" ref="cartService"/>
                                <property name="commerceCheckoutService" ref="commerceCheckoutService"/>
                                <property name="customerEmailResolutionService" ref="customerEmailResolutionService"/>
                                <property name="flexibleSearchService" ref="flexibleSearchService"/>
                                <property name="modelService" ref="modelService"/>
                                <property name="userService" ref="userService"/>
                                <property name="creditCardPaymentSubscriptionDao" ref="creditCardPaymentSubscriptionDao"/>
                                <property name="siteConfigService" ref="siteConfigService"/>
                                <property name="businessProcessService" ref="businessProcessService"/>
                                <property name="paymentResponseInterpretation" ref="paymentResponseInterpretationStrategy"/>
                                <property name="hopPaymentResponseInterpretation" ref="hopPaymentResponseInterpretationStrategy"/>
                                <property name="customerBillToDataConverter" ref="customerBillToDataConverter"/>
                                <property name="customerShipToDataConverter" ref="customerShipToDataConverter"/>
                                <property name="paymentDataConverter" ref="paymentDataConverter"/>
                                <property name="paymentInfoDataConverter" ref="paymentInfoDataConverter"/>
                                <property name="createSubscriptionResultConverter" ref="createSubscriptionResultConverter"/>
                                <property name="paymentFormActionUrlStrategy" ref="paymentFormActionUrlStrategy"/>
                                <property name="erroCodeToFormFieldMappingStrategy" ref="erroCodeToFormFieldMappingStrategy"/>
                                <property name="commonI18NService" ref="commonI18NService"/>
                                </bean>
```

yacceleratorfacades Extension

This extension exposes the payment related services to the `yacceleratorstorefront` extension via the `paymentFacade` Spring bean.

→ Tip

The `paymentFacade` Spring bean defines a generic payment facade that is fully configurable and extensible to meet your project requirements. The important property of that bean is *<paymentService>*. You can inject a different provider implementation here to provide the specific function offered by your provider API. See the code extracts below.

Configuration File	Description
<${HYBRIS_BIN_DIR}>/ext-accelerator/yacceleratorfacades/resources/yacceleratorfacades-spring.xml	This file contains the payment Facade definition used by the Accelerator storefronts. The code extract below shows the existing bean configuration.

```
<alias name="defaultPaymentFacade" alias="paymentFacade"/>
                                <bean id="defaultPaymentFacade" class="de.hybris.platform.acceleratorfacades.payment.impl.DefaultPaymentFacade" >
                                <property name="baseSiteService" ref="baseSiteService"/>
                                <property name="paymentSubscriptionResultDataConverter" ref="paymentSubscriptionResultDataConverter"/>
                                <property name="siteConfigService" ref="siteConfigService"/>
                                <property name="paymentService" ref="acceleratorPaymentService"/>
                                <property name="siteBaseUrlResolutionService" ref="siteBaseUrlResolutionService"/>
                                <property name="userService" ref="userService"/>
                                <property name="checkoutCustomerStrategy" ref="checkoutCustomerStrategy"/>
                                </bean>
```

Mocked Configuration

Learn about the configuration options for the Mocked HOP and SOP implementation in the SAP Commerce Accelerator and the affected extensions.

Implementation of payment requires following options to be configured:

- Generic Payment configuration, described in the Generic Configuration document.
- Mocked Configuration, described in this document.

acceleratorservices Extension

This extension contains an additional web extension that is used to drive the Mocked Hosted Order Page in SAP Commerce Accelerator.

Mocked HOP Configuration

The `acceleratorservices` extension currently contains one controller, `HostedOrderPageMockController`, for handling the HOP. This controller is an annotation configured controller which contains several properties worth mentioning. You can see the main ones with the `@Resource` annotation in the code extract below:

```
@Controller
                                        @RequestMapping("/hop-mock")
                                        public class HostedOrderPageMockController
                                        {
                                        ...   ...   ...

                                        @Resource(name = "supportedCardTypes")
                                        private Map<String, String> supportedCardTypes;
```

```
@Resource(name = "defaultPaymentDetailsForm")
private PaymentDetailsForm defaultPaymentDetailsForm;

...   ...   ...
}
```

You can find the bean definitions for the two controller properties above in the ***${HYBRIS_BIN_DIR}***`ext-accelerator/acceleratorservices/web/webroot/WEB-INF/config/hop-mock-config.xml` file. The table below, explains what each property is and how they are configured.

Property	Description	Configuration Option
<supportedCardTypes>	Maintains a `MapString, String` of supported credit card types where each map entry contains a code and a description of the card type, for example: `001, Visa`. This collection of credit card types are the only ones shown on the Mocked Hosted Order Page in the Credit Card Type drop down list shown in the image below: 	`<util:map id="supportedCardTypes" key-type="java.lang.String" value-type="java.lang.String">` `<entry key="001" value="Visa"/>` `<entry key="002" value="MasterCard"/>` `<entry key="003" value="American Express"/>` `<entry key="005" value="Diners Club"/>` `<entry key="024" value="Maestro (UK Domestic)"/>` `</util:map>`
<defaultPaymentDetailsForm>	This is a custom form model object used to hold the default credit card values that are pre-populated on the Mocked Hosted Order Page. You can change the values for the properties in the bean definition to suite your demo.	`<bean id="defaultPaymentDetailsForm" class="de.hybris.platform.acceleratorservices.web.hostedorderpage.forms.PaymentDeta` `<property name="cardTypeCode" value="001"/>` `<property name="cardNumber" value="41111111111111` `<property name="verificationNumber" value="123"/>` `<property name="issueNumber" value="01"/>` `<property name="startMonth" value="1"/>` `<property name="startYear" value="#{T(java.util.` `<property name="expiryMonth" value="12"/>` `<property name="expiryYear" value="#{T(java.util` `</bean>`

HOP Mock performs URL validation for payment response and merchant callback URLs. If the `properties acceleratorservices.payment.response.url.allowlist` and `acceleratorservices.merchant.callback.url.allowlist` are not defined, the system evaluates a set of default allowed values. These default values are usually based on all possible base site URLs.

In a case where you need to provide an alternative host for these URLs, you must first populate this allowlist.

ℹ Note

The default list stops working once you populate the allowlist. You must provide all possible values in the allowlist, once you decide to edit it.

This validation is not required if you are in a dev setup environment are security problems are not of concern. In this instance, you can simply disable it by executing`acceleratorservices.payment.url.strict.enabled=false`.

This configuration is defined in the *<${HYBRIS_BIN_DIR}>*`/ext-templates/acceleratorservices/project.properties` file as shown in the extract below:

```
############################
### Configuration to enable or disable strict checking of payment and merchant urls (to prevent redirection attacks)
############################
acceleratorservices.payment.url.strict.enabled=true
```

```
############################
### Comma separated list of allowed payment hosts (to prevent redirection attacks)
### only works when acceleratorservices.payment.url.strict.enabled=true
### example: https://localhost:9002,https://electronics.local:9002
############################
acceleratorservices.payment.response.url.allowlist=

############################
### Comma separated list of allowed merchant callback and merchant extended callback hosts (to prevent redirection attacks)
### only works when acceleratorservices.payment.url.strict.enabled=true
### example: https://localhost:9002,https://electronics.local:9002
############################
acceleratorservices.merchant.callback.url.allowlist=
```

By default, the HOP Mock is disabled. To activate the HOP Mock, set the property `acceleratorservices.payment.hopmock.enabled` to *<true>*.

```
acceleratorservices.payment.hopmock.enabled=true
```

Mocked SOP Configuration

The `acceleratorservices` extension contains an additional web extension which is used to drive the Mocked Silent Order Post in the SAP Commerce Accelerator. For handling the SOP it contains one controller `SilentOrderPostMockController`.

```
@Controller
                                    @RequestMapping("/sop-mock")
                                    public class SilentOrderPostMockController
                                    {
                                    ...   ...   ...

                                    @Resource(name = "sopPaymentDetailsValidator")
                                    private SopPaymentDetailsValidator sopPaymentDetailsValidator;

                                    ...   ...   ...
                                    }
```

As you can see in the code above the `SilentOrderPostMockController` has `sopPaymentDetailsValidator` that is used to validate data provided by customer in the online storefront.

SOP Mock also performs URL validation for payment response and merchant callback URLs. It shares the same properties as the HOP Mock URL validation.

By default, the SOP Mock is disabled. To activate the SOP Mock, set the property `acceleratorservices.payment.sopmock.enabled` to *<true>*. The property is defined in the `<${HYBRIS_BIN_DIR}>/ext-templates/acceleratorservices/project.properties` file.

```
acceleratorservices.payment.sopmock.enabled=true
```

yacceleratorstorefront Extension

This extension contains the main configuration option that allow the HOP or SOP request to go to the Mocked HOP or SOP, respectively.

Mocked HOP Configuration

The *<hop.post.url>* property defines the URL for the Hosted Order Page, in this case the Mocked HOP. All parameters which contain order details are sent to this URL for processing.

i Note

Remember to use the Mocked HOP for demo or development purposes only. Also when the Mocked HOP is used then the Mocked Payment provider that comes with the SAP Commerce Accelerator must be used since mocked subscription IDs cannot be used against a production payment provider.

This configuration is defined in the *<${HYBRIS_BIN_DIR}>*`/ext-templates/yacceleratorstorefront/project.properties` file as shown in the extract below:

```
############ Hosted Order Page settings ###############
                                    ##########################################################
                                    ##### Common Properties
                                    hop.post.url=/acceleratorservices/hop-mock
```

Mocked SOP Configuration

The *<sop.post.url>* property defines the URL for the Silent Order Post, in this case the Mocked SOP. All parameters which contain order details are sent to this URL for processing.

i Note

Remember to use the Mocked SOP for demo or development purposes only. Also when the Mocked SOP is used then the Mocked Payment provider that comes with the SAP Commerce Accelerator must be used since mocked subscription IDs cannot be used against a production payment provider.

This configuration is defined in the *<${HYBRIS_BIN_DIR}>*`/ext-templates/yacceleratorstorefront/project.properties` file as shown in the extract below:

```
sop.post.url=/acceleratorservices/sop-mock/process
```

Checkout PCI Strategies

This document provides a brief overview of the checkout PCI strategies offered for customers to create a credit card payment subscription in their account that they can use to pay for purchases. However, it mainly looks at how these strategies should be managed for the implementation required by your project.

→ Tip

The expectation is any project implemented using SAP Commerce Accelerator removes the ability for the customer to decide which checkout PCI strategy is used and instead adapts the Checkout PCI Strategy required.

Checkout PCI Strategies Overview

The B2C storefronts offered with Commerce Accelerator are currently shipped with a number of methods for collecting customer payment details, called **"Creating a customer subscription"**:

- A payment details form for capturing a customer's credit card details (Single Page checkout only).
- A secure page hosted by a third party payment provider used for collecting a customer's card details, that is a Hosted Order Page (HOP).
- A payment details form provided in Accelerator that POSTs to an external service for creating the subscription, this is known as Silent Order Post (SOP).

We discuss how the `subscriptionPciOption` is chosen for each of the B2C storefronts.

ⓘ Note

The checkout PCI strategies can only be used during the multi-step checkout experience.

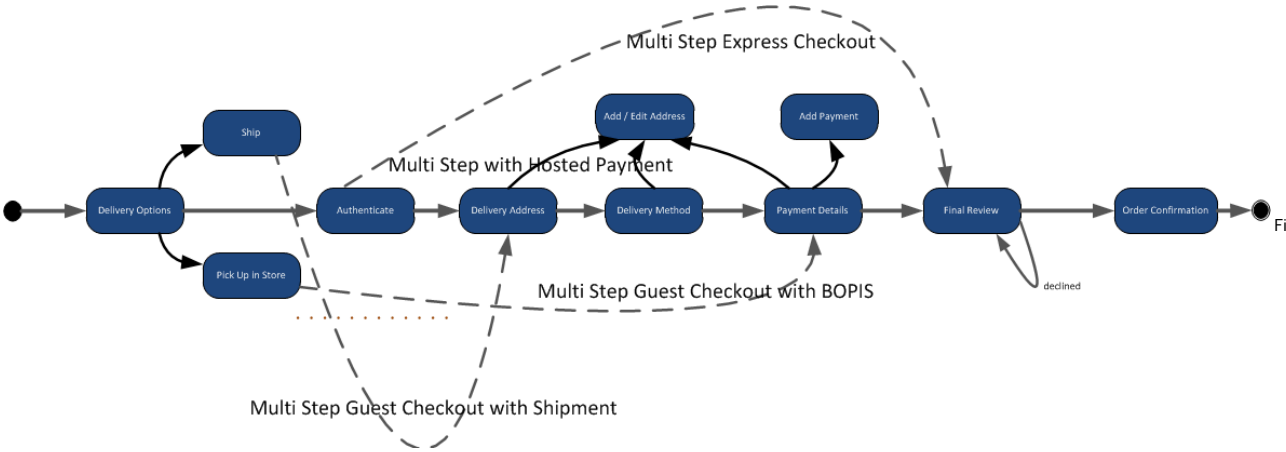


Figure: Multi-step checkout page flow

Before the multi-step checkout flow is initiated, the following conditions must be fulfilled:

- Customer adds items to the cart.
- Customer clicks the **Checkout** button.
- If the user is not already logged in, they have to authenticate.

In the multi-step checkout pages flow, the order of data entry is dictated by the checkout journey. At each stage, the customer has the option to return to a previous step in the journey.

- If the customer has no delivery address, the **Add/Edit Address** page is shown for a new address to be entered. If, however, the customer has one or more addresses he is taken to the **Select Delivery Address** page where one of the addresses listed can be selected as the delivery address.
- On submitting the delivery address details the customer is taken to the **Delivery Method** page for choosing the desired available delivery method.
- The customer is then taken to the **Payment Method** page where payment details can either be added or selected for payment.
    - If the customer has saved payment cards, he can select one and proceed to the next step.
    - If the customer has no saved cards or wants to create a new one, it is at this point that the `MultiStepCheckoutController` queries the `Checkout Facade` for the `SubscriptionPciOption`. Depending on the results, the customer is either taken to the Hosted Order Page or a payment information form where the new card details are to be entered.
- The user is taken to the **Checkout Summary** page on submitting the **Payment Method** page.
- Once the customer is satisfied with the details provided, he can click the **Place Order** button to submit payment for the items in the cart.
- Following successful payment, the customer is directed to the **Order Confirmation Page**.

In the multi-step checkout pages flow the customer can also select the **Express Checkout** option where the customer starts the checkout process on **Proceed To Checkout Page** and next they are taken to the **Checkout Summary** page. Therefore, the customer skips the pages in-between where the required fields are filled within the values saved in the account profile.

For more information, see Managing Multi Step Checkout Strategies.

Implemented Checkout PCI Strategies

Currently there are three Checkout PCI Strategies in the Accelerator. Two are fixed strategies specified by a `CheckoutPciOptionEnum`, the other is a dynamic one in that it defaults to one of the fixed strategy if its selection criteria is not met.

Checkout PCI Strategy	PCI Selection Type	CheckoutPciOptionEnum Value
`configuredCheckoutPciStrategy`	Dynamic	• `DEFAULT` • `HOP` • `SOP`

Checkout PCI Strategy	PCI Selection Type	CheckoutPciOptionEnum Value
wsCheckoutPciStrategy	Fixed	DEFAULT
hopCheckoutPciStrategy	Fixed	HOP

These enumerations are defined in the acceleratorservices extension in the acceleratorservices/resources/acceleratorserivces-items.xml file as follows:

```
<enumtype generate="true" code="CheckoutPciOptionEnum"
                              autocreate="true" dynamic="true">
                              <value code="Default" />
                              <value code="HOP" />
                              <value code="SOP" />
                              </enumtype>
```

Currently we use the dynamic configuredCheckoutPciStrategy strategy for selecting the Subscription PCI Option used during checkout. This strategy simply checks which strategy is configured for the currently running CMS site.

→ Tip

The appearance of the checkout strategy selector is now controlled on the site level with system property. The following example from the project.properties file shows the configuration keys are:

```
site.pci.strategy.electronics.Desktop=SOP
                                    site.pci.strategy.apparel-uk.Desktop=SOP
                                    site.pci.strategy.apparel-de.Desktop=SOP
```

The default behavior is set to show the selector in the cart page. You can change the option on the fly using the SAP Commerce Administration Cockpit in the Platform Configuration section.

The table below shows the implemented checkout strategies used for each B2C storefront in Accelerator:

Storefront	Available Checkout Flow	Available Subscription PCI Option
Desktop B2C Electronics	• Multi Step • Multi Step with PCI	• DEFAULT • HOP • SOP
Desktop B2C Apparel UK	• Multi Step • Multi Step with PCI	• DEFAULT • HOP • SOP
Desktop B2C Apparel DE	• Multi Step • Multi Step with PCI	• DEFAULT • HOP • SOP

Checkout PCI Strategy Configuration

This document looks at the configuration options for the checkout PCI strategy configurations within SAP Commerce Accelerator and the affected extensions.

SAP Commerce Accelerator offers different checkout PCI strategies. You can find a description of implemented strategies in the Checkout PCI Strategies document.

Checkout PCI Strategy Configuration

You can easily configure the strategies, described in the Implemented Checkout PCI Strategies section of the Checkout PCI Strategies, by making some changes to several Spring configuration files. This section describes which extension and related files require configuration.

yacceleratorfacades Extension

This extension provides a checkout flow facade to the storefronts for choosing the checkout PCI strategy to use.

→ Tip

If the entire proposed default Checkout PCI Strategy needs to be replaced or removed this is where it is done. This is the highest level where configuration can be done. Finer grained configurations can be done via the yacceleratorcore extension. For more information, see the **yacceleratorcore Extension** section below.

The checkoutFlowFacade bean is defined in the *<${HYBRIS_BIN_DIR}>*/ext-templatesyacceleratorfacades/resources/yacceleratorfacades-spring.xml file. This bean has a *<checkoutFlowStrategy>* property that contains the logic for selecting a checkout page flow to use during checkout. It also contains a *<checkoutPciStrategy>* property that contains the logic for selecting the checkout PCI option. In the reference implementation it is the *<SubscriptionPciOption>* property.

```
<alias name="defaultCheckoutFlowFacade" alias="checkoutFlowFacade"/>
                                    <bean id="defaultCheckoutFlowFacade" class="de.hybris.platform.yacceleratorfacades.flow.impl.DefaultCheckoutI
                                    <property name="checkoutFlowStrategy" ref="checkoutFlowStrategy"/>
                                    <property name="checkoutPciStrategy" ref="checkoutPciStrategy"/>
                                    </bean>
```

The SessionOverrideCheckoutFlowFacade, which allows the CheckoutFlow and the SubscriptionPciOption to be overridden in the session on the shopping cart page, is also defined in the mentioned file. This is typically done for demonstration purposes and you may not need this behavior in your environment, in which case the defaultCheckoutFlowFacade

should be sufficient.

The following example is from the `yacceleratorfacades-spring.xml` file.

```
<alias name="sessionOverrideCheckoutFlowFacade" alias="checkoutFlowFacade"/>
                                        <bean id="sessionOverrideCheckoutFlowFacade" class="de.hybris.platform.yacceleratorfacades.flow.impl.SessionO
                                        <property name="sessionService" ref="sessionService"/>
                                        </bean>
```

yacceleratorcore Extension

This extension provides all the actual checkout PCI strategies implemented by the SAP Commerce Accelerator storefronts.

→ Tip

Here finer grained configuration can be done depending on your project requirements.

Related files:

- *<${HYBRIS_BIN_DIR}>*`/ext-templates/yacceleratorcore/resources/yacceleratorcore-items.xml`
- *<${HYBRIS_BIN_DIR}>*`/ext-templates/yacceleratorcore/resources/yacceleratorcore-spring.xml`

The Main PCI Strategy

The PCI option to use during checkout is determined by the *<checkoutPciStrategy>* property of the `defaultCheckoutFlowFacade`.

→ Tip

This property can be set to any of the fixed checkout PCI strategies or the `configuredCheckoutPciStrategy`, all discussed in the Fixed Checkout PCI Strategies and Dynamic Checkout PCI Strategies sections.

The following example is from the `yacceleratorcore-spring.xml` file.

```
<alias alias="checkoutPciStrategy" name="configuredCheckoutPciStrategy"/>
```

Fixed Checkout PCI Strategies

As discussed in the Implemented Checkout PCI Strategies section of the Checkout PCI Strategies document, the two fixed strategies are:

- `wsCheckoutPciStrategy`
- `hopCheckoutPciStrategy`

These strategies are identified by the `FixedCheckoutPciStrategy` defined in its *<subscriptionPciOption>* property. The enum is defined in the *<${HYBRIS_BIN_DIR}>*`/ext-templatesyacceleratorcore/resources/yacceleratorcore-items.xml` file.

→ Tip

In the following example, additional enum values can be defined for any extra PCI Options.

```
<enumtypes>
                                        <enumtype generate="true" code="CheckoutPciOptionEnum" autocreate="true" dynamic="true" >
                                        <value code="Default"/>
                                        <value code="HOP"/>
                                        <value code="SOP"/>
                                        </enumtype>
                                        </enumtypes>
```

The code extract below from the `yacceleratorcore-spring.xml` file contains the bean definition for the fixed checkout PCI strategies.

→ Tip

Here additional strategies can be defined for any extra functionality that cannot be implemented using the defaults. It is also here that additional PCI options can be added and example would be a Payment PCI Option that determines the PCI option to use when making payments during checkout. That is HOP instead of the currently used webservice option.

```
<bean id="wsCheckoutPciStrategy" class="de.hybris.platform.yacceleratorcore.checkout.pci.impl.FixedCheckoutPciStrategy">
                                        <!-- Use the Payment extension web services to create credit card details -->
                                        <property name="subscriptionPciOption" value="DEFAULT" />
                                        </bean>
                                        <bean id="hopCheckoutPciStrategy" class="de.hybris.platform.yacceleratorcore.checkout.pci.impl.FixedCheckoutP
                                        <!-- Use the Hosted Order Payment to create credit card details -->
                                        <property name="subscriptionPciOption" value="HOP" />
                                        </bean>
```

Dynamic Checkout PCI Strategies

As discussed in the Implemented Checkout PCI Strategies section of the Checkout PCI Strategies document, there is only one dynamic strategy, `configuredCheckoutPciStrategy`.

The following code extract from the `yacceleratorcore-spring.xml` file contains the bean definition for the `configuredCheckoutPciStrategy`.

→ Tip

Here the `defaultCheckoutPciStrategy` can be changed to any of the fixed strategies. This is the fallback strategy to use if its internal checks fail.

```
<!-- This strategy checks which PCI strategy is set in properties (site.pci.strategy).
                                    You can set one of the option defined in CheckoutPciOptionEnum (for example site.pci.strategy=SOP, site.pci.
                                    <bean id="configuredCheckoutPciStrategy" class="de.hybris.platform.yacceleratorcore.checkout.pci.impl.Config
                                    <property name="siteConfigService" ref="siteConfigService"/>

                                    <!-- Default to the wsCheckoutPciStrategy -->
                                    <property name="defaultCheckoutPciStrategy" ref="wsCheckoutPciStrategy" />
                                    </bean>
```

Related Information

yacceleratorcore Extension
yacceleratorfacades Extension

CMS Actions

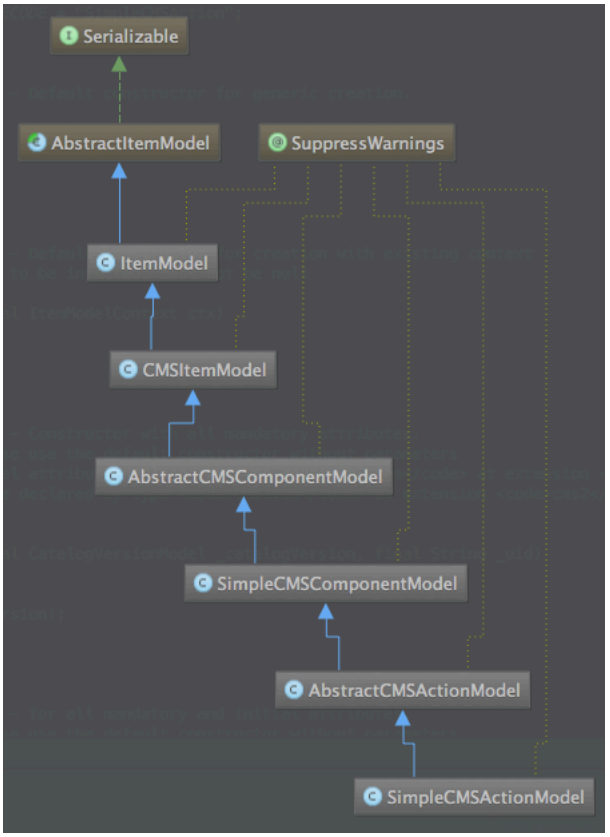
CMS actions add functionality to CMS components. You can use actions on a component that performs an operation, such as modifying the cart, or linking to another page. The add-to-cart button or the pick-up-in-store button are examples of such actions.

i Note
An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Overview

CMS actions are related to CMS components in a many-to-many relationship. Actions are evaluated and rendered when the CMS component is rendered through a CMS tag library, such as `de.hybris.platform.acceleratorcms.tags2.CMSComponentTag`. A CMS action can be restricted to one or more types of `CMSComponentType` through the `ApplicableCmsActionsTypeForCmsComponent` relation. The CMS action and related relationships are defined by the `acceleratorcms` extension.

The following is a UML diagram of the `SimpleCMSActionModel`:



The following are examples of action implementations:

```
PickUpInStoreActionModel (Action for Pick Up In Store button
ShareOnSocialNetworkActionModel (Action for social network widget)
AddToCartActionModel (Action for add to cart button)
SimpleCMSActionModel (Base action for all action implementations
```

Creating a New CMS Action

To create a new CMS action item:

1. Define the item type, as follows:

```
<itemtype code="MyNewAction" jaloclass="de.hybris.platform.acceleratorcms.jalo.actions.MyNewAction" extends="SimpleCMSAction"   autocreate="tru
</itemtype>
```

2. Add the action view to your storefront extension, as shown in the example from the `/yacceleratorstorefront/web/webroot/WEB-INF/views/desktop/cms/mynewaction.jsp` file:

```
<%@ page trimDirectiveWhitespaces="true" %>
<button type="button">Click Me!</button>
```

When creating a new CMS Action, if you create a .js file related to your action, make sure to include the corresponding js files in the `yacceleratorstorefront/web/webroot/WEB-INF/wro.xml` file by specifying them as below:

```
<js>/_ui/responsive/common/js/cms/addtocartaction.js</js>
<js>/_ui/responsive/common/js/cms/listaddtocartaction.js</js>
<js>/_ui/responsive/common/js/cms/listorderformaction.js</js>
<js>/_ui/responsive/common/js/cms/listpickupinstoreaction.js</js>
<js>/_ui/responsive/common/js/cms/pickupinstoreaction.js</js>
<js>/_ui/responsive/common/js/cms/vieworderaction.js</js>
<js>/_ui/responsive/common/js/cms/viewstoreaction.js</js>
```

3. Add the action to a CMS component. The following example ImpEx shows how you would add your new action to the `ProductAddToCartComponent` in the Accelerator.

```
# this relation is used by CMSActionRestriction, by adding ActionRestriction to your action
INSERT_UPDATE ApplicableCmsActionsTypeForCmsComponent;target(code)[unique=true];source(code)[unique=true];
;ProductAddToCartComponent;MyNewAction;

# CMS Action Restrictions
INSERT_UPDATE CMSActionRestriction;$contentCV[unique=true];uid[unique=true];name;
;;ActionRestriction;A CMSAction restriction;

# CMS ProductAddToCart Components
INSERT_UPDATE ProductAddToCartComponent;$contentCV[unique=true];uid[unique=true];name;actions(&actionRef);&componentRef
;;AddToCart;Product Add To Cart;PickUpInStoreAction,AddToCartAction,ShareOnSocialNetworkAction,MyNewAction;AddToCart
# My New Action,
INSERT_UPDATE MyNewAction;$contentCV[unique=true];uid[unique=true];url;name;restrictions(uid,$contentCV);&actionRef
;;MyNewAction;/mynewaction/act/{productCode};My New Action;ActionRestriction;MyNewAction
```

4. Add the action to the appropriate CMS component view if it is not already present, as shown in the example from the `/yacceleratorstorefront/web/webroot/WEB-INF/tags/desktop/product/productAddToCartPanel.tag` file:

```
...
<%@ taglib prefix="action" tagdir="/WEB-INF/tags/desktop/action" %>
...
<div id="actions-container-for-${component.uid}" class="productAddToCartPanelContainer clearfix">
        <ul class="productAddToCartPanel clearfix">
                <action:actions element="li" styleClass="productAddToCartPanelItem span-5" parentComponent="${component}"/>
        </ul>
</div>
```

i Note

The `parentComponent` attribute is required to properly evaluate the restrictions on the CMS action.

Related Information

acceleratorcms Extension - Technical Guide
acceleratorservices Extension - Technical Guide
cms2 Extension - Technical Guide
cms2lib Extension - Technical Guide

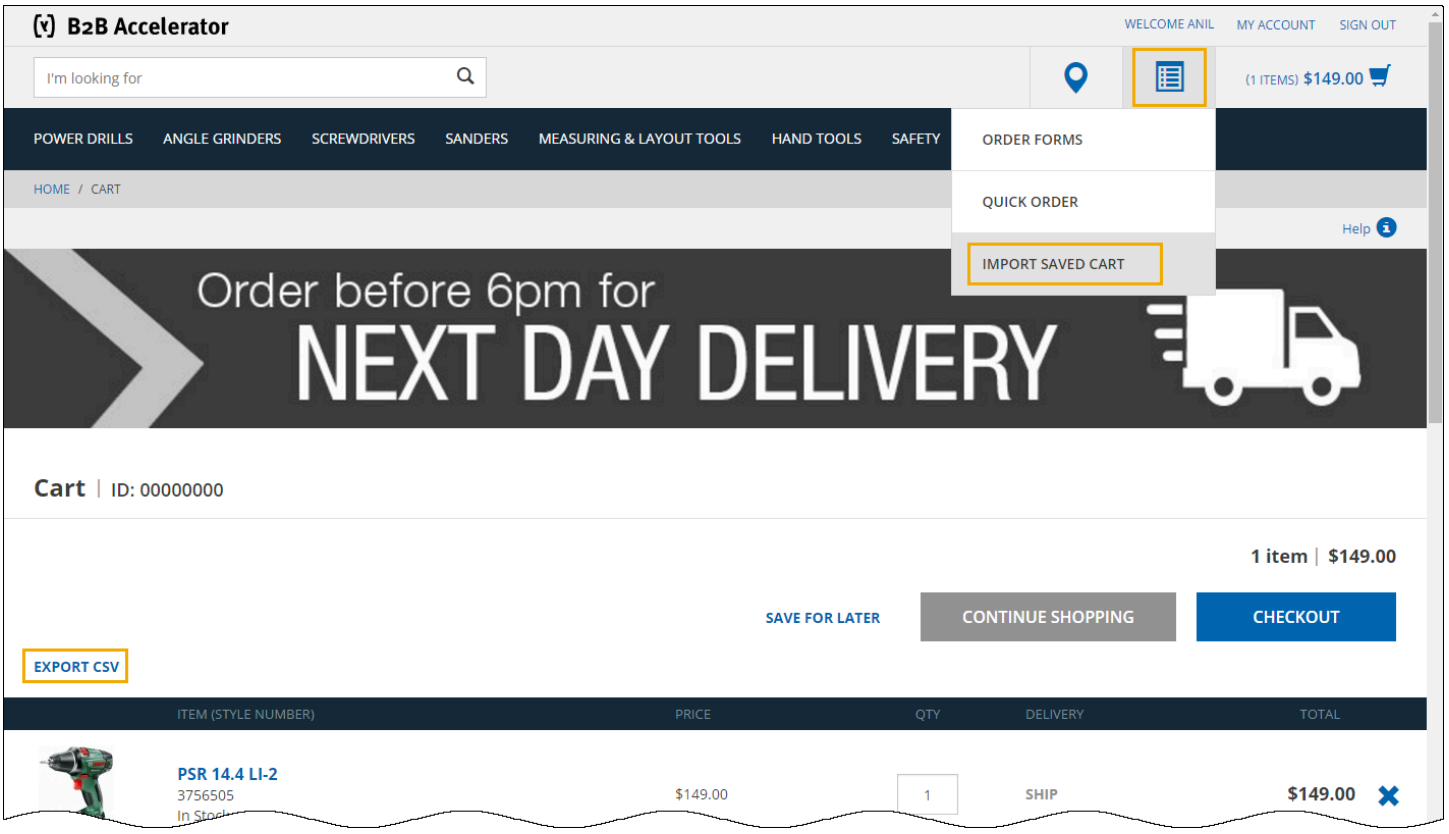
CSV Import and Export

Users can export and import their carts as a CSV file so they can view or adjust their cart outside the storefront.

Users can export their active cart to a CSV file by clicking the **Export CSV** link in the cart page; they do not need to be logged into the storefront to export carts.

To import carts, users must be logged into the storefront. Users import carts by clicking the **Order Tools** icon ( ) and then selecting **Import Saved Cart**. Users then choose the file to upload, and if the file is valid, a new saved cart is created from the CSV file (note that it does not become the active cart when imported).

This screenshot shows the UI controls to import and export carts:



All imported carts can be viewed in the Saved Carts page. The imported cart's description includes the number of lines that were imported successfully, the number of lines where the quantity was adjusted due to low stock, and the number of lines that were not imported. Details of the import log can be viewed in the Backoffice Administration Cockpit. For more information, see Viewing CSV Import Error Messages.

Backend services and facades to convert CSV files into carts and vice versa are implemented in the `coreaccelerator` module. Controllers to handle relevant UI requests are added in `yacceleratorstorefront`. CSV import and export is enabled for all storefronts included in B2B Accelerator and B2C Accelerator.

CSV File Technical Details

The default encoding of the CSV file is UTF-8 as specified in the JVM system property file, `file.encoding`. The encoding can be modified, but ensure that changing the encoding for the entire platform does not negatively affect other modules.

When a cart is exported, the SKU, quantity, product name and unit price are saved in the CSV file, but only the SKU and quantity fields are used when importing the file. Here's an example of the contents of the CSV file:

```
SKU,Quantity,Description,Price
00123,2,test sku,$100.00
00124,3,test sku 2,$200.00
00125,3
XC60S90,8,
XC60S90,9
```

The default delimiter for the file is a comma. You can change the delimiter by changing the value of the `acceleratorservices.savedcartupload.file.delimiter` property in your `local.properties` file.

Two properties specify the maximum file size allowed for importing. Both these properties are contained in the `project.properties` file in `yacceleratorstorefront`.

The `import.csv.file.max.size.bytes` property defines the maximum size in bytes of the CSV file that could be uploaded. This limit is checked on both client and server sides and it is also displayed on the Import CSV page (converted to Bytes/KB/MB/GB as needed). The default limit size is 10 KB. When changing this limit, ensure that the system performance is not drastically affected. If the value for this property is not set, a fallback value of 0 is used, which triggers the rejection of the uploaded CSV file.

The `import.csv.max.upload.size.bytes` property defines the maximum allowed size in bytes of the upload request. In addition to the CSV file, other parameters are required in the upload request, such as a CSRF token. Set this limit appropriately to support the desired size of the uploaded CSV file. Note that the size of the upload request varies between browsers. Typically, a value that is 1 KB more than `import.csv.file.max.size.bytes` is appropriate. The `import.csv.max.upload.size.bytes` property is used to set the `maxUploadSize` property of the `org.springframework.web.multipart.commons.CommonsMultipartResolver` resolver used for the CSV file upload.

Export Process

The `exportCart.tag` file in `yacceleratorstorefront` defines the Export CSV link, and is referenced by `cartDisplay.jsp` to add the link to the cart page.

The `exportCsvFile()` handler method in `CartPageController` for `yacceleratorstorefront` is invoked when the Export CSV link is clicked. The handler method generates the localized headers of the CSV file (SKU, Quantity, Name, and Price), and calls `CsvFacade` to convert the line items of the active session cart into a CSV file for download.

The `CsvFacade` and its default `DefaultCsvFacade` implementation are defined in the `acceleratorfacades` extension. The `DefaultCsvFacade` converts each cart line item into a CSV file entry. The line separator (EOL symbol) is `\n` (Unix style), and the field delimiter is a comma. If more sophisticated logic is needed to meet clients' business needs, you can use your own implementation of the `CsvFacade` interface and overwrite the `CsvFacade` alias (the default alias assignment is specified in `acceleratorfacades-spring.xml`).

Import Process

The import process is triggered when users select **Import Saved Cart** from the **Order Tools** menu. The Import Saved Cart page (`importCSVSavedCartPage.jsp`) loads, together with the relevant Javascripts defined in the `yacceleratorstorefront` extension. On this page, users browse and select a CSV file to import through an HTTP file upload POST request.

The `yacceleratorstorefront` extension includes a `FileUploadFilter` that is used to configure a filter and resolver to process multipart HTTP POST file upload requests for specific configurable URLs (through `urlFilterMapping`). This allows for different URLs to support different file size limits. You can find an example of the configuration in `spring-filter-config.xml` in the `yacceleratorstorefront` extension. For CSV import, `importCSVMultipartResolver` is configured for the `/import/csv/*` URL pattern. The `maxUploadSize` property of this resolver limits the size of the upload request, and its value is defined in `import.csv.max.upload.size.bytes`.

The POST request is then mapped to `ImportCSVSavedCartForm` (defined in the `acceleratorstorefrontcommons` extension) and handled by `ImportCSVPageController` (defined in `yacceleratorstorefront`). The controller uses `importCSVSavedCartFormValidator` (defined in `acceleratorstorefrontcommons`) to validate the actual file size. The value of the maximum file size is specified in the `import.csv.file.max.size.bytes` property. A similar check based on the value of this property is also done on the UI (webpage) side. In addition to the checks for maximum file and upload request size, checks are also performed on both the client side (HTML 5) and server side to ensure the imported file has a `.csv` file extension, and that one of the following MIME (content) types is used:

- `text/csv` for MacOS
- `application/vnd.ms-excel` for Windows
- `text/comma-separated-values` for Android

If you want to use another MIME type to verify the imported `.csv` file, you can add the required MIME type to the following files:

- `acceleratorstorefrontcommons/commonweb/src/de/hybris/platform/acceleratorstorefrontcommons/forms/validation/ImportCSVSavedCartFormValic`
- yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/ybase-0.1.0/js/acc.csv-import.js

The controller then calls `SavedCartFileUploadFacade` (defined in the `acceleratorfacades` extension), whose default implementation, `DefaultSavedCartFileUploadFacade`, in turn creates media using the file and publishes a `SavedCartFileUploadEvent`. The event is handled by `SavedCartFileUploadEventListener` in an asynchronous manner and triggers `savedCartFileUPloadProcess`. Once started, the process invokes `SavedCartFromUpladFileAction` (in `acceleratorservices`), which then creates an empty saved cart and calls `SavedCartFileUploadStrategy` (in `acceleratorservices`) to parse the CSV file line by line to convert the lines into cart line items.

If `DefaultSavedCartFileUploadStrategy` parses the CSV file in a way that doesn't meet your business needs, you can create your own implementation of the `SavedCartFileUploadStrategy` interface and overwrite the Spring alias `savedCartFileUploadStrategy` (the default alias assignment is defined in `acceleratorservices-spring.xml`). Similarly, you can also replace `DefaultSavedCartFileUploadFacade` with your own implementation if you need to customize the entire file processing logic.

Related Information

Adding the Saved Cart Field in Advanced Search
Viewing CSV Import Error Messages

Adding the Saved Cart Field in Advanced Search

To effectively search for the business process in the Backoffice Administration Cockpit, a search field must be manually added through configuration files.

Context

The business process provides details on any errors that occurred when importing a cart as a CSV file. The **Saved cart** field must be added as an advanced search field in the Backoffice so that the corresponding business process for the imported cart can be quickly found. The SAP Commerce server should be running for this procedure.

Procedure

1. Add the following code in `acceleratorservices-backoffice-config.xml` (located in *<HYBRIS_BIN>*`/ext-accelerator/acceleratorservices/resources`):

```
<context type="SavedCartFileUploadProcess" component="advanced-search" merge-by="type" parent="BusinessProcess">
    <advanced-search:advanced-search xmlns:advanced-search="http://www.hybris.com/cockpitng/config/advancedsearch">
        <advanced-search:field-list>
            <advanced-search:field xmlns="http://www.hybris.com/cockpitng/config/advancedsearch" name="savedCart" selected="true"/>
        </advanced-search:field-list>
    </advanced-search:advanced-search>
</context>
```

2. Log into the Backoffice.
3. Press F4 to open edit mode.
4. Click the SAP Commerce logo ( ) at the top-right corner of the page, and then select **Reset Everything**.
5. Press F4 to exit edit mode.

Results

The **Saved cart** field is added as an advanced search field. Enter the cart ID in this field to find the business process for that cart. For more details, see Viewing CSV Import Error Messages.

Related Information

CSV Import and Export
Viewing CSV Import Error Messages

Viewing CSV Import Error Messages

Use the Backoffice Administration Cockpit to view details on items that were not imported successfully from a CSV file.

Prerequisites

You must add a specific advanced search field to the Backoffice to use this procedure. For instructions, see Adding the Saved Cart Field in Advanced Search.

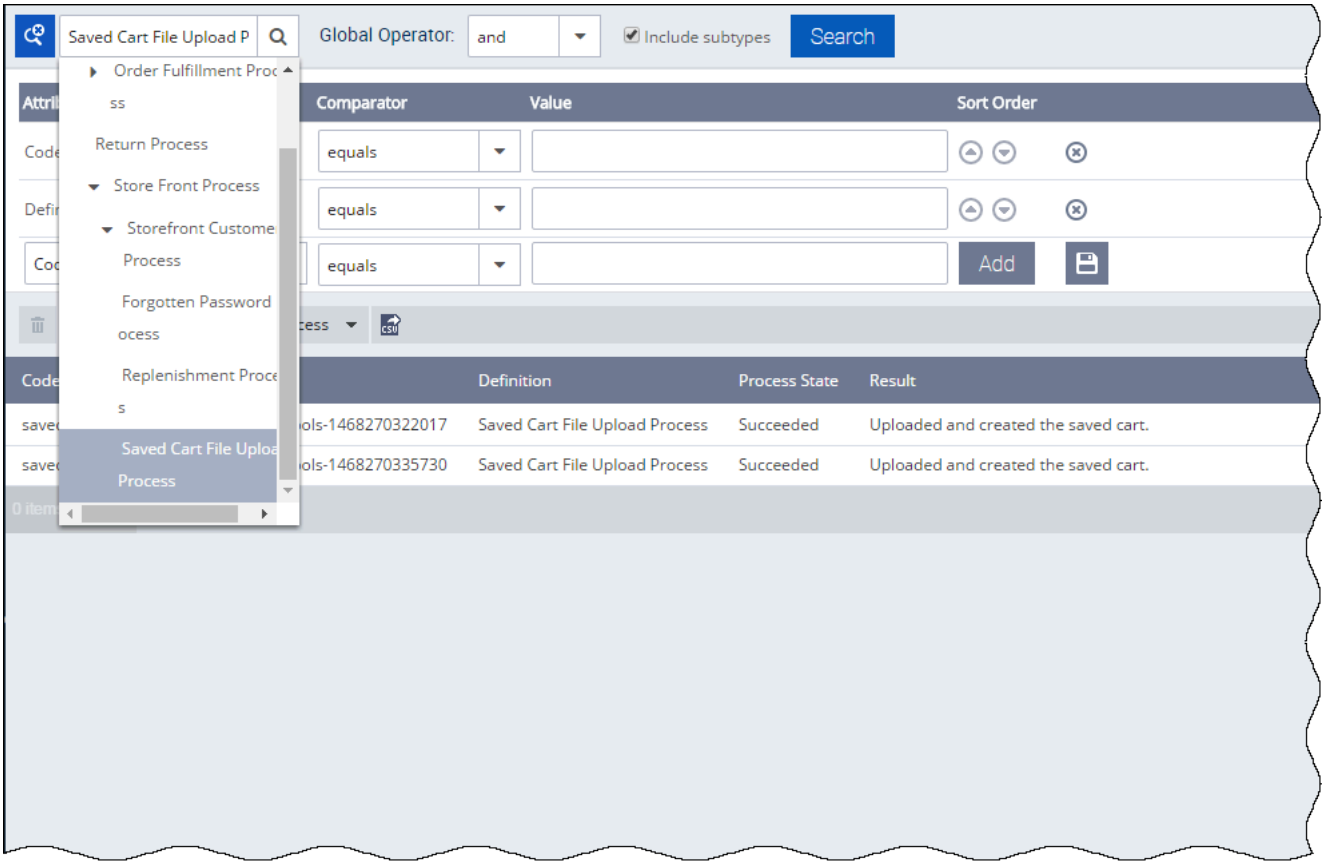
Context

When importing a cart using a CSV file, some items may not be imported as expected. If the stock level of an item is lower than the quantity specified in the CSV file, then the quantity is adjusted to the current stock level upon import. Items are not imported if the item number does not exist or there is no stock for a valid item number. A summary is provided in the description of the saved cart, but it does not provide the details of the import errors. Details of the import errors are available in the Backoffice.

Procedure

1. Log into the Backoffice.
2. Navigate to System > Business Processes > Business Processes.
3. In the top bar, click the Switch search mode icon ( ).

   The search mode is changed and Business process appears in the search box.
4. Click Business Process in the search box, then expand Business Process > Storefront Process > Storefront Customer, and then select Saved Cart File Upload Process.



   The Advanced Search fields are updated.
5. In the Saved Cart search field, enter the cart ID (not the cart name) and Click Search.

   The business process for that cart ID appears in the search results.
6. Click the business process to open its details.
7. In the Process Task Logs field, double-click the import error log to open its details.
8. In the Messages field, all items that could not be imported or whose quantities were adjusted are listed in this field.

Results

In the Messages field, all items that could not be imported or whose quantities were adjusted are listed.

Related Information

CSV Import and Export
Adding the Saved Cart Field in Advanced Search

Backoffice Operations

Various Backoffice operations are available for SAP Commerce Accelerator that you can use for customer support, configuring products, and setting up the online store layout.

Refer to the following topics for details on each Backoffice module.

Product Cockpit

SAP Commerce Accelerator provides the fully functional Product Cockpit, enhanced with new components and features. The Product Cockpit is completely integrated in a way that is familiar to SAP Commerce users.

Moderating Reviews

SAP Commerce Accelerator provides a customer review functionality on the storefront. Each review consists of rating, review title, and review description. All of these are mandatory fields for a customer.

Storefront Configuration

Use the Backoffice Administration Cockpit for Accelerator storefront configurations, including base store settings, languages and territories, tax settings, and email templates.

Product Cockpit

SAP Commerce Accelerator provides the fully functional Product Cockpit, enhanced with new components and features. The Product Cockpit is completely integrated in a way that is familiar to SAP Commerce users.

⚠ Caution

This page refers to deprecated software. For further details, see Deprecation Status.

All the functions that are available for the editing of products in the Product Cockpit are also applicable for the Accelerator. For more information about how to use the Product Cockpit, see Product Cockpit.

You can also find more detailed information in the following topics:

- Getting Started with the Product Cockpit
- Product Perspective
- Catalog Perspective

What is More in the Accelerator?

Managing Extended Product Images

You have more ways to manage product images. Each product can have multiple images of various sizes for defined purposes. For more information, go to the Managing Media in the SAP Commerce Product Cockpit document.

Creating User-Friendly URLs

You can define the URL for the product page using the product name. For more information, see SEO URL Tutorial.

Managing Additional Attributes and Multiple Variants

SAP Commerce Accelerator helps you manage multiple product variants, such as style and size. You just need to follow the hierarchical structure of variants. For more information, see Styles and Size Variants in the SAP Commerce Product Cockpit.

To find out how to add the product attribute in the Product Cockpit, see Setting Up Product Attributes.

Easy Selection for Cross-Selling and Up-Selling

With the SAP Commerce Accelerator, you can easily cross-sell and up-sell your products.

Related Information

Styles and Size Variants in the SAP Commerce Product Cockpit
Product Cockpit
Getting Started with the Product Cockpit
Product Perspective
Catalog Perspective

Managing Media in the SAP Commerce Product Cockpit

Learn how to manage media in the SAP Commerce Product Cockpit.

⚠ Caution

This page refers to deprecated software. For further details, see Deprecation Status.

SAP Commerce Accelerator provides you with the opportunity to manage multiple images of various sizes for defined purposes. Media in Commerce Accelerator are grouped into media containers. A media container holds various media formats of the same image. The Commerce Accelerator follows the best practice that the images displayed in the front-end should have the right proportions for performance reasons.

Available Media Formats

In the SAP Commerce Product Cockpit for Commerce Accelerator, the following media formats are available:

- The `superZoom` media format, grouping images of size 1200Wx1200H.

  This media format is currently not used in the SAP Commerce Accelerator sample data.

- The `zoom` media format, grouping images of size 515Wx515H.

This media format is used for zooming the product images. To see the bigger image, click on the Zoom button, located in the bottom right corner of the standard image on the Product Page.

Figure: Example of the media in the *zoom* format.

- The product media format, grouping images of size 300Wx300H.

  This media format is used for displaying the standard product image.

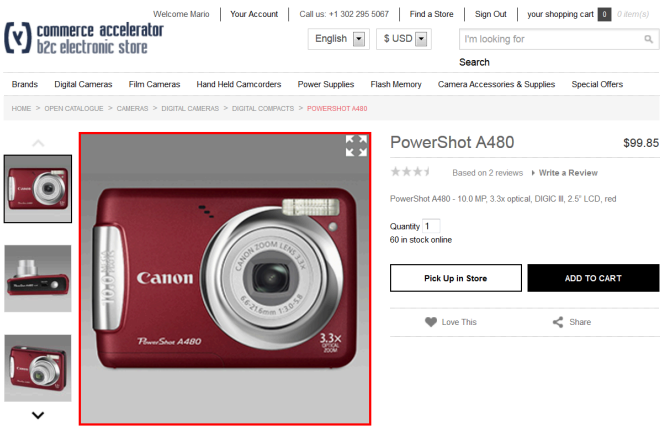
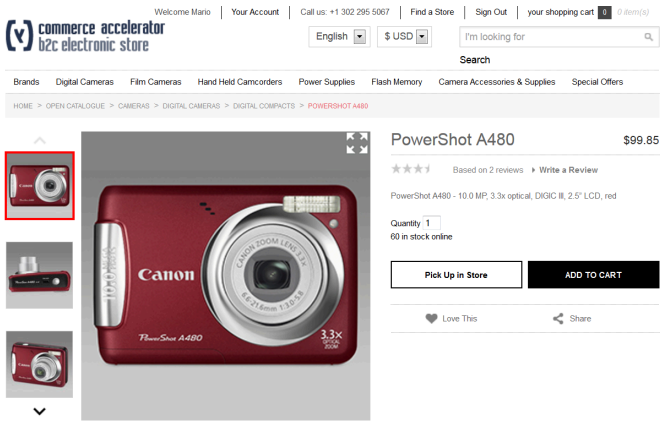


Figure: Example of the media in the *product* format.

- The `thumbnail` media format, grouping images of size 96Wx96H.

  This media format is used for showing the auxiliary pictures of a product. Media in this format are also displayed in the Cart page.



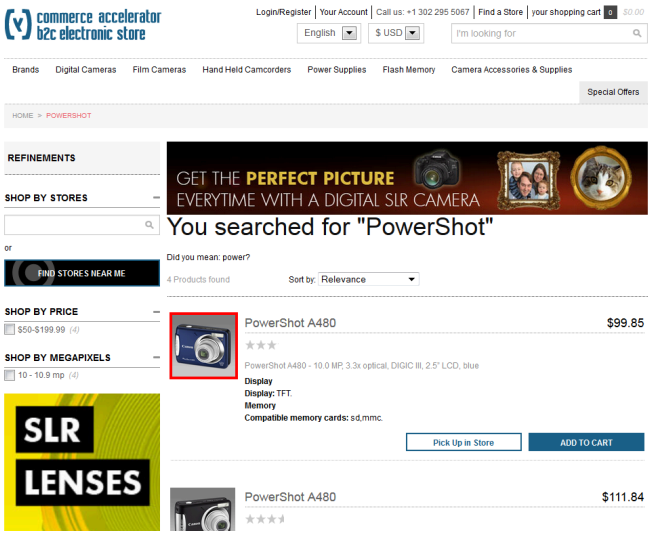


Figure: Example of the media in the *thumbnail* format.

- The `cartIcon` media format, grouping images of size 65Wx65H.

  This media format is used in the mini-cart. To access the mini-cart, move your mouse over the cart area in the top right corner.

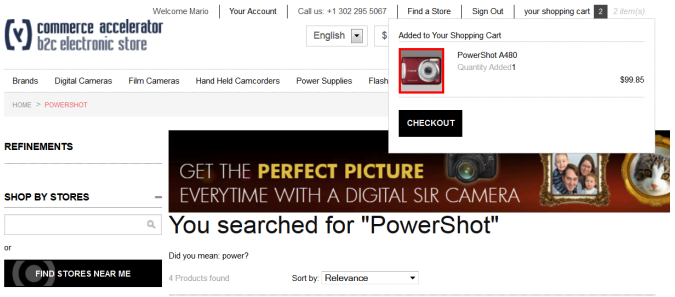


Figure: Example of the media in the *cartIcon* format.

- The `styleSwatch` media format, grouping images of size 30Wx30H.

  This media format is used for showing product variants, for example different colors of a given product.

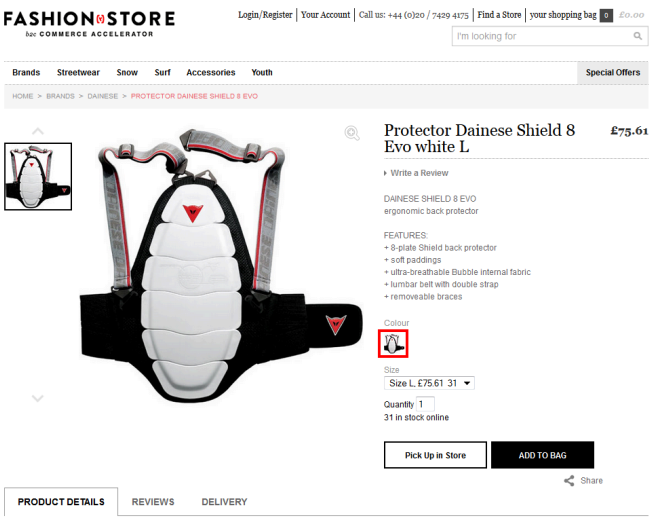


Figure: Example of the media in the *styleSwatch* format.

Multimedia Section in the Editor Area

The editor area in the SAP Commerce Product Cockpit is composed of several sections. One of these sections is the **Multimedia** section that you use for attaching new images to a selected product. The **Multimedia** section contains several attributes:

- Image
- Thumbnail
- Specifications
- Details
- Logos
- Normals
- Others

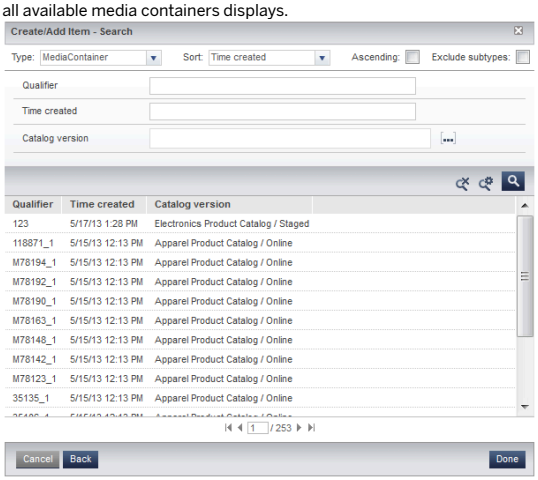
- Thumbnails

You can attach media to each one separately.

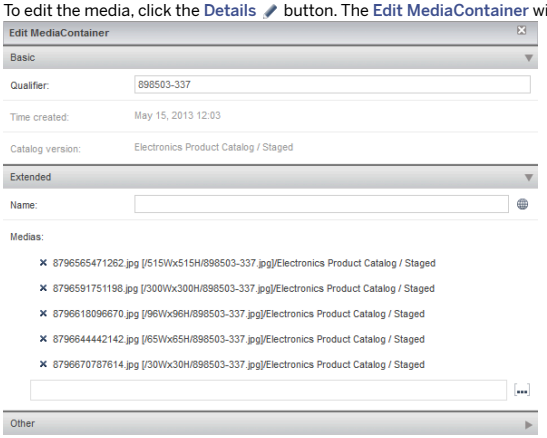
The Gallery Images is a collection of media containers. The images included in the gallery are displayed in the front end in various media formats.

Editing Media Containers

1. To display the included media containers, click the Add reference button.
2. Select the Select an existing reference option, as you only want to edit a media container.
3. Search for a given media container by typing its name in the Qualifier field and clicking the Search button. If you leave the Qualifier field empty and click the Search button, a list of all available media containers displays.



4. Select a media container from the list and click the Done button.
5. To edit the media, click the Details button. The Edit MediaContainer wizard opens.



6. In the Basic section, you can remove media by clicking the Remove button. Also, you can add the name in this section.

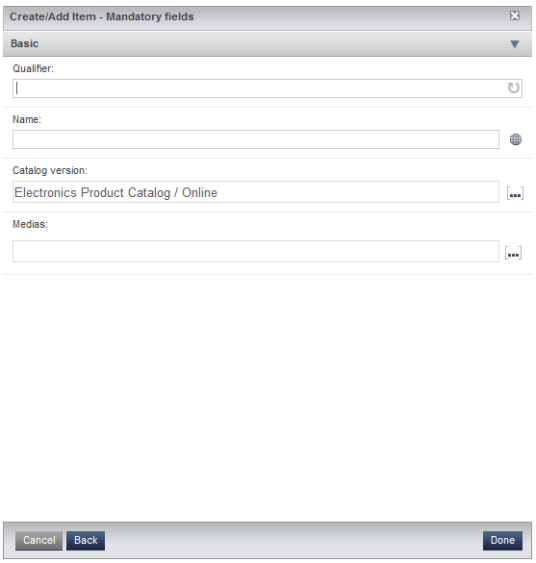
   To upload a new image to an existing media container, click the Add reference button and select the type of object you wish to add.

7. To leave the Edit Media wizard, click the Close button.

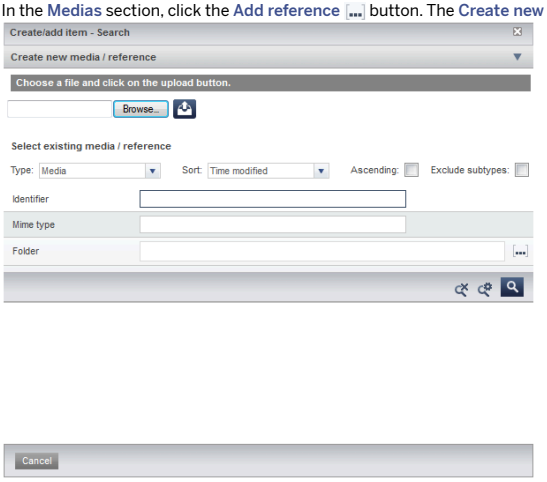
Creating New Media Container

To create a new media container:

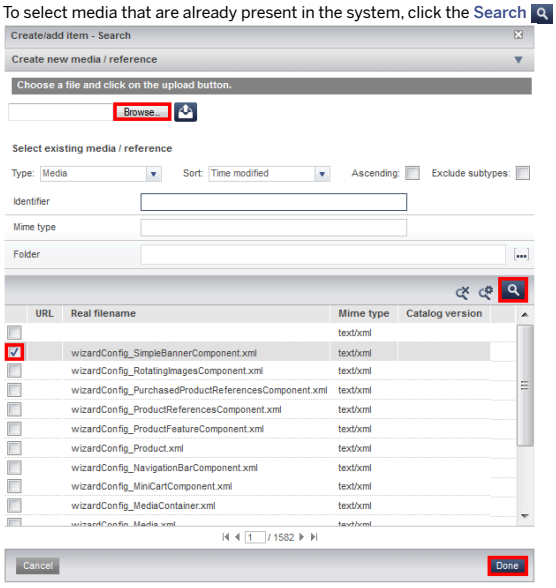
1. Click the Add reference button.
2. Select the Create a new item option. The Create/Add Item wizard displays.



3. Enter the Qualifier, for example `Qualifier8`.
4. In the Medias section, click the Add reference button. The Create new media / reference wizard displays.

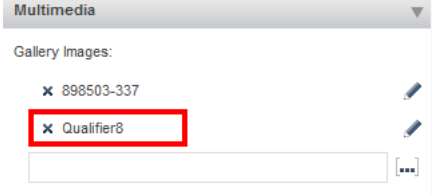


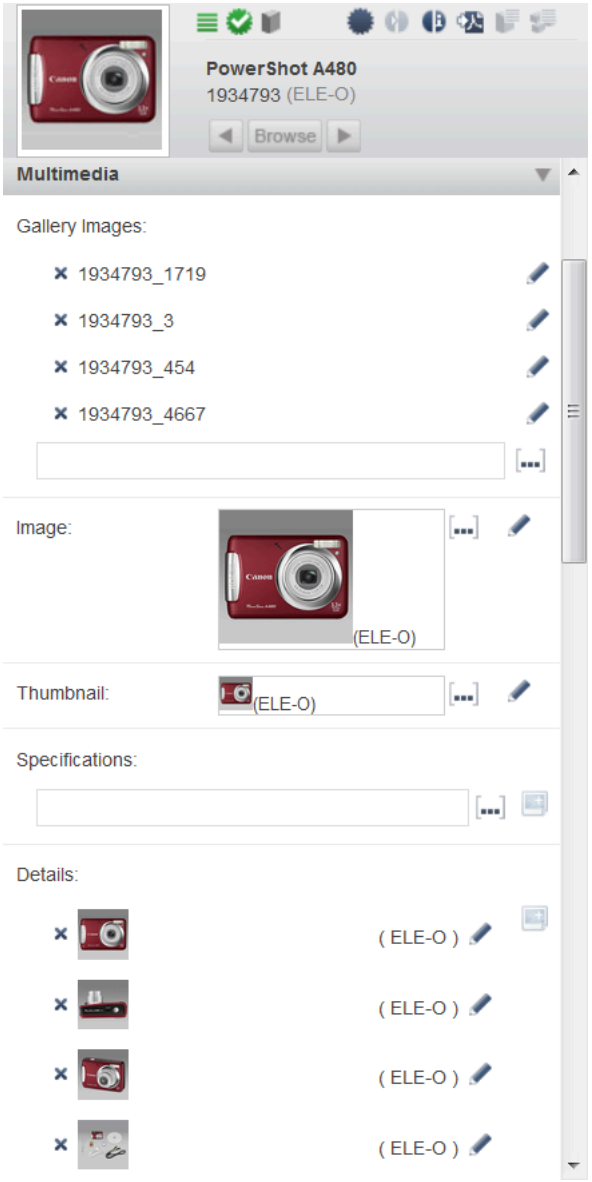
5. Select medias to insert into the media container. You can select medias in two different ways:
   - To select new media, click the Browse button. When you select your medias, click the Upload button to upload them to the system.
   - To select media that are already present in the system, click the Search button and select a media from the search results list.



   When you have finished selecting media, click the Done button.

6. Your new media container appears in the Gallery images section of the Editor Area.





Related Information

commerceservices Extension
Catalog Perspective
Product Perspective

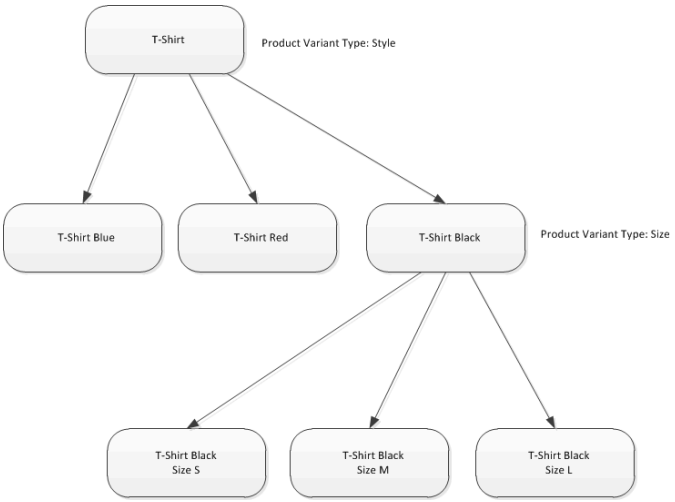
Styles and Size Variants in the SAP Commerce Product Cockpit

The SAP Commerce Accelerator enables you to manage not only products, but also their variants, such as style and size.

⚠ Caution
This page refers to deprecated software. For further details, see Deprecation Status.

This document describes the recommended structure for the implementation of a product with its variants.

The diagram below presents the hierarchical structure of variants and how such structure should be defined for your products:



For information about the data model, see yacceleratorcore Extension.

Overview

The SAP Commerce Accelerator supports four product variants:

- Style and size variants for B2C Apparel storefronts
- Color variant for B2C Electricals storefront
- Size variant for B2B Powertools storefront

The following sections show the implementation of the base product and its variants structure based on the B2C Apparel storefront.

Base Product Implementation

The product is a fundamental base in the SAP Commerce Accelerator. First, you need to have the implementation of the product to properly define your catalog framework.

The base product need to be specified by these parameters:

- **$productCatalog=apparelProductCatalog**
- **$productCatalogName=Apparel Product Catalog**
- **$catalogVersion=catalogversion(catalog(id**default=$productCatalog),versiondefault='Staged')Staged
- **$prices=europe1pricestranslator=de.hybris.platform.europe1.jalo.impex.Europe1PricesTranslator**
- **$baseProduct=baseProduct(code,catalogVersion(catalog(iddefault='$productCatalog'),versiondefault='Staged'))**
- **$approved=approvalstatus(code)default='approved'**
- **$taxGroup=Europe1PriceFactory_PTG(code)default=eu-vat-full**

In the B2C Apparel storefront products may have no variations, only color or style variations, or both color and size variations. Obviously, if you do not have the implementation of product, there is no point to provide it for the variants.

The implementation of the base product you can find in the ***${HYBRIS_BIN_DIR}*/ext-data/apparelstore/resources/apparelstore/import/sampledata/productCatalogs/apparelProductCatalog/product.impex** file:

```
INSERT_UPDATE ApparelProduct;codeunique=true;$catalogVersion;unit(code)default='pieces';$prices;supercategories(code,$catalogVersion);varianttype(co
                             ;102277;;pieces;40,46 GBP , 49,95 EUR;450100,snow,Red;ApparelStyleVariantProduct;;;;MALE
```

Style Product Variants

When the base product is created, then you can add or modify the variants of it. The typical example of the style variant of the product is color.

For example, for the base product with code **102277**shown in the above section, there are three style variants:

- **black_matte**
- **trench_green**
- **white_pearl**

The style variants are implemented as shown from this example for a **product.impex** file:

```
INSERT_UPDATE ApparelStyleVariantProduct;codeunique=true;$catalogVersion;$baseProduct;supercategories(code,$catalogVersion);swatchColors(code);unit(
                                       ;102277_black_matte;;102277;450100,snow,Red;BLACK;pieces;ApparelSizeVariantProduct;
                                       ;102277_trench_green;;102277;450100,snow,Red;GREEN;pieces;ApparelSizeVariantProduct;
                                       ;102277_white_pearl;;102277;450100,snow,Red;WHITE;pieces;ApparelSizeVariantProduct;
```

Size Product Variants

For the particular style variants the variants of the size are assigned. The implementation of the size product variants should look as shown from this example for a **product.impex** file:

```
INSERT_UPDATE ApparelSizeVariantProduct;codeunique=true;$catalogVersion;$baseProduct;supercategories(code,$catalogVersion);unit(code)default='pieces
                                       ;300453357;;102277_black_matte;450100,snow,Red;pieces;40,46 GBP , 49,95 EUR;;;
                                       ;300453356;;102277_black_matte;450100,snow,Red;pieces;40,46 GBP , 49,95 EUR;;;
```

```
;300453365;;102277_trench_green;450100,snow,Red;pieces;40,46 GBP , 49,95 EUR;;;
;300453379;;102277_white_pearl;450100,snow,Red;pieces;40,46 GBP , 49,95 EUR;;;
;300453378;;102277_white_pearl;450100,snow,Red;pieces;40,46 GBP , 49,95 EUR;;;
;300453377;;102277_white_pearl;450100,snow,Red;pieces;40,46 GBP , 49,95 EUR;;;
```

Related Information

Modeling Product Variants
yacceleratorcore Extension
Setting Up Product Attributes

Moderating Reviews

SAP Commerce Accelerator provides a customer review functionality on the storefront. Each review consists of rating, review title, and review description. All of these are mandatory fields for a customer.

Merchant employees have to moderate the reviews using the Backoffice Administration Cockpit before they are released.

Moderating Reviews in the Backoffice

To moderate reviews using the Backoffice

1. Log into the Backoffice.
2. Navigate to **Marketing** > **Product reviews**. A list of reviews appears in the main pane.
3. Search for the required review or reviews:
    a. Click beside the search bar to display the advanced search settings.
    b. Enter the required search parameters and then click **Search**.

    → **Tip**
    Usually you start working with the reviews that are pending approval. To search for these reviews only, select **is equal** as the **Comparator** and **Pending** as the **Value** for the **Approval status** attribute.

4. Click the review you want to moderate.
5. Select the **Approval status** from the drop-down list.
6. Click **Save**.

Related Information

Customer Reviews

Storefront Configuration

Use the Backoffice Administration Cockpit for Accelerator storefront configurations, including base store settings, languages and territories, tax settings, and email templates.

Base Store Management

This section describes how to configure following options:

- Prices (net/gross)
- Currencies
- Website theme

To set net or gross prices:

1. Log into Backoffice (`https://localhost:9002/backoffice`).
2. Navigate to **Base Commerce** > **Base Store**. A list of storefronts appears in the main pane.
3. Click the storefront you want to set up.
4. In the **Properties** tab, select **True** under **Net** to display net prices, or select **False** to display gross prices.
5. Click **Save**.

To set the currencies for your storefront:

1. In Backoffice, navigate to **Base Commerce** > **Base Store**, and then click the storefront you want to set up.
2. In the **Properties** tab, click under **Currencies** and then select the required currency. Repeat the process if you need to add more currencies to your storefront.

    → **Tip**
    You can add new currencies by clicking , selecting **Create new currency**, and then completing the required fields.

3. To change the default currency displayed in the storefront, click to remove the current currency, and then click and select the default currency.
4. Click **Save**.

To change the theme of your storefront:

1. In Backoffice, navigate to **WCMS** > **Website**, and then click the storefront you want to set up.

2. In the Properties tab, select a theme from the Theme drop-down list.
3. Click Save.

Languages and Territories

This section describes how to set the languages and delivery territories for a website. The options are set in the content catalog properties.

1. Log into Backoffice.
2. Navigate to Catalog > Catalog Versions. A list of catalogs appears in the main pane.
3. Click the catalog you want to edit.
4. In the Catalog version tab, add Languages and Territories by clicking and selecting the required entries.
5. Click Save.

Assigning Taxes to Customers

To assign a tax group to a customer:

1. Log into Backoffice.
2. Navigate to User > Customers. A list of customers is displayed in the main pane.
3. Click the customer you want to edit.
4. In the Prices tab, select the Tax Group from the drop-down list.
5. Click Save.

Managing Email Templates

The system offers an option to communicate with customers in specific situations. These responses are available by default:

- Customer registration
- Forgotten password
- Order confirmation.

You can configure the contents of each of the emails in Backoffice:

1. Log into Backoffice.
2. Navigate to System > Output Documents > Communication Template. A list of message template codes are displayed in the main pane.
3. Click the template you want to edit.
4. In the General tab, edit the Template script or other required parameters. If you want to associate the template with a new event within the system, provide the appropriate class name in the Context class text box.

   Note

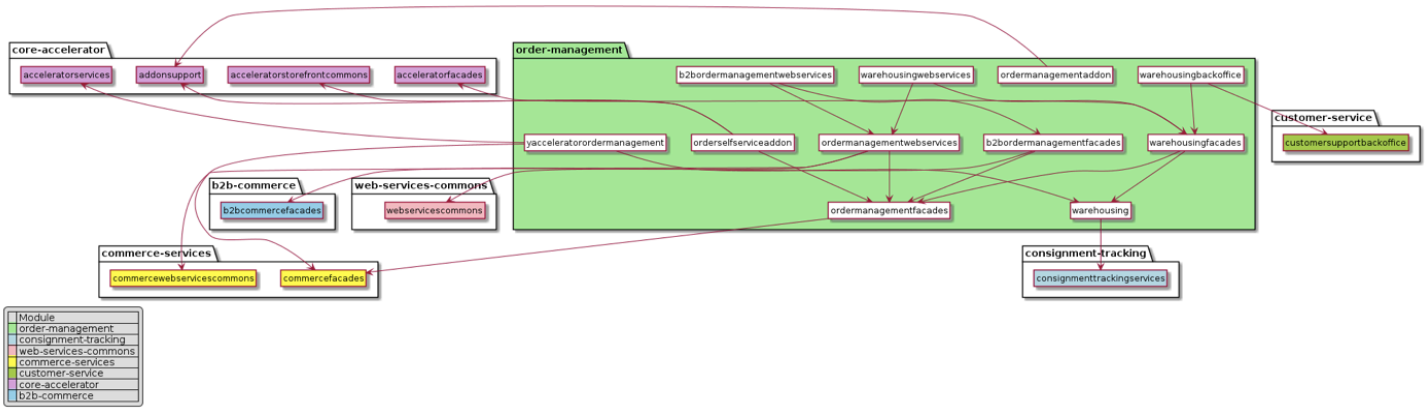
   The template script accepts the default HTML format of messages as well as plain text.

5. Click Save.

Core Accelerator Architecture

The Core Accelerator Module is a set of extensions providing features such as payment services, extensions to the OCC API, and additional WCMS components.

Dependencies



Recipes

For a complete list of SAP Commerce recipes that may include this module, see Installer Recipes.

For a complete list of the SAP Commerce Cloud, integration extension pack recipes that may include this module, see Installer Recipe Reference.

Extensions

The Core Accelerator Module consists of the following extensions:

- acceleratorcms Extension
- acceleratorfacades Extension
- acceleratorservices Extension
- acceleratorstorefrontcommons AddOn
- acceleratorwebservicesaddon AddOn
- addonsupport Extension
- acceleratorbackoffice Extension

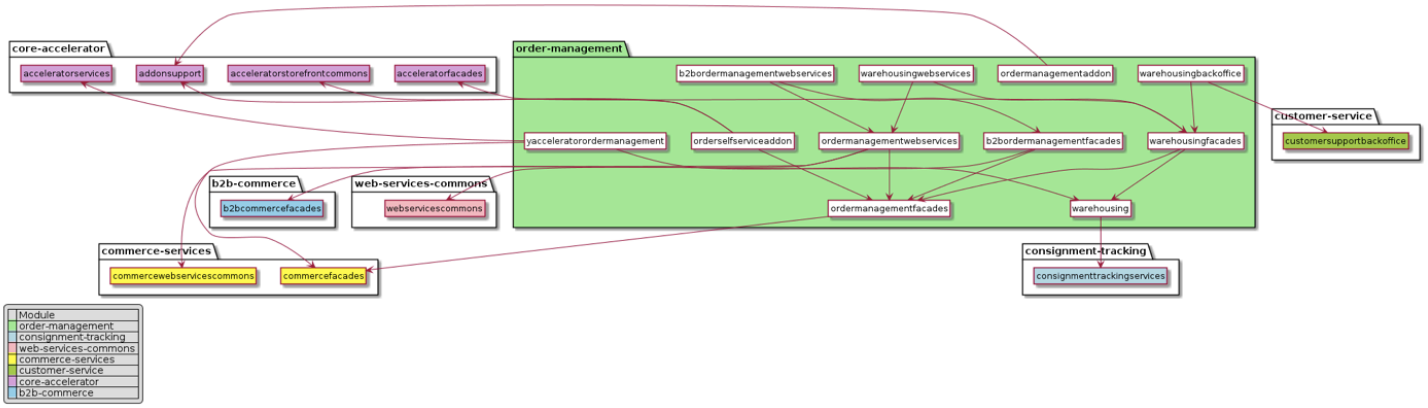
acceleratorbackoffice Extension

The `acceleratorbackoffice` extension provides backoffice interfaces for customer support, configuring products, and setting up the online store layout.

About the Extension

Name	Directory	Related Module
`acceleratorbackoffice`	`bin/modules/core-accelerator`	Core Accelerator Module

Dependencies



Related Information

Extensions and AddOns

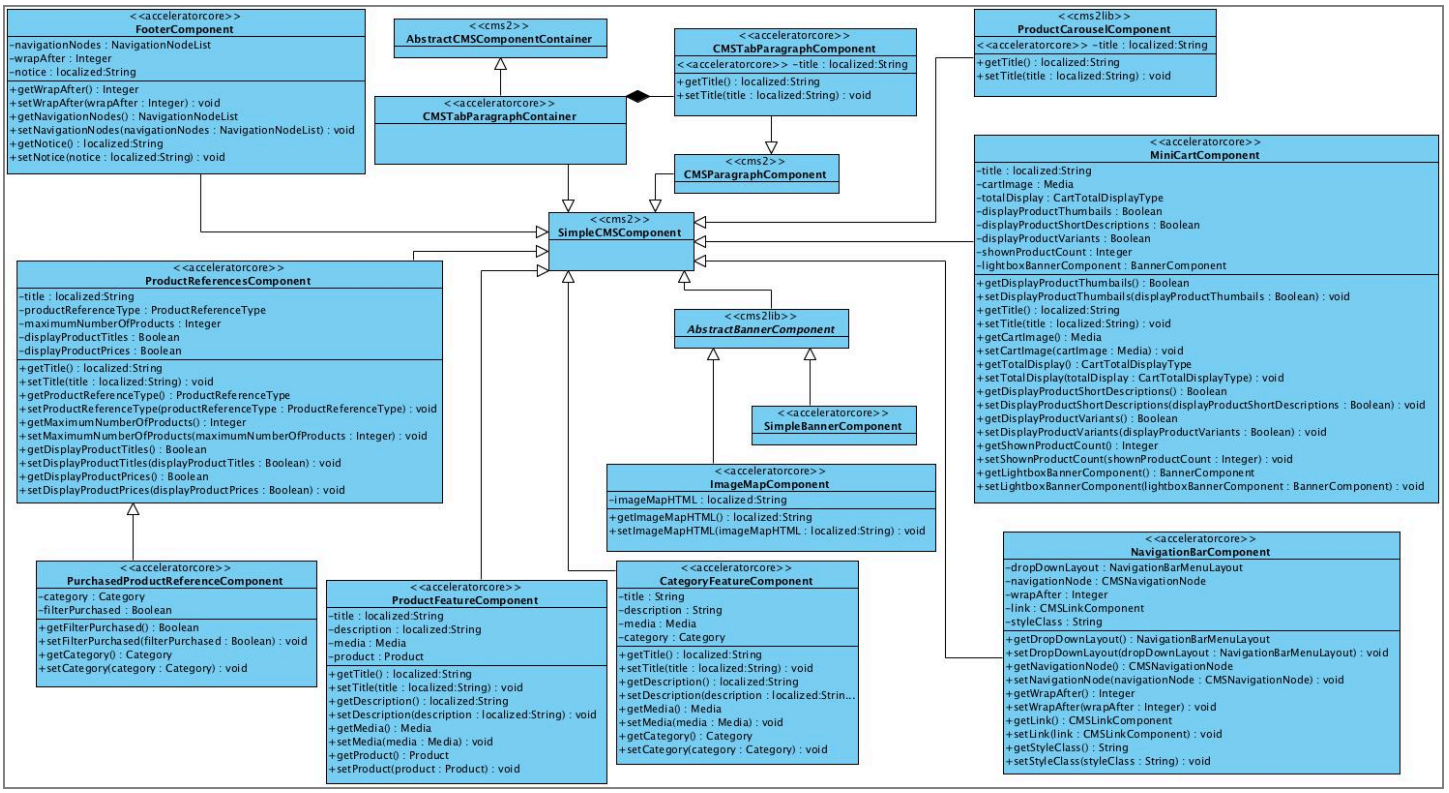
acceleratorcms Extension

The `acceleratorcms` extension provides a number of WCMS components that are designed to be used with your generated storefronts.

WCMS Components

The `acceleratorcms` extension provides WCMS components in addition to those provided by default in the `cms2` and `cms2lib` extensions. Some existing components have also been extended with additional attributes.

The following class diagram shows the extensions made to the WCMS data model in the `acceleratorcms` extension. For your reference, the declaring extension is stereotyped on the class. If an existing type has been extended, the attribute is stereotyped.



- Category Feature Component: The category feature component spotlights a particular category of products. With this component, you can add marketing text, an image, and a title. The link is auto-generated.
- CMS Tab Paragraph Component: The CMS tab paragraph component supports the ability to add additional tabs of content on the product details page.
- Footer Component: The footer component is a counterpart to the navigation component. It allows you to build a footer menu.
- Image Map Component: The image map component offers all the attributes of the simple banner component, but also adds the ability to embed image map HTML.

> **Note**
>
> The image component is only available when the Accelerator user experience is set to **responsive**. For more information, see Responsive Storefronts.

- Mini Cart Component: The mini cart component configures how the mini cart works on the storefront. This includes title, how totals are calculated, how many recently added products to display, and a merchandising banner.
- Navigation Bar Component: The navigation bar component is a placeholder component for building the ever-present top navigation on a website.
- Product Carousel Component: The product carousel component is a `cms2lib` component that has been extended to add a title.
- Product Feature Component: The product feature component spotlights a particular product. With this component, you can add marketing text, an image, and an emotive title. The link is auto-generated.
- Purchased Product References Component: The purchased product references component is provided mainly for sample purposes. It extends the product references component by incorporating a sample suggestion service. This service demonstrates the functionality of advanced personalization and restrictions.
- Product References Component: The product references component chooses what types of cross-sells to show on a product page. The system fills the slot with cross-sells of the specified type if they exist for the currently viewed product.
- Simple Banner Component: The simple banner component contains only image and link attributes.
- Responsive Image Component: The responsive image component is used for containing media. You can configure it for different image components corresponding to different UI experiences.

CMS Component Caching

The `acceleratorcms` extension contains integration with the region cache to allow caching of rendered CMS components. Components are cached based on their modified time, and are expired based on the configured cache eviction policy. CMS component caching is disabled by default.

Enabling CMS Caching

The following settings in the `project.properties` file located in the `hybris/bin/modules/core-accelerator/acceleratorcms` folder are used to control the caching behavior :

```
cms.cache.enabled=false
cms.cache.use.default.fallback=false

regioncache.cmsregion.size=20000
regioncache.cmsregion.evictionpolicy=LFU
regioncache.cmsregion.ttl=300
```

By default, `cms.cache.use.default.fallback` is set to `false`. As a result, the `CacheKey DefaultCmsCacheService.getKey(final HttpServletRequest request, final AbstractCMSComponentModel component)` method returns null for a cache key because it does not fall back to use the defaultCmsCacheKeyProvider.

If `cms.cache.use.default.fallback` is set to `true`, the `CacheKey DefaultCmsCacheService.getKey(final HttpServletRequest request, final AbstractCMSComponentModel component)` method uses defaultCmsCacheKeyProvider for every component type that doesn't have an explicit CacheKeyProvider mapping. The provider caching is based on the PK and modified time of a component, as well as the current language and currency, so it should not be used for components that display user or session-specific information. As a result, `cms.cache.use.default.fallback` must be set to `false` until all components are correctly mapped in the project.

Cache Key Providers

The default Spring configuration in the `hybris/bin/modules/core-accelerator/acceleratorcms/resources/acceleratorcms-spring.xml` file shows that most components are cached; other types can be added if needed:

```
	<alias name="defaultCacheKeyProviders" alias="cacheKeyProviders"/>
	<util:map id="defaultCacheKeyProviders" key-type="java.lang.String">
		<entry key="AbstractCMSComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="NavigationBarComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="BannerComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="SimpleBannerComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="RotatingImagesComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="ImageMapComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="ProductCarouselComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="CategoryFeatureComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="ProductFeatureComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="CMSParagraphComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="CMSImageComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="CMSTabParagraphComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="FooterComponent" value-ref="cmsCacheKeyProvider"/>
		<entry key="ProductReferencesComponent" value-ref="currentProductCmsCacheKeyProvider"/>
		<entry key="ProductDetailComponent" value-ref="currentProductCmsCacheKeyProvider"/>
		<entry key="ProductVariantSelectorComponent" value-ref="currentProductCmsCacheKeyProvider"/>
		<entry key="DynamicBannerComponent" value-ref="currentCategoryCmsCacheKeyProvider"/>
	</util:map>
```

You can add explicit provider mappings for custom component types to the defaultCacheKeyProviders map in Spring and a custom CacheKeyProivder can be implemented for caching based on other criteria.

If you want to extend the cache implementation to cover your custom components, add new CmsCacheKeyProviders, or configure one of the defaults the followoing key providers are available:

- DefaultCmsCacheKeyProvider: This cache is based on PK, modified time, current currency, current language, and any component restrictions.
- CurrentUserCmsCacheKeyProvider: This cache extends the default implementation to also cache on the current user.
- CurrentProductCmsCacheKeyProvider: This cache extends the default implementation to also cache on the current product. As a result, this is only effective on product pages.
- CurrentCategoryCmsCacheKeyProvider: This cache extends the default implementation to also cache on the current category. As a result, this is only effective on category pages.

Entry Group Support for Cart Grouping Functionality

To add entry group support there is a `ListAddToEntryGroupAction` itemtype implemented to `acceleratorcms-items.xml`. Localization for it is defined in `acceleratorcms-locales_en.properties`.

acceleratorcms/resources/acceleratorcms-items.xml

```
[...]
<itemtype code="ListAddToEntryGroupAction"
        jaloclass="de.hybris.platform.acceleratorcms.jalo.actions.ListAddToEntryGroupAction"
        extends="SimpleCMSAction" autocreate="true" generate="true">
</itemtype>
[...]
```

acceleratorcms/resources/localization/acceleratorcms-locales_en.properties

```
[...]
type.ListAddToEntryGroupAction.name=List Add To Entry Group Action
type.ListAddToEntryGroupAction.description=
[...]
```

acceleratorcms/src/de/hybris/platform/acceleratorcms/jalo/actions/ListAddToEntryGroupAction.java

```
package de.hybris.platform.acceleratorcms.jalo.actions;
public class ListAddToEntryGroupAction extends GeneratedListAddToEntryGroupAction
{
    // Deliberately empty class
}
```

Related Information

acceleratorservices Extension - Technical Guide
cms2 Extension
cms2lib Extension
Legacy Components
Cart Entry Grouping

acceleratorfacades Extension

The `acceleratorfacades`, `yacceleratorfacades`, and `yb2bacceleratorfacades` extensions contain the features that are common for both B2B and B2C facades of the SAP Commerce Accelerator.

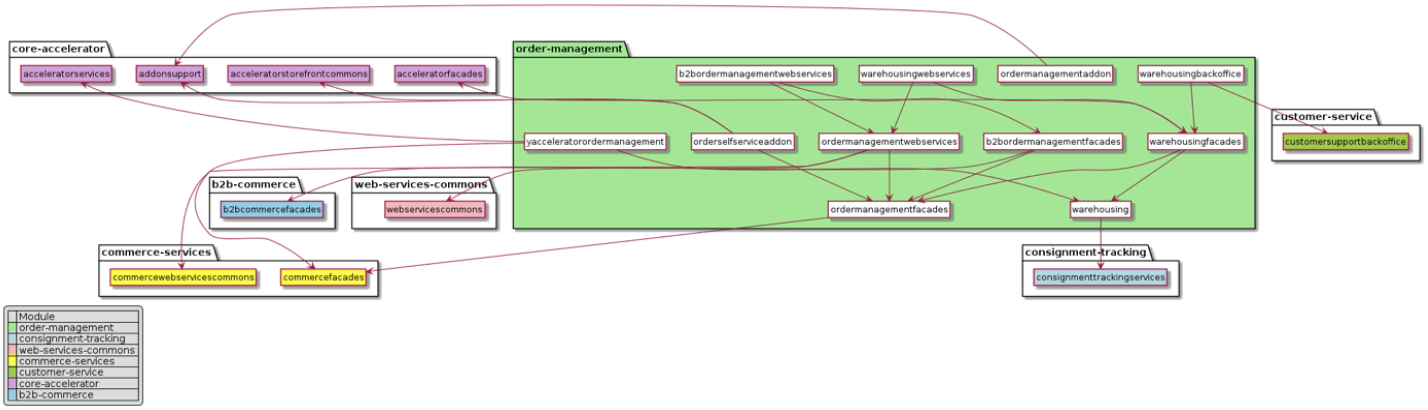
> **i Note**
> An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

The `accProductConfiguredPopulator` and `accProductFacade` are deprecated. They are aliases for `productVariantConfiguredPopulator` and `productVariantFacade` respectively. There is no need to create a new bean to modify populator list. `configurablePopulatorModification` is used to add new options to `productVariantConfiguredPopulator`.

acceleratorfacades/resources/acceleratorfacades-spring.xml

```
[...]
<!-- deprecated since version 6.4.0.0, use productVariantConfiguredPopulator instead -->
<alias alias="accProductConfiguredPopulator" name="productVariantConfiguredPopulator" />
<!-- deprecated since version 6.4.0.0, use productVariantFacade instead -->
<alias alias="accProductFacade" name="productVariantFacade" />
<bean parent="configurablePopulatorModification">
    <property name="target" ref="productVariantConfiguredPopulator" />
    <property name="keyType" value="de.hybris.platform.commercefacades.product.ProductOption" />
    <property name="key" value="VOLUME_PRICES" />
    <property name="add" ref="productVolumePricesPopulator" />
</bean>

<bean parent="configurablePopulatorModification">
    <property name="target" ref="productVariantConfiguredPopulator" />
    <property name="keyType" value="de.hybris.platform.commercefacades.product.ProductOption" />
    <property name="key" value="KEYWORDS" />
    <property name="add" ref="productKeywordsPopulator" />
</bean>
[...]
```

Dependencies



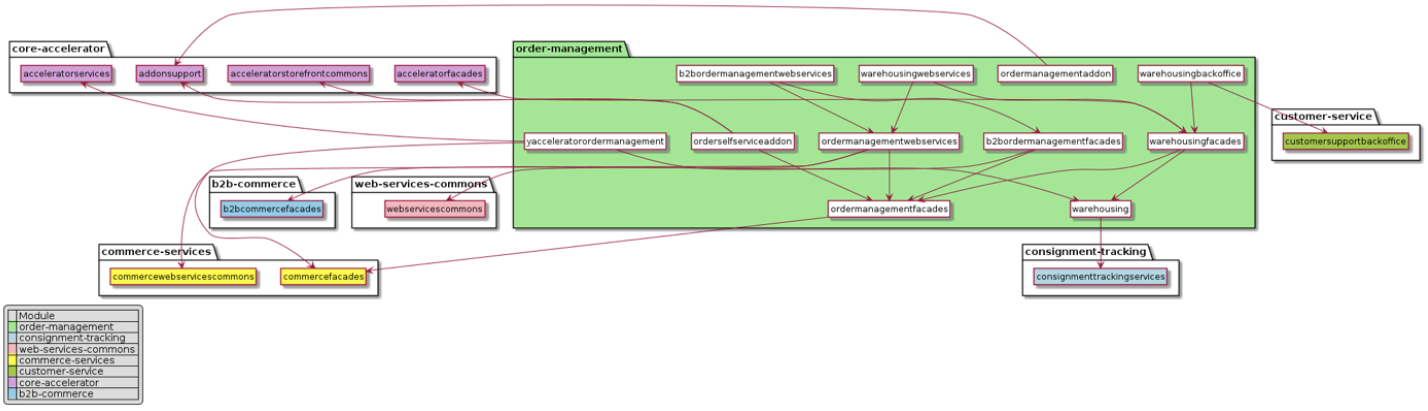
acceleratorocc Extension

The `acceleratorocc` extension exposes saved functionality of the Core Accelerator module in the form of the standardized OCC (Omnichannel Commerce) API.

About the Extension

Name	Directory	Related Module
`acceleratorocc`	`hybris/bin/modules/core-accelerator/`	Core Accelerator Module

Dependencies



Related Information

Extensions and AddOns

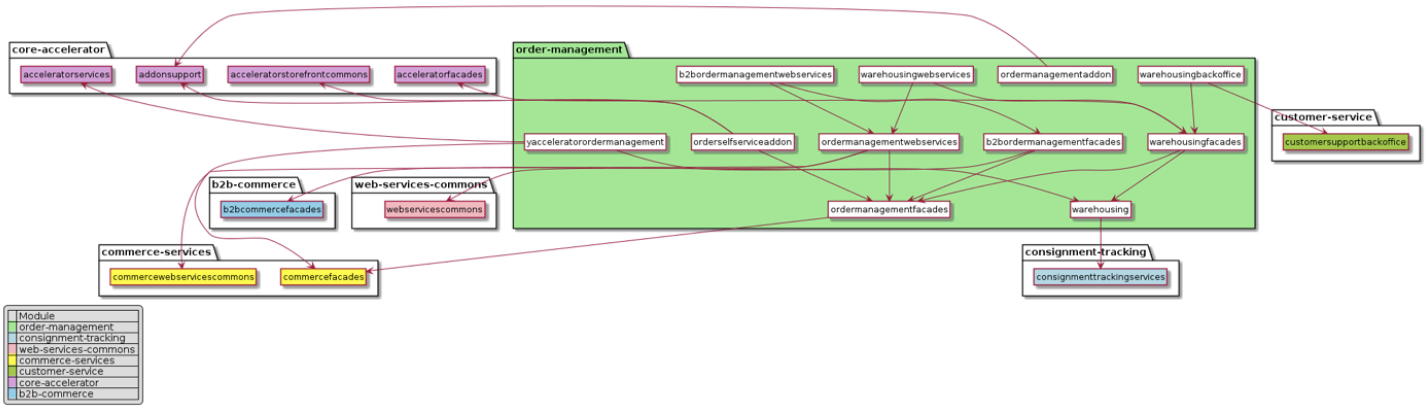
acceleratorservices Extension

The `acceleratorservices` extension provides data import and export capabilities, as well as email and location services.

i Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Dependencies



Payment Services

The following sections describe the payment services of the `acceleratorservices` extension.

Calls Sequence

For both Silent Order Post (SOP) and Hosted Order Page (HOP), the sequence of the payment flow in Accelerator is as follows:

1. The form URL to POST customer information to is retrieved.
2. The different form containing the hidden fields to POST to the payment provider is generated. (For SOP you have additional credit card and address input fields. For HOP, this information is added on another form hosted by the payment provider.)
3. The form is POSTed.
4. For HOP, the customer is redirected to an additional form hosted by the payment provider, and then the same form is POSTed as well.
5. The customer gets redirected by the payment provider to the storefront.
6. The payment provider response gets interpreted.
7. If there are any errors, they are mapped to the payment form field when possible (for SOP).
8. The signature is validated.

Payment Gateway Mock

The `acceleratorservices` extension provides a payment gateway mock that integrates with the Accelerator storefront checkout process. This is designed to assist your project with adding your own payment gateways, which is often required for PCI.

As a part of the multi-step checkout, developer options are presented that allow the selection of either an HOP or an SOP checkout processes.

ⓘ Note

The `acceleratorservices` `project.properties` file contains a number of Cross-Origin Resource Sharing (CORS) settings that have been set up to work with the payment gateway mock. If you are running a production environment, the following `corsfilter` settings need to be adjusted:

```
corsfilter.acceleratorservices.allowedOrigins=http://localhost:4200
corsfilter.acceleratorservices.allowedMethods=GET POST PUT OPTIONS
corsfilter.acceleratorservices.allowedHeaders=origin content-type accept authorization
```

For more information, see yacceleratorstorefront Extension, Accelerator Payment Mock, and Cross-Origin Resource Sharing Support.

Payment Form URLs

Before rendering payment forms, the system needs to determine where to post the form. To make this dynamic, the `PaymentFormActionUrlStrategy` is provided. The default Accelerator implementation looks at the following properties as shown in the following example from the `local.properties` file:

```
hop.post.url=
sop.post.url=
```

Payment Form Content

For different payment providers, you might need to send different values. This part is covered by the `CreateSubscriptionRequestStrategy`. This strategy generates a `CreateSubscriptionRequest` that holds customer data, order data, payment data, and so on. The default implementation should cover most of the required data for different payment providers, but this can still be overridden for handling specific cases.

The `CreateSubscriptionRequest` is then converted to an instance of `PaymentData` that holds a map of keys-values that represent the different fields to send to the payment provider. These keys-values are used to generate the form to render during the checkout process. This conversion is done by the `bean paymentDataConverter` and its populators. By default, Accelerator comes with populators that are designed for integration with CyberSource. The same populators are also designed to work with a payment gateway mock (as mentioned above). To implement a specific payment provider, these populators need to be changed.

ⓘ Note

The CyberSource extension has been removed from the SAP Commerce distribution.

Payment Response Interpretation

Accelerator must be able to handle the responses returned from different Payment Service Providers (PSPs). For this purpose, the `PaymentResponseInterpretationStrategy` is provided. The default implementation handles the responses from CyberSource and the payment gateway mock.

Mapping Between Errors and Fields

Possible errors returned by the PSP need to be mapped to the form fields. This allows you to display an error message next the appropriate field on the front-end. This is handled by the `CreateSubscriptionResultValidationStrategy`. The default implementation handles the responses from CyberSource and the payment gateway mock.

Signature Validation

Some of the payment providers (such as CyberSource) include an encrypted signature of the sensitive fields. This allows you to verify that no sensitive data has been modified by any third party before reaching Accelerator. The strategy that validates the digital signatures is `SignatureValidationStrategy`.

The default implementation handles the responses from CyberSource and the payment gateway mock.

Order Updates

After sending an authorization, the payment provider has the option to send updates for possible fraud verification.

Handling Received Order Update

Received order updates are handled by the `PaymentService-handleFraudUpdate` method, which is implemented in the `DefaultCISPaymentService` in the `cispayment` AddOn. In other words, it is sent to the CIS service. Based on the CIS service response, a proper review decision entry is processed by the `setPaymentTransactionReviewResult` method.

Saving Review Decision and Sending Event for Process Order

The `CyberSourceAcceleratorPaymentService` class adds a new `PaymentTransactionEntry` of the `REVIEW_DECISION` type to the order and sends the `process.order.code_ReviewDecision` event.

Data Import and Export

Data import and export is describes in detail in Data Importing and System Context.

Email Services

The `acceleratorservices` extension adds email generation and audit functionality, combined with additional processes, for use with the SAP Commerce process engine.

For more information, see WCMS Email and Process Engine Integration.

Email Service

The `EmailService` handles the creation and sending of emails. Services are provided to allow a record of the email to be persisted for auditing purposes.

For more information, see processing Extension.

Email Business Process

The `acceleratorservices` extension provides a set of business process actions that support the ability to asynchronously generate email content with CMS-managed sections, and then send the emails using an SMTP server.

The following actions have been implemented:

- Generate an email body
- Send the email
- Remove the email from the database.

CMS Email Page Service

The `CMSEmailPageService` adds the ability to manage parts of the email content in the WCMS Cockpit. A specialized kind of page template, called the `EmailPageTemplate`, allows the template to be linked to a `RendererTemplate` from the `commons` extension. A special type of CMS page, called an `EmailPage`, allows you to add WCMS components to allocated slots that are defined by your `EmailPageTemplate`, and to place them in the email by the `RendererTemplate`.

> **Note**
> The interface has an explicit dependency on the `cms2` extension.

For more information, see cms2 Extension and commons Extension.

Email Generation Service

The `EmailGenerationService` is responsible for generating the email content and attaching it to a new `EmailMessage`.

Site Base URL Resolution Service

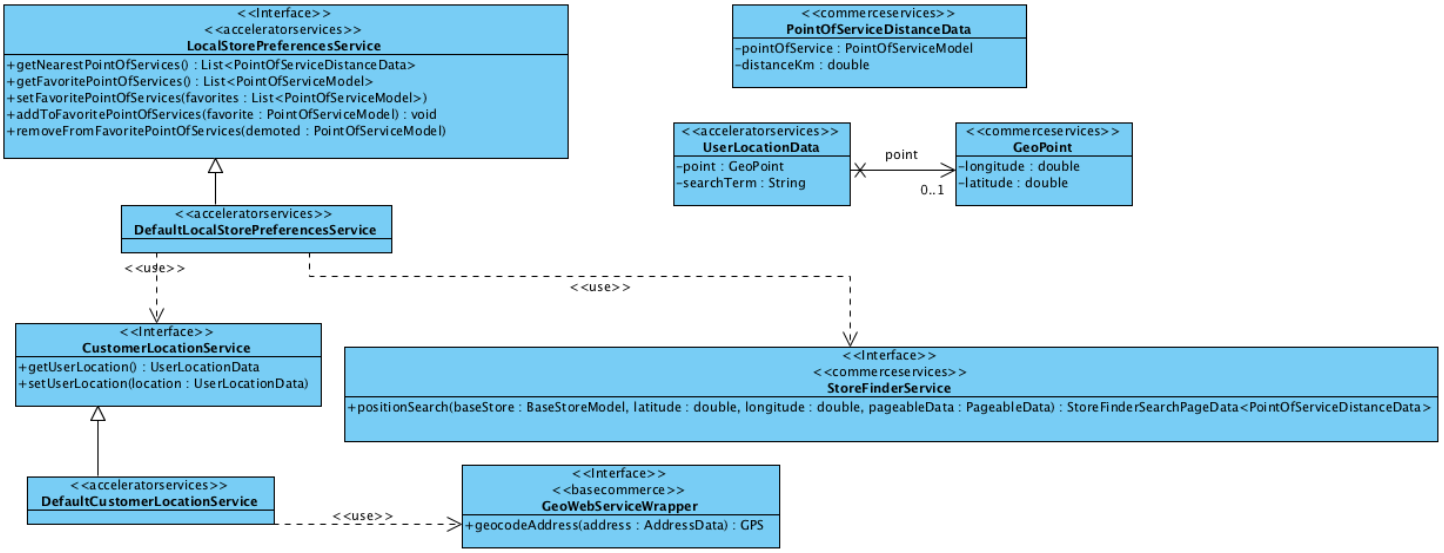
Emails are typically sent asynchronously. The option of having the `HttpRequest` object to build up URL paths is not available. The `SiteBaseUrlResolutionService` returns the base site or media URL for the specified `CMSSite`. This enables you to generate emails with valid links to site content or imagery. This service can also be applied for other asynchronous exports that might require the base URL, such as site map generation, for example.

The default implementation uses the `ConfigurationService`, and a naming convention using the site UID, as shown in the following example from the `local.properties` file:

```
website.apparel-uk.http=http://apparel.uk.local:9001/yacceleratorstorefront
website.apparel-uk.https=https://apparel.uk.local:9002/yacceleratorstorefront
website.apparel-de.http=http://apparel.de.local:9001/yacceleratorstorefront
website.apparel-de.https=https://apparel.de.local:9002/yacceleratorstorefront
website.electronics.http=http://electronics.local:9001/yacceleratorstorefront
website.electronics.https=https://electronics.local:9002/yacceleratorstorefront

media.apparel-uk.http=http://apparel.uk.local:9001
media.apparel-uk.https=https://apparel.uk.local:9002
media.apparel-de.http=http://apparel.de.local:9001
media.apparel-de.https=https://apparel.de.local:9002
media.electronics.http=http://electronics.local:9001
media.electronics.https=https://electronics.local:9002
```

Customer Location and Nearby Store Preferences

The `acceleratorservices` extension offers services for retaining the user's current location and identifying stores that are nearby to the user. These features can be useful for mobile-stores-near-me, or pickup-in-store functionality.



Customer Location Service

The `CustomerLocationService` provides the capability to cache the customer's location for use in other business logic. Currently, the location can be decoded by the `basecommerce` extension's `GeoWebServiceWrapper` using a supplied search term, where a longitude and latitude is obtained, or just by caching a `GeoPoint` directly.

The default implementation does not persist this information in the customer profile and is lost on session invalidation. However, the `UserLocationData` could be stored in a cookie if persistence is required.

Local Store Preference Service

The `LocalStorePreferencesService` uses the `CustomerLocationService` to return the nearest `PointOfService` of type STORE, given the user's current location. Favourite stores can also be saved for use in other business logic.

URL Decoding

The `acceleratorservices` extension offers URL decoding, using either a given regular expression, or else a path matcher. This logic offers the ability to decode a business object from the given URL. The `FrontendUrlDecoder` provides a generic interface for decoding these URLs. The abstract classes `BaseFrontendPathMatcherUrlDecoder` and `BaseFrontendRegexUrlDecoder` provide a default implementation for decoding the URLs, either by path matcher or regex, respectively.

HtmlMetaTag Tag Library

A tag library `HtmlMetaTag` is a convenient way to render HTML meta tags, based on a collection of `de.hybris.platform.acceleratorservices.storefront.data.MetaElementData` objects in the requests that are passed to the tag using the *<items>* attribute.

The tag can be used as follows:

```
<%@ taglib prefix="htmlmeta" uri="http://hybris.com/tld/htmlmeta" %>
<htmlmeta:meta items="${metatags}"/>
```

The result might render as follows:

```
<meta content="index,follow" name="robots">
<meta content="True" name="HandheldFriendly">
<meta content="970" name="MobileOptimized">
<meta content="width=970, target-densitydpi=160, maximum-scale=1.0" name="viewport">
```

Save Cart Support

In order to delete a saved cart, the functionality of the `CartRemovalJob` cron job has been extended. If a saved cart's parameter has been set to null as a result of calling the `flagForDeletion()` method, the `CartRemovalJob` cron job removes the cart after its period of validity expires.

For more information on save cart functionality, see commerceservices Extension.

acceleratorstorefrontcommons AddOn

The `acceleratorstorefrontcommons` AddOn is a special type of AddOn that encapsulates the common web resources for a storefront. This allows you, for example, to re-use web code that is common to the different Accelerators, such as B2B and Telco. It is also acceptable to have other AddOns depend on the `acceleratorstorefrontcommons` AddOn.

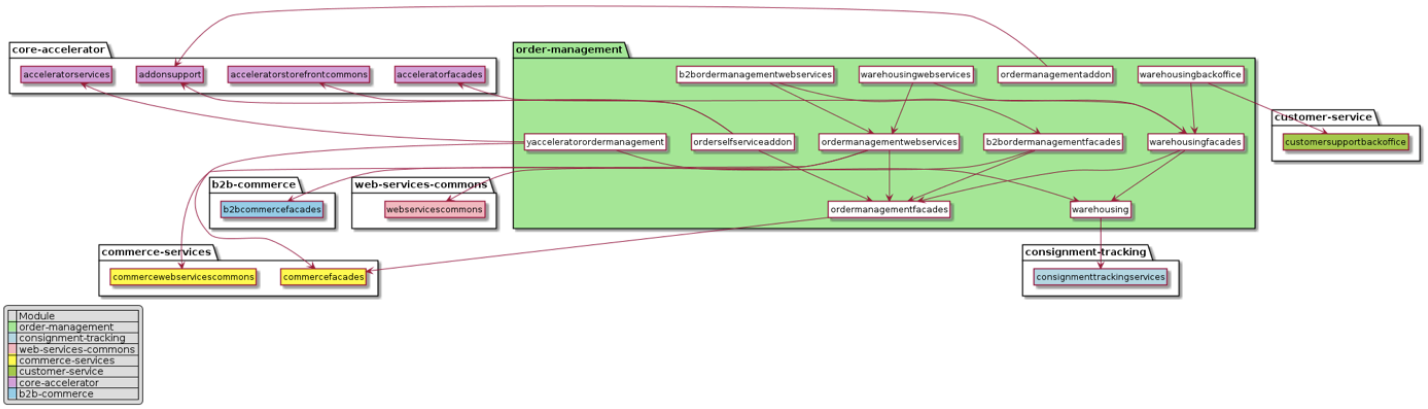
⚠ Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

ℹ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Dependencies



AddOn Definition

Name	`acceleratorstorefrontcommons`
Description	The `acceleratorstorefrontcommons` extension encapsulates common code used by the storefronts.
Requires	⛔
Author	SAP Commerce

commonweb Folder

What makes this extension a common Web extension is the existence of the `/hybris/bin/modules/core-accelerator/acceleratorstorefrontcommons/commonweb` folder. This folder contains an `src` folder, which contains defined common Web resources, such as abstract controllers, form data objects, or validators. At build time, for every extension that depends on the `acceleratorstorefrontcommons` AddOn, the Java files are compiled into the storefront extension `/hybris/bin/modules/base-accelerator/yacceleratorstorefront/web/commonwebsrc/acceleratorstorefrontcommons`.

Cart Restoration Strategies

The `NoOpCartRestorationStrategy`, `DefaultCartRestorationStrategy`, and `MergingCartRestorationStrategy` strategies give you the option to configure cart restoration behavior after login. `DefaultCartRestorationStrategy` is used as the default strategy. Of the three strategies, `NoOpCartRestorationStrategy` provides the best performance.

During login (as well as registration in B2C Accelerator), the strategies work as described in the following table:

Strategy	You Have an Empty Anonymous Cart...	You Have an Anonymous Cart with Entries...
`NoOpCartRestorationStrategy`	The cart remains empty.	The same cart from before login is used.
`DefaultCartRestorationStrategy`	The most recent session cart is used, if it exists. Otherwise, the cart remains empty.	The same cart from before login is used.
`MergingCartRestorationStrategy`	The most recent session cart is used, if it exists. Otherwise, the cart remains empty.	The anonymous cart is merged with the most recent session cart, if it exists. Otherwise, the same cart from before login is used.

During a login at checkout, we always assume that the cart contains items. The strategies work as described in the following table:

Strategy	You Have an Anonymous Cart with Entries...
`NoOpCartRestorationStrategy`	The same cart from before login is used.

Strategy	You Have an Anonymous Cart with Entries...
`DefaultCartRestorationStrategy`	The same cart from before login is used.
`MergingCartRestorationStrategy`	The anonymous cart is merged with the most recent session cart, if it exists. Otherwise, the same cart from before login is used.

The following steps describe how to switch the strategy:

1. Open `/acceleratorstorefrontcommons/resources/acceleratorstorefrontcommons/web/spring/acceleratorstorefrontcommons-spring.xml`.
2. Change the `name` attribute of the `cartRestorationStrategy` alias. The following is an example:

   ```
   <alias name="defaultCartRestorationStrategy" alias="cartRestorationStrategy"/>
   ```

3. Restart the server.

ConfigureForm Class

`ConfigureForm` is a form for transferring configuration between **product details**, **product search** page, and **product configuration** page.

to support adding entry group to the cart

AddToEntryGroupForm Form

The `AddToEntryGroupForm` form supports adding entry group to the cart.

```
package de.hybris.platform.acceleratorstorefrontcommons.forms;
/**
 * Form for adding entry group to the cart.
 */
public class AddToEntryGroupForm
{
    private String productCode;
    private Integer entryGroupNumber;
    public String getProductCode()
    {
        return productCode;
    }
    public void setProductCode(String productCode)
    {
        this.productCode = productCode;
    }
    public Integer getEntryGroupNumber()
    {
        return entryGroupNumber;
    }
    public void setEntryGroupNumber(Integer entryGroupNumber)
    {
        this.entryGroupNumber = entryGroupNumber;
    }
}
```

Installation

The `acceleratorstorefrontcommons` AddOn is installed in a similar manner to other AddOns, except that you do not need to run the `antaddoninstall` command. If the `acceleratorstorefrontcommons` AddOn is included in `extensioninfo.xml` file.

1. Add the `acceleratorstorefrontcommons` extension to your `localextensions.xml` file, as follows.

   ```
   <extension dir="${HYBRIS_BIN_DIR}/ext-accelerator/acceleratorstorefrontcommons"/>
   ```

2. Add the `acceleratorstorefrontcommons` AddOn as a required extension of your storefront extension by including it in the `extensioninfo.xml` file, as follows:

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

   i Note
   If you initialized your system before installing the `acceleratorstorefrontcommons` extension, run `ant clean all` to update the system.

   i Note
   The location of the `acceleratorstorefrontcommons` Spring context file is defined as a property in the storefront extension. The following is an example:

   ```
   yacceleratorstorefront.additionalWebSpringConfigs.acceleratorstorefrontcommons=classpath:/acceleratorstorefrontcommons/web/spring/accelerato
   ```

Modifications Checklist

The modifications that this AddOn makes to the Accelerator are listed below:

Impex Configuration Scripts	⛔
Core Data Listeners	⛔
Model Layer	⛔
Model Interceptors	⛔

Cockpit Configuration	⛔
Cockpit Beans	⛔
Validation Rules	⛔

Service Layer	✅
Façade DTO	✅
Façade Layer	⛔
CMS Components	⛔
Page Templates	⛔
JavaScript	⛔
CSS	⛔

Page Controllers	✅
Tags	⛔
TLD	⛔
Filters	⛔
MVC Interceptors	⛔
Spring Security	⛔
Message Resources	✅

Related Information

yacceleratorstorefront Extension

acceleratorwebservicesaddon AddOn

The `acceleratorwebservicesaddon` is the AddOn that extends the Omni Commerce Connect (OCC) API. Its function is to provide resources which depend on the SAP Commerce Accelerator

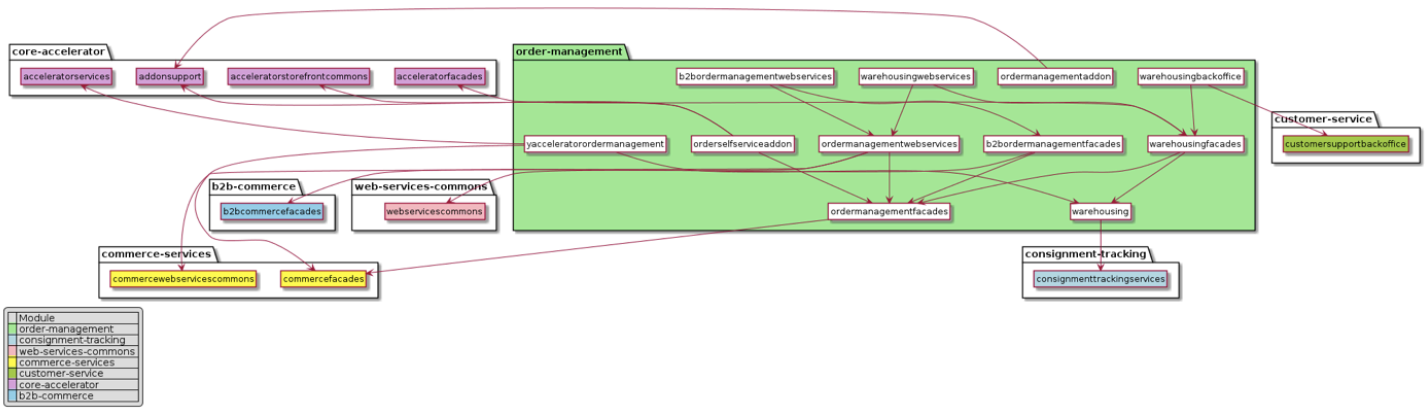
⚠ Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

ℹ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Dependencies



AddOn Definition

Name	acceleratorwebservicesaddon
Description	The `acceleratorwebservicesaddon` is an AddOn for the `ycommercewebservices` extension. Its purpose is to extend the Commerce Web Services with functionalities which depend on the SAP Commerce Accelerator.
Requires	• commercewebservicescommons Extension • acceleratorfacades Extension

Author	SAP

Supported Markets and Channels

The `acceleratorwebservicesaddon` AddOn extends the Omni Commerce Connect (OCC) API.

Available Resources

Version v2

Resource: /{site}/users/{userId}/carts/{cartId}/consolidate

Method	**GET**
Description	This method returns consolidated result consisting of items to be picked up in two different stores.
Example URL	`https//:localhost:9002/rest/v2/electronics/users/127bf550-3092-4faa-af6d-f168f8ee85fc/carts/00000036/consolidate`
Authentication	None
Request Parameters	Not applicable.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be `application/xml` or `application/json`.
Representations	`application/xml`, `application/json`

Resource: /{site}/users/{userId}/carts/{cartId}/consolidate

Method	**POST**
Description	This method consolidates different locations the items may be picked up at.
Example URL	• User already logged in: `https://localhost:9002/rest/v2/electronics/users/127bf550-3092-4faa-af6d-f168f8ee85fc/carts/00000036/consolidate?storeName=Shinbashi` • Anonymous user: `https://localhost:9002/rest/v2/electronics/users/anonymous/carts/81918b5fbc416eba4a5f71feebb04eb1b282bb7d/consolidate?storeName=Shinbashi`
Authentication	None
Request Parameters	Parameters to be provided as a POST body: • storeName (Style: storeName, Required: true): Name of store where items will be picked up.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	`application/xml`, `application/json`

Resource: /{site}/users/{userId}/carts/{cartId}/payment/sop/request

Method	**GET**
Description	This method returns information required to create a subscription contacting directly with the payment provider.These details contain payment provider url and a list of parameters required to create a subscription. These parameters are partially filled in with appropriate values based on information stored in the cart.
Example URL	• User already logged in: `https://localhost:9002/rest/v2/electronics/users/127bf550-3092-4faa-af6d-f168f8ee85fc/carts/00000036/payment/sop/request` • Anonymous user: `https://localhost:9002/rest/v2/electronics/users/anonymous/carts/81918b5fbc416eba4a5f71feebb04eb1b282bb7d/payment/sop/request`
Authentication	Access is available for ROLE_CUSTOMERGROUP, ROLE_GUEST, ROLE_CUSTOMERMANAGERGROUP, ROLE_TRUSTED_CLIENT
Request Parameters	• responseUrl (Required : true) Value for orderPage_cancelResponseURL, orderPage_declineResponseURL, orderPage_receiptResponseURL • extendedMerchantCallback (Default:false) The flag decides which url for merchant callback will be returned. **ℹ Note** Remember to set this flag to true if you want to use the SOP flow with extended merchant callback.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.

Representations	`application/xml`, `application/json`

ℹ Note

For details on SOP Payment, see: Payment in OCC

Resource: /{site}/users/{userId}/carts/{cartId}/payment/sop/request

Method	**GET**
Description	This method returns information required to create a subscription contacting directly with the payment provider.These details contain payment provider url and a list of parameters required to create a subscription. These parameters are partially filled in with appropriate values based on information stored in the cart.
Example URL	• User already logged in: `https://localhost:9002/rest/v2/electronics/users/127bf550-3092-4faa-af6d-f168f8ee85fc/carts/00000036/payment/sop/request` • Anonymous user: `https://localhost:9002/rest/v2/electronics/users/anonymous/carts/81918b5fbc416eba4a5f71feebb04eb1b282bb7d/payment/sop/request`
Authentication	Access is available for ROLE_CUSTOMERGROUP, ROLE_GUEST, ROLE_CUSTOMERMANAGERGROUP, ROLE_TRUSTED_CLIENT
Request Parameters	• responseUrl (Required : true) Value for orderPage_cancelResponseURL, orderPage_declineResponseURL, orderPage_receiptResponseURL • extendedMerchantCallback (Default:false) The flag decides which url for merchant callback will be returned. **ℹ Note** Remember to set this flag to true if you want to use the SOP flow with extended merchant callback.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be `application/xml` or `application/json`.
Representations	`application/xml`, `application/json`

Resource: /{site}/users/{userId}/carts/{cartId}/payment/sop/response

Method	**GET**
Description	This method returns information about create subscription request results. Available status values: • 202 (Accepted): no response from the payment provider. • 200 (OK): payment provider response is positive and payment details are returned. • 400 (Bad request): payment provider response is negative and an error with details information is returned.
Example URL	• User already logged in: `https://localhost:9002/rest/v2/electronics/users/127bf550-3092-4faa-af6d-f168f8ee85fc/carts/00000036/payment/sop/response` • Anonymous user: `https://localhost:9002/rest/v2/electronics/users/anonymous/carts/81918b5fbc416eba4a5f71feebb04eb1b282bb7d/payment/sop/response`
Authentication	Access is available for ROLE_CUSTOMERGROUP, ROLE_GUEST, ROLE_CUSTOMERMANAGERGROUP, ROLE_TRUSTED_CLIENT
Request Parameters	Not applicable.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be `application/xml` or `application/json`.
Representations	`application/xml`, `application/json`

Resource: /{site}/users/{userId}/carts/{cartId}/payment/sop/response

Method	**DELETE**
Description	This method deletes payment provider response associated with a given cart.
Example URL	• User already logged in: `https://localhost:9002/rest/v2/electronics/users/127bf550-3092-4faa-af6d-f168f8ee85fc/carts/00000036/payment/sop/response` • Anonymous user: `https://localhost:9002/rest/v2/electronics/users/anonymous/carts/81918b5fbc416eba4a5f71feebb04eb1b282bb7d/payment/sop/response`
Authentication	Access is available for ROLE_CUSTOMERGROUP, ROLE_GUEST, ROLE_CUSTOMERMANAGERGROUP, ROLE_TRUSTED_CLIENT
Request Parameters	Not applicable.

Resource: /{site}/users/{userId}/carts/{cartId}/payment/sop/response

Method	**POST**
Description	This method handles response from payment provider subscription request. • If the payment provider response is positive, the payment details are defined.

	• If the payment provider response is negative, an error with details information is returned.
Example URL	• User already logged in: `https://localhost:9002/rest/v2/electronics/users/127bf550-3092-4faa-af6d-f168f8ee85fc/carts/00000036/payment/sop/response` • Anonymous user: `https://localhost:9002/rest/v2/electronics/users/anonymous/carts/81918b5fbc416eba4a5f71feebb04eb1b282bb7d/payment/sop/response`
Authentication	Access is available for ROLE_CUSTOMERGROUP, ROLE_GUEST, ROLE_CUSTOMERMANAGERGROUP, ROLE_TRUSTED_CLIENT
Request Parameters	Parameters to be provided as a POST body: • parameters returned by payment provider in create subscription process • savePaymentInfo If the payment details should be saved for the customer and than could be reused for future orders • fields Response configuration (list of fields, which should be returned in response). For more information, see: DTO Mapping and Response Configuration
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be **application/xml** or **application/json**.
Representations	`application/xml`, `application/json`

Version v1

Resource: /{site}/cart/consolidate

Method	**GET**
Description	This method returns consolidated result consisting of items to be picked up in two different stores.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/consolidate`
Authentication	None
Request Parameters	Not applicable.
Headers	• **Accept** (Style: header, Required: true): Accept header needs to be sent with each request. It can be `application/xml` or `application/json`.
Representations	`application/xml`, `application/json`

Resource: /{site}/cart/consolidate

Method	**POST**
Description	This method consolidates different locations the items may be picked up at.
Example URL	`https://localhost:9002/rest/v1/electronics/cart/consolidate?storeName=Shinbashi`
Authentication	None
Request Parameters	Parameters to be provided as POST body: • storeName (Style: storeName, Required: true) Name of store where items will be picked up.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be `application/xml` or `application/json`.
Representations	`application/xml`, `application/json`

Resource: /{site}/customers/current/locationLatLong

Method	**PUT**
Description	This method is required to calculate the distance between the user and store. It sets the location of a current user based on latitude and longitude parameters to return user's location. Used in facets that return information about the stores available near the user's current location.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/locationLatLong`
Authentication	None
Request Parameters	Parameters to be provided as PUT body: • latitude (Style: body, Required: true): Latitude location parameter. • longitude (Style: body, Required: true):

	Longitude location parameter.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be `application/xml` or `application/json`.
Representations	`application/xml`, `application/json`

Resource: /{site}/customers/current/location

Method	**PUT**
Description	This method is required to calculate the distance between a user and the store. It sets the location of a current user based on a search term and returns the location of a such a user. If no location is found, the `NoLocationFoundException` is thrown.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/location`
Authentication	None
Request Parameters	Parameters to be provided as PUT body: • location (Style: body, Required: true): Free-text location parameter.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be `application/xml` or `application/json`.
Representations	`application/xml`, `application/json`

Resource: /{site}/customers/current/location

Method	**GET**
Description	This method returns the location of a current user. Returns **null** if not set.
Example URL	`https://localhost:9002/rest/v1/electronics/customers/current/location`
Authentication	None
Request Parameters	Not applicable.
Headers	• Accept (Style: header, Required: true): Accept header needs to be sent with each request. It can be `application/xml` or `application/json`.
Representations	`application/xml`, `application/json`

How to Install

To install the `acceleratorwebservicesaddon` AddOn perform the following steps:

1. Add the `acceleratorwebservicesaddon` AddOn to your `localextensions.xml` file, making sure the listed required extensions are also included.

```
<extension name="acceleratorwebservicesaddon" />
<extension name="acceleratorfacades"/>
<extension name="commercewebservicescommons"/>
<extension name="ycommercewebservices"/>
<extension name="addonsupport"/>
```

2. Run the `ant addoninstall` command. This generates the correct properties in the `project.properties` file of the `acceleratorwebservicesaddon` AddOn and adds a dependency from your commerce web services to the `acceleratorwebservicesaddon` AddOn.

   For `ycommercewebservices`, the command line looks like the following:

```
ant addoninstall -Daddonnames="acceleratorwebservicesaddon" -DaddonStorefront.ycommercewebservices="ycommercewebservices"
```

   For extensions generated from the `ycommercewebservices` template which name is `mycommercewebservices`.

```
ant addoninstall -Daddonnames="acceleratorwebservicesaddon" -DaddonStorefront.ycommercewebservices="mycommercewebservices
```

3. After the script has completed its work successfully, rebuild the system.

Modifications Checklist

The modifications that this AddOn introduces to OCC are listed below:

Impex Configuration Scripts	⛔
Core Data Listeners	⛔
Model Layer	⛔

Model Interceptors	⛔
Cockpit Configuration	⛔
Cockpit Beans	⛔
Validation Rules	⛔

OCC API	✅
OCC DTO	✅
Service Layer	⛔
Facade DTO	⛔
Facade Layer	⛔
CMS Components	⛔
Page Templates	⛔
JavaScript	⛔
CSS	⛔

Page Controllers	⛔
Tags	⛔
TLD	⛔
Filters	⛔
MVC Interceptors	⛔
Spring Security	⛔
Message Resources	⛔

Related Information

Creating an AddOn for OCC Web Services

addonsupport Extension

The `addonsupport` extension provides a small collection of convenient classes for use when constructing your own Accelerator AddOns.

⚠ Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

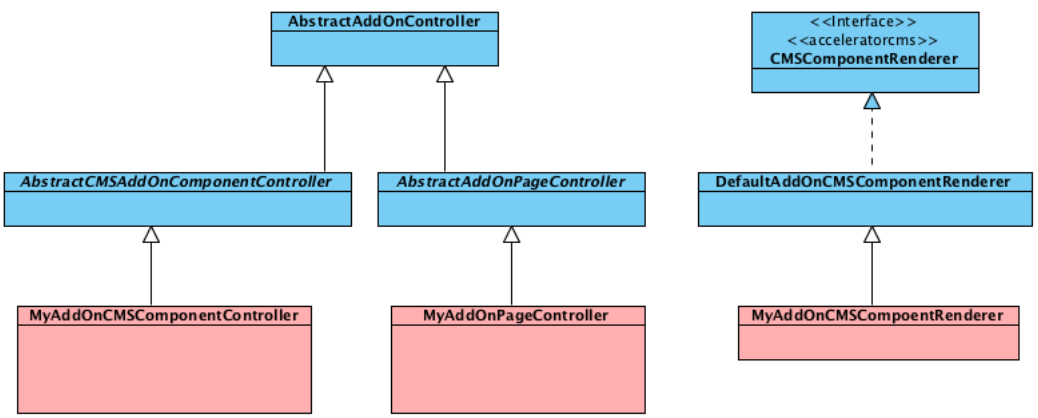
i Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Overview

Developers working with Accelerator AddOns can use the `addonsupport` extension as an optional toolkit to extend the controllers and renderers. It is recommended to use functionality in this extension rather than in the B2C or B2B-specific storefront versions. Doing so maximizes the possibility of reusing your AddOns in more than one storefront template like B2C and B2B. Basically, you can reuse the AddOn with many storefronts.

The following class diagram shows the extension points for your AddOn. In addition to a base CMS component controller, you can also use a JSP Includes directly in a renderer rather than using an MVC Controller to process the business logic.



Javascript Framework

The Javascript Framework and API can inject localized properties defined in an AddOn into the JavaScript of a page. The properties need to be defined in the message resource bundle with the prefix `base.js` found in `<myAddon>/acceleratoraddon/web/webroot/WEB-INF/messages`.

To expose the localized properties to JavaScript in the view, some beans need to be defined in your AddOn's Spring configuration file, as follows:

```
<bean id="defaultMyAddonJavaScriptMessageResourceAccessor" parent="abstractJavaScriptMessageResourceAccessor">
                <property name="addOnName" value="MyAddon" />
        </bean>

        <bean id="defaultMyAddonBeforeViewJsPropsHandlerAdaptee" parent="abstractBeforeViewJsPropsHandlerAdaptee" class="de.hybris.platform.addonsup
                <property name="messageSource" ref="defaultMyAddonJavaScriptMessageResourceAccessor" />
        </bean>

        <bean id="getMyAddonBeforeViewHandler" parent="beforeViewHandlerAdapter">
                <property name="adaptee">
                        <ref bean="defaultMyAddonBeforeViewJsPropsHandlerAdaptee" />
                </property>
        </bean>
        <bean id="MyAddonBeforeViewHandlerListMergeDirective" depends-on="beforeViewHandlersList" parent="listMergeDirective" >
                <property name="add" ref="getMyAddonBeforeViewHandler" />
```

This will expose every property defined in `base.js_<locale>.properties` to the view.

```
<script>
        /*<![CDATA[*/
        ACC.addons = {};        //JS holder for addons properties
        ACC.addons.MyAddon = [];
        ACC.addons.MyAddon['key.1'] = 'value 1';
        ACC.addons.MyAddon['key.2'] = 'value 2';

        /*]]>*/
</script>
```

Default AddOn CMS Component Renderer

The `addonsupport` extension provides a default CMS component renderer, `DefaultAddOnCMSComponentRenderer`, which implements `CMSComponentRenderer` to inject the frontend properties of a component into a view.

The renderer is declared as a bean with the ID `addOnJspIncludeCMSComponentRenderer`, as follows:

```
<bean id="addOnJspIncludeCMSComponentRenderer"
                class="de.hybris.platform.addonsupport.renderer.impl.DefaultAddOnCMSComponentRenderer">
                <property name="typeService" ref="typeService" />
                <property name="uiExperienceService" ref="uiExperienceService" />
                <property name="modelService" ref="modelService" />
                <property name="cmsComponentService" ref="cmsComponentService" />
        </bean>
```

CMS Component Renderer Registry

The `acceleratorcms` extension provides the `DefaultCMSComponentRendererRegistry` class, which holds a map of `CMSComponentRenderer`, which is keyed by the `CmsComponentType` (for example, `ProductAddToCartComponent`), and allows you to inject a custom renderer for an existing or new CMS component. If a CMS component does not have a renderer assigned to it, a `defaultCmsComponentRenderer` is used. The `defaultCmsComponentRenderer` can also be overwritten using Spring.

Registering a Custom Renderer for a Specific CMS Component Type Code

A special extended version of the `AddOnCMSComponentRendererRegistry` registry is provided that allows an AddOn to register a custom renderer for a specific CMS component type code. You can do all of this the Spring configuration.

ⓘ Note

If you use the `ComponentController` approach, the following steps are not necessary.

1. Implement a custom renderer extending from `DefaultAddOnCMSComponentRenderer`, as follows:

   ```
   public class MyCustomComponentRenderer<C extends SimpleCMSComponentModel> extends DefaultAddOnCMSComponentRenderer<C>
   {
           private final static String COMPONENT = "component";
           @Override
           protected Map<String, Object> getVariablesToExpose(final PageContext pageContext, final C component)
           {
                   final Map<String, Object> model = new HashMap<String, Object>();
                   model.put(COMPONENT, component);
                   return model;
           }
   }
   ```

2. Declare your custom renderer with the `addOnJspIncludeCMSComponentRenderer` as a parent, as follows:

   ```
   <bean id="myCustomComponentRenderer"  class="com.hybris.addon.MyCustomComponentRenderer" parent="addOnJspIncludeCMSComponentRenderer"/>
   ```

3. Register your custom renderer against the CMS components type code by declaring a renderer mapping bean with `addonCmsComponentRendererMapping` as a parent, as follows:

```
<bean id="myCustomComponentRendererMapping" parent="addonCmsComponentRendererMapping" >
      <property name="typeCode" value="MyCustomComponent" />
      <property name="renderer" ref="myCustomComponentRenderer" />
</bean>
```

i Note

If your CMS component does not execute any back-end business logic, and is purely a place to capture configuration to affect the HTML markup that is produced, we strongly recommend writing a custom renderer. This will result in a performance improvement by not having to include the various context-includes and forwards that come by invoking a Spring MVC Controller and View Resolving.

Automatic Setup Services

AddOns may wish to automatically run configuration code during the initialization process (typically using ImpEx scripts) to self-configure themselves. The standard initialization process supports a Convention over Configuration (CoC) approach that allows extensions to add one or more `essentialdata` or `projectdata` scripts to hook into the initialization process. Furthermore, an extension can register beans annotated with the `SystemSetup` annotation to execute more complex initialization steps.

An AddOn may need to hook into the standard Accelerator initialization process to execute its site-specific configuration scripts immediately after Accelerator has run the Core Data Import, and before the storefront sample data is imported. The Accelerator supports the raising of Core and Sample Data Import Events. These are convenient plugin points that AddOns can use to step in and import additional data, or to run other business logic. Furthermore, these events also deliver as metadata the IDs of the product and content catalogs, and store names for the content just imported. This allows AddOns to restrict scripts only to imported catalogs or stores.

There are many examples of core data that an AddOn may want to import using this event listener approach. The following are some common examples:

- CMS page templates, including Structure View Velocity Scripts
- Default CMS pages
- Default CMS components
- Solr index configuration.

The `addonsupport` extension delivers a number a convenient features to enable self-configuring, automated imports of data that are dependent on the Accelerator store configuration.

- An event listener enabling AddOns to easily hook into the initialization process
- Channel awareness that is only loaded for B2B stores
- A template CoC folder and filename structure that matches the Accelerator import steps, to reduce the need for AddOns to include their own custom event listeners or system setup classes
- Support for parameterising ImpEx scripts, which removes the need to hard-code macros, such as the catalog name. This enables configuration-based ImpEx scripts to be linked less to actual sample data.

Core Data Import

You can register an Event Listener in the Global Application Context. The simplest way is to specify the parent modifier as the `abstractCoreDataImportedEventListener` bean. This will give your listener the default setup, as follows:

```
<bean id="myAddonCoreDataImportedEventListener" parent="abstractCoreDataImportedEventListener"  >
        <property name="supportedChannels">
            <list value-type="de.hybris.platform.commerceservices.enums.SiteChannel">
                <value>B2C</value>
                <value>B2B</value>
            </list>
        </property>
        <property name="addonExtensionMetadata">
            <bean class="com.hybris.addon.data.AddonExtensionMetadata">
                <property name="baseExtensionName" value="myaddon" />
            </bean>
        </property>
    </bean>
```

At a minimum, you must specify the base extension name for your AddOn. This will be used as the extension to find the templated ImpEx configuration. If you need different AddOns for different channels, but you have a common library and configuration extension, you can add a `suffixChannel` flag to the AddOn Extension metadata. This will assume an AddOn named with the convention `$baseExtensionName$lowercase(channel)`, such as `myaddonb2c`.

```
<bean class="de.hybris.platform.addonsupport.addon.data.AddonExtensionMetadata">
    <property name="baseExtensionName" value="myaddon" />
    <property name="suffixChannel" value="true" />
</bean>
```

You can use the `supportedChannels` modifier to restrict the event listener to processing only a subset of channels, for example, if you have a B2B-specific AddOn:

```
<property name="supportedChannels">
    <list value-type="de.hybris.platform.commerceservices.enums.SiteChannel">
        <value>B2B</value>
    </list>
</property>
```

Once a core data event listener has been registered, ImpEx scripts can be stored in your AddOn resources folder, which will get automatically run on notification of the `CoreDataImportedEvent`. In this case, your AddOn scripts will be run straight after the extension's project data is invoked, rather than your AddOn project data.

To invoke these scripts, you have to publish a `CoreDataImportedEvent`.

Although none of the following files are mandatory, this table indicates the order in which they are executed. It is the same order as in the Accelerator systems, and the names follow the standard Accelerator naming and structure:

Step	Resource Type	Filenames
1	Catalog configuration	*<addonextensionname>*/resources/*<addonextensionname>*/import/productCatalogs/template/catalog.impex
2	CMS configuration	*<addonextensionname>*/resources/*<addonextensionname>*/import/contentCatalogs/template/catalog.impex *<addonextensionname>*/resources/*<addonextensionname>*/import/contentCatalogs/template/cms-content.impex
3	Solr configuration	*<addonextensionname>*/resources/*<addonextensionname>*/import/solr/template/solr.impex

All scripts are parameterised with the following attributes, which are defined in `de.hybris.platform.commerceservices.impex.data.ImpexMacroParameterData`:

Parameter	Sample Data Example
contentCatalog	apparel-ukContentCatalog
productCatalog	apparelProductCatalog
siteUid	apparel-uk
storeUid	apparel-uk
configExtensionName	myaddon
addonExtensionName	myaddon
solrIndexedType	apparel-ukProductType
channel	B2C

In addition, you can add populators to your import configuration, which allows you to inject additional attributes. The simplest approach is to add extra macro fields to the `ImpexMacroParameterData.additionalParameterMap` map property. They key of the map is the macro name, so it is important to use a unique name. If you prefer a more type-safe approach, you can extend the `ImpexMacroParameterData` class using your AddOn `beans.xml` file, as follows:

```
<bean class="com.hybris.addon.impex.ImpexMacroParameterData">
    <property name="myExtraAddonAttribute" type="String"/>
</bean>
```

To populate these extra ImpEx macro attributes, you can use the populator pattern. Four groups of populators are run during the import process. You can merge your populator using the `ModifyPopulatorList` feature. Since an import for a store may involve multiple content catalogs and sites, you might have content catalog, product catalog, or site-specific configurations. In this case, you need to ensure your populator is invoked at the best time. The following table offers some guidelines:

Populator Bean Name	Type
impexMacroParametersConverter	Global macro value
selectedProductCatalogImpexMacroParametersPopulators	Product catalog-specific macro value
selectedBaseSiteImpexMacroParametersPopulators	Base site-specific macro value
selectedContentCatalogImpexMacroParametersPopulators	Content catalog specific macro value

AddOn Sample Data Import

Similar to the Core Data Import, the `GenericAddonSampleDataEventListener` waits for a `SampleDataImportedEvent` to be published by an extension and will import your AddOn's sample data using the `AddOnSampleDataImportService`. This can be useful when adding new attributes to stores, catalogs, and so on. The `DefaultAddonSampleDataImportService` imports the files in a similar (though not exact) structure to the import of `SampleDataImportService` provided in the `acceleratorservices` extension.

```
<bean id="myAddOnSampleDataEventListener" parent="abstractGenericAddOnSampleDataEventListener"  >
        <property name="extensionName" value="myAddon" />
</bean>
```

The `DefaultAddonSampleDataImportService` import structure begins looking at the files in *<addonextensionname>*/resources/*<addonextensionname>*/import/*. The steps in the loading are as following:

Step	Resource	File Directories / Name
1	Common Data	*<addonextensionname>*/resources/*<addonextensionname>*/import/cockpits *<addonextensionname>*/resources/*<addonextensionname>*/import/common-addon-extra.impex
2	Product Catalog Data	*<addonextensionname>*/resources/*<addonextensionname>*/import/productCatalogs
3	Content Catalog & Store Data related to the catalog	*<addonextensionname>*/resources/*<addonextensionname>*/import/contentCatalogs *<addonextensionname>*/resources/*<addonextensionname>*/import/stores/
4	Store Locations	*<addonextensionname>*/resources/<addonextensionname>/import/stores/
5	Store Initial Data	*<addonextensionname>*/resources/<addonextensionname>/import/stores/

i Note

To see the specific files that are loaded, you can inspect `DefaultAddonSampleDataImportService`.

AddOn extensions should not have explicit dependencies on other AddOns, because it must always be possible to install one AddOn without requiring another AddOn to be present. However, there are cases when you may wish an AddOn to load specific sample data if another specific AddOn is present, and to safely ignore this sample data if the other AddOn is not present. To allow for this, the `GenericAddOnSampleDataEventListener` and `AddOnSampleDataImportService` classes have been extended to support such cases.

To address the undeterministic order in which two independent AddOns may be loaded, AddOns are now capable of handling events that indicate when sample data imports for other AddOns have finished, and will load the corresponding sample data by convention in the case of such an event. To assist in this process, an event cache is used to defer events to a later point in the initialization process, if necessary.

Let's look at a scenario where you have an AddOn, `addon1`, and you want it to load a specific set of sample data that depends on another AddOn, `addon2`, but you only want this sample data to be loaded if `addon2` is present. The following details ensure that the specific sample data is loaded at the correct time:

- The following entry is added to `addon1/project.properties.template`:

  ```
  addon1.awareof.addons=addon2
  ```

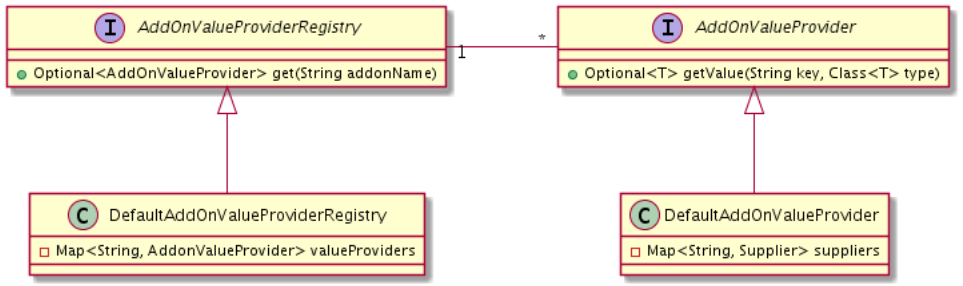
- The impex file that contains the dependent sample data must be placed in a subfolder of `addon1/resources/addon1/import/addons/addon2`, where the subfolder structure below the `addon2` folder follows the same structure as the subfolders that appear below `addon1/resources/addon1/import`.

  For example, you may have a folder structure similar to the following: `addon1/resources/addon1/import/stores`. In this case, the corresponding folder structure for where to place the impex file that contains the dependent sample data would look as follows: `addon1/resources/addon1/import/addons/addon2/stores`.

AddonValueProvider

It is considered best practice that Accelerator AddOns are not tightly coupled. In other words, there shouldn't be an explicit dependency between two AddOns because they also have to work independently of each other in different scenarios and recipes. However, there are cases where a loose coupling is necessary to allow one AddOn to access values provided by a service or facade that is only visible to another AddOn.

As a result, a value provider mechanism has been introduced in the `addonsupport` extension. The `addonsupport` extension now provides an `AddOnValueProviderRegistry` that enables AddOns to access value providers defined and registered by other AddOns using their extension name. Such value providers must implement the `AddOnValueProvider` interface, which has a generic `getValue(String key, Class<T> type)` method and is also defined in the `addonsupport` extension.



Configuring an AddonValueProvider for an AddOn

Let's take a scenario where you want to create an AddOnValueProvider for a custom AddOn called `myaddon`, and the AddOnValueProvider provides a Boolean value for key `booleanValue` and a String value for key `stringValue`. The first step is to implement `Suppliers` for the two values. `Suppliers` have to implement the `java.util.function.Supplier` interface. Using the example AddOn `myaddon`, the following is an example from `myaddon/acceleratoraddon/web/src/de/hybris/platform/myaddon/valueprovider/BooleanSupplier.java`:

```
public class BooleanSupplier implements Supplier<Boolean>
{
	@Override
	public Boolean get()
	{
		// Retrieve and return Boolean value
	}
}
```

Continuing with the example AddOn `myaddon`, the following is an example from `myaddon/acceleratoraddon/web/src/de/hybris/platform/myaddon/valueprovider/StringSupplier.java`:

```
public class StringSupplier implements Supplier<String>
{
	@Override
	public String get()
	{
		// Retrieve and return String value
	}
}
```

The next step is to create Spring beans for the `Suppliers` and the value provider in the web spring configuration file of `myaddon`. The value provider should be based on the `DefaultAddOnValueProvider` implementation and has to be added to the Map of value providers which is defined in the `addonsupport` extension. Continuing with the example AddOn `myaddon`, the following is an example from `myaddon/myaddon/resources/assistedservicestorefront/web/spring/myaddon-web-spring.xml`:

```
<bean id="booleanSupplier" class="de.hybris.platform.myaddon.valueprovider.BooleanSupplier">
<bean id="stringSupplier" class="de.hybris.platform.myaddon.valueprovider.StringSupplier">

<bean id="myAddOnValueProvider" class="de.hybris.platform.addonsupport.valueprovider.impl.DefaultAddOnValueProvider">
	<property name="suppliers">
		<map key-type="java.lang.String" value-type="java.util.function.Supplier">
			<entry key="booleanValue" value-ref="booleanSupplier" />
			<entry key="stringValue" value-ref="stringSupplier" />
		</map>
	</property>
```

```
</bean>

<bean depends-on="addOnValueProviderMap" parent="mapMergeDirective">
        <property name="key" value="myaddon" />
        <property name="value" ref="myAddOnValueProvider" />
</bean>
```

Usage

The code of another AddOn which doesn't have a dependency on `myaddon` has to inject the `addOnValueProviderRegistry` bean and can access the Boolean and String value provided by `myaddon` in the following manner:

```
final Optional<AddOnValueProvider> valueProvider = getAddOnValueProviderRegistry().get("myaddon");

if (valueProvider.isPresent())
{
        final Optional<Boolean> booleanValue = valueProvider.get().getValue("booleanValue", Boolean.class);
        final Optional<String> stringValue = valueProvider.get().getValue("stringValue", String.class);

        // check if values are present and do sth. with them
}
```

ⓘ Note

The framework uses the `java.util.Optional<T>` class for return types in order to force consumers to check if the expected provider and values are actually present. This makes sure situations in which `myaddon` is not installed are handled gracefully.

Core Accelerator Implementation

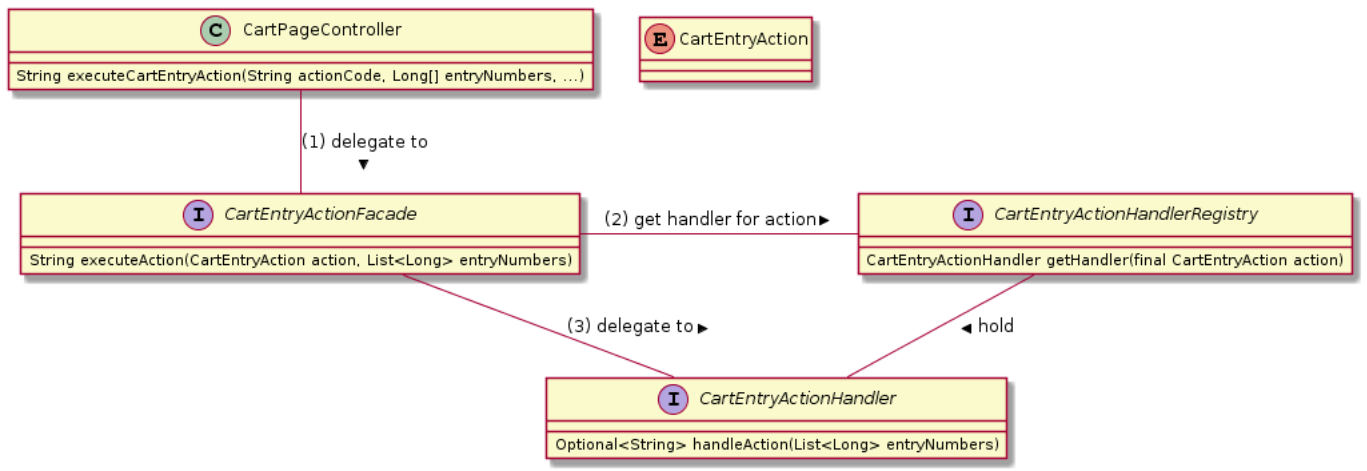
The Core Accelerator provides a fully responsive storefront implementation, and also allows you to add new actions that trigger custom processing on cart entries with the extensible cart item menu.

Extensible Cart Item Menu

The Extensible Cart Item Menu feature allows you to add new actions that trigger custom processing on cart entries.

In the Cart page, the menu that appears at the far right of cart items is now extendable. You can add new actions that trigger custom processing on cart entries. A custom action can be displayed only for certain cart entries, depending on the rules implemented in the custom action. After the action is performed, the cart page can be refreshed, or the user can be directed to a new page, depending on what is required for your use case.

This is achieved by providing a generic request mapping for `CartEntry`-related actions in the `CartPageController`, which delegates the execution of a `CartEntryAction` to the configured implementation of the `CartEntryActionFacade`. The facade in turn retrieves the configured `CartEntryActionHandler` implementation for the given action from a registry. Finally, the handler executes the action on the cart entry and returns a URL that the controller redirects to. Since the `CartEntryActionFacade` assumes that the operation is executed on the current session cart, the action and the cart entry number are the only necessary parameters. This is illustrated in the following diagram.



ⓘ Note

If there is more than one multi-dimensional product that has the same base product added to the cart, the action on the entry is provided with all the multi-dimensional products that belongs to this entry.

Adding a Custom Action

Learn to add a new action that triggers custom processing on cart entries

Context

The following procedure describes how to define a custom cart entry action in an AddOn

Procedure

1. Add the new action value to the `CartEntryAction` enum in the `beans.xml` file of your AddOn.

   The following is an example from `resources/myaddon-beans.xml`:

   ```
   <enum class="de.hybris.platform.acceleratorfacades.cart.action.CartEntryAction">
       <value>MY_ACTION</value>
   </enum>
   ```

2. Implement the `CartEntryActionHandler` interface for you action in your AddOn.

   The following is an example from `acceleratoraddon/web/src/<package>/MyCartEntryActionHandler.java`:

   ```
   public class MyCartEntryActionHandler implements CartEntryActionHandler
   {
       // Example redirect url string.  Same format as you would use in the return
       // value from a SpringMVC controller.
       private static final String REDIRECT_URL = "redirect:/my-url";

       @Override
       public Optional<String> handleAction(final List<Long> entryNumbers) throws CartEntryActionException
       {
           try
           {
               // execute your handler's logic
           }
           catch (final Exception e) // your impl should handle specific exceptions
           {
               // When a CartEntryActionException is thrown, the CartPageController will redirect back to the cart page
               // and display the custom error message with the key provided by the getErrorMessageKey() method.
               throw new CartEntryActionException("Failed to handle action", e);
           }
           // Returning an empty optional tells the CartPageController to redirect back to the cart page.
           // Provide a redirect url if you need to redirect somewhere else.
           return Optional.empty();
       }

       @Override
       public String getSuccessMessageKey()

       {
           return "my.success.message.key";
       }

       @Override
       public String getErrorMessageKey()
       {
           return "my.error.message.key";
       }

       @Override
       public boolean supports(final CartEntryModel cartEntry)
       {
           // execute logic to determine whether your handler supports a cart entry here
           return true;
       }
   }
   ```

3. Add your handler implementation to the registry in the web Spring configuration of your AddOn.

   The following is an example from `resources/myaddon/web/spring/myaddon-web-spring.xml`:

   ```
   <!-- Define the handler -->
   <bean id="myCartEntryActionHandler" class="de.hybris.platform.myaddon.cart.action.impl.MyCartEntryActionHandler">
       <!-- inject properties here -->
   </bean>

   <!-- Define the action as a spring bean. -->
   <bean id="myCartEntryAction" class="de.hybris.platform.acceleratorfacades.cart.action.CartEntryAction" factory-method="valueOf">
       <constructor-arg>
           <value>MY_ACTION</value>
       </constructor-arg>
   </bean>

   <!-- Use the action as the key to retrieve the handler -->
   <bean id="myMergeDirective" depends-on="cartEntryActionHandlerMap" parent="mapMergeDirective">
       <property name="key" ref="myCartEntryAction"/><!-- The key must be of type CartEntryAction-->
       <property name="value" ref="myCartEntryActionHandler"/>
   </bean>
   ```

4. Add storefront localization properties for the tool menu label as well as error and success messages in the base*.properties files of your AddOn.

   The following is an example from `acceleratoraddon/web/webroot/WEB-INF/messages/base_en.properties`:

   ```
   # This is the menu item label.
   # The key has to follow a convention: basket.page.entry.action.[ACTION_CODE]
   basket.page.entry.action.MY_ACTION=My Action
   ```

```
# The keys for success and error messages don't follow a convention, just use the keys defined in your handler implementation.
my.success.message.key=Success Message
my.error.message.key=Error Message
```

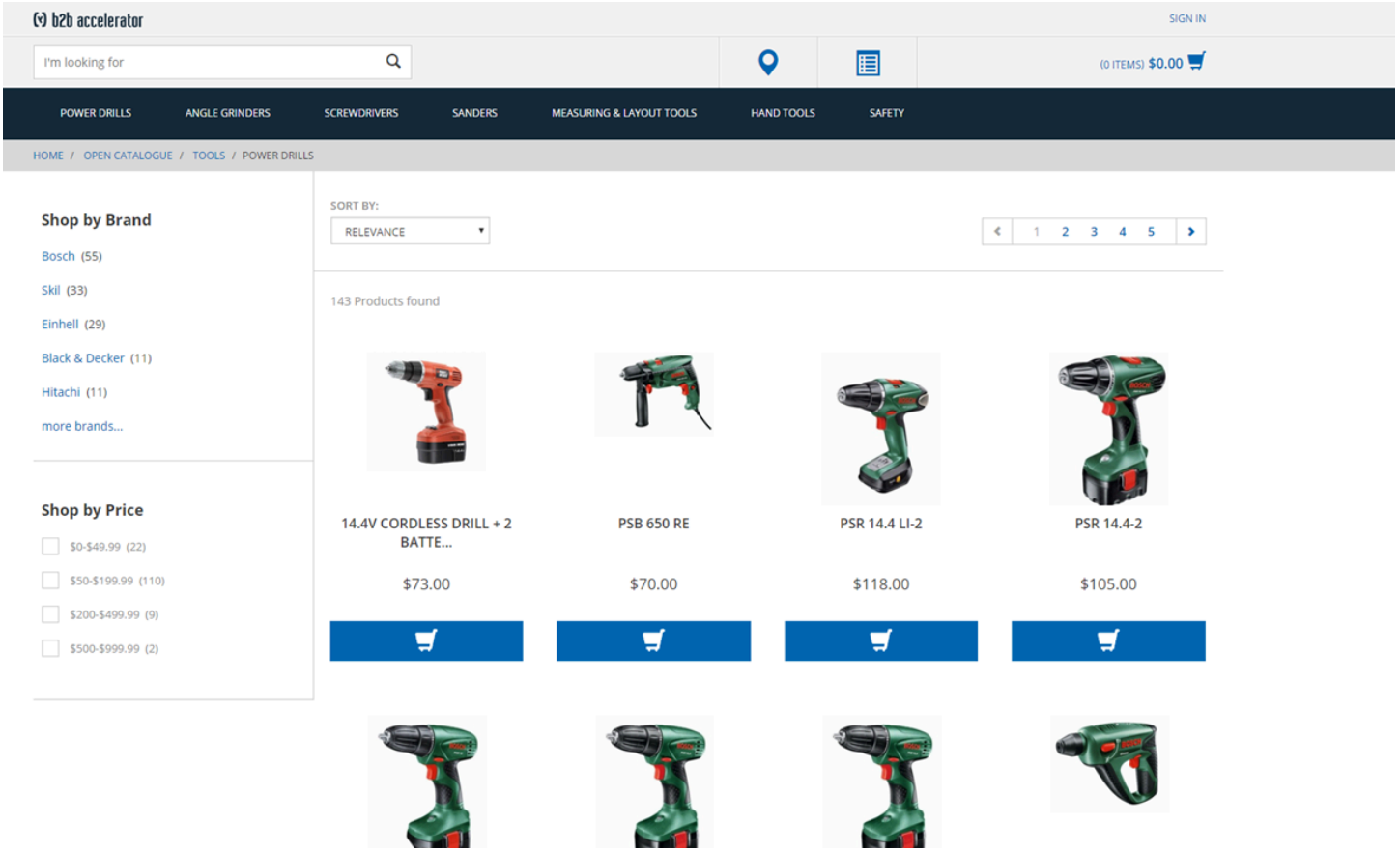
Responsive Storefronts

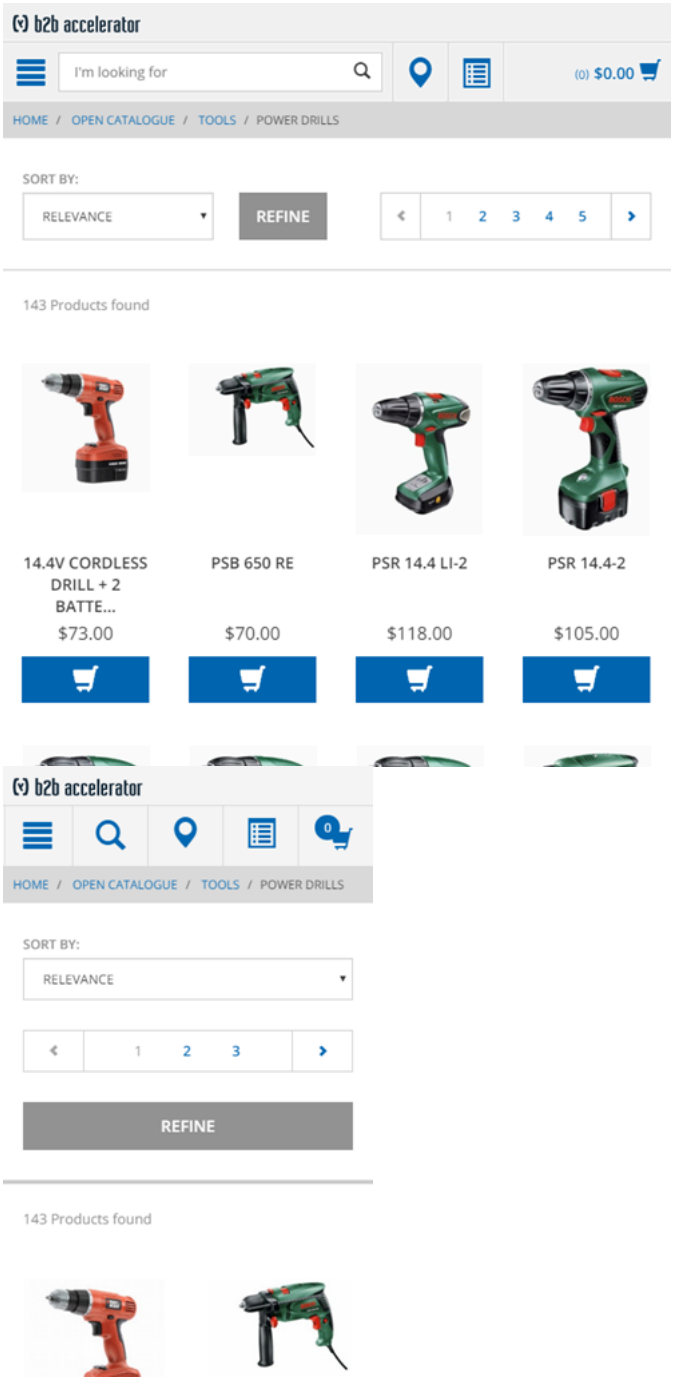
SAP Commerce Accelerator provides a fully responsive storefront implementation for B2B, B2C, financial, and telco storefronts.

Overview

Commerce Accelerator includes a complete responsive storefront implementation that is enabled by default for B2B, B2C, Financial Services Accelerator, and Telco and Utilities Accelerator. There is no need to separately configure desktop and mobile storefronts; website pages are automatically rendered in one of four sizes (mobile, tablet, desktop, and ultrawide desktop) depending on the device used to view the page.

The next images show the same product list page in desktop, tablet, and phone sizes respectively.





Note

The responsive storefront is currently available for B2B, B2C, Financial Services Accelerator, and Telco and Utilities Accelerator only. The desktop version is still available for use with other Accelerator implementations.

Themes

Two themes that contain the storefront color and font definitions are provided. The **alpha** (blue) theme is used for the B2C storefront by default, and the **lambda** (black) theme is used for the B2B storefront. Themes use Less to build the responsive storefront pages. You can use these themes as starting points for your own, or even create new themes from scratch. You can find more information about themes in these topics:

- For more information on customizing and creating themes, see Responsive Themes
- To learn about the theme build process using Less, see Responsive Website Build Process
- To learn how to use the alpha and lambda themes as a base for new themes, see Creating New Themes

Available Responsive WCMS Components

The responsive implementation of Commerce Accelerator does not define all of the WCMS components that are included in the desktop implementation of Accelerator. However, there is no restriction to prevent the addition of these components to a responsive page. The fully functional responsive pages use the WCMS components that are defined. You can see which components are defined for responsive by referring to files in the `/yacceleratorstorefront/web/webroot/WEB-INF/views/responsive/cms/` folder. If a non-implemented component is required, you can provide a JSP definition in this directory.

Responsive Images

In Accelerator, the `simpleResponsiveImageComponent` handles images in a responsive way. For information, see Responsive Image WCMS Component.

Responsive Website Build Process

The `yacceleratorstorefront` extension provides a process for building a responsive website front end that supports LESS.

Overview

To support LESS and other JS libraries, a build process to generate a `_ui` folder that contains the appropriate JS and CSS styling is available. This process can be started either through Ant or Grunt.

There are two ways that you can modify the existing front end:

- Modify the files in the `_ui` folder directly. If you are using this approach, the `_ui-src` and `GruntFile.js` may be removed. Note that this is not the recommended approach to modify the storefront.
- Modify any required storefront files in the `_ui-src` folder and then generate the `_ui` folder using one of the methods described in the Generating the Storefront Files section. This is the recommended approach and it is described in more details in the following sections.

Using the _ui-src Folder

The recommended way to modify the existing front end is to modify the `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive` folder.

This directory is used as an input to generate the `_ui` folder that is used to render the different pages of the storefront, which are located in the `yacceleratorstorefront/web/webroot/_ui/responsive` folder. The `_ui-src` folder contains JS testing, full libraries, and the LESS files used to generate the CSS. These files are not all required to be loaded by the web server. Only the required files are transferred from the `_ui-src` folder to the `_ui` folder.

The `_ui-src` folder has the following structure:

- `responsive`: Contains all files related to the responsive user experience
  - `lib`: Contains third-party and SAP Commerce framework to enable the responsive web design
    - `bootstrap-x.x.x` (where `x.x.x` represents the version number): Contains the Twitter bootstrap framework (http://getbootstrap.com/)
    - `common`: Contains third-party JavaScript libraries, such as JQuery libraries.
    - `ybase-x.x.x`: Contains Accelerator-related JS and LESS SAP Commerce framework. The `addons.less` file (located in the `less` subfolder) stores references to *<AddOn name>*`.less` files of any installed AddOns. This file is initially empty but is populated when AddOns are installed if those AddOns have a corresponding *<AddOn name>*`.less` file. See the Responsive AddOn Support section below for more details on AddOns.
  - `themes`: Contains the different themes
    - `lambda`: Contains the black theme example
      - `fonts`: Contains specific fonts for the theme
      - `less`: Contains LESS files that are used to generate CSS later on
    - `alpha`: Contains the blue theme example
      - `fonts`: Contains specific fonts for the theme
      - `less`: Contains LESS files that are used to generate CSS later on
- `shared`: Contains all files that are common across the different UI experiences
  - `less`: Contains the LESS variables and their mappings to be generated. For more information, see JS, LESS and JSPs: Build Process for Generating Variables.

Responsive AddOn Support

Some AddOns support responsive pages out of the box. These AddOns have an *<AddOn name>*`.less` file located in the *<AddOn Name>*`/acceleratoraddon/web/webroot/WEB-INF/_ui-src/responsive/less` folder.

If an AddOn does not have this file, create the `_ui-src/responsive/less` folders under `WEB-INF` and then create the *<AddOn name>*`.less` file in the `less` folder. This file contains import references to the AddOn's LESS files (for example, `@import "buttons.less"`). The references must be added to this file manually.

When `addoninstall` is executed, the references are added to the `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/ybase-x.x.x/less/addons.less` file if that reference does not exist already. When the build process is triggered by executing either `ant clean all` or `ant build`, LESS files in the AddOn's `_ui-src/responsive/less` folder are copied to the storefront in the `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/addons/`*<addon name>*`/responsive/less` folder.

Build Process

When the build process is initiated using `ant` or Grunt, the `_ui` folder is generated from the `_ui-src` folder by:

- Generating the CSS files from the LESS files
- Copying the JS files
- Copying the fonts

The following sections provide details on which files are affected in each step of the build process.

Generating the CSS Files from the LESS Files

The LESS files contained in `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/themes/<theme>/less` and `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/shared/less/` are used to generate the CSS files. If AddOn are installed, LESS files from the `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/addons/<addon name>/responsive/less/` folder are also used to generated the CSS files as long as a reference to `<addon name>.less` exists in the `addons.less` file.

The `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/themes/<theme>/less` contains the following files:

- `style.less`: This file references the LESS files present in the `ybase` library (which, in turn, references the Twitter bootstrap). Any updates to the LESS files can be added here or referenced within this file. This file is used to generate the theme-specific CSS, such as `yacceleratorstorefront/web/webroot/_ui/responsive/theme-alpha/css/style.css`. If AddOns are installed, this file also contains relative import file references pointing to the `addons.less` file (for example, `@import "../../../lib/ybase-0.1.0/less/addons.less";`)
- `variables.less`: This file imports both the `generatedVariables.less` file and the `variableMapping.less` file, which are used to define global variables. For AddOns, the variables used within the AddOn's LESS files should be available and set within this theme-specific variable file.
- `theme-variables.less`: This file contains the color variables and values for the storefront.

The `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/shared/less` contains the following files:

- `generatedVariables.less`: This file is the generated global variable file.
- `variableMapping.less`: This file contains the mapping from variable names to names appropriate for LESS.

For more information, see JS, LESS and JSPs: Build Process for Generating Variables. For more information on the LESS compiler itself, see http://lesscss.org/ .

Copying the Libraries and JavaScript Files

To have the web server load the appropriate JS files, the appropriate JS files are copied from the `_ui-src` to the `_ui` folder. The following JavaScript files are copied:

- `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/ybase-x.x.x/js/*` to `yacceleratorstorefront/web/webroot/_ui/responsive/common/js`
- `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/common/js/*` to `yacceleratorstorefront/web/webroot/_ui/responsive/common/js`
- `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/bootstrap-x.x.x/*` to `yacceleratorstorefront/web/webroot/_ui/responsive/common/bootstrap`

When the `ant` build is triggered, the ANT task `yaccleratorstorefront_compileuisrc` `ant` task in `yacceleratorstorefront/buildcallbacks.xml` copies JS files from these folders:

- `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/ybase-x.x.x/js/*`
- `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/common/js/*`

The JS files are copied to the `yacceleratorstorefront/web/webroot/_ui/responsive/common/js` folder.

Any modifications done to JS files in the `yacceleratorstorefront/web/webroot/_ui/responsive/common/js` folder are always replaced by the contents of JS files in the `_ui-src` folders as soon as an `ant` build is run. We recommended to always modify the JS files in the `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/ybase-x.x.x/js/*` and `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/common/js/*` folders and let the `ant` task copy the JS files to the `yacceleratorstorefront/web/webroot/_ui/responsive/common/js` folder.

Copying the Fonts

To allow the addition and modification of fonts, the fonts files are also copied from `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/themes/<theme>/fonts` to `yacceleratorstorefront/web/webroot/_ui/responsive/theme-<theme>/fonts`.

HTML

There is one source of HTML that is located in the following directories:

- `/hybris/bin/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/tags`
- `/hybris/bin/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/views`

For localization purposes, you can localize the files located in the `/hybris/bin/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/messages` directory.

Tables are avoided for CSS-based styling and Google optimization. Tables have a predefined structure that cannot be changed using CSS, so a DIV-based HTML structure is used instead. The DIV-based HTML structure is more generic and flexible.

You can use the Firefox Firebug extension to explore and analyze the structure of the web front end of Accelerator. This extension allows you to select any item in the DOM tree by clicking on it in the browser. It also displays all of the available styles for this item, and which CSS files contain this information.

Generating the Storefront Files

To allow out-of-the-box functionality, as well as extensibility,SAP Commerce Accelerator provides two ways to generate the `_ui` folder, as described in the following sections.

Ant

The tasks performed by the `yaccleratorstorefront_compileuisrc` target are triggered at the end of the build process for the storefront. The `yaccleratorstorefront/buildcallbacks.xml` file defines the target `yaccleratorstorefront_compileuisrc`. To invoke this, run `ant yaccleratorstorefront_compileuisrc`. This will perform the steps defined in the Build Process described above. When working with AddOns that contain LESS files, the `ant clean` command must be followed by either `ant build` or `ant clean all`. For the exact execution rules for this command, refer to the callbacks files.

Grunt

SAP Commerce Accelerator does not provide a Grunt setup out of the box, but it does provides the configuration. For more information about Grunt and how to install it, see http://gruntjs.com/ . SAP Commerce Accelerator provides a `Gruntfile.js` file, which performs the same actions (that is, compile and copy) as the `ant` command.

ⓘ Note

`Gruntfile.js` does not support LESS for AddOns; the provided configuration file only handles updates to storefront files

Responsive Themes

Theme variables make it easy to change the look and feel of your storefront. Learn about the variables and their default values.

Overview

SAP Commerce Accelerator themes use variables for storefront UI elements so that colors can quickly be changed. For example, the *<@text-color>* variable defines the text color and the *<@link-color>* variable defines the link text color. SAP Commerce Accelerator comes with two themes: alpha (blue) and lambda (black).

Colors that are used in both themes are defined in the `variables.less` file, located in the `/hybris/bin/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/ybase-0.1.0/less` directory. Colors that are used in specific themes are defined in the `theme-variables.less` file, located in the `/hybris/bin/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/themes/<theme name>/less` directory.

For more information on how to generate updated storefront colors, see Responsive Website Build Process

Default Themes

Two default themes are used for the storefronts out-of-the-box. The lambda theme is used for the B2B storefront, and the alpha theme is used for the B2C storefront. These themes can be used as a starting point to create your own themes by changing any variable value as required. The following images show the main colors, fonts, and icons used in each theme.

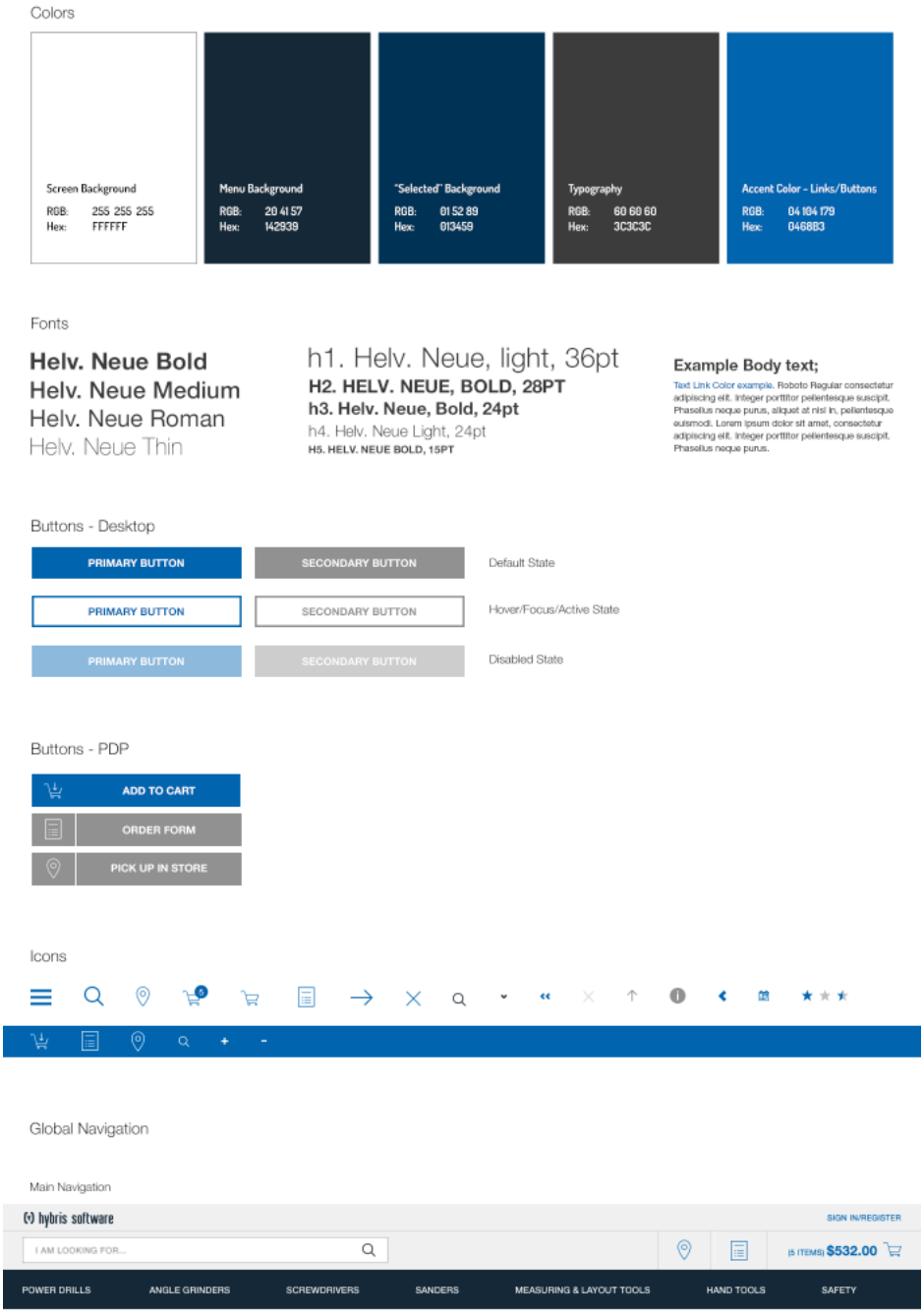


Figure: Lambda theme overview

Figure: Alpha Theme Overview

The following sections provide details on all the theme variables and what they represent, as well as the default color values for each theme.

Brand Colors

Brand colors are used as global colors for multiple UI elements. By default, UI elements that use the *<@brand-primary>* color value include:

- navigation icons
- link text
- buttons

For example, the variable that defines the navigation icon colors (*<@nav-icon-color>*) has a value of *<@brand-primary>*.

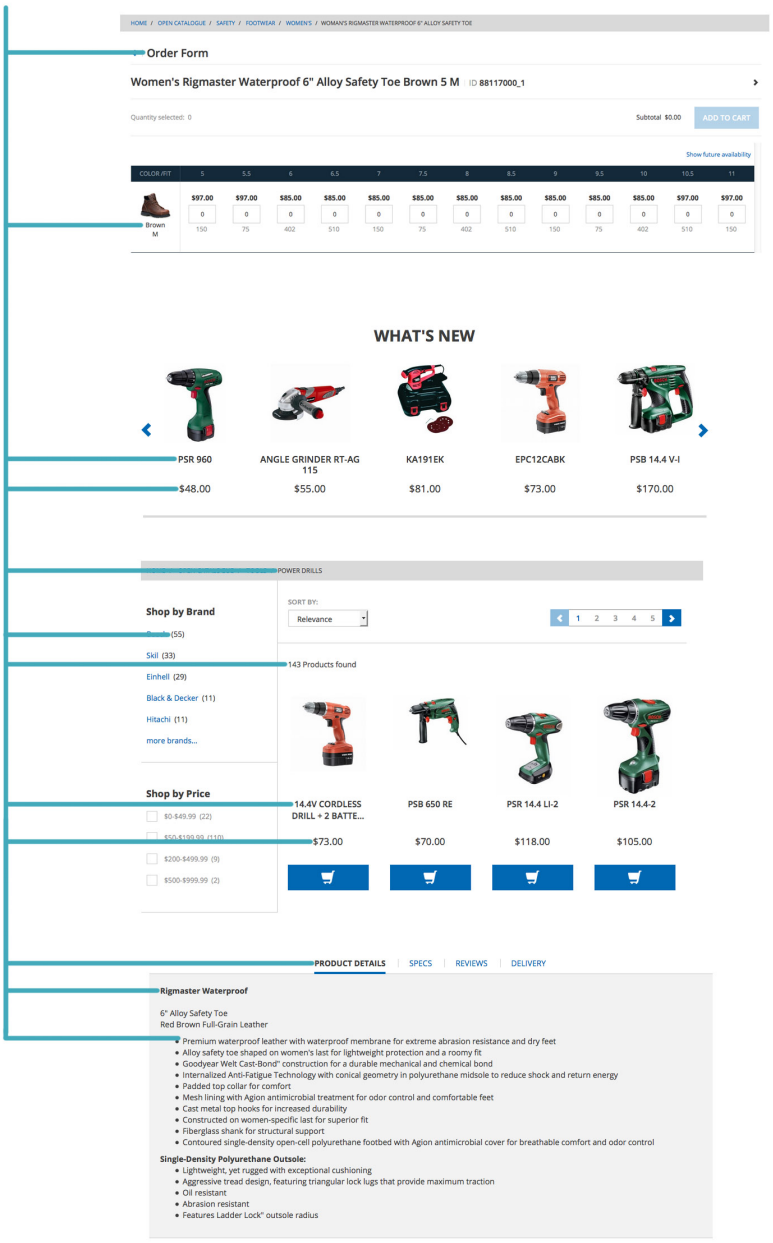
Note that by default, the lambda theme has two brand colors, and the alpha theme has three brand colors. This table shows the default values for the brand colors of each theme.

Theme Variable	Lambda Theme Value	Alpha Theme Value
@brand-primary	#0068b3	#47b6b1
@brand-secondary	#929292	#ec7205
@brand-tertiary	n/a	#5adfd9

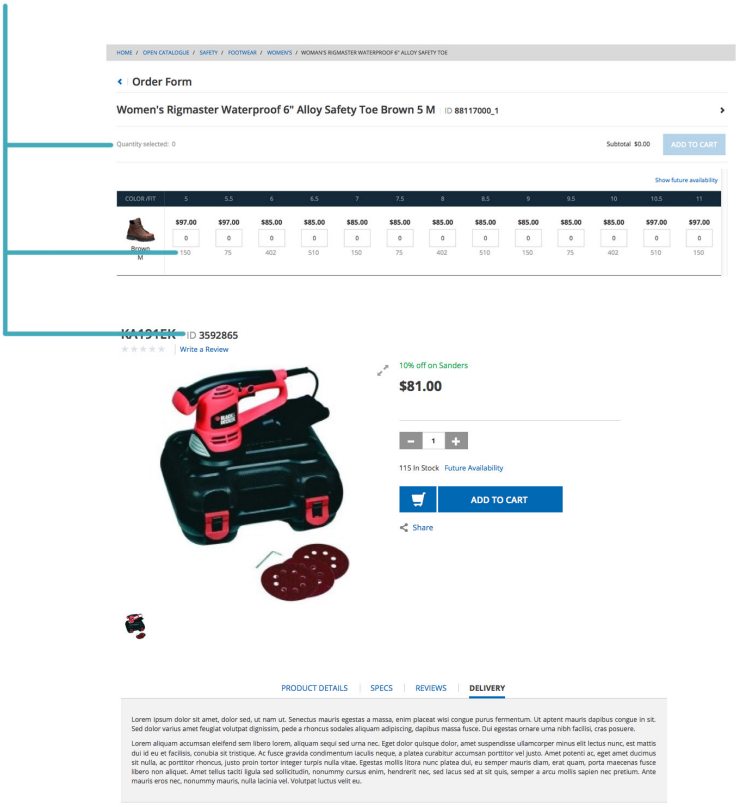
Typography

The following images identify the various text elements (primary and secondary text, link text, heading text) and their corresponding variable name. The subsequent table provides the default color values for each of these variables for both themes.

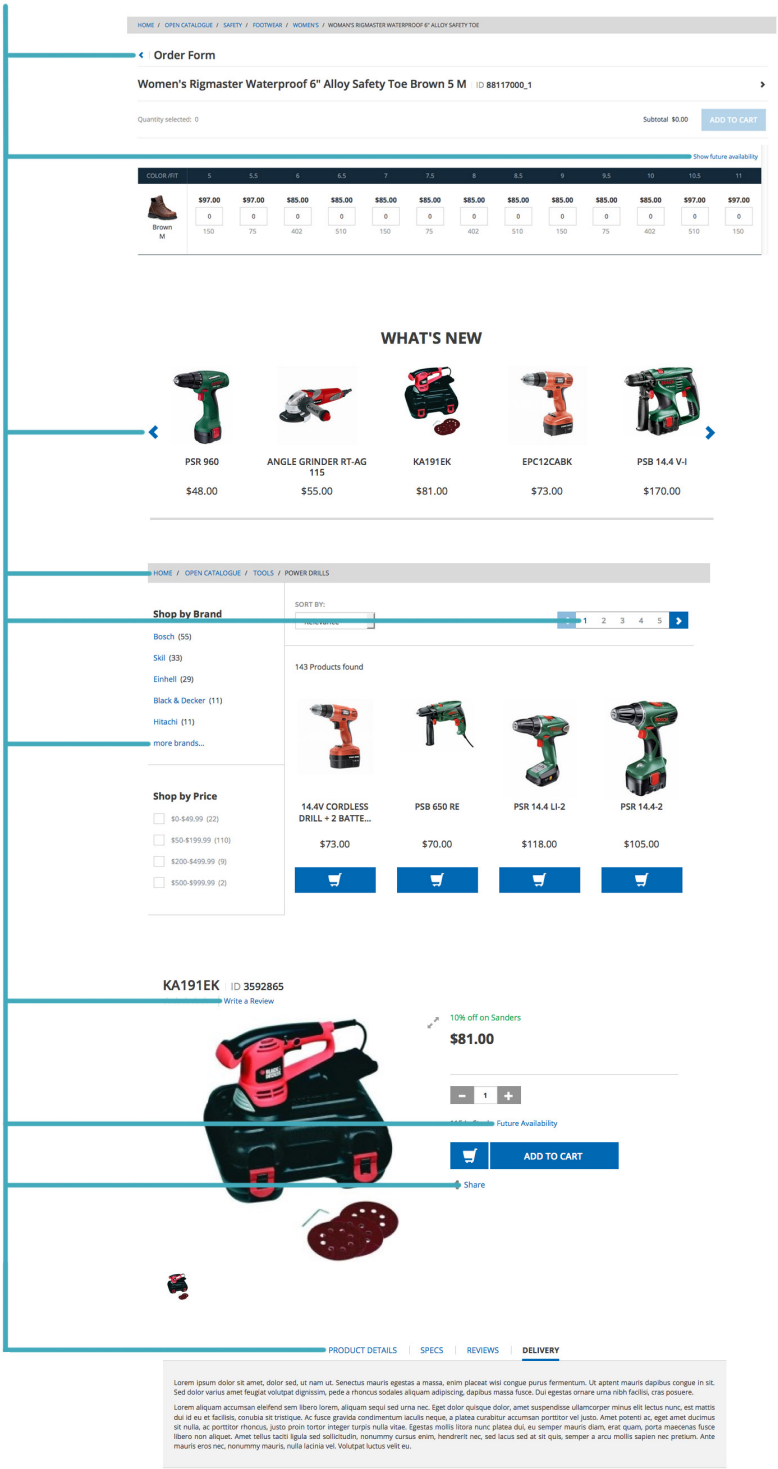
@text-color


Order Form
Women's Rigmaster Waterproof 6" Alloy Safety Toe Brown 5 M
WHAT'S NEW
PSR 960
$48.00
ANGLE GRINDER RT-AG 115
$55.00
KA191EK
$81.00
EPC12CABK
$73.00
PSB 14.4 V-I
$170.00
POWER DRILLS
Shop by Brand
Skil (33)
Einhell (29)
Black & Decker (11)
Hitachi (11)
more brands...
Shop by Price
143 Products found
14.4V CORDLESS DRILL + 2 BATTE...
$73.00
PSB 650 RE
$70.00
PSR 14.4 LI-2
$118.00
PSR 14.4-2
$105.00
PRODUCT DETAILS
SPECS
REVIEWS
DELIVERY
Rigmaster Waterproof
6" Alloy Safety Toe
Red Brown Full-Grain Leather
Premium waterproof leather with waterproof membrane for extreme abrasion resistance and dry feet
Alloy safety toe shaped on women's last for lightweight protection and a roomy fit
Goodyear Welt Cast-Bond™ construction for a durable mechanical and chemical bond
Internalized Anti-Fatigue Technology with conical geometry in polyurethane midsole to reduce shock and return energy
Padded top collar for comfort
Mesh lining with Agion antimicrobial treatment for odor control and comfortable feet
Cast metal top hooks for increased durability
Constructed on women-specific last for superior fit
Fiberglass shank for structural support
Contoured single-density open-cell polyurethane footbed with Agion antimicrobial cover for breathable comfort and odor control
Single-Density Polyurethane Outsole:
Lightweight, yet rugged with exceptional cushioning
Aggressive tread design, featuring triangular lock lugs that provide maximum traction
Oil resistant
Abrasion resistant
Features Ladder Lock™ outsole radius

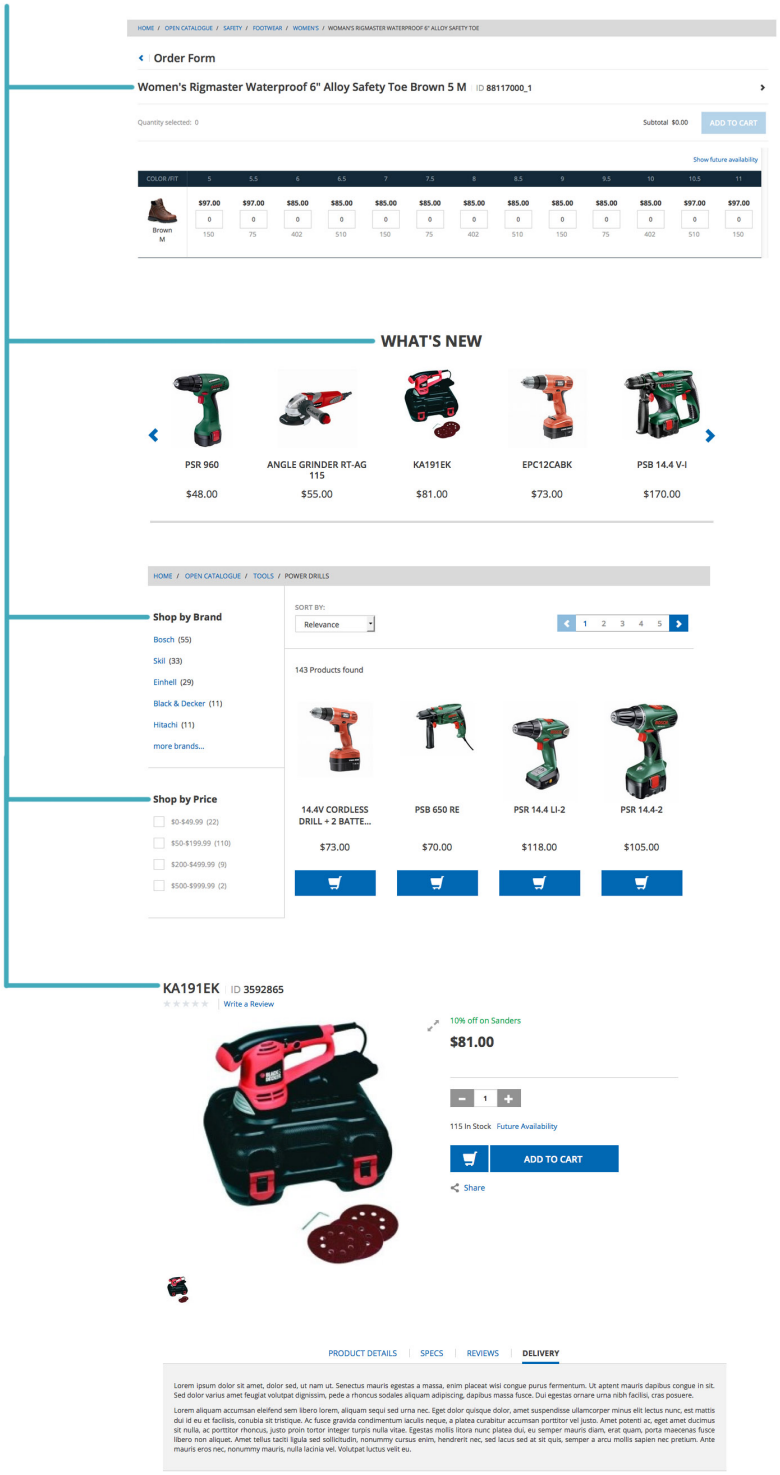
@text-color-secondary



@link-color


Order Form
Women's Rigmaster Waterproof 6" Alloy Safety Toe Brown 5 M
WHAT'S NEW
PSR 960
ANGLE GRINDER RT-AG 115
KA191EK
EPC12CABK
PSB 14.4 V-I
Shop by Brand
Shop by Price
143 Products found
14.4V CORDLESS DRILL + 2 BATTE...
PSB 650 RE
PSR 14.4 LI-2
PSR 14.4-2
KA191EK
ID 3592865
Write a Review
10% off on Sanders
$81.00
Future Availability
ADD TO CART
Share
PRODUCT DETAILS
SPECS
REVIEWS
DELIVERY

@headline-color

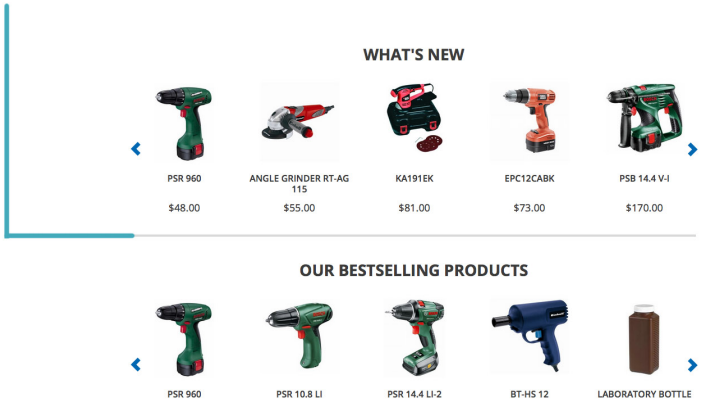


Theme Variable	Lambda Theme Value	Alpha Theme Value
@text-color	#3c3c3c	#19212b
@text-color-secondary	@brand-secondary: #929292	#929292
@link-color	#0068b3	@brand-secondary: #ec7205
@headline-color	#3c3c3c	#154057

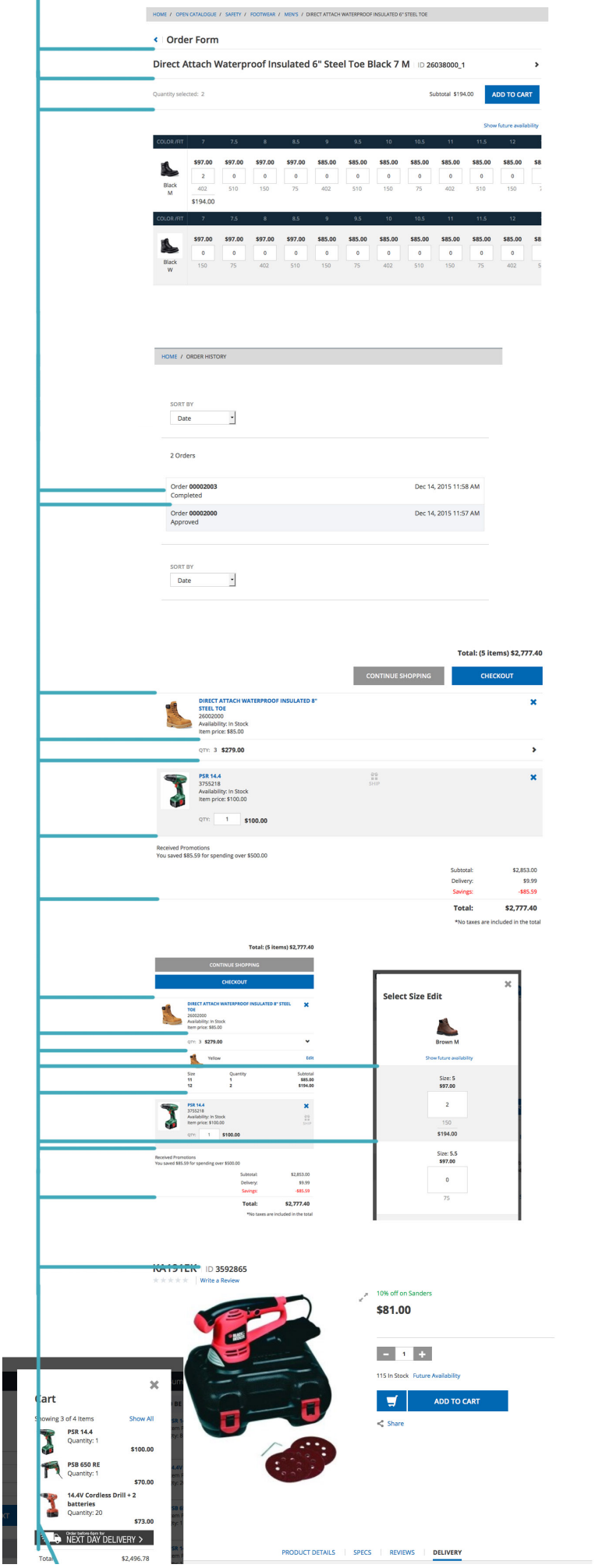
Border Colors

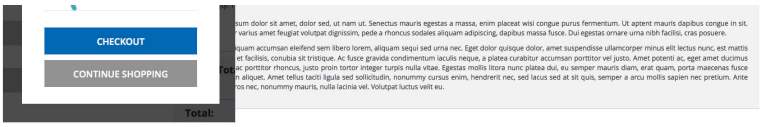
The following images identify the various border elements (primary, secondary, and tertiary) and their corresponding variable name. The subsequent table provides the default color values for each of these variables for both themes.

@border-color

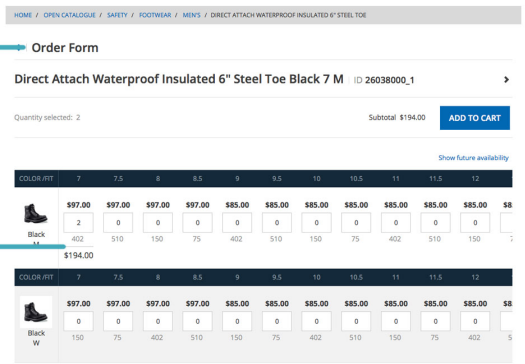


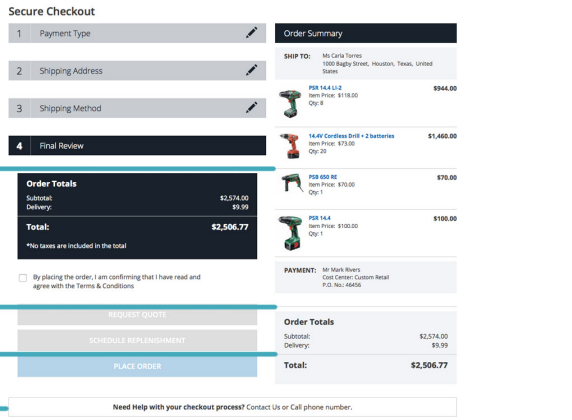
@border-color-2

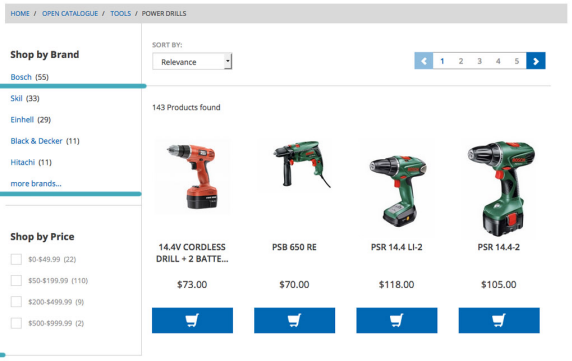


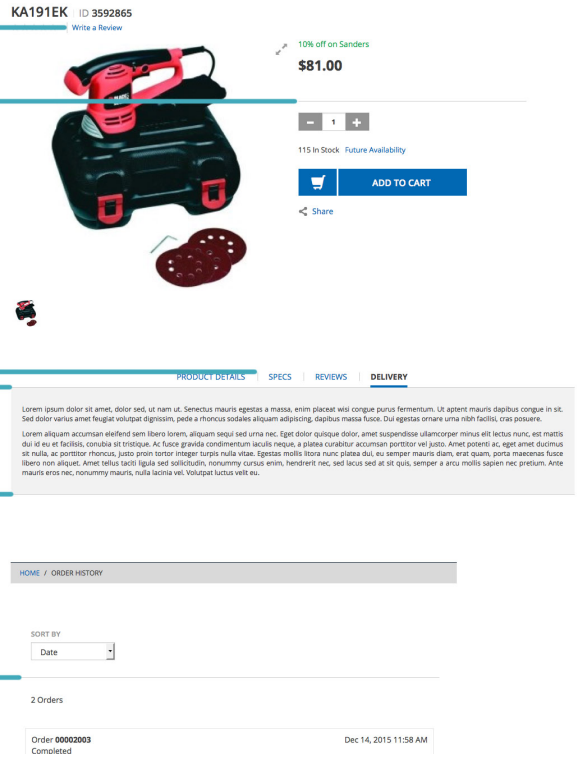

CHECKOUT
CONTINUE SHOPPING
Total:

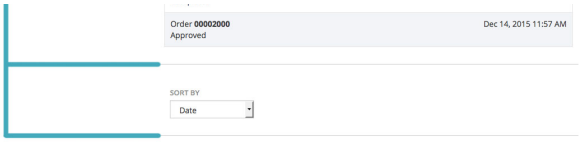
@border-color-3







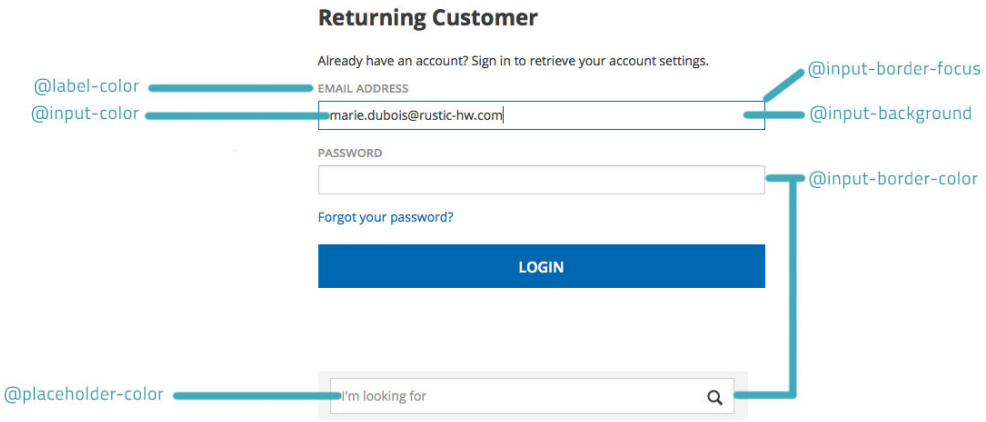


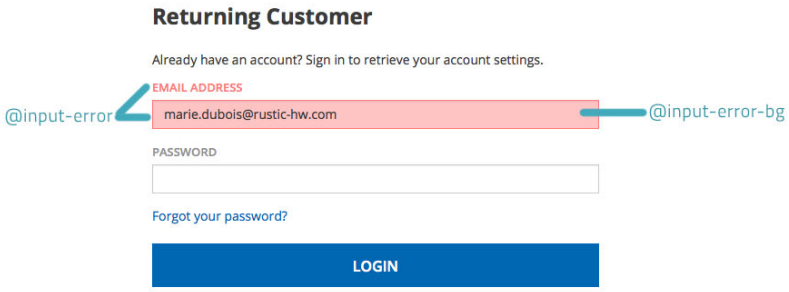


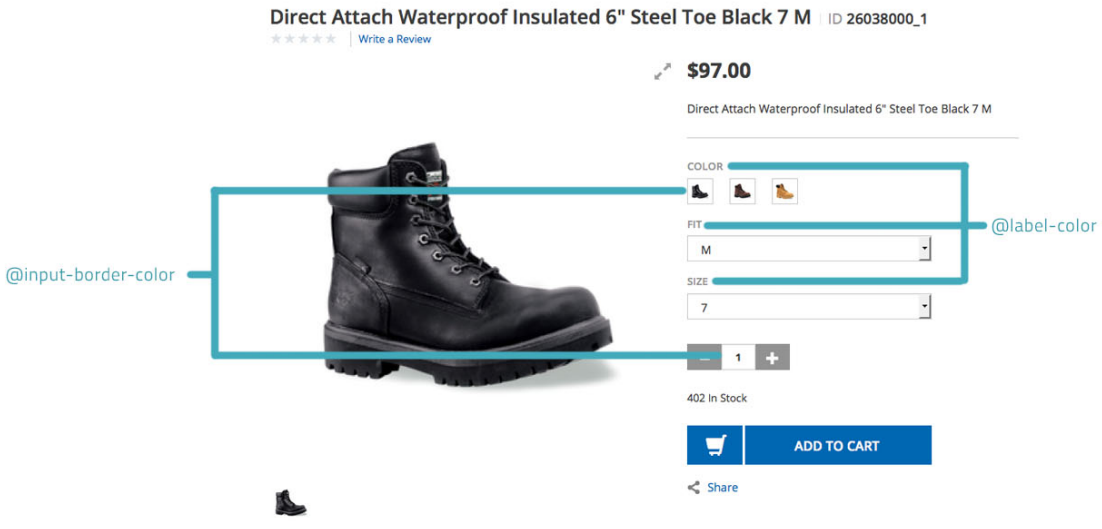
Theme Variable	Lambda Theme Value	Alpha Theme Value
@border-color	#d9d9d9	#f2f2f2
@border-color-2	#e5e5e5	#e5e5e5
@border-color-3	#cccccc	#cccccc

Form Colors

The following image identifies UI elements used in forms and their corresponding variable name. The subsequent table provides the default color values for each of these variables for both themes.



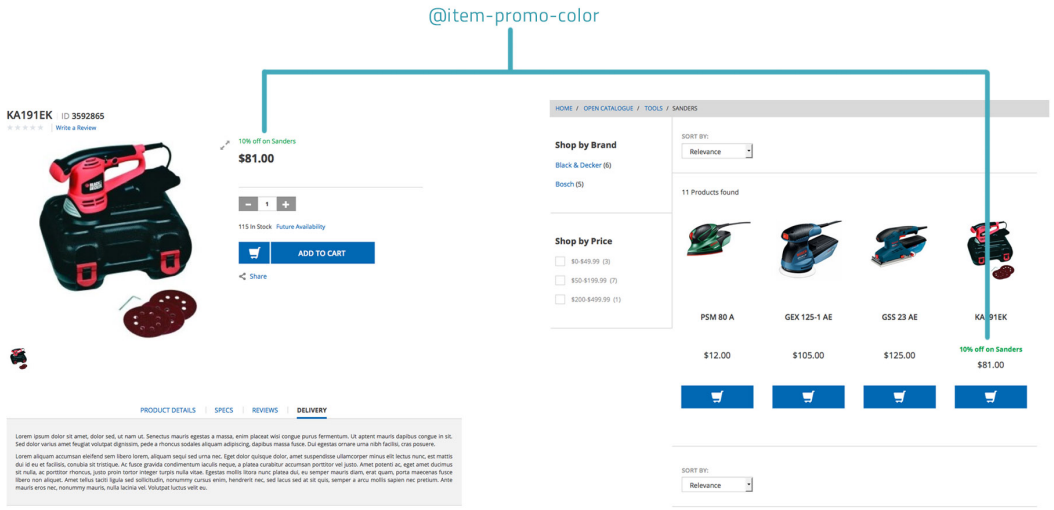




Theme Variable	Lambda Theme Value	Alpha Theme Value
@label-color	@brand-secondary: #929292	#153a59
@input-color	@text-color: #3c3c3c	@text-color: #19212b
@input-background	#ffffff	#ffffff
@input-border-color	@border-color-3: #cccccc	@border-color-3: #cccccc
@placeholder-color	@brand-secondary: #929292	#b9bdc2
@input-border-focus	@brand-primary: #0068b3	@brand-tertiary: #5adfd9
@input-error	#fd7b7b	#c53131
@input-error-bg	#fec3c3	#f6e0e0

Promotion Item Colors

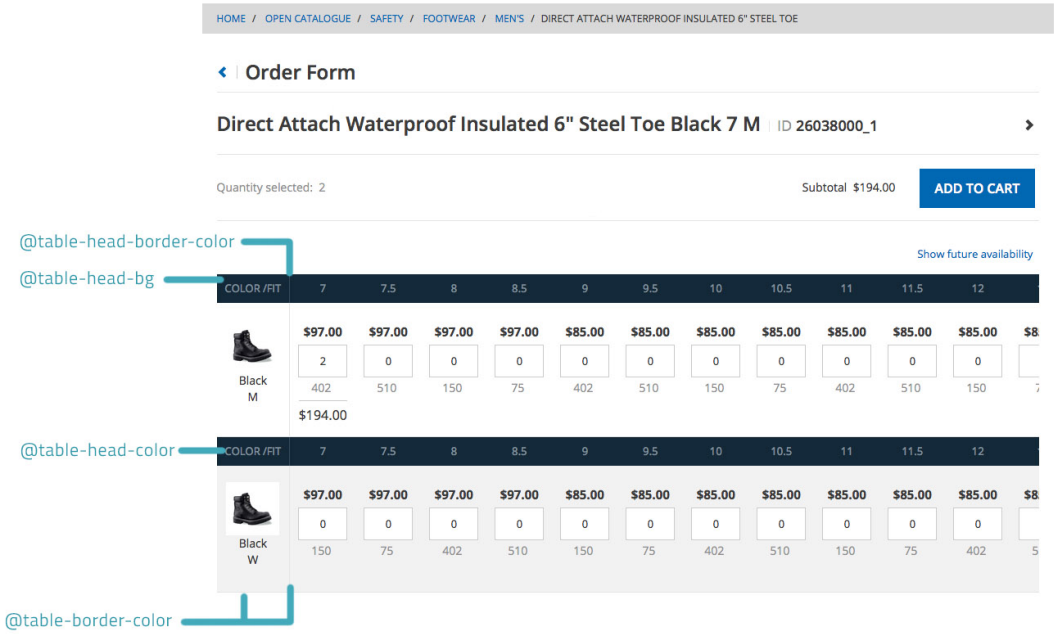
The following image identifies the UI elements used for promotional items. The subsequent table provides the default color values for each of these variables for both themes.



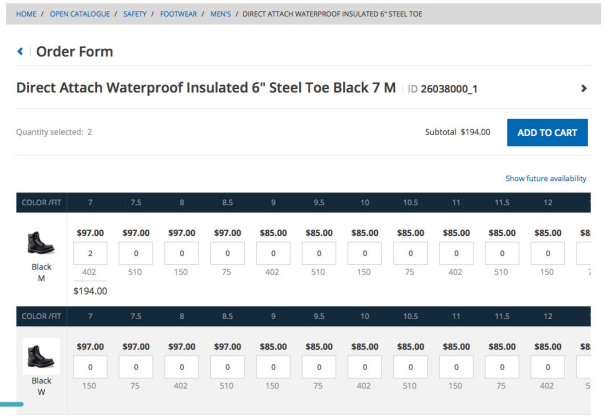
Theme Variable	Lambda Theme Value	Alpha Theme Value
@item-promo-color	#00a651	#00a651
@item-sale-color	#ed1c24	#ed1c24
@item-sale-price-color	@brand-secondary: #929292	@text-color-secondary: #929292

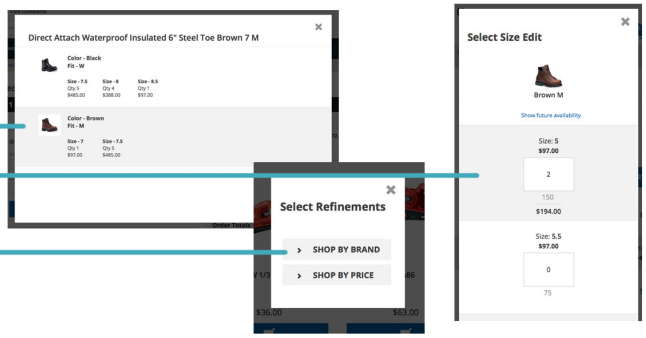
Table Colors

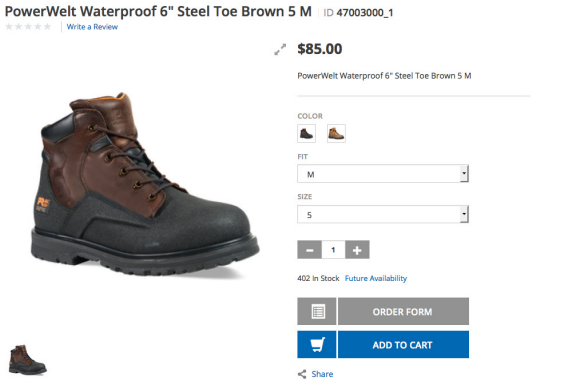
The following images identify the UI elements used for tables and their corresponding variable name. The subsequent table provides the default color values for each of these variables for both themes.

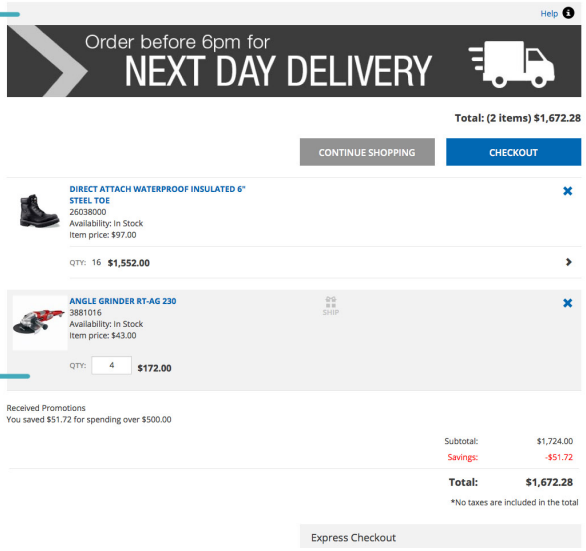

HOME / OPEN CATALOGUE / SAFETY / FOOTWEAR / MEN'S / DIRECT ATTACH WATERPROOF INSULATED 6" STEEL TOE
Order Form
Direct Attach Waterproof Insulated 6" Steel Toe Black 7 M
ID 26038000_1
Quantity selected: 2
Subtotal $194.00
ADD TO CART
Show future availability
@table-head-border-color
@table-head-bg
COLOR /FIT
7
7.5
8
8.5
9
9.5
10
10.5
11
11.5
12
$97.00
$85.00
Black M
$194.00
@table-head-color
Black W
@table-border-color

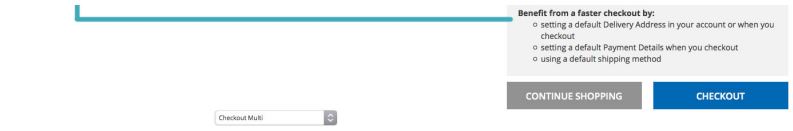
@table-2n-line-bg







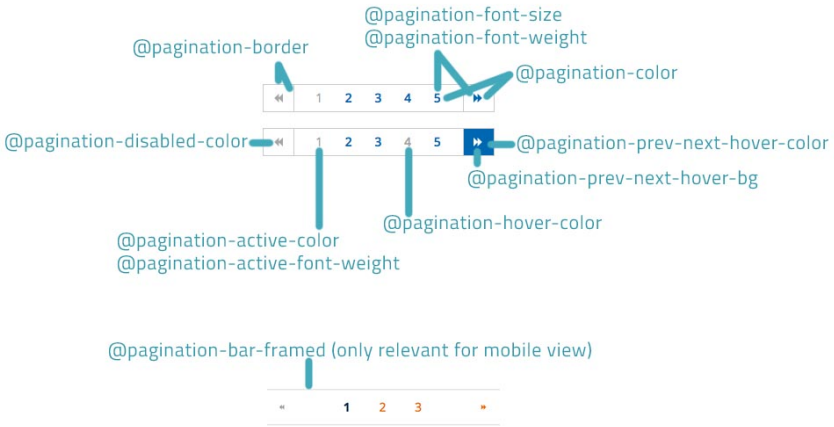




Theme Variable	Lambda Theme Value	Alpha Theme Value
@table-head-bg	@nav-cat-background: #142939	@header-background-color: #128a9e
@table-head-color	#7f919e	#ffffff
@table-head-border-color	#334b5c	@nav-border-color: #359bac
@table-border-color	@border-color-2: #e5e5e5	@border-color-2: #e5e5e5
@table-2n-line-bg	#f2f2f2	@border-color: #f2f2f2

Pagination Control Colors

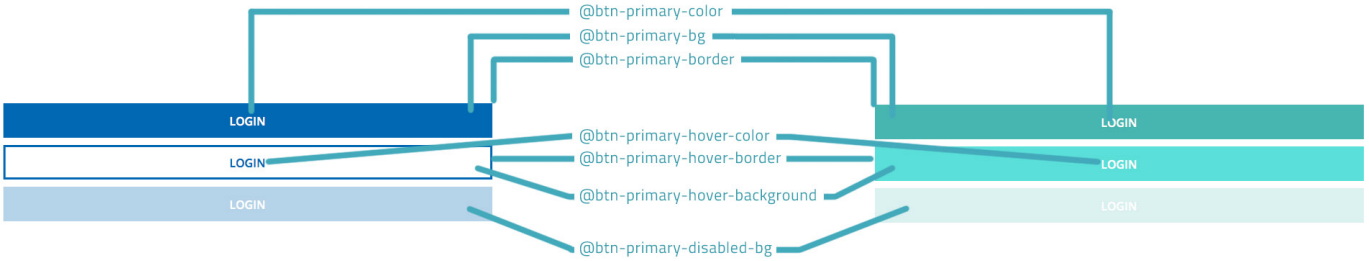
The following image identifies the UI elements used for pagination controls and their corresponding variable name. The subsequent table provides the default color values for each of these variables for both themes.

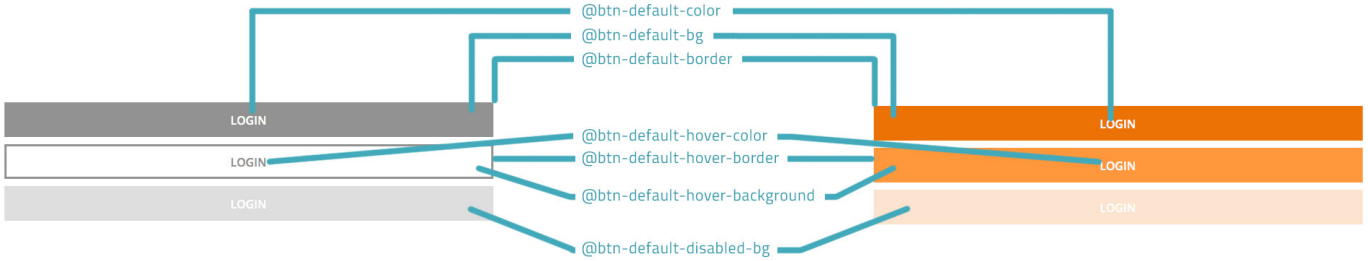


Theme Variable	Lambda Theme Value	Alpha Theme Value
@pagination-border	@border-color-3: #cccccc	transparent
@pagination-font-size	@font-size-base	24px
@pagination-font-weight	600	400
@pagination-color	@brand-secondary: #929292	@border-color-3: #cccccc
@pagination-active-color	#a6a6a6	@label-color: #153a59
@pagination-active-font-weight	400	700
@pagination-disabled-color	@pagination-active-color: #a6a6a6	#a6a6a6
@pagination-hover-color	@pagination-active-color: #a6a6a6	@pagination-active-color: #153a59
@pagination-prev-next-hover-bg	@btn-primary-bg: #0068b3	@pagination-hover-bg: transparent
@pagination-prev-next-hover-color	@btn-primary-color: #ffffff	@pagination-hover-color: #153a59
@pagination-bar-framed	false	true

Button Colors

The following images identify the UI elements used for buttons and their corresponding variable name. The subsequent table provides the default color values for each of these variables for both themes.

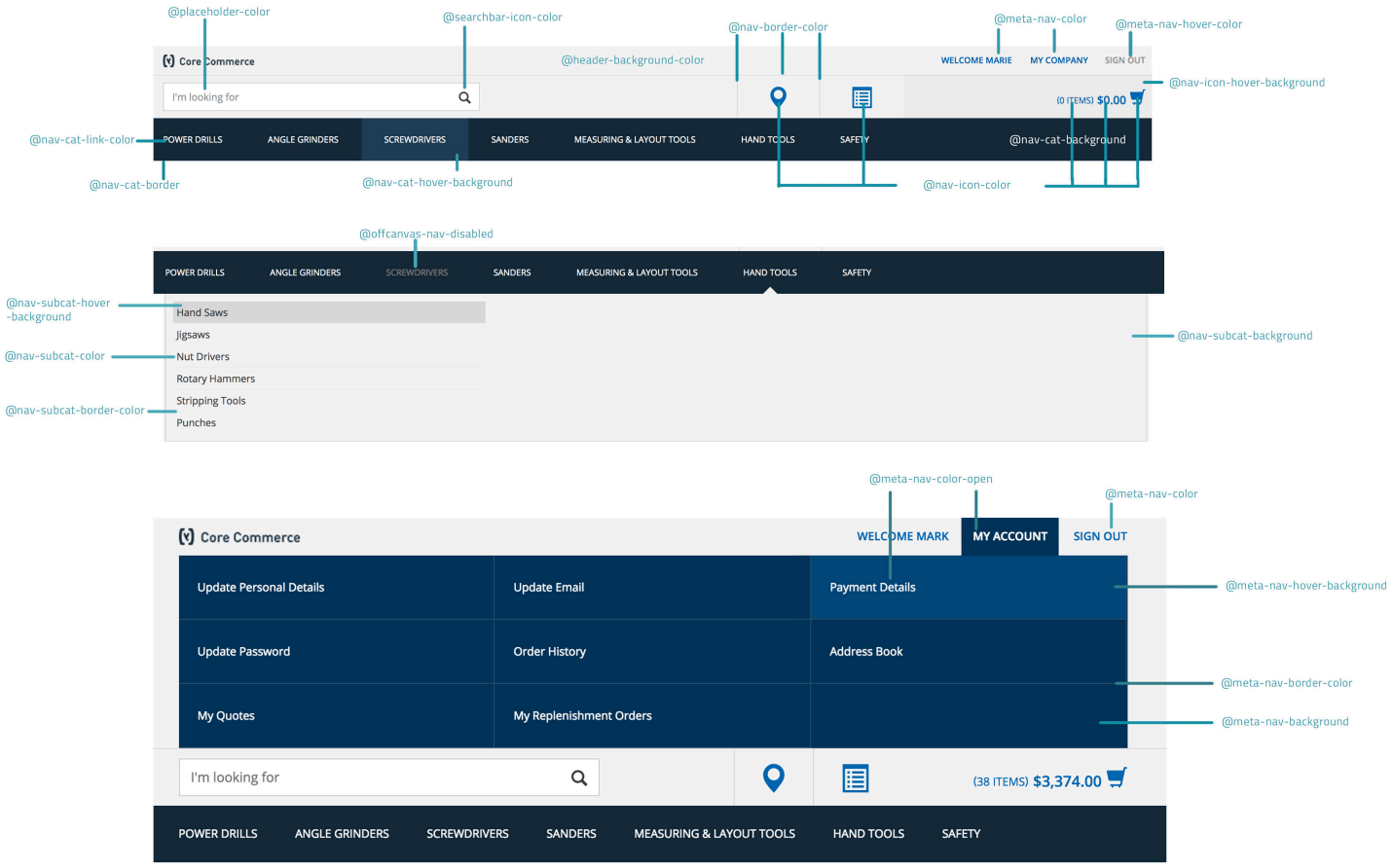




Theme Variable	Lambda Theme Value	Alpha Theme Value
@btn-primary-color	#ffffff	#ffffff
@btn-primary-bg	@brand-primary: #0068b3	@brand-primary: #47b6b1
@btn-primary-border	@brand-primary: #0068b3	@brand-primary: #47b6b1
@btn-primary-hover-color	@brand-primary: #0068b3	#ffffff
@btn-primary-hover-border	@brand-primary: #0068b3	@brand-tertiary: #5adfd9
@btn-primary-hover-background	#ffffff	@brand-tertiary: #5adfd9
@btn-primary-disabled-bg	#8cbbdd	#c7e9e7
@btn-default-color	#ffffff	#ffffff
@btn-default-bg	@brand-secondary: #929292	@brand-secondary: #ec7205
@btn-default-border	@brand-secondary: #929292	@brand-secondary: #ec7205
@btn-default-hover-color	@brand-secondary: #929292	#ffffff
@btn-default-hover-border	@brand-secondary: #929292	#ff973b
@btn-default-hover-background	#ffffff	@btn-default-hover-border: #ff973b
@btn-default-disabled-bg	@border-color-3: #cccccc	#f9d4b4
@modal-close-btn-color	@brand-secondary: #929292	#797979
@gray	#f2f4f7	#f2f4f7

Header and Navigation Colors

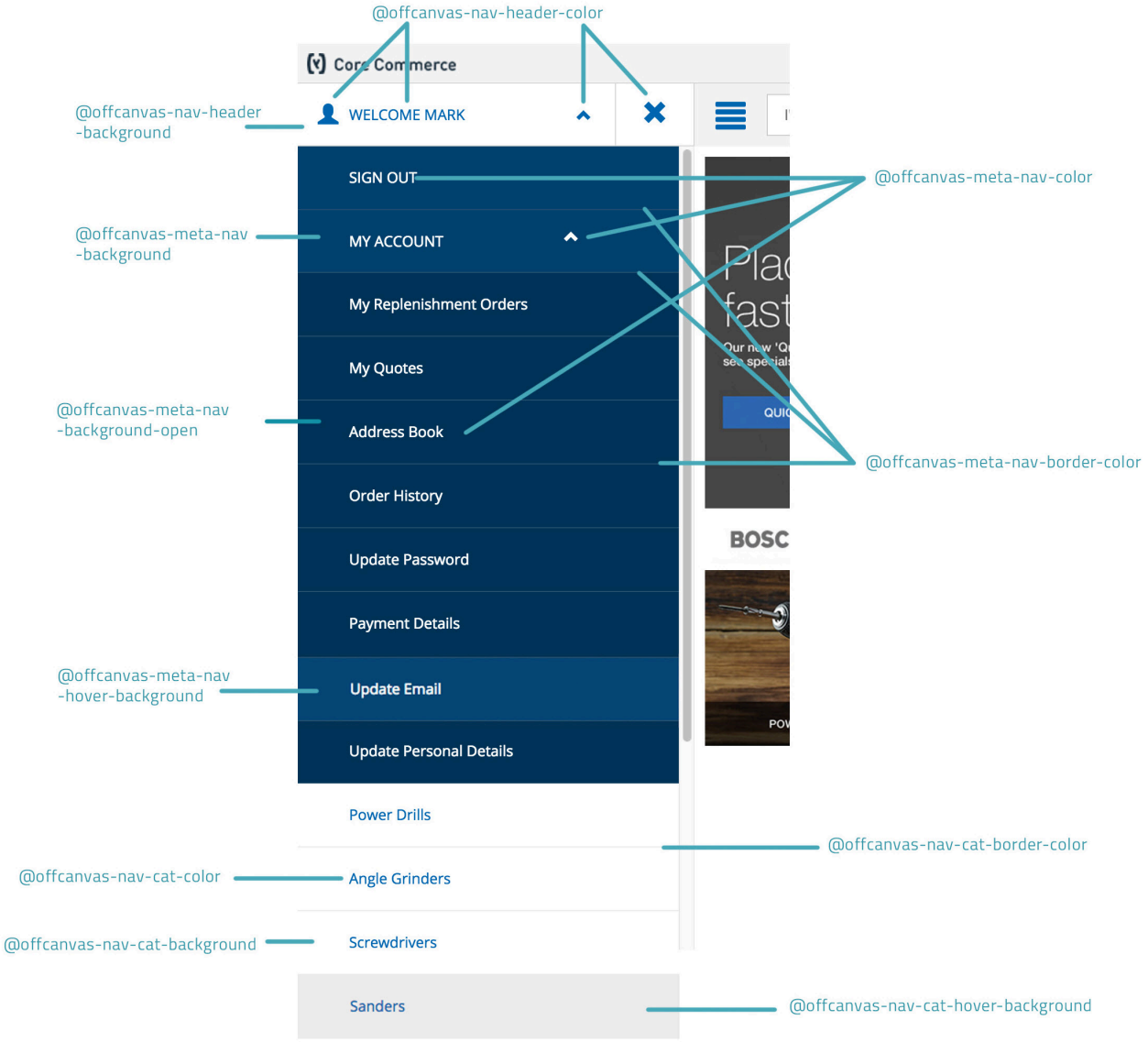
The following images identify the UI elements used for navigation and the storefront header. The subsequent table provides the default color values for each of these variables for both themes.



Theme Variable	Lambda Theme Value	Alpha Theme Value
@header-background-color	#f4f4f4	#128a9e
@nav-border-color	@border-color: #d9d9d9	#359bac
@meta-nav-color	@brand-primary: #0068b3	#ffffff
@meta-nav-color-open	#ffffff	#ffffff
@meta-nav-hover-color	@brand-secondary: #929292	@brand-primary: #47b6b1
@meta-nav-border-color	#265272	#317f8c
@meta-nav-background	#003459	#0d6878
@meta-nav-hover-background	#004678	#0f7384
@searchbar-icon-color	@text-color: #3c3c3c	#b9bdc2
@nav-icon-color	@brand-primary: #0068b3	#ffffff
@nav-icon-hover-background	@border-color-2: #e5e5e5	#1492a7
@nav-cat-background	#142939	#ffffff
@nav-cat-border	@nav-cat-background: #142939	@border-color: #f2f2f2
@nav-cat-hover-background	#1e3d55	@border-color: #f2f2f2
@nav-cat-link-color	#ffffff	@text-color: #19212b
@nav-subcat-color	@text-color: #3c3c3c	#ffffff
@nav-subcat-background	@header-background-color: #f4f4f4	@header-background-color: #128a9e
@nav-subcat-border-color	@border-color-2: #e5e5e5	@nav-border-color: #359bac
@nav-subcat-hover-background	@border-color: #d9d9d9	@nav-icon-hover-background: #1492a7

Off-Canvas Navigation Colors

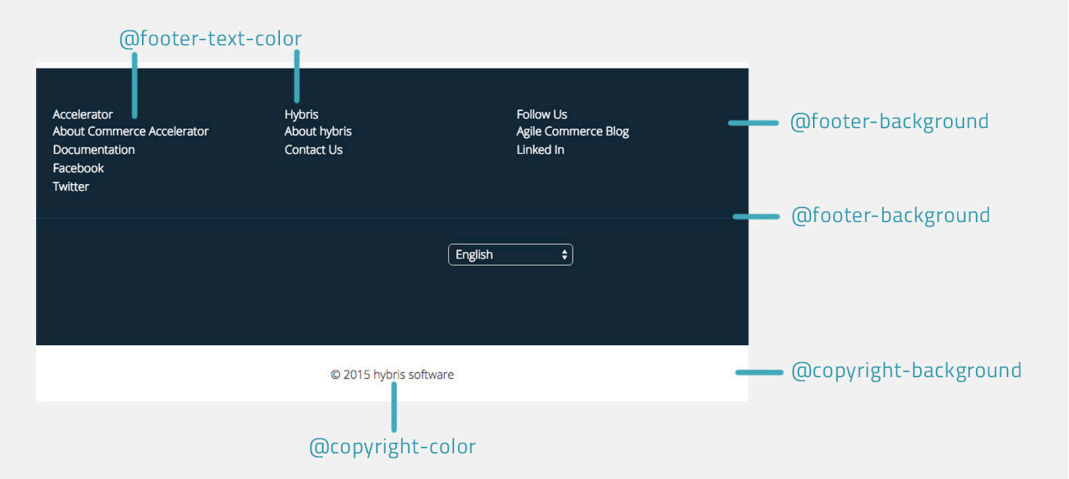
The following image identifies the UI elements used for the off-canvas menu (also known as the navigation drawer). The subsequent table provides the default color values for each of these variables for both themes.



Theme Variable	Lambda Theme Value	Alpha Theme Value
@offcanvas-nav-header-color	@brand-primary: #0068b3	#ffffff
@offcanvas-nav-header-background	#ffffff	@header-background-color: #128a9e
@offcanvas-nav-disabled	@brand-secondary: #929292	@text-color-secondary: #929292
@offcanvas-meta-nav-color	#ffffff	#ffffff
@offcanvas-meta-nav-border-color	@meta-nav-border-color: #265272	@meta-nav-border-color: #317f8c
@offcanvas-meta-nav-background	#03426f	@meta-nav-hover-background: #0f7384
@offcanvas-meta-nav-background-open	@meta-nav-background: #003459	@meta-nav-background: #0d6878
@offcanvas-meta-nav-hover-background	@meta-nav-hover-background: #004678	#107d8f
@offcanvas-nav-cat-color	@brand-primary: #0068b3	@text-color: #19212b
@offcanvas-nav-cat-background	#ffffff	#ffffff
@offcanvas-nav-cat-border-color	@border-color-2: #e5e5e5	@border-color: #f2f2f2
@offcanvas-nav-cat-hover-background	@border-color-2: #e5e5e5	@border-color: #f2f2f2
@breadcrumb-bg	@border-color: #d9d9d9	#d9d9d9

Footer Colors

The following image identifies the UI elements used for the storefront footer. The subsequent table provides the default color values for each of these variables for both themes.



Theme Variable	Lambda Theme Value	Alpha Theme Value
@footer-background	@nav-cat-background: #142939	@meta-nav-hover-background: #0f7384
@footer-text-color	#ffffff	#ffffff
@footer-border-color	@nav-cat-hover-background: #1e3d55	@nav-border-color: #359bac
@copyright-background	#ffffff	#005361
@copyright-color	@text-color: #3c3c3c	#ffffff

Creating New Themes

Use the alpha and lambda themes as a source to create your own custom themes.

Context

The alpha (blue) and lambda (black) themes can be used to create new themes.

Procedure

1. Execute the ant target `yacceleratorstorefront_responsive_theme_setup` with the following parameters:
    a. `sourceThemeCode`: string representing the code of one of the existing responsive themes to be used as the source. We recommend using the **alpha** or **lambda** themes as the source.
    b. `targetThemeCode`: string representing the code of the newly added theme (single word, all lowercase characters, or numbers [a-z;0-9]).

    ```
    ant yacceleratorstorefront_responsive_theme_setup -DsourceThemeCode=alpha -DtargetThemeCode=code1
    ```

2. Add the new theme code (set in the `targetThemeCode` parameter of the `yacceleratorstorefront_responsive_theme_setup` ant command) to the `yacceleratorcore/resources/yacceleratorcore/import/common/themes.impex` file.
3. Add the new theme name in all localization files (`themes_<language>.properties`) found in `yacceleratorcore/resources-lang/yacceleratorcore/import/common` .

    This code snippet shows the default themes listed in the `themes_en.properties` file:

    ```
    SiteTheme.black.name=Black
    SiteTheme.blue.name=Blue
    SiteTheme.alpha.name=Alpha
    SiteTheme.lambda.name=Lambda
    ```

4. Build the system using the command: `ant clean all`. The build process will generate the themes_<language>.impex file.
5. Initialize or update the system to add the new theme.
6. Switch to the newly added theme through the Backoffice:
    a. Log into the Backoffice and navigate to **WCMS** > **Website** .
    b. Double-click the website you want to change the theme for.
    c. In the **Properties** tab, change the **Theme** to the newly created one.
    d. Click **Save** to apply the theme change.

Ant Target Process

Context

The `yacceleratorstorefront_responsive_theme_setup` ant target performs the following actions when executed:

Procedure

1. Adds a group element with name attribute set to *<theme_code>*`_responsive` to `yacceleratorstorefront/web/webroot/WEB-INF/wro.xml` with a child set to `<css>/_ui/responsive/theme-<theme_code>/css/style.css</css>`.

```
<group name="<theme_code>_responsive">
   <css>/_ui/responsive/theme-<theme_code>/css/style.css</css>
</group>
```

2. Creates a new folder using the theme code in the `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/themes` folder.
   a. Adds the `fonts` subfolder under the newly created folder and copies any font-related files required for the theme.
   b. Adds the `less` subfolder under the newly created folder and copies any Less-related files required for the theme.
3. Adds property files to the `yacceleratorstorefront/web/webroot/WEB-INF/messages`. The property files use the following naming convention: `theme-<theme_code>-desktop_<locale_code>.properties`. Adds properties to the file, pointing to images that are theme-specific and that may require localization.

   Note that the scope of the script is limited to the following properties:

   - `img.favIcon`
   - `img.iconArrowCategoryTile`
   - `img.missingProductImage.responsive.product`
   - `img.spinner`
4. Creates a new folder named `theme-<theme_code>` in the `yacceleratorstorefront/web/webroot/_ui/responsive` folder.
   a. Adds the `images` subfolder under the newly created folder. Adds the images pointed to by `theme-<theme_code>-desktop_*.properties` files to this folder or to the `yacceleratorstorefront/web/webroot/_ui/responsive/common/images` folder (if not already there).

   Note that the scope of the script is limited to the images associated with the following properties:

   - `img.favIcon`
   - `img.iconArrowCategoryTile`
   - `img.missingProductImage.responsive.product`
   - `img.spinner`

Site Channel Validation

SAP Commerce Accelerator allows you to define site channels, which can be used to indicate different commerce channels, such as B2C or B2B.

Overview

Site channels are defined in `commerceservices-items.xml`. By using site channel definitions, you can ensure that different sites are handled differently based on their corresponding channels.

SiteChannel Validation Strategy

One integral use of `SiteChanel` is provided by the `acceleratorservices` extension with the `AbstractAcceleratorSiteEventListener` class. This class is wired up to an appropriate implementation of `SiteChannelValidationStrategy`. The following is an example:

```
public interface SiteChannelValidationStrategy
                                {
                                /**
                                * Validates the {@link SiteChannel}.
                                *
                                * @param siteChannel
                                *           the {@link SiteChannel} to validate.
                                * @return <code>true</code> if the {@link SiteChannel} is supported, <code>false</code> otherwise
                                */
                                boolean validateSiteChannel(SiteChannel siteChannel);
                                }
```

This is a very simple strategy that only has the purpose of determining whether or not a `siteChannel` is valid when called. By default, it is wired up in `acceleratorservices-spring.xml` to have the default valid channels of B2C and B2B, as shown in the following example:

```
<alias name="defaultSiteChannelValidationStrategy" alias="siteChannelValidationStrategy"/>
                                <bean id="defaultSiteChannelValidationStrategy" class="de.hybris.platform.acceleratorservices.site.strategies.impl.D
                                <property name="supportedSiteChannels">
                                <util:set value-type="de.hybris.platform.commerceservices.enums.SiteChannel">
                                <ref bean="SiteChannel.B2C"/>
                                <ref bean="SiteChannel.B2B"/>
                                </util:set>
                                </property>
                                </bean>
                                <bean id="SiteChannel.B2C" class="de.hybris.platform.commerceservices.enums.SiteChannel" scope="prototype" factory-m
                                <constructor-arg value="B2C"/>
                                </bean>

                                <bean id="SiteChannel.B2B" class="de.hybris.platform.commerceservices.enums.SiteChannel" scope="prototype" factory-m
                                <constructor-arg value="B2B"/>
                                </bean>
```

Using this configurable strategy, we allow for the `AbstractAcceleratorSiteEventListener` class, and any class that extends it, to decide whether or not to handle events based on the supported `siteChannels`. There are many event listeners in the `yacceleratorcore` extension that extend `AbstractAcceleratorSiteEventListener`, and this allows you to indicate if an event is supported without having to hard-code the `siteChannel`, especially in the case of a single server running both B2C and B2B sites. `AbstractAcceleratorSiteEventListener.getSiteChannelForEvent` is an abstract method that must be implemented for any custom event extending the class. This allows you to configure how a `siteChannel` is determined for a particular event, and makes the logic even more flexible.

B2C and B2B Configuration

As mentioned previously, this configurable strategy allows for overrides and supported channels to be defined within different Spring web contexts. For the B2C and B2B configurations, there is the case of not wanting a user to access the B2B web application while using the B2C site. To handle this, the `spring-filter-config.xml` file of `yacceleratorstorefront` defines a new

`acceleratorSiteChannels` that only support B2C. This is then injected into `CmsSiteFilter`, which throws a `500 Error` if the user attempts to access the wrong site. The `b2bacceleratoraddon` AddOn then overrides this `acceleratorSiteChannels` in its `b2bacceleratoraddon-web-spring.xml` to only support B2B. Since running the two sites on the same server requires an `ext-gen`, these are two separate web applications and these configurations will not clash with each other.

Localization of ImpEx Using Build Time Generation

SAP Commerce Accelerator offers the feature of localizing ImpEx files using build time generation. The purpose of this feature is to enable non-technical employees of the customer to handle the ImpEx files.

This feature takes on the input the velocity template and evaluates it. While in, there are special API calls which resolve the localized property data for configured localizations and languages. This localized data is used to evaluate the template and on the outcome it produces a ImpEx script into desired place.

→ Tip

Generation is to be run only for specially prepared extensions containing velocity templates and localized data with active macro call.

Abstraction

You have to prepare following items in order to activate this feature for a certain extension:

- **Template**

  A template file having the content evaluated by the velocity engine.

- **Localized Property File**

  File containing localized data to be evaluated into target ImpEx file using velocity template.

- **Target File**

  ImpEx script generated out of the template using given properties data - wired on execution time.

For more information, see http://velocity.apache.org .

Resource Structure

Determining Available Language ISO Codes

Currently there is simple `LocaleResolver` implementation which gets a language `Locale#getLanguage()` for a configured locale.

Available locales are configured by the `project.property` entry, as follows:

```
impex.generation.supported.locales=de,en,jp,zh
```

Resolving a Localized Property File

You can reach the localized property file by giving it a relative path to the velocity template path. The relative path is passed using `ant sourcePath` attribute:

```
<sourcePath propertyFilePath=".">
```

Resolving a Velocity Template

You can also embed a fileset using a `sourcePath` attribute, which filters out a velocity template files:

```
<sourcePath propertyFilePath=".">
                                    <fileset dir="${ext.@{extname}.path}/resources-lang/@{extname}">
                                    <include name="**/*.vt"/>
                                    </fileset>
                                    </sourcePath>
```

The above code snippet definition means that every file with the `.vt` extension in the `${ext.@{extname}.path}/resources-lang/@{extname}` folder is used as a template to render an ImpEx script. For the `apparelstore` extension, the syntax presented above would evaluate into *<${HYBRIS_BIN_DIR}>*`/ext-data/apparelastore/resources/apparelstore/import/sampledata/productCatalogs/apparelProductCatalog` folder.

For more information, see http://ant.apache.org/...tasks-filesets-properties.html .

Generating Target ImpEx Scripts

There are two possibilitiees to configure target script effective path:

- **Relative path generation**

  Similar for localized property you can pass a relative path like: `.\sampledirectory`, `..\sampledirectory`, `\genDir` where the ImpEx scripts are generated.

  This is achieved by configuring an ant target in the `buildcallbacks.xml` file, as follows:

  ```
  <targetPath targetFilePath="../maingenDir" relative="true" />
  ```

- **Side-by-side structure generation**

  Other approach is to have a side-by-side structure of the `template\properties` scripts for generating the final ImpEx files and structure for files which you created manually and are not localized. Side-by-side structure is also called mirror structure. This the default approach used in SAP Commerce Accelerator.

The input data for the ImpEx generation task is placed in the `resources-lang` folder, while the target is to be placed into the `resources` folder among rest of the ImpEx scripts.

To configure this, the Ant needs the following in the `buildcallbacks.xml` file:

```
<targetPath targetFilePath="${ext.@{extname}.path}/resources" relative="false" />
```

→ Tip

Currently the SAP Commerce Accelerator stack uses the `impexGenerate` Ant target, which expects the mirrored structure of the resources.

Build Integration

The `impexGenenerate` macro is defined in the `acceleratorservices` extension. If this macro is called in any child extension of the `acceleratorservices` extension, and there are available input data fro the ImpEx generator, then this feature is activated while in build process of a particular extension.

For more information, see acceleratorservices Extension.

The conditions for activation are as follows:

- There must be a macro call in the `buildcallback.xml` file:

```
<macrodef name="acceleratorservices_after_build">
                                                <sequential>
                                                <impexGenenerate extname="acceleratorservices" />
                                                </sequential>
                                                </macrodef>
```

  It is hooked after the build since it uses internally the Ant target class from sources so it needs it to be compiled before.

- There must be available `resources-lang` directory side-by-side to the `resources` folder. This folder should contain an appropriate velocity template files and corresponding localized properties files.

  A sample folder looks as follows:

```
resources
                                                resources-lang
                                                └──acceleratorsampledata
                                                └──import
                                                └──productCatalogs
                                                └──apparelProductCatalog
                                                categories.vt
                                                categories_de.properties
                                                categories_en.properties
                                                products-stocklevels.vt
                                                products-stocklevels_de.properties
                                                products-stocklevels_en.properties
                                                products.vt
                                                products_de.properties
                                                products_en.properties
                                                promotions.vt
                                                promotions_de.properties
                                                promotions_en.properties
```

- You have to create a task and reference binaries it needs. It is as a create Ant macro which uses a `GenerateLocalizedImpexesTask` binary. The following is an example from the `accantmacros.xml` file:

```
<macrodef name="impexGenerate">
    <attribute name="extname"/>
    <sequential>
        <typedef name="impexGen" classname="de.hybris.platform.acceleratorservices.velocity.ant.GenerateLocalizedImpexesTask" onerror="report">
            <classpath>
                <pathelement path="${ext.acceleratorservices.path}/classes" />
                <pathelement path="${ext.acceleratorservices.path}/resources" />

                <fileset dir="${ext.core.path}/lib">
                    <include name="*.jar"/>
                </fileset>
                <fileset dir="${platformhome}/bootstrap/bin/">
                    <include name="*.jar"/>
                </fileset>

                <fileset dir="${platformhome}/lib">
                    <include name="*.jar"/>
                </fileset>

                <fileset dir="${ant.home}/lib">
                    <include name="ant*.jar"/>
                </fileset>
            </classpath>
        </typedef>
        <if>
            <available file="${ext.@{extname}.path}/resources-lang/@{extname}"/>
            <then>
                <outofdate>
                    <sourcefiles>
                        <fileset dir="${ext.@{extname}.path}/resources-lang/">
                            <include name="**/*.vt"/>
                        </fileset>
                        <fileset dir="${ext.@{extname}.path}/resources-lang/">
                            <include name="**/*.properties"/>
                        </fileset>
                    </sourcefiles>
                    <targetfiles path="${ext.@{extname}.path}/resources-lang/touch_impexgen"/>
                    <sequential>
                        <echo message="Generating localized impex scripts for @{extname}"/>
                        <impexGen>
                            <sourcePath propertyFilePath=".">
                                <fileset dir="${ext.@{extname}.path}/resources-lang/@{extname}">
                                    <include name="**/*.vt"/>
```

```
                                </fileset>
                            </sourcePath>
                            <targetPath targetFilePath="${ext.@{extname}.path}/resources" relative="false"/>
                        </impexGen>
                        <touch file="${ext.@{extname}.path}/resources-lang/touch_impexgen"/>
                    </sequential>
                </outofdate>
            </then>
            <else>
                <echo message="No impex templates found for @{extname}"/>
            </else>
        </if>
    </sequential>
</macrodef>
```

- You have to reference it from somewhere else, for example in the `acceleratorservices` extension. Yo do so, you have to adjust your `buildcallbacks.xml`, for example:

```
<macrodef name="acceleratorservices_after_build">
                                                <sequential>
                                                <impexGenerate extname="acceleratorservices" />
                                                </sequential>
                                                </macrodef>
```

  To use just refer a `impexGenenerate`, for details see the Build Integration section above.

→ Tip

The task is being performed after build but only if there was any change to the `*.properties` localized property file or to the `*.vt` template file since last build.

The mechanism of differential calls has granularity of the extension.

Impex Generation Velocity API

The table below describes objects available in the velocity context:

Key	Type	Remarks
header	String	Textual representation of the header to be placed into generated ImpEx files. It is configured interna
esc	`org.apache.velocity.tools.generic.EscapeTool`	Util object used for dealing with unparsable phrases. Find more information on unparsable phrases here: http://velocity.apache.org/.../EscapeTool.html
lang	String	Textual representation of the language ISO code. **ℹ Note** Currently it is assumed that a locale language matches a corresponding `LanguageModel` ISO co
query	`de.hybris.platform.acceleratorservices.velocity.eval.PropertyEvaluator`	Sample usage: `#set ( $currencies = $query.load('products-data', 'currency') )` `UPDATE Currency;is` `#foreach( $row in` `;$row.key;$row.val` `#end`

Migration

Migrating Existing Localized ImpEx Scripts to the New Structure

You have to fulfill following steps in order to properly migrate ImpEx scripts:

1. Adjust the Ant calls in the `buildcallback.xml` file, see the Build Integration section above.
2. Prepare the input data:
   a. For every locale you have to prepare separate localized value property file. This file has to fulfill the convention. Every key has to consist of 3 main parts separated by `'.'`.

   ℹ Note
   You should change non visible characters in property key to Unicode standard, see the first bullet in the Migrating Data Problems section below.

   Example of the key property:

   ```
   Product.Shoe.name = Nike
                                                                    ...
                                                                   Product.Pants.color = Red
   ```

   These values are used by related velocity template to render the target ImpEx script.

   ℹ Note
   `Proper Property Encoding`

   To ensure all national signs are rendered properly, make sure that prepared properties are encoded in the UTF-8 standard.

   b. The ImpEx script properties should match the target property file. For example, there is a localized ImpEx script as follows:

```
UPDATE Category;$catalogVersion;code[unique=true];name[lang=$lang];description[lang=$lang]
                                                  ;;1;Open Catalogue
                                                  ;;106;Komponenten;"Eine Hardware-Komponente eines Computers."
                                                  ;;206;Datenspeicher;
```

For such a script, the result in the example `category_de.properties` should be as follows:

```
Category.1.code=Open Catalogue
                                                                  Category.106.code=Komponenten
                                                                  Category.106.description="Eine Hardware-Komponente eines
                                                                  Category.206.code=Datenspeicher
```

c. Velocity template is processed by the velocity engine. It provides a custom context to give possibility to use hybris API. For more details, refer to the Impex Generation Velocity API section above.

d. Load a property. You have to use a `load` method of the context `query` object as follows:

```
#set ( $categories= $query.load('category', 'category') )
```

This results in loading a `products-data` localized property file for every available and configured locale. Additionally this call also uses only key with type identifier equal to `currency`. To load all properties from this file you can use following call:

```
#set ( $categories = $query.load('category') )
```

From now on the currencies variable has available following attributes:

- `key`: in this case values used as `key`are as follows:

```
1
                                                                                  106
                                                                                  206
```

- `value map`: loaded with appropriate localized properties. In this case values used as `values.code` are as follows:

```
Open Catalogue
                                                                                  Komponenten
                                                                                  Datenspeicher
```

Values used as `values.description` are as follows:

```
"Eine Hardware-Komponente eines Computers."
```

- `render` property. Use it as in the following example:

```
;$row.key;$row.values.code;$row.values.description;
```

Migrating Data Problems

There are some known issues connected to the migration of scripts. Find description of the most common ones below:

- `Special characters` in property keys:

  To solve the problem and make the data consistent, use Unicode key according to the information provided in the document linked in the `See Also`panel on the right hand side. This applies only for the key values, values do not need this.

```
Category.adidas\u0020eyewear.name=adidas eyewear
                                                        Category.Adio.name=Adio
                                                        Category.ainwear.name=ainwear
                                                        Category.Airhole.name=Airhole
                                                        Category.Al\u0020Merrick.name=Al Merrick
                                                        Category.Alien\u0020Workshop.name=Alien Workshop
```

- `Multicolumn` column `uniqueness` issue:

  Normalize ImpEx data so there is only one unique column for a sequence of ImpEx rows. Note that scripts similar to the one below, have basically only one unique column, the second one is evaluated and is the same for all rows.

```
UPDATE Category;$catalogVersion;code[unique=true];name[lang=$lang];description[lang=$lang]
                                                  ;;1;Open Catalogue;
                                                  ....
                                                  ;;1421;Binoculars;
```

Case
`org.apache.velocity.exception.ParseErrorException: Lexical error,  Encountered: "u" (117), after : "" at *unset*`

Old Code
`$contentCV=catalogVersion(CatalogVersion) ...` `INSERT_UPDATE Media;$contentCV[unique=true];....`

- Beware of the `multiline data` from ImpEx script; it has to be as one line in the localized property files. The code presented in the first code snippet is the multi-line, it should be transformed to the code presented in the second example:

`Multi Line Script`

```
INSERT_UPDATE Type;column[unique=true];multiColumn;lastColumn;
                                                  ;identifier;some multilined data
                                                  with  many white spaces ....;next column Value;
```

```
Properly Organized Script

 Type.identifier.multiColumn=some multilined data with  many white spaces ....
                                                                                Type.identifer.lastColumn=next column Value
```

- Beware of the semicolons in the end of lines in localized properties files; each separates column data evaluated into final ImpEx script.
- If the original ImpEx content for column was in, for example double quote, this has to stay the same in the property file:

```
UPDATE ProductMultiBuyPromotion;code[unique=true];enabled[default=true][unique=true];messageFired[lang=$lang]
                                                                    ;MultiBuyElectronics;;"Kaufen Sie {0,number,integer} für {2} - Sie sparen {4}"
```

Localized property for German:

```
ProductMultiBuyPromotion.MultiBuyElectronics.messageCouldHaveFired="Kaufen Sie {0,number,integer} Filme für {2} - Geben Sie {3,number,integer}
                                                                    ProductMultiBuyPromotion.MultiBuyElectronics.messageFired="Kauf
```

Related Information

Using ImpEx with Backoffice or SAP Commerce Administration Console
ImpEx Syntax
Data Importing

Flashbuy Promotion

Learn how to install the Flashbuy Promotion AddOn (`timedaccesspromotionsstorefront`) on SAP Commerce B2C Accelerator.

A Flashbuy promotion is an extension of the `ProductFixedPricePromotion` class, which means a Flashbuy product has a fixed price, which compared with its original price, is very attractive, for example: original price of $200 and a Flashbuy price of $10.

Since a Flashbuy product is sold at such low price, its quantity is limited. Only a limited number of customers are eligible for a Flashbuy promotion and these are the ones to first to click the **Flashbuy** button on the storefront. Because of the low price of a Flashbuy product, thousands of requests may possibly be sent within several seconds.

To integrate the Flashbuy function into your B2C Accelerator, you need to install the `timedaccesspromotionsstorefront` AddOn. Then the Flashbuy function can work out of the box on B2C Accelerator.

i Note

This document uses the term *<HYBRIS_HOME>* to refer to the directory where you unzipped the SAP Commerce package.

Integrating Flashbuy into B2C Accelerator

To integrate the Flashbuy function into your B2C Accelerator:

1. Open the `localextensions.xml` file located under *<HYBRIS_HOME>*`/hybris/config`. Add the `timedaccesspromotionsstorefront` AddOn into the `localextensions.xml` file.

```
<hybrisconfig xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xsi:noNamespaceSchemaLocation='../bin/platform/resources/schemas/extensions
 <extensions>
 <path dir='${HYBRIS_BIN_DIR}' />
 <extension name='timedaccesspromotionsstorefront' />
 <extension name='yacceleratorstorefront' />
 <extension name='ycommercewebservices' />
 <extension name='electronicsstore' />
 <extension name='acceleratorwebservicesaddon' />
 <extension name='yaddon' />
 <extension name='apparelstore' />
 <extension name='liveeditaddon' />
 <extension name='yacceleratorinitialdata' />
 <extension name='yacceleratorcockpits' />
 </extensions>
</hybrisconfig>
```

2. Open the `local.properties` file located under *<HYBRIS_HOME_DIR>*`/hybris/config`. Add the following line into the `local.properties` file:

```
addon.china.install.strategy=B2C
```

3. Open a command interpreter window and navigate to the *<HYBRIS_HOME_DIR>*`/hybris/bin/platform` directory.
4. Set up your ant environment by running the following command:

```
setantenv
```

5. Install the `timedaccesspromotionsstorefront` AddOn by running the following command:

```
ant addoninstall -Daddonnames="timedaccesspromotionsstorefront" -DaddonStorefront.yacceleratorstorefront="yacceleratorstorefront"
```

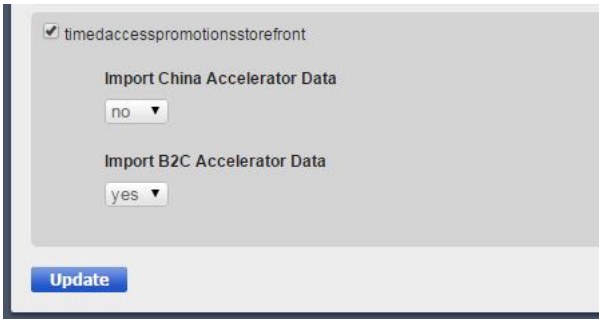
6. Run the following command to ensure that the `timedaccesspromotionsstorefront` AddOn is compiled and included into your build:

```
ant clean all
```

7. Start the SAP Commerce server by running the following command:

```
hybrisserver.bat
```

8. Open a web browser and go to `http://localhost:9001/platform/update`. At the bottom of this page, select the **timedaccesspromotionsstorefront** checkbox, and select **no** for **Import China Accelerator Data** and **yes** for **Import B2C Accelerator Data**. Click the **Update** button.



9. Go to `https://electronics.local:9002/yacceleratorstorefront/electronics/en/Open-Catalogue/Cameras/Digital-Cameras/Digital-Compacts/PowerShot-A480/ta/1934793`. This is the Flashbuy promotion page.

   To view the original product detail page, go to `https://electronics.local:9002/yacceleratorstorefront/electronics/en/Open-Catalogue/Cameras/Digital-Cameras/Digital-Compacts/PowerShot-A480/p/1934793`.

Base Accelerator Module

The Base Accelerator module provides the majority of Accelerator functionality, from configurable checkout to Spring security, as well as link and navigation management, SEO, pick up in store, customizations, search and navigation, and much more.

Features

Quick Orders
Customer Reviews
Data Importing
Essential and Project Data
Functional Storefront Interaction
Importing Product Tax Codes
Pickup In Store
Search and Navigation
SEO Directives
B2C Spring Security
Spring Usage
Stock Management
Store Locator Configuration
Storefront and Catalog Modeling
System Context
Top Navigation and Link Management
User Interface and Creatives: Responsive Experience
WCMS Integration

Architecture

yacceleratorcockpits Extension
yacceleratorcore Extension
yacceleratorfacades Extension
yacceleratorinitialdata Extension
yacceleratorfulfilmentprocess Extension
yacceleratorstorefront Extension
yacceleratortest Extension
yaddon Extension
yacceleratorcore Extension Related Data
B2C Store Extensions Related Data

Implementation

Sitemap Configuration
JS, LESS and JSPs: Build Process for Generating Variables
JSP Tag Scripts: Installing in an Accelerator Storefront
Customizing the B2C Accelerator
Adding a New CMS Page Template
Just One Storefront
Managing Multi-Step Checkout Strategies
SEO URLs

Base Accelerator Features

The Base Accelerator module provides a range of storefront features, from configurable checkout to Spring security, as well as link and navigation management, SEO, pick up in store, customizations, search and navigation, and much more.

Business Processes and Eventing

The SAP Commerce Accelerator provides integration with the SAP Commerce Process Engine out of the box. This comes implicitly with its integration with the `yacceleratorfulfilmentprocess` extension as well as a new processes to handle storefront user events and asynchronously constructing and sending emails.

Configurable Checkout

You can customize multiple checkout flows in SAP Commerce, and each checkout flow can be further customized to respond to user input.

Customer Reviews

Customer reviews are built on top of the `customerreview` extension. The features offered include customer reviews and ratings that can be moderated.

Data Importing

The SAP `acceleratorservices` extension template comes with a batch package that enables automated importing of data from hot folders.

Essential and Project Data

SAP Commerce Accelerator provides example sites that you can load as required through the Essential and Project Data load processes.

Functional Storefront Interaction

The SAP Commerce Accelerator uses Spring MVC for the storefront, and integrates the SAP Commerce WCMS Module to allow creation and editing of pages and components.

Importing Product Tax Codes

You can use an ImpEx script file to import external product tax codes into the SAP Commerce Accelerator.

Pickup In Store

The SAP Commerce Accelerator for B2C supports a complete end-to-end Buy Online Pickup in Store (BOPiS) feature. The feature is built directly into the default purchasing process and is available across areas such as Search, Stock, Cart and Checkout.

Quick Orders

The Quick Order feature provides users a speedy way to add multiple items to their cart.

Search and Navigation

The SAP Commerce Accelerator uses and extends the capabilities of the Search & Navigation module.

SEO Directives

Meta tags provide structured metadata about a storefront page. You can use meta tags to specify various attributes of a page, such as page description and keywords. These tags are placed in the `head` section of the page. This document describes the kinds of pages that are indexed and how meta tags are implemented within SAP Commerce Accelerator.

B2C Spring Security

SAP Commerce Accelerator is based on the Spring Framework. It uses the Inversion of Control (IoC) container to manage its components, as well as other frameworks that are built on top of the Spring Framework.

Spring Usage

The SAP Commerce Accelerator, like the SAP Commerce, is based on the Spring Framework. It not only uses the Inversion of Control(IoC) container to manage its components, but also other frameworks that are built on top of the Spring Framework. This document provides an overview of which frameworks the SAP Commerce Accelerator uses.

Stock Management

The handling of Product SKU Inventory, often referred to as **stock**, is an important part of the SAP Commerce Accelerator. The Accelerator maintains stock levels at both the BaseStore and PointOfService level. This makes it possible for project implementations to expose both **Delivery** and **Pickup in Store** purchasing options and differentiate stock levels accordingly.

Store Locator Configuration

The SAP Commerce Store Locator helps consumers to find stores in the proximity of a postal code, city name, or by using Global Positioning System (GPS) information. The Store Locator can also provide directions to the selected store. From a more technical point of view, the Store Locator functionality provides services that can work with standard online interfaces, as well as with mobile solutions.

Storefront and Catalog Modeling

Learn the basics of how SAP Commerce Accelerator and its sample stores are configured and modeled with respect to the storefront setup and distribution of customer data, as well as product and content catalogs between multiple storefronts. A multiple storefront setup is a single SAP Commerce deployment that serves multiple web sites, rather than separate SAP Commerce deployments for each web site.

System Context

This document discusses potential extension points for SAP Commerce Accelerator and how Accelerator fits into the wider context of an eCommerce System. This includes pointers to where code should be extended, additional documentation to review, or recommendations for implementation decisions.

Top Navigation and Link Management

The SAP Commerce Accelerator offers customizable, top navigation and link management functionality. The customization is done through the WCMS components.

User Interface and Creatives: Responsive Experience

Learn about the interface and design of the default Accelerator responsive storefronts, including how to change the design of the storefronts using themes and CSS.

WCMS Integration

This document describes the WCMS integration of the `yacceleratorstorefront` extension and how to enable the content management and preview functionality based on the `cms2` extension. This integration is implemented by reusing existing classes of the `storetemplate` extension while adding and modifying existing functionality.

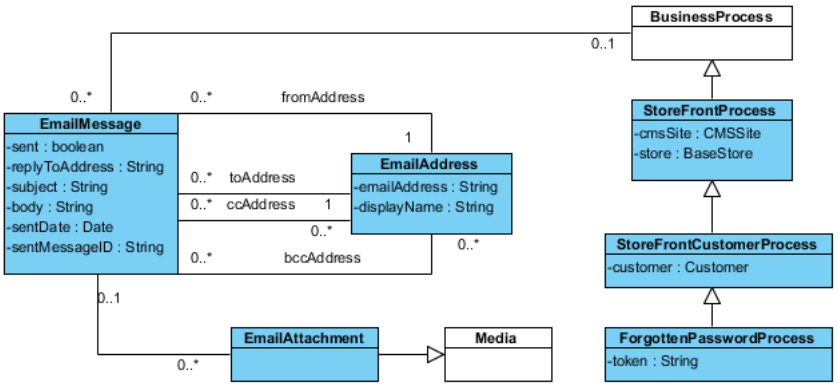
Business Processes and Eventing

The SAP Commerce Accelerator provides integration with the SAP Commerce Process Engine out of the box. This comes implicitly with its integration with the `yacceleratorfulfilmentprocess` extension as well as a new processes to handle storefront user events and asynchronously constructing and sending emails.

Business Processes and Eventing

Generating and sending emails is integrated into WCMS and the platform `core` extension. The process is triggered by sending an event which is then delivered by Spring to configured event listeners for this particular event.

The data model used for sending emails is shown in the next diagram:



The main persistent items are as follows:

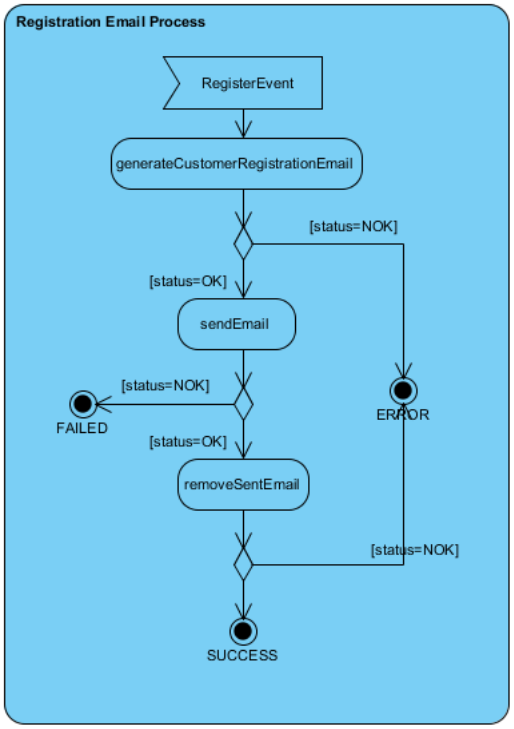
Class	Description
`EmailMessage`	Stores an email message, mainly the subject, the body and status related information.
`EmailAttachment`	Optionally, stores attachments as `Media`.
`EmailAddress`	Stores an email address together with the display name for all different address types (to, from, cc, bcc).
`StoreFrontProcess`	Extends a business process with site and store related information required while rendering the email content.

Class	Description
`StoreFrontCustomerProcess`	Extends `StoreFrontProcess` with an additional customer attribute.
`ForgottenPasswordProcess`	Extends `StoreFrontCustomerProcess` with an additional token attribute used for the forgotten password process.

The extended business process models have been added to support capturing of email content as well as to trigger storefront processes such as handling registration of forgotten password.

Customer Registration Email Process

This process is used for creating and sending an email as shown in the next diagram:



The steps triggered after receiving an `RegisterEvent` are:

Step	Description
`generateCustomerRegistrationEmail`	Creates a new `EmailMessage` and initialise it using CMS integration classes and EmailGenerationService
`sendEmail`	Sends the `EmailMessage` to the intended recipients using `EmailService`
`removeSentEmail`	Removes the `EmailMessage` from the database

If an error occurs during processing, the process is stopped with an error status.

Example

How a business process is started in response to an previously published event using the platform `EventService` is shown in this section. The approach leverages the built-in support in Spring for publishing and receiving events using the Spring application context.

This example shows how to:

- Create and start a business process in response to a received event.
- Configure an event listener in Spring

1. Provide an event listener parameterized with the event type as shown in the example for the `RegistrationEventListener`:

```
public class RegistrationEventListener extends AbstractEventListener<RegisterEvent>
            {
            private ModelService modelService;

            public BusinessProcessService getBusinessProcessService()
            {
            return (BusinessProcessService) Registry.getApplicationContext().getBean("businessProcessService");
            }

            protected ModelService getModelService()
            {
            return modelService;
            }

            public void setModelService(final ModelService modelService)
            {
            this.modelService = modelService;
            }

            @Override
            protected void onEvent(final RegisterEvent registerEvent)
```

```
{
final StoreFrontCustomerProcessModel storeFrontCustomerProcessModel = (StoreFrontCustomerProcessModel) getBusinessProcessService()
.createProcess("customerRegistrationEmailProcess" + System.currentTimeMillis(), "customerRegistrationEmailProcess");
storeFrontCustomerProcessModel.setCmsSite(registerEvent.getCmsSite());
storeFrontCustomerProcessModel.setCustomer(registerEvent.getCustomer());
getModelService().save(storeFrontCustomerProcessModel);
getBusinessProcessService().startProcess(storeFrontCustomerProcessModel);
}
}
```

This code first creates a new process using `BusinessProcessService` and initializes the newly created process instance using received event data. In a second step, the process is started.

2. Configure the event listener in Spring as shown in the example from the `yacceleratorcore-spring.xml` file:

```
<bean id="customerRegistrationEventListener" class="de.hybris.platform.yacceleratorcore.event.RegistrationEventListener">
            <property name="modelService" ref="modelService"/>
            </bean>
```

Related Information

processing Extension
yacceleratorfulfilmentprocess Extension
WCMS Email and Process Engine Integration

Configurable Checkout

You can customize multiple checkout flows in SAP Commerce, and each checkout flow can be further customized to respond to user input.

You can assign a different checkout flow to each base store. For example, the electronics store can be configured with a checkout flow that is different from the apparel stores. By using an AddOn, each checkout flow can be further modified to include additional steps.

You can add or remove as many steps in the checkout flow as required, and the sequence of steps can also be configured.

Furthermore, the checkout flow can be customized to change dynamically, depending on choices made the user, or depending on what appears in the user's shopping cart.

Use Case

If you were running a promotion that applied to only one of your base stores (such as the electronics store), you could add a step that allows customers to participate in the promotion based on the contents of their cart, and this promotion would only appear in the checkout flow of the electronics store, without affecting the checkout flows in your other base stores (such as the apparel stores).

Features

The following is an overview of the key features of Configurable Checkout.

Checkout Groups

Checkout groups are the starting point for creating a customized checkout flow. Each base store has one checkout group, which allows you to have a different, customized checkout flow for each base store.

Dynamic Checkout Flows

This feature allows you to change the checkout path, or to override the next configured checkout step, based on user action or based on the contents of the user's cart.

Dynamic Progress Bar

With the each customized checkout flow, you can configure the progress bar to reflect the checkout flow. For example, if you have added steps to your checkout flow, or the checkout flow updates dynamically based on user choice, the progress bar can be configured to accurately reflect the customer checkout journey every step of the way.

Dependencies

There are no specific dependencies for using Configurable Checkout.

Related Information

Customizing Configurable Checkout

Customer Reviews

Customer reviews are built on top of the `customerreview` extension. The features offered include customer reviews and ratings that can be moderated.

Language

If a website is available in two languages, for example English and German, and a review is posted on the German version of the website it will only be displayed on that version of the site. The language of a review can be changed in the Backoffice Administration Cockpit.

The current language for the storefront is managed by the `commercefacades` extension `StoreSessionFacade`.

Anonymous Reviews

All reviews are anonymous - the user can provide an optional alias. If the user does not do so, **by Anonymous** is shown as the author of the review.

This behavior can be altered to show the logged in user by editing the `reviewsTab.jsp` file.

For example, if the review was posted by a user who has loggerd in, and the desired behavior is to show that user's ID if he does not provide an alias the following code:

```
<c:choose>
                                <c:when test="${not empty review.alias}">
                                ${review.alias}
                                </c:when>
                                <c:otherwise>
                                <spring:theme code="review.submitted.anonymous"/>
                                </c:otherwise>
                                </c:choose>
```

This behavior can be changed to:

```
<c:choose>
                                <c:when test="${not empty review.alias}">
                                ${review.alias}
                                </c:when>
                                <c:when test="${not empty review.principal.name && empty review.alias }">
                                ${review.principal.name}
                                </c:when>
                                <c:otherwise>
                                <spring:theme code="review.submitted.anonymous"/>
                                </c:otherwise>
                                </c:choose>
```

Rating System

Accelerator supports a 5-star rating system out of the box. Each review has a rating associated with it. An average review is calculated for the product and is displayed next to it.

Accelerator uses the JQuery Star Rating Widget. Changing it to support a different rating system such as 1-4 or 1-10 stars is just a matter of changing the graphics, CSS and HTML.

For example to provide support for a 10-star rating system change the `productPageReviewsTab.tag` HTML markup from:

```
<div id="stars-wrapper">
                                <form:select path="rating" >
                                <form:option value='1'>1/5</form:option>
                                <form:option value='2'>2/5</form:option>
                                <form:option value='3'>3/5</form:option>
                                <form:option value='4'>4/5</form:option>
                                <form:option value='5'>5/5</form:option>
                                </form:select>
                                </div>
```

to the following one:

```
<div id="stars-wrapper">
                                <form:select path="rating" >
                                <form:option value='1'>1/10</form:option>
                                <form:option value='2'>2/10</form:option>
                                <form:option value='3'>3/10</form:option>
                                <form:option value='4'>4/10</form:option>
                                <form:option value='5'>5/10</form:option>
                                <form:option value='6'>6/10</form:option>
                                <form:option value='7'>7/10</form:option>
                                <form:option value='8'>8/10</form:option>
                                <form:option value='9'>9/10</form:option>
                                <form:option value='10'>10/10</form:option>
                                </form:select>
                                </div>
```

The ratings widget is not used to display the ratings next to the product or the reviews so those graphics and CSS need to be altered.

Commerce Facades

There are two ways to get the reviews for a product. The first method is more efficient if only the reviews are desired. The second method actually returns a `ProductData` object which contains the reviews for the product.

- Method 1:

  ```
  List<ReviewData> productReviews = productFacade.getReviews(productCode)
  ```

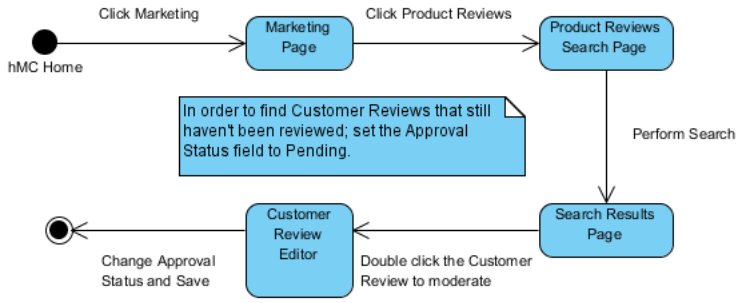
  This returns the reviews that the current user can view based on the language and search restrictions.

- Method 2:

  ```
  final ProductData productData = productFacade.getProductForCode(productCode, Arrays.asList(ProductOption.BASIC, ProductOption.PRICE, ProductOpt
                                          List<ReviewData> productReviews = productData.getReviews();
  ```

For more information on the `commercefacades` extension, refer to commercefacades Extension.

New Approval Process



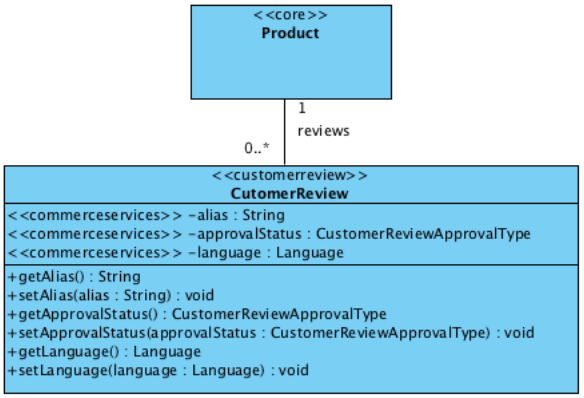
There has been a new `approvalStatus` attribute added to the `CustomerReviewModel` which is set to `Pending` by default.

A search restriction on Reviews will ensure that only users in the customergroup will only view reviews that have been approved through the Backoffice.

The search restriction is created in `de.hybris.platform.commerceservices.setup.CommerceServicesSystemSetup`.

For more information, see Moderating Reviews.

Extended Data Model



Related Information

Customer Reviews
Managing Customer Reviews per Product

Data Importing

The SAP `acceleratorservices` extension template comes with a batch package that enables automated importing of data from hot folders.

Hot Folder Data Importing

The infrastructure enables the import of simple CSV files that are internally translated into fast, multi-threaded SAP ImpEx scripts that import content directly into your product catalogs. Files can also be grouped into batches to split the data load across multiple files. This feature is intended for automated updates, such as stock status, price, and product catalog feeds that you may receive from external systems, or from other areas of your business. The infrastructure uses Spring integration to provide a pluggable, highly configurable, service-based design that enables you to extend the service to fit your specific requirements.

With hot folder data importing, CSV files are imported automatically when they are moved to a folder that is scanned periodically by the system. The system processes any files found in the folder.

ℹ Note

If you are using Accelerator in a clustered environment, using the same hot folder on multiple nodes is not supported.

For more information on clustering, see Clustered Environment.

Diagram Of Components

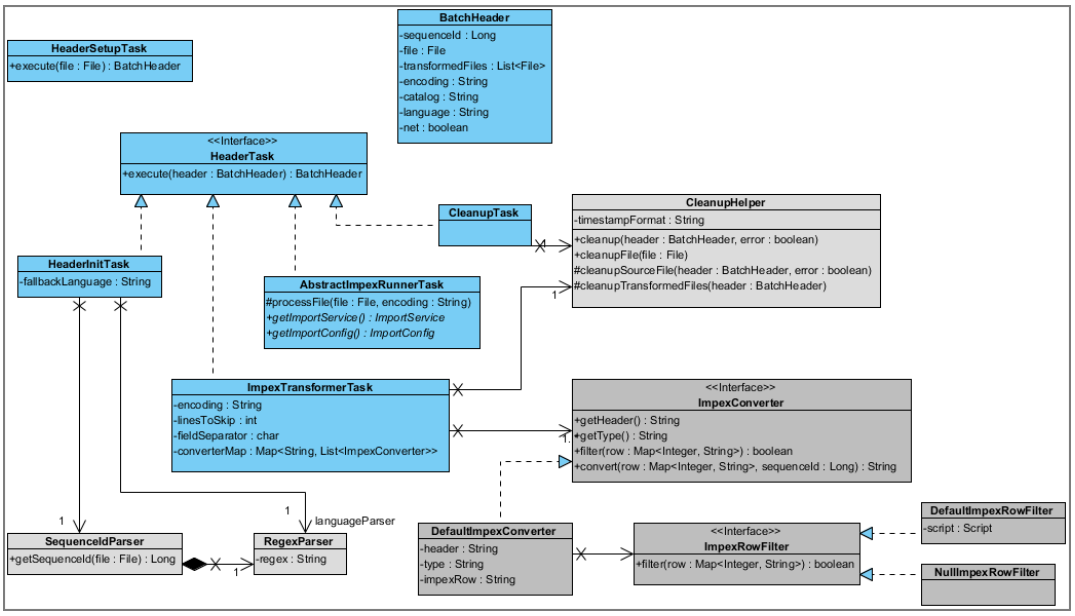


Figure: Components behind hot folder data importing.

The classes are structured into three major parts:

- Tasks executed by the Spring integration infrastructure.
- Converters providing the ImpEx header and converting CSV rows into ImpEx rows with optional filtering.
- Helper and utility classes: `SequenceIdParser`, `RegexParser`, and `CleanupHelper`

Tasks obey the general contract of a `HeaderTask`, with the exception of the initial `HeaderSetupTask` creating the `BatchHeader`. The contract allows for:

- Flexible configuration of processing pipelines
- Providing a shared process context
- Cleanup of all generated process files
- Error handling by means of an `ErrorHandler` listening on an error channel

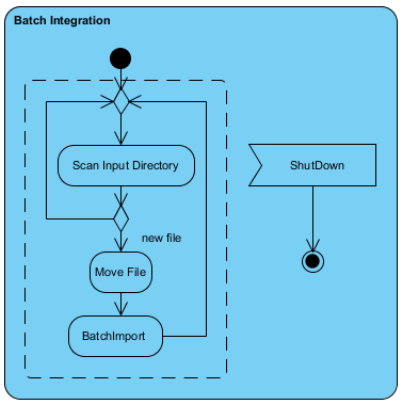
CSV files are transformed into ImpEx files by the `ImpexTransformerTask`, which performs the following tasks:

1. It retrieves all configured converters matching the file name prefix.
2. For every converter found, it converts the input file as follows:
    - It adds the ImpEx file header once with substitutions
    - It converts all rows if they are not filtered
    - If the line has missing input fields, it adds the line along with the error message to a file in the `error` subdirectory

All generated ImpEx files are then sent to the platform `ImportService` in the `AbstractImpexRunnerTask` sequentially.

Diagram of General Flow

The general flow is shown in this diagram:



Spring integration periodically scans the configured input directory for new files. If new files are found, they are moved to the processing subdirectory and then sent to the Batch Import pipeline, which consists of the following tasks:

- `HeaderSetupTask`: Creates a new `BatchHeader`.
- `HeaderInitTask`: Retrieves a sequence ID and (optionally) a language from the file name.
- `ImpexTransformerTask`: Creates one or many ImpEx files from the CSV input and writes error lines to the `error` subdirectory.
- `ImpexRunnerTask`: Processes all ImpEx files sequentially with multiple threads.

- `CleanupTask`: Deletes all transformed files and moves the imported file with an optionally appended timestamp to the `archive` subdirectory.
- `ErrorHandler`: Deletes all transformed files and moves the imported file with an optionally appended timestamp to the `error` subdirectory.

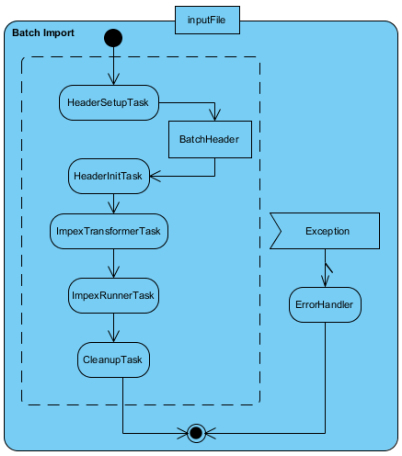


Figure: Batch import diagram.

Configuration

The hot folder import processes are configured in the following files:

File	Location	Configuration Items
`hot-folder-spring.xml`	*<HYBRIS_BIN_DIR>*`/modules/core-accelerator/acceleratorservices/resources/acceleratorservices/integration`	• Spring integration infrastructure including AOP setup. • Common ImpEx converters, filters, and converter mappings.
`hot-folder-common-spring.xml` `hot-folder-store-electronics-spring.xml` `hot-folder-store-apparel-spring.xml`	*<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorcore/resources/yacceleratorcore/integration`	• Extension-related Spring integration infrastructure including hot folder setup. • Extension-related ImpEx converters, filters, and converter mappings.
`hot-folder-store-powertools-spring.xml`	*<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yb2bacceleratorcore/resources/yb2bacceleratorcore/integration`	• Extension-related spring integration infrastructure including hot folder setup. • Extension-related ImpEx converters, filters, and converter mappings.
`project.properties`	*<HYBRIS_BIN_DIR>*`/modules/core-accelerator/acceleratorservices`	• **acceleratorservices.batch.impex.max-threads**: number of threads used for ImpEx. • **acceleratorservices.batch.impex.basefolder**: base folder of the import infrastructure containing catalog specific subfolders.

Task Configuration

The following table lists the relevant configuration options for tasks:

Configuration Option	Parameters
HeaderSetupTask	• `catalog`: The catalog to use. This setting is applied to the default header substitution: `$CATALOG$` • `net`: The net setting to apply to prices. This setting is applied to the default header substitution: `$NET$`
HeaderInitTask	• `sequenceIdParser`: The regular expression used to extract the sequence ID from the file name. In the sample configuration, the sequence ID precedes the file extension separated by a hyphen, for example: `base_product-1.csv` • `languageParser`: The regular expression used to extract the language from the file name. In the sample configuration, the language optionally precedes the sequence ID, for example: `base_product-de-1.csv` • `fallbackLanguage`: The language to use if the language is not set in the file name.
ImpexTransformerTask	• `converterMap`: Used to map file prefixes to one or multiple converters to produce ImpEx files. • `encoding`: The file encoding to use (default value is `UTF-8`). • `linesToSkip`: The lines to skip in all CSV files (default value is `0`). • `fieldSeparator`: The separator to use to read CSV files (default value is ,).
CleanupTask	• `CleanupHelper.timeStampFormat`: If set, appends a timestamp in the specified format to input files moved to the `archive` or `error` subdirectory.

Converter Configuration

Converters are the central configuration element used for importing. The following configuration options are available:

Configuration Option	Parameters
`header`: The ImpEx header to use including header substitutions listed in the next column.	• `$NET$`: the net setting • `$CATALOG$`: the catalog prefix • `$LANGUAGE$`: the language setting • `$TYPE$`: an optional type attribute that can be applied if filtering is configured
`impexRow`: The template for an ImpEx row adhering to the syntax in the next column.	`Syntax: {('+')? (<columnId> | 'S')}` The '+' character adds a mandatory check to this column. Any lines with missing attributes are written to an error file in the `error` subdirectory. For example, writing the first CSV column to the ImpEx file can be accomplished by configuring `{+0}`. The 'S' can be used for writing the current sequence ID at the template position. Optionally, columns can be quoted by enclosing the column in the template with quotation marks. **Optional Parameters** • `rowFilter`: An optional row filter. The supplied expression must be a valid Groovy expression. The current row map consisting of column ID and value is referenced by row. The Groovy semantics for converting String to Boolean apply. For example, querying whether the first column is not empty can be written as `!row1`. Configuring multiple converters with row filters gives the option to split a supplied CSV input file into different ImpEx files according to specified filter criteria. • `type`: An optional type that can be retrieved in the header using the header substitution `$TYPE$`.

Spring Integration Configuration

To allow for easier configuration, the following elements leverage the Spring integration namespace:

- `int:channel`: Sets up a channel.
- `int:service-activator`: Activates a referenced bean when receiving a message on a configured channel. The bean response is again wrapped in a message and sent to the configured output channel.
- `file:inbound-channel-adapter`: Scans a directory in a configurable interval and sends files to a configured channel under the following conditions:
  - Only files matching a specified regular expression are retrieved (filename-regex).
  - Files are processed in the order defined by the `FileOrderComparator`, using the following priority rule:
    - If a priority is configured for the file prefix, it uses the specified priority.
    - For files with equal priority, the older file is processed first.
- `file:outbound-gateway`: Moves a file to the `processing` subdirectory.

Spring Configuration of a New Converter

To plug in the new converter, you must define it in Spring and define its mapping. The following code samples provide examples of the converter definition and mapping.

Converter Definition

The following is a sample converter definition:

```
<bean id="batchTaxConverter" class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.impl.DefaultImpexConverter">
                <property name="header">
                        <value>...</value>
                </property>
                <property name="impexRow">
                        <value>...</value>
                </property>
        </bean>
```

Mapping Definition

The following is a sample converter mapping:

```
<bean id="batchTaxConverterMapping"
                  class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"
                  p:mapping="tax"
                  p:converter-ref="batchTaxConverter"/>
```

The batch Transformer Task collects every mapping (i.e. every bean that implements the `ConverterMapping` interface) and uses it for mapping the right converter. There is no need for further configuration; just define a mapping bean in the Spring context.

Configuration Guide

The next sections describe typical configuration tasks and how you can achieve them.

Adding a New Import

To add a new import:

1. Define the file name schema for the new import, for example: `customer-<sequenceId>.csv`.
2. Set up a new `ImpexConverter` configuration describing the ImpEx header and the row expression.
3. Add the file prefix and the converter reference to the `converterMap` property of the `ImpexConverter`.

Restricting to a Certain Catalog Version

Typically, the version is defined as an ImpEx variable in the ImpEx header. You can reference this variable easily in the ImpEx import statements. The sample configuration imports into the Staged catalog only (price imports are imported into both Staged and Online catalogs).

Applying Updates to Multiple Catalog Versions

In order to update multiple catalogs, an input line is written to multiple transformed lines with differing catalog versions. An example of this approach is shown in the price converter configuration `batchPriceConverter` in the `hot-folder-spring.xml` file:

```
<property name="impexRow">
    <value>;{+0}:$catalog:Staged;{+1};{+2};{3};;;;$catalog:Staged;{S}
;{+0}:$catalog:Online;{+1};{+2};{3};;;;$catalog:Online;{S}</value>
</property>
```

Sequencing Support

Sequencing, which is also known as version support, defines the ability to associate updates with a unique ID to prevent stale updates. The recommended way to provide this ID is to use UNIX timestamps while exporting data. In Accelerator, this ID is called the sequence ID. Support for this feature is available for Product and `PriceRow` updates. The data model for these two Items was extended with an additional property `sequenceId`.

The sequence ID is associated to an input file via a configurable naming convention `SequenceIdParser`. This sequence ID is then provided for ImpEx in every row by using standard converter row syntax.

The sequence ID is then imported by means of a specialized ImpEx translator, `SequenceIdTranslator`. The provided value is compared with the current value, and the row is rejected if the value is outdated. Rejected lines can be viewed as unresolved lines of the responsible cron job in the Backoffice Administration Cockpit.

By default, tax and media import is configured in the `GreaterSequenceIdTranslator` to validate the sequence ID while tolerating an equal sequence ID.

Guide to Sample Imports

Accelerator provides sample import feeds which you can easily adjust to different requirements. This section describes the available file formats.

New Product Feed

New products are imported from files named using following convention: `base_product-`*`<language>`*`-`*`<sequenceID>`*`.csv`. If the product variant type is not empty, the product is imported as `ApparelProduct`, otherwise it is imported as Product. This functionality is achieved by using row filtering and configuring multiple converters for base_product.

The optional language part of the file name is added to the ImpEx header by using a header substitution. Thereby it is possible to update localized attributes like name or description for this language only.

Find the description of file format in the table below:

Column	Attribute	Type	Optional	Localized	Default	Values	Constraints	Description
0	code	String						
1	variantType	String	yes			`ApparelSizeVariantProduct` `ApparelStyleVariantProduct`		
2	name	String	yes	yes				
3	description	String	yes	yes				
4	ean	String	yes					
5	manufacturerName	String	yes					
6	manufacturerAID	String	yes					
7	unit	String	yes		pieces			
8	approved	String	yes		check	check approved		
9	product tax group	String	yes		eu-vat-full	eu-vat-full eu-vat-half		

Stock Update Feed

Stock updates are imported from files named like `stock-`*`<sequenceID>`*`.csv` by calling the `StockService` for each line. This is achieved by declaring a dedicated `StockTranslator` delegating to a `StockImportAdapter` configured in `hot-folder-spring.xml`. Thereby it is possible to configure a different warehouse for the stock updates and also leverage the stock history maintained by the platform `StockService`.

The following table describes the file format:

Column	Attribute	Type	Optional	Localized	Default	Values	Constraints	Description
0	product code	String						
1	stock	Integer					>= 0	
2	warehouse	String	yes					

Variant Feed

When using the `yacceleratorcore` extension, the variant products are imported as `ApparelSizeVariantProduct` or `ApparelStyleVariantProduct`, depending on the column value. If no product variant type is given, the product is imported as `ApparelSizeVariantProduct` if the size column is not empty, otherwise it is imported as `ApparelStyleVariantProduct`. This can easily be adapted to different requirements by changing the filter and converter configuration.

The optional language part of the file name is added to the ImpEx header by using a header substitution. Thereby it is possible to update localized attributes, like style or size, for this language only.

The following table describes the file format:

Column	Attribute	Type	Optional	Localized	Default	Values	Constraints	Description
0	parent product code	String						
1	code	String						
2	variantType	String	yes			**ApparelSizeVariantProduct** **ApparelStyleVariantProduct**		
3	style	String	yes	yes				
4	size	String	yes	yes				

Price Update Feed

Price rows are imported from files named like `price-<sequenceID>.csv`. Prices are imported to both the Online and Staged catalog. The `Unit`, `unitFactor`, and `minqtd` attributes are set to default values.

The following table describes the file format:

Column	Attribute	Type	Optional	Localized	Default	Values	Constraints	Description
0	product code	String						
1	price	Double					>= 0	
2	currency	String					EUR USD JPY	ISO code of the price currency
3	net	Boolean	yes		Configurable per site	true false		net/gross setting

Merchandising Feed

Product references are imported from files named like `merchandise-<sequenceID>.csv`.

The following table describes the file format:

Column	Attribute	Type	Optional	Localized	Default	Values	Constraints	Description
0	source product code	String						Source
1	reference type	String				see enum **ProductReferenceType**		
2	target product code	String						Target

Media Feed

Media is imported from files named like `media-<sequenceID>.csv`. Images have to be provided in the `images/<media format>` subdirectory in the following formats: 30Wx30H, 65Wx65H, 96Wx96H, 300Wx300H, 515Wx515H, 1200Wx1200H.

The import is processed in three steps:

1. Import media in all specified formats.
2. Import a media container for the product gallery.
3. Assign the media and media container to the product.

The following table describes the file format:

Column	Attribute	Type	Optional	Localized	Default	Description
0	product code	String				Code of product
1	code	String				Code of media

Customer Feed

Customers are imported from files named like `customer-<sequenceID>.csv`.

The following table describes the file format:

Column	Attribute	Type	Optional	Localized	Default	Values	Constraints	Description
0	email	String						Email and ID of a customer
1	first name	String	yes					
2	surname	String	yes					
3	title	String	yes			mr mrs miss ms dr rev		Code of title

If there is no value for `first name` and `surname` then the name for `Customer` is created from the email address.

Example import file:

```
testCustomer@test.com,firstName,lastName,dr
testCustomer1@test.com,firstName,,
testCustomer2@test.com,,,dr
testCustomer3@test.com,,,
```

Modifying File Mappings

Adding additional CSV columns can be achieved by updating the corresponding converter:

- Update the header property to import the new column.
- Update the impexRow to supply the CSV column in the transformed file.

For example, importing a unit as a fifth column in the `price-`*`<sequenceId>`*`.csv` file can be configured by changing following line:

```
<value>;{+0}:$catalog:Staged;{+1};{+2};{3};;;;$catalog:Staged;{S}</value>
```

to the following:

```
<value>;{+0}:$catalog:Staged;{+1};{+2};{3};{4};;;$catalog:Staged;{S}</value>
```

Related Information

ImpEx API
Using ImpEx with Backoffice or SAP Commerce Administration Console
Event System
yacceleratorcore Extension
acceleratorservices Extension
Localization of ImpEx Using Build Time Generation
Importing Product Tax Codes
Styles and Size Variants in the SAP Commerce Product Cockpit

Essential and Project Data

SAP Commerce Accelerator provides example sites that you can load as required through the Essential and Project Data load processes.

This document describes these processes and the options that are available to control the data that is loaded.

General Concept of Core and Sample Data

The Essential/Project data provides a starting point for a project.

The Essential/Project data setup for the `yacceleratorcore` extension template is designed to:

- Create languages, currencies, delivery options, themes, and URL bindings.
- Create CMS components, page templates, and pages and configure CMS restrictions.
- Configure system user groups and roles.

These need to be altered and customized for any site based on the Accelerator but provide a starting point for a project. Loading the `yacceleratorcore` extension essential and project data, creates fully functional, but empty, web sites.

The Essential/Project data setup for the `apparelstore` and `electronicsstore` extensions is designed to:

- Create base stores.
- Create empty catalogs.

- Create CMS sites and bind the base stores and catalogs to them.
- Create Solr search indexes and triggers.
- Load sample product data, categorization, media, suppliers, reviews, promotions, and stock.
- Load sample store data, that is locations, facilities, and imagery.
- Load and configure sample CMS pages and components.

The sample data provided with the `apparelstore` and `electronicsstore` AddOn extensions helps to boost productivity during development but should not be loaded into production systems. It should be removed from the `localextensions.xml` file and functionality, or tests should not rely upon it being present. The sample data provided is primarily an example of how complex data sets can be set up in the SAP Commerce-driven systems.

Features

SAP Commerce Accelerator provides the class **AbstractSystemSetup** to assist with adding new extensions and additional ImpEx files.

This features:

- A simplified mechanism for adding **yes/no** options for controlling **Project** data setup using the **createBooleanSystemSetupParameter**.
- Automatic loading of localized ImpEx files following a naming convention: loading `my_products.impex` causes the files `my_products_en.impex`, `my_products_de.impex` and so on to be loaded - the list of languages that are detected is taken from the **CommonI18NService**.
- Optimized configuration of catalog synchronization jobs for product and content catalogs.
- Configuration of Solr Indexer jobs.

For more information, see Using ImpEx with Backoffice or SAP Commerce Administration Console.

Functional Storefront Interaction

The SAP Commerce Accelerator uses Spring MVC for the storefront, and integrates the SAP Commerce WCMS Module to allow creation and editing of pages and components.

Controllers CMS Integration

Each page controller in the SAP Commerce Accelerator storefront extends the **AbstractPageController**. Pages that require CMS integration, for components or page title for example, should call the **storeCmsPageInModel(final Model model, final AbstractPageModel cmsPage)** method to populate the model with the **CMSPage**, as shown in the following **AbstractPageController.java** code snippet.

```
protected void storeCmsPageInModel(final Model model, final AbstractPageModel cmsPage)
{
	if (model != null && cmsPage != null)
	{
		model.addAttribute(CMS_PAGE_MODEL, cmsPage);
		if (cmsPage instanceof ContentPageModel)
		{
			storeContentPageTitleInModel(model, getPageTitleResolver().resolveContentPageTitle(cmsPage.getTitle()));
		}
	}
}
```

Next, the **CmsPageInterceptor postHandle** method runs, which takes each CMS slot for the page and inserts it into the model. This allows the **cms2lib** extension tags to pick up the slots and insert the components.

Each component is driven off its own JSP, and the name of the JSP corresponds to that of the CMS component.

Rendering of JSP Pages

JSPs are constructed from tags. Out of the box all JSPs derive from the `page.tag` and `master.tag` files, get their style sheets from the `styleSheets.tag` tag file and get their scripts from the `javaScript.tag` file. There is no use of JSP includes.

JSPs are never referenced directly via a URL. All JSPs are packaged under `WEB-INF` directory and so are not externally visible. After a request is handled by an Spring MVC controller it returns the name of a JSP - its view - as its response. This JSP is located under the `WEB-INF/views/pages/` directory. This JSP is then used to render the model.

Spring Themes

Themes are applied across a site. The SAP Commerce **SiteTheme** item is implemented via Spring Themes in the storefront. Each theme is driven from a `theme-` ***themeName*** `.properties` file, which sets the appropriate CSS and imagery.

For more information, see http://static.springsource.org/.../mvc.html : SpringSource Theme Resolver.

CSS and JavaScript Frameworks

The Accelerator storefronts use Blueprint CSS to drive the layout of the sites, and JQuery as their JavaScript framework.

For more information on JQuery, see http://jquery.com/ .

Related Information

http://static.springsource.org/.../mvc.html

Importing Product Tax Codes

You can use an ImpEx script file to import external product tax codes into the SAP Commerce Accelerator.

External Taxes for Product

The **ProductTaxCode** object stores the taxes for the particular product. The attributes of the object are:

- **productCode**: Product identifier
- **taxCode**: Tax identifier
- **taxArea**: ISO code for country

Import External Taxes Through ImpEx Script File

Taxes can be loaded into the SAP Commerce Accelerator with the help of the ImpEx script file. The instruction syntax is as follows:

```
INSERT_UPDATE ProductTaxCode;productCode[unique=true];taxCode;taxArea[unique=true]
```

Sample taxes for a product can be found in the **`products-tax.impex`** file localized in the ***$[internal:HYBRIS_BIN_DIR]*`/ext-accelerator/acceleratorsampledata/resource/acceleratorsampledata/import/productCatalogs/electronicsProductCatalog`** directory. Loading this file is done by the **yacceleratorinitialdata** extension.

Sample ImpEx Script File

The following is a sample from the **products-tax.impex** file:

```
INSERT_UPDATE ProductTaxCode;productCode[unique=true];taxCode;taxArea[unique=true]
;23191;PC080601;US
;23210;P0000000;US
;23231;P0000000;US
;107701;PE070000;US
;23191;PC080601;GB
;23210;P0000000;GB
;23231;P0000000;GB
;107701;PE070000;GB
```

Import External Taxes using the Hot Folder

Taxes can be loaded into the SAP Commerce Accelerator through the hot folder. The idea of the hot folder data import is that CSV files are imported automatically by moving them to a folder that is scanned periodically by the system. For more details on hot-folders, see Data Importing.

External taxes are imported from the files named with the following convention: **`external_tax-`** ***<sequenceID>*** **`.csv`**. The following table provides the file format description.

Column	Attribute	Type	Required	Description
0	product	String	yes	Product code
1	tax	String	yes	Tax code
2	country	String	yes	Isocode for country

The following sample shows an example of the import file content from the **external_tax-1.csv** file:

```
11160,PC040400,GB
11160,PC040400,US
117211,PC040400,DE
```

Related Information

Data Importing

Pickup In Store

The SAP Commerce Accelerator for B2C supports a complete end-to-end Buy Online Pickup in Store (BOPiS) feature. The feature is built directly into the default purchasing process and is available across areas such as Search, Stock, Cart and Checkout.

The BOPiS feature provides the following functionality:

- Check stock availability in nearby pickup locations.
- Add items to cart for pick up at proposed locations that have available stock along with items to be shipped.
- Consolidate order entries for multiple pickup locations in to a single pickup location.
- Pay online, collect order in store.
- View the order history and receive e-mail notifications when the items are available to be picked up.

Google Maps Embed API Key for Pick-Up-In-Store

The Google Maps Embed API is used to enable the Pick-Up-In-Store pop-up window to display a map that shows the location of a selected store. This implementation requires an associated Google account, as well as a generated Google Maps Embed API key for that account.

Add the API key to the `project.properties` file, as follows:

```
googleEmbedApiKey=<yourAPIkey>
```

i Note

You must obtain your own API Key to use the Google Maps Embed API. For more information, see https://developers.google.com/maps/documentation/embed/guide .

Enabling and Disabling the BOPiS Feature for a Storefront

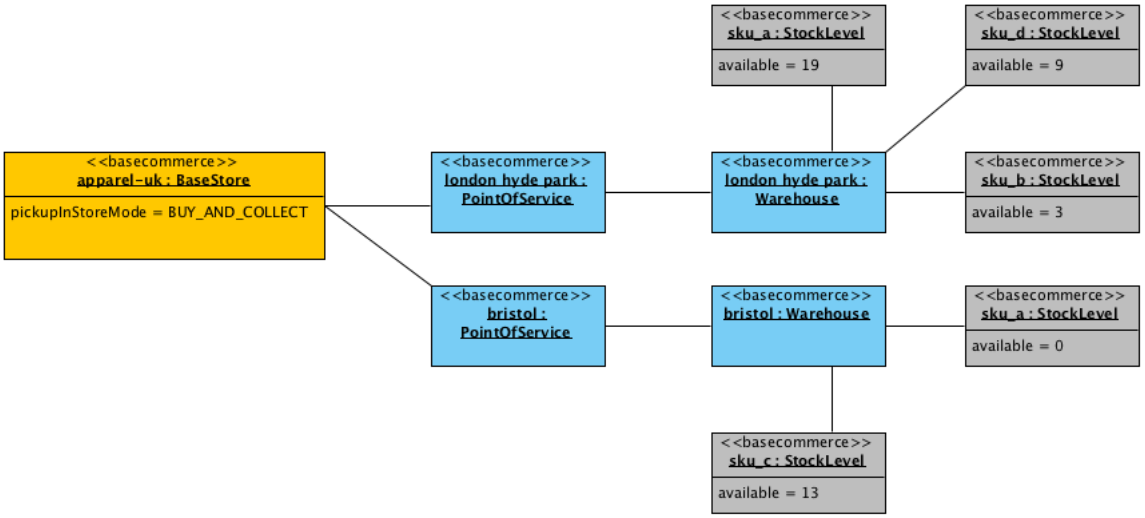
SAP Commerce Accelerator for B2C provides a Pickup In Store feature that is a standard feature of the `commercefacades` and `commerceservices` extensions. However, to fully enable the functionality, you must add relevant data to an instance. Commerce Accelerator ships with a fully functional Pickup In Store feature with sample data loaded in the reference Apparel UK and Electronics storefronts.

This feature can be enabled and disabled for each individual storefront. To demonstrate this, the BOPiS feature is disabled in the reference Apparel DE storefront.

Enabling the BOPiS Feature

To enable the BOPiS feature, you must:

- Set `PickupInStoreMode` on the BaseStore: `DISABLED` and `BUY_AND_COLLECT` are the currently supported values. When `BUY_AND_COLLECT` is specified as the value, a process in the `yacceleratorstorefront` extension is enabled; the process allows a user to add items to a cart that the user can pick up at one or more stores that currently have available stock. The user must pay online prior to picking the items up in store.
- Add one or more `PointsOfService` of the type `STORE`. These `PointOfService` will also appear in the Store Locator.
- For each `PointOfService` that offers the Pickup In Store Service, link a Warehouse.
- Assign the `StockLevel` to each `PointOfService` Warehouse.



Disabling the BOPiS Feature

You can disable the BOPiS feature as follows:

- For a base store: Set `PickupInStoreMode` on the `BaseStore` to `DISABLED` to disable the entire BOPiS feature per `BaseStore`.
- For a specific store location: Ensure that the `PointOfService.warehouse` attribute is set to `null`.
- For a specific SKU: Remove `ForceOutOfStock` or `StockLevel` items for the `PointOfService` warehouses for the SKU's article number.

You can also completely disable the BOPiS feature. You must:

1. Ensure `PickupInStoreMode` is set to `DISABLED` on the Base Store to disable the BOPiS feature for each `BaseStore`.
2. Do not load the `StockLevel` for `PointOfService Warehouse`.

Checking Available Inventory for Pick Up

Ensuring that there is sufficient inventory available in stock is a key feature of a customer's Pickup In Store buying experience. Therefore, SKU stock levels can be checked for both shipping and pick up in store options. For the pick up in store option, Availability-to-Sell (ATS) stock level can be exposed at the `PointOfService` level so that customers can check the stock available in store.

SAP Commerce Accelerator allows users to check stock for SKUs at nearby stores or refine search results by available stock at proposed pickup stores in the area.

For more information, see Stock Management.

Cart

`CommerceCartService` supports the functionality to add items to cart at a proposed pickup location that has sufficient available stock. The supported functionality includes:

- Adding items to cart for pick up at a proposed pickup location with stock verified to ensure sufficient available inventory.
- Editing carts by changing pickup locations or by changing to shipping on a entry by entry level.
- Combining entries picked up from more than one pickup location with entries shipped to a delivery address in a single cart.
- Restoring the cart, using fallback logic, for out of stock items.

Adding Items to Cart for Pickup

`CommerceCartService` offers overloaded `addToCart` methods supporting the ability to add an item to the cart that preempts whether it is for shipping to a delivery address or pickup in store.

Shipping Delivery Address

```
CommerceCartModification addToCart(CartModel cartModel, ProductModel productModel, long quantity, UnitModel unit,
                        boolean forceNewEntry) throws CommerceCartModificationException;
```

Pickup In Store

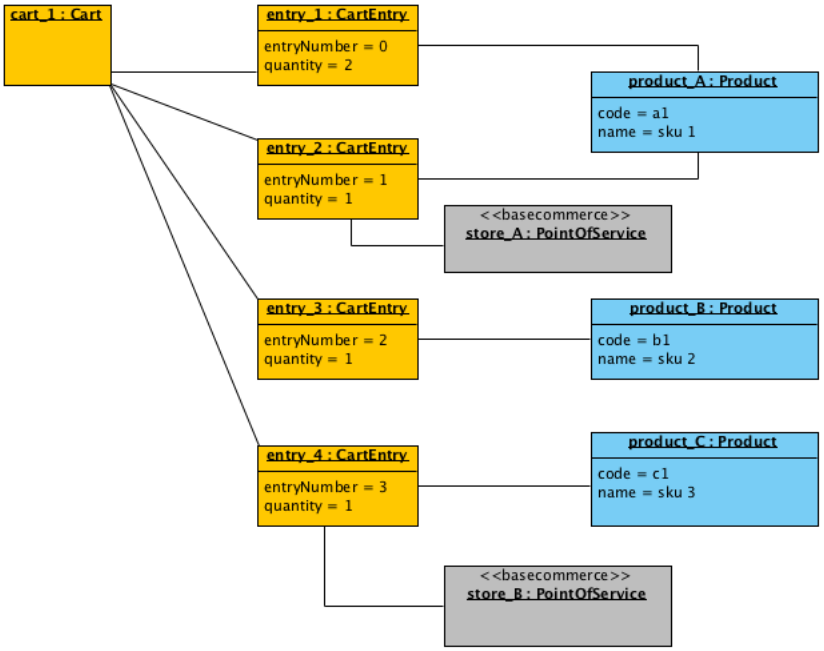
```
CommerceCartModification addToCart(CartModel cartModel, ProductModel productModel, PointOfServiceModel deliveryPointOfService,
                        long quantity, UnitModel unit, boolean forceNewEntry) throws CommerceCartModificationException;
```

When multiple pickup locations have been specified for the same SKU, or the user has nominated to pickup the SKU from a location, but also has the same SKU delivered to them, multiple entries are created for the same SKU.

For pick up in store, `CommerceCartService` does the following:

- Checks that there is sufficient available stock at the pickup location for the requested quantity. The stock is reduced if there is insufficient quantity. The operation fails if there is no stock at all.
- Creates a new entry if an entry for the same SKU does not already exist.
- Creates a new entry if an entry for the same SKU and pickup location does not already exist.
- Updates the quantity of an existing entry if an entry with the same SKU and pickup location already exists.

The `forceNewEntry` flag can be used to override this behavior. This flag forces the creation a new entry when a new SKU is added to a cart. However, this is not standard behavior of the cart page shipped with the `yacceleratorstorefront` extension.



Changing Cart Entry Pickup Location or Shipping Mode

For an item already in the cart, `CommerceCartService` supports the functionality to change pickup location or shipping mode.

Change Pickup Location

```
CommerceCartModification updatePointOfServiceForCartEntry(CartModel cartModel, long entryNumber,
                        PointOfServiceModel pointOfServiceModel) throws CommerceCartModificationException;
```

Change to Shipping

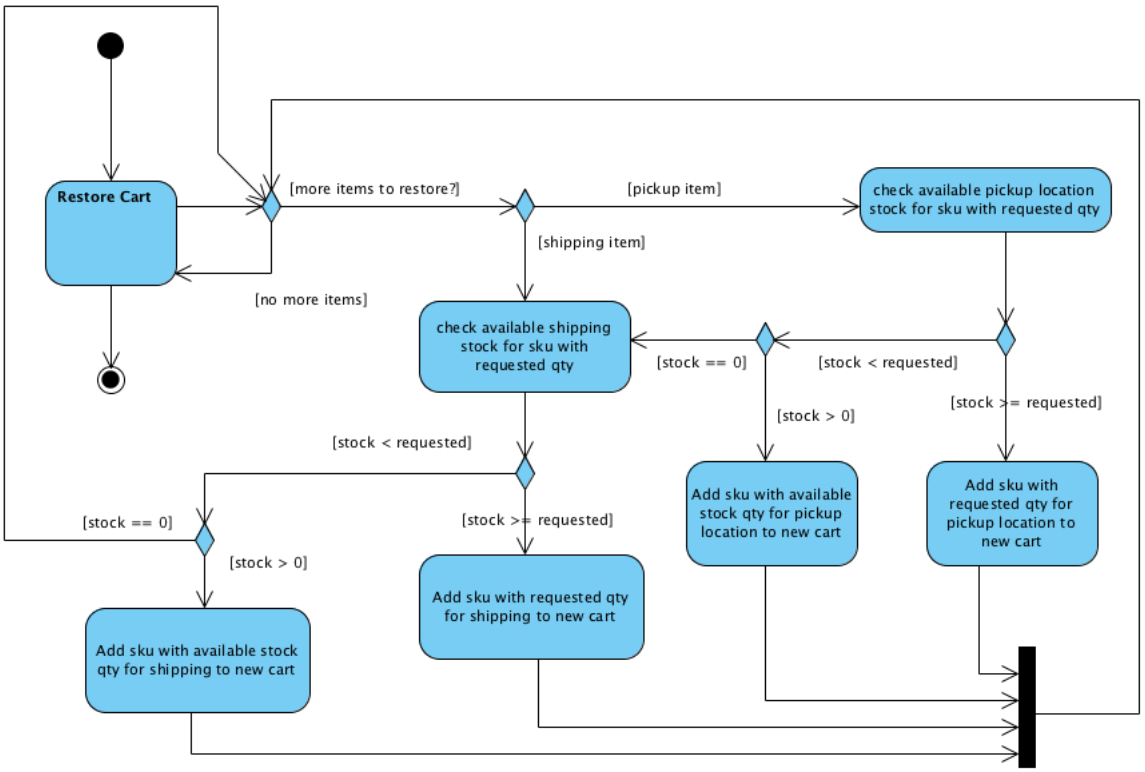
```
CommerceCartModification updateToShippingModeForCartEntry(CartModel cartModel, long entryNumber)
                        throws CommerceCartModificationException;
```

Note that the logic regarding checking stock described above also applies here.

Cart Restoration

`CommerceCartService` supports restoring both saved shipping and pickup entries from a saved cart to a new cart. The following activity diagram shows the stock fallback rules implemented by the `restoreCart` method:

```
public CommerceCartRestoration restoreCart(final CartModel oldCart) throws CommerceCartRestorationException
```



Checkout Process

The SAP Commerce Accelerator BOPiS functionality is supported by the desktop and mobile multistep checkout process.

The checkout process provides the following functionality:

- Checkout entries picked up from more than one pickup location as well with entries shipped to a delivery address in a single order.
- Consolidation of an order with multiple pickup locations into a single location.
- Online payment for pickup items prior to picking them up in store.
- Grouping of cart entries by delivery address and pickup locations.
- Splitting up an order into consignments for order history progress updates.

Supported Checkout Flows

BOPiS is currently only supported with SAP Commerce Accelerator for B2C, for both desktop and mobile multistep checkout flows. If the cart contains an item for pickup, non-supported checkout options will not be available.

Order Entry Groups vs Consignments

For the purpose of developing order tracking and status update features, as well as pre-grouping entries in orders into relevant batches for information and display purposes, the `commercefacades` extension `Cart` and `Order` data object model supports the grouping of a subset of Cart and Order entries into additional grouping components called `OrderEntryGroups` and `Consignments`. This data can be used to render order summary screens, order history, and order update notifications, such as item dispatched and item ready to collect emails.

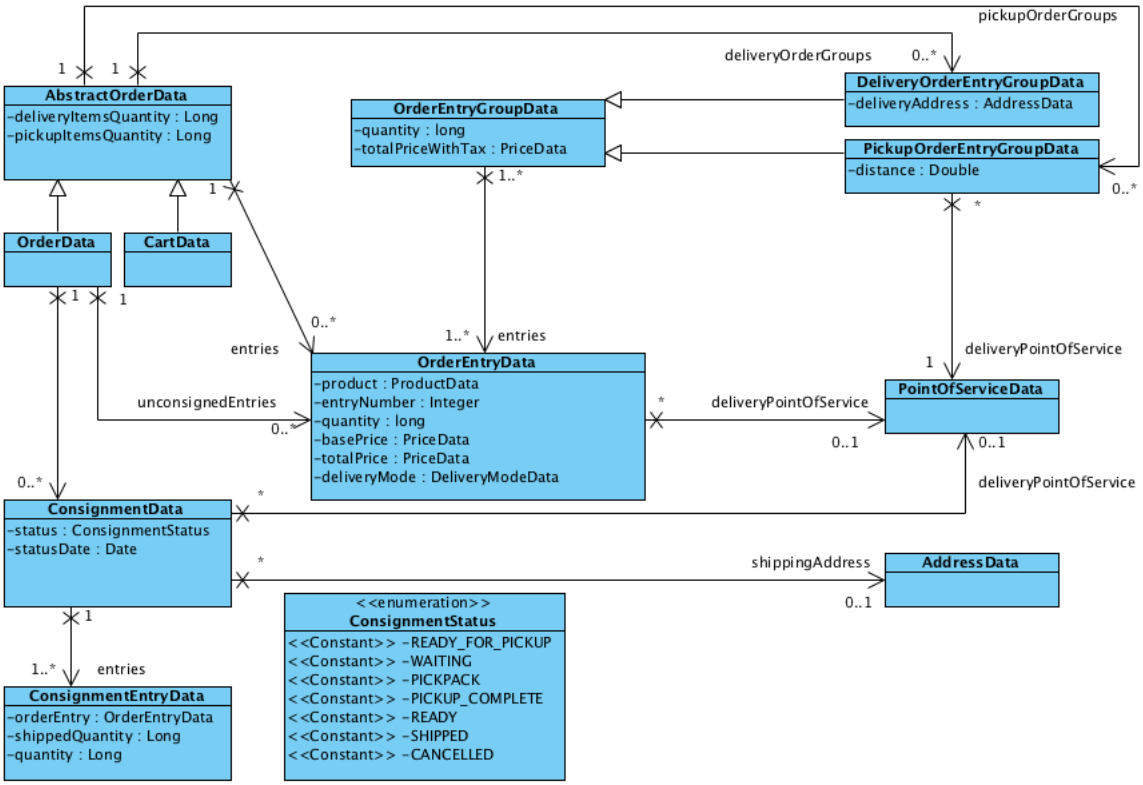
`Order Entry Groups` and `Consignments` have different purposes.

Order Entry Groups

- Are applicable for both `Carts` and `Orders`.
- Allow items in a cart or order to be broken into relevant groups for easier processing, for example, through order information pages.
- Group related order entries by a characteristic, for example, in the context of this document this is group by delivery address, `DeliveryOrderEntryGroupData`, and pickup location, `PickupOrderEntryGroupData`.

Consignments

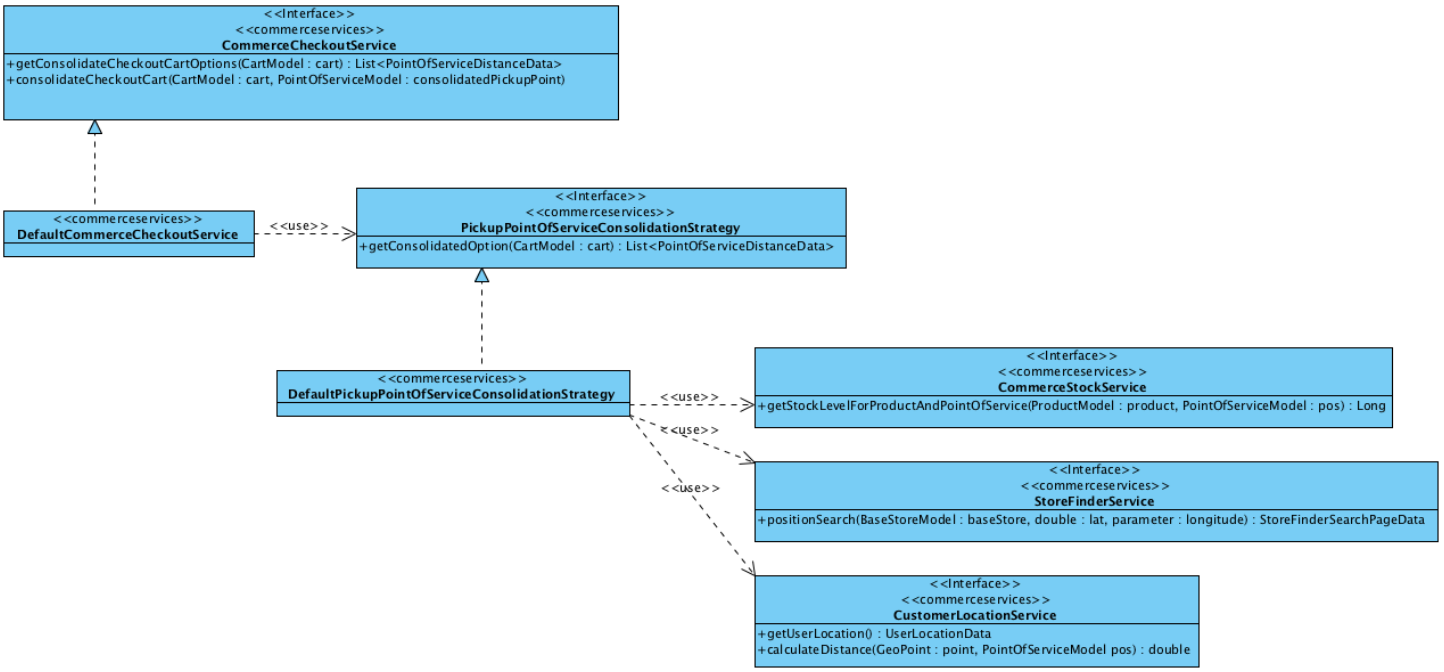
- Applicable only for `Orders` and a by-product of order fulfillment processes.
- Order entries, or a quantity subset of an order entry, grouped together in a single delivery unit, which could be either a shipment or a customer pickup.
- Typically asynchronously created by order management processes following the placement of the Order.
- Transient, that is order entry quantities can be further split into additional consignments later on in the fulfillment process such as warehouse mis-pick.
- Usually one consignment per pickup location in an order, unless a pickup location can only source part of its assigned order initially.
- Have their own status, for example, Picked/Shipped/Cancelled/Ready and so on, the order status is usually an aggregation of all the current consignment status.
- An order with an entry with quantity of 2 could result in 2 consignments if each item is shipped separately.
- Some entries may be linked to consignments, but it is possible for others to be unconsigned.



Consolidating Pickup Locations

A customer can pick up items from multiple stores locations in a single cart and can check out this cart as a single order. However, it is also possible that all the items the customer has selected for pickup can be picked up from a single store location. When a user selects to pick up from multiple locations, the `CommerceCheckoutService` exposes the services to:

1. List alternative pickup locations that can source the entire pickup part of the order; this is an optional action the user can select during checkout.
2. Propose an alternative pickup location to consolidate the pickup entries in a cart to be collected at the suggested location.



The default behavior of the `DefaultPickupPointOfServiceConsolidationStrategy` is as follows:

1. For all pickup store locations in the Cart, check if there is sufficient stock at the proposed location for the customer to collect the entire pickup part of the order.
2. If the above is not true, and the user location is known, expand the search until a location is found where the entire pickup part of the order can be collected.

The objective of the pickup location consolidation is to have fewer pickup `OrderEntryGroups`.

Related Information

commerceservices Extension
commercefacades Extension

Quick Orders

The Quick Order feature provides users a speedy way to add multiple items to their cart.

Feature Overview

Quick Order is ideal for users who order multiple items and know the item IDs. It is available in both B2B Accelerator and B2C Accelerator. Users enter the item ID and then press the Tab key to move focus to the Quantity field. By default, three rows appear in the form when it is accessed, and a maximum of 10 items can appear on the form. You can change these default values by modifying the following entries in the `project.properties` file in the `yacceleratorstorefront` extension:

```
yacceleratorstorefront.quick.order.rows.min=3
yacceleratorstorefront.quick.order.rows.max=10
```

⚠ Caution

We recommend that the default maximum value should not be higher than 10 to prevent performance issues.

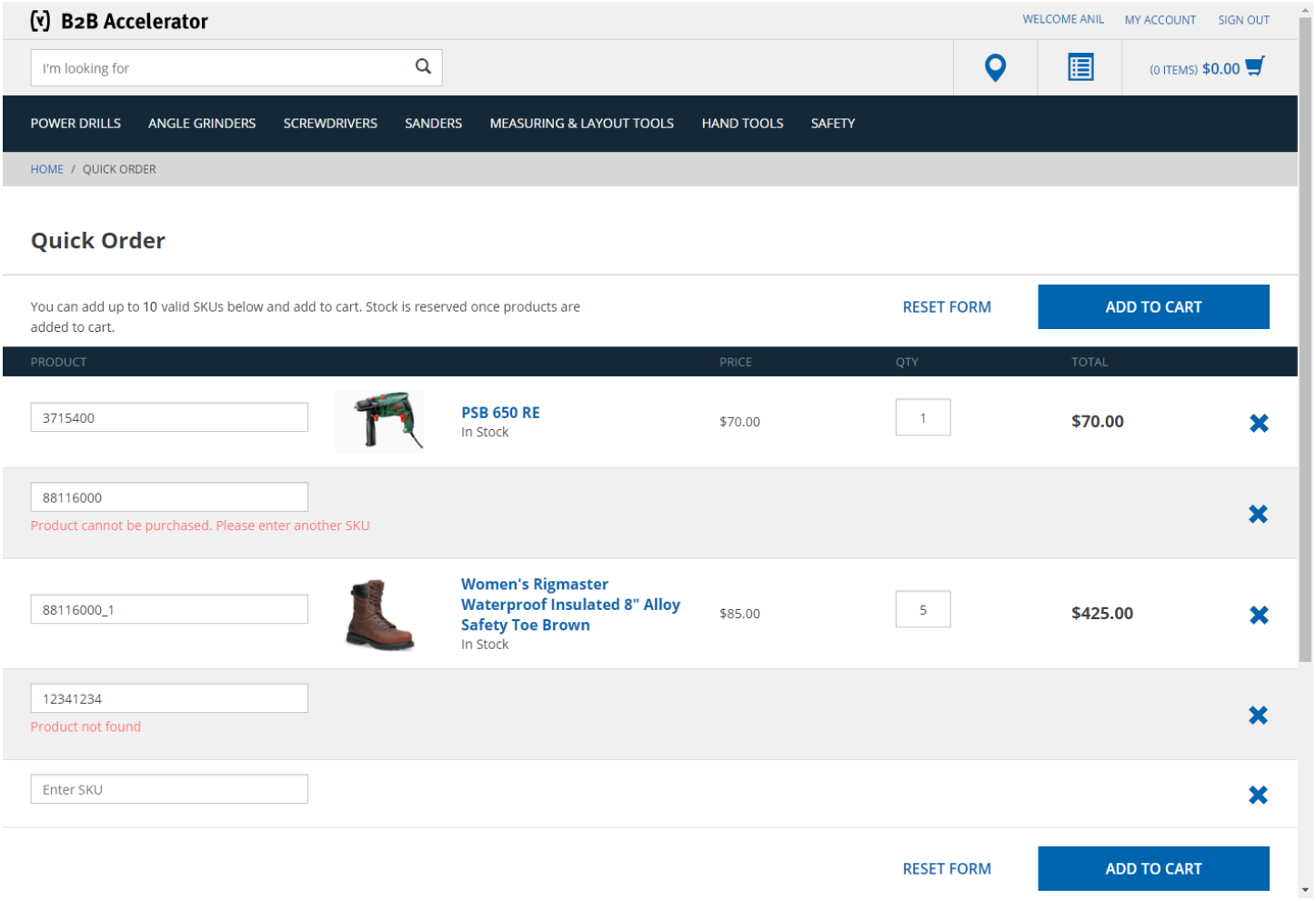
User Experience

The Quick Order page is accessed by selecting Quick Order from the Order Tools menu ( ). Three empty rows appear on the form by default. Users enter the item ID and then press the Tab key to move focus to the Quantity field and then the next empty row. Once a valid item ID is entered and focus is moved away from the field, the item image, description, and unit price appear in the row, as well as a value of "1" in the QTY field. If the entered item does not have any stock, then the QTY field displays a value of "0" and is grayed out. If the quantity entered is more than the available stock, then the value is adjusted to match the available stock.

In some cases, the item ID entered cannot be processed. These error messages are displayed for the following scenarios:

- `Product not found`: Displayed when an invalid item ID is entered. Users must re-enter a valid ID.
- `Product cannot be purchased. Please enter another SKU`: Displayed when the base ID for a multi-dimensional product is entered. Users must enter the ID of the specific variant.
- `SKU already exists in the form`: Displayed when the item was previously entered on the form. Users can adjust the quantity of the original entry.

This screenshot shows the Quick Order form with valid and invalid entries:



As users fill the form, a new row is automatically added when the last empty row is being filled out (up to the maximum number of rows specified in the `project.properties` file). At any time, users can modify their order:

- Individual line items can be removed by clicking the X for that row.
- Quantities for any item can be modified.
- The entire form can be cleared by clicking the Reset Form link.

Once all line items are entered as required, users click the **Add to Cart** button to add all the items to their cart. Note that line items that have errors are not added to the cart.

Search and Navigation

The SAP Commerce Accelerator uses and extends the capabilities of the Search & Navigation module.

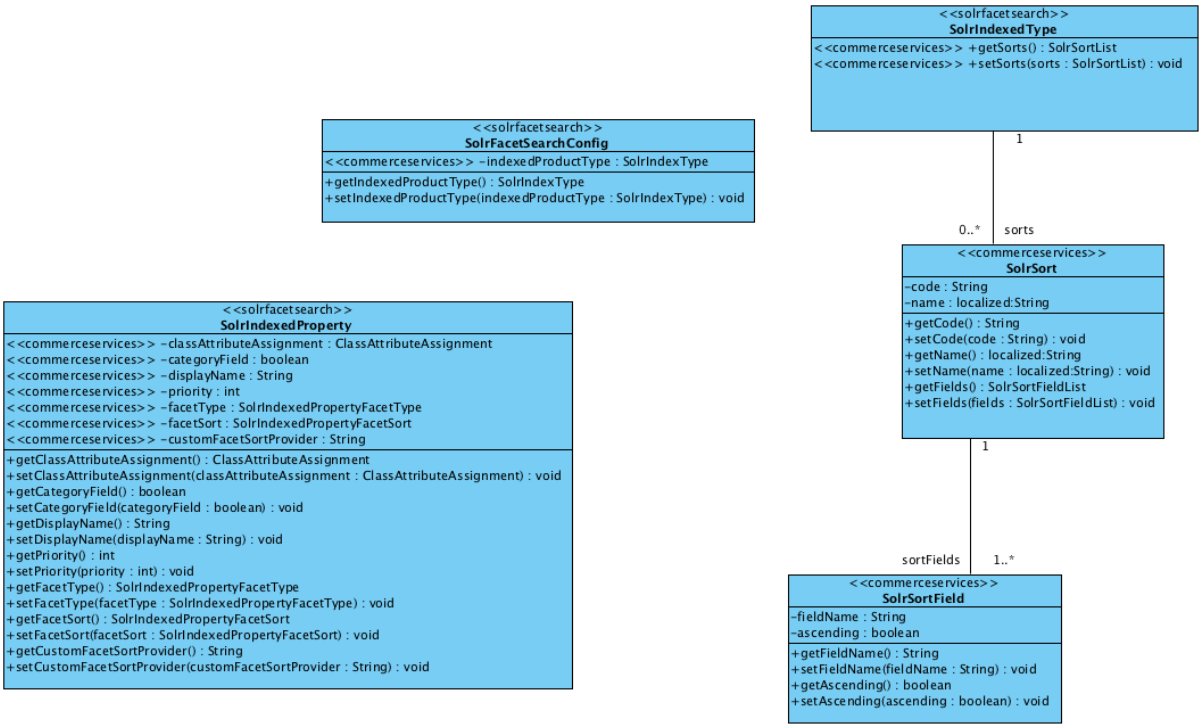
For details on how to configure the Search & Navigation module, see Search and Navigation Module Implementation.

Solr Configuration

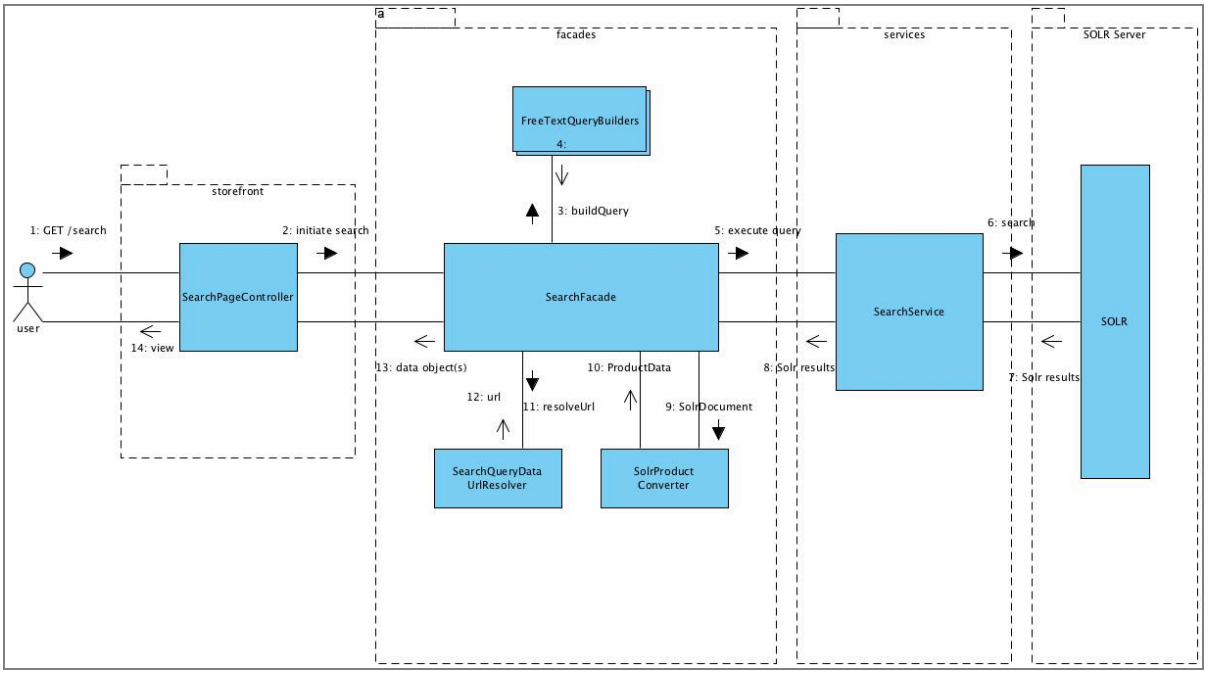
Using the solrfacetsearch Extension, you can configure the types and properties to index into Solr in the following ways:

- You can create the entire search configuration in the database as hybris items. This allows you to extend and modify the configuration via ImpEx and the Backoffice Administration Cockpit. For more information, see Solr Facet Search Configuration.
- You can create a separate Solr index for each product catalog. You will then have two indexes, one for the Apparel product catalog and one for the Electronics product catalog. The configuration for each of these ImpEx instances is imported through ImpEx using the following files:
  - `<HYBRIS_BIN_DIR>/modules/base-accelerator/yacceleratorcore/resources/yacceleratorcore/import/productCatalogs/apparelProductCatalog/solr.impex`
  - `<HYBRIS_BIN_DIR>/modules/base-accelerator/yacceleratorcore/resources/yacceleratorcore/import/productCatalogs/electronicsProductCatalog/solr.impex`

The following diagram illustrates the extended data model.



The following diagram illustrates search interaction.



Commerce Services

The `commerceservices` extension contains item types extended from the `solrfacetsearch` extension. The `commerceservices` extension provides the following additional functionality.

Feature	Description
Product results ordering	The `solrfacetsearch` extension provides some support for configuring the order in which results are returned, allowing the default sort order to be set. The `commerceservices` extension allows all the supported ordering to be configured on the `SolrIndexedType`. Each sort option has a display name and a list of the Solr fields to sort against. Sorting in ascending and descending order is supported. This ensures that all results sorting is performed in Solr and that the available ordering is part of the configuration and not the business logic.
Display name for each facet	The `commerceservices` extension adds a localized display name to the `SolrIndexedProperty`. This is used in the frontend when displaying the facet.
Facet ordering priority	The `commerceservices` extension adds a priority for each facet. This is used to provide an ordering of the facets in the display, with the highest priority facet appearing first. Facets with a negative priority are not displayed. This allows for internal filtering facets that are not displayed to the user.
Multi-select facets	The `solrfacetsearch` extension supports refinement facets, where one facet value can be selected within the facet. This is sufficient for most drill-down search refinement requirements. The `commerceservices` extension supports multi-select facets, where more than one value can be selected in the same facet. These values can be combined using either the **AND** or the **OR** operators. For example, you can allow multiple shoe sizes to be selected as facet filters at the same time.
Facet value sorting	The `solrfacetsearch` extension does not support different sorts per facet. The `commerceservices` extension supports sorting the values within each facet through the following methods: • By facet value count, with the highest count first • Sorted alphabetically, A to Z • Using a custom sort provider
Link indexed properties to classification attributes	The `commerceservices` extension links indexed properties to the classification system, providing the following additional features: • If the `ClassAttributeAssignment.searchable` property is set, the free text query is also matched against this classification feature. • If the `ClassAttributeAssignment.listable` property is set, the classification feature values are included in the product data shown on the search results and category navigation pages.
Category facet flag	Category facets are typically treated differently on search and navigation pages. The `commerceservices` extension has a flag for the indexed property to indicate if it should be treated as a category facet.

For more information, see commerceservices Extension.

CommerceSearchService

The `commerceservices` extension provides the interface to execute free text searches and category searches, as well as the ability to page, sort, and refine these searches. To ensure performance and scalability, its data model is designed to ensure that the states are held by the client and not on the server.

The user's search journey is essentially a stateful one. The user runs a search, reviews the results, selects a refinement option, reviews the results, and repeats as required.

More detailed information about the implementation can be found by looking directly in the `ProductSearchService` interface.

Indexing

Indexing the correct products into Solr is critical. The Solr index must include only purchasable products. It is not possible to programmatically filter the products returned from the Solr search as this would invalidate the total results count, the facet counts, and the paging.

The `Solrfacetsearch` extension supports full index rebuilding as well as incremental index updates. The products to index are retrieved through FlexibleSearch queries. It is important to ensure that all purchasable products are included in the full index rebuild query.

The query for incrementally updating the index is often much more complex than the full rebuild query. It has to take into account any modified products and any items related to the product that may also have been modified.

For more information, see FlexibleSearch.

Category Hierarchies

The SAP Commerce categories feature is very flexible. For example, one category hierarchy could be used to represent product categories, and another could be used to represent brands.

These separate category hierarchies must be indexed into separate indexed properties. The `commerceservices` extension allow these hierarchies to be configured via a `CategorySource` interface that is injected into the category `FieldValueProviders`. The `CategorySource` implementation is configured with the code of the root category and finds all the category paths between the product and the root category. You can also configure it to ignore classification class categories.

For more information, see commerceservices Extension.

Facet Value Display Name Providers

In certain cases, the facet value indexed into Solr is not the value that you would like to display to the user. An example of this is with categories, where the indexed value is the unique category code, but the value that you would like to display is the category localized name.

The `solrfacetsearch` extension provides this facet display name provider that looks up the localized category name to show instead of the facet value indexed into Solr. When sorting the facet values alphabetically, it is the display name that is sorted and not the facet value.

Sorting Product Results

The ordering of the product results returned from the Solr is critical to correct operation of search. The ordering must be achieved within the Solr; the results cannot be resorted programmatically once they have been returned from the Solr, as this invalidates the paging support.

Results ordering is typically not overly complex, since the user can only select from a number of predefined orderings. This is precisely what the `comerceservices` extension provides. The supported sorts are configured for each indexed type through database items. Each `SolrSort` item has a localized name to display to the user, an order list of `SolrSortField` items which indicate which field and direction to sort on.

The `comerceservices` extension ensures that the results are always finally sorted by the Solr score. The Solr score is the result measure of relevance to the search terms. If you have a sort that orders based on the price field, then it is very likely that there are many results with the same price value. The sort does not specify what the relative ordering of these results with the same price value should be. To ensure that there is a defined and repeatable ordering, the score field is automatically added as a final ordering field.

Configuration in the Backoffice Administration Cockpit

You can configure the Solr facet search options using the Backoffice Administration Cockpit. For more information, see Solr Facet Search Configuration.

Commerce Facades

The `ProductSearchFacade.java` file provides a similar interface to that of the `ProductSearchService` interface, but the return type is a `ProductData` object.

Starting a Search

Each search is initiated by calling either `textSearch(String text)` or `categorySearch(String categoryCode)` in the `ProductSearchFacade.`

These search methods are used for free text search and category browsing.

With free text search:

- The search is triggered by the user entering a search string.
- The search string is translated into Solr search terms.
- The fields to search can be configured (see below).
- The products that match the free text search string are returned.

With category browsing:

- The search is triggered for a category.
- The products that are in the specified category are returned.

The search returns the matching products in the default sort ordering. All the search methods return the same type, `SearchPageData`.

The `ProductSearchPageData` type contains the following data:

- Search query that was executed
- Matching products
- Pagination data (total number of results, page size, current page number, current sort)
- Currently applied filters
- Available facets and facet values that can be applied to filter the results
- Available sorts

SearchStateData

The search state data contains the data required to run a search. The result of running a search, `ProductSearchPageData`, contains `SearchStateData` instances for applying and removing search filters. See the following sections for more information.

Each `SearchStateData` has a URL that allows the search to be performed.

Category Landing Pages

Category search is used to browse the products within a category. The products in the category are returned in the default sort order.

The category search also supports category landing pages where products are not shown. Selecting a subcategory navigates directly to the selected category rather than filtering the results to only show the products that are in the selected category. This category navigation mode can be configured by setting the category search method Boolean parameter to true in the `SearchFacade`.

Free Text Search

The free text search string is turned into Solr query terms which are then executed to select the matching products. The `ProductSearchService` is configured through spring with a list of `FreeTextQueryBuilders` that are responsible for building up the Solr query terms from the free text search. There is a free text search mechanism that can be used instead. For more details, see Search Process.

Paging

The `ProductSearchPageData` includes `PaginationData` that includes information on the total number of products matched, the page size, the current page, and the total number of pages.

In order to get a specific page of search results, the `ProductSearchFacade` provides methods that take the `currentQuery` from the `SearchPageData` and a `PageableData` with the `currentPage` set to the requested page. If the requested page is beyond the range of available pages, the last page is returned.

Sorting

The `PaginationData` also includes the current sort option, which is one of the available sorts in the `ProductSearchPageData`. The sort order can be changed by calling `ProductSearchFacade`, passing the `currentQuery` from the `ProductSearchPageData` and a `PageableData` with the sort set to the requested sort. If the requested sort is valid, the sort is changed, the search rerun, and the results returned.

Building the Product Results

The Solr search returns Solr documents, which are converted into `ProductData` by a specialized converter. The converter has access to all the data about the product that has been indexed into Solr. The intent is to create and populate the `ProductData` without loading the `Product` item from the database, which improves the performance and scalability of the search functionality.

ⓘ Note

Ensure that the entire product data set required for display on the search results page be indexed into the Solr index, even if it is not required for search queries.

Available Facets

The `ProductSearchPageData` has a list of `FacetData`. Each `FacetData` has a list of `FacetValueDatas` which represents the available values within each of the available facets. The `FacetData` has a user-displayable name. The `FacetValueData` also has a user-displayable name and the facet count (the total number of results that would be returned should this facet value be applied as a filter). The facet values are sorted in the configured order.

The `FacetData` also has another list of `FacetValueDatas` in the `topValues` property. This list holds the top five facet values ordered by count. The number of held values is configurable. This top values list is not always populated. If the total number of facet values is less than the top facet count then the top values are not populated. If the facet is configured as a multi-select facet, the top values list is also not populated.

Filtering Results

Each `FacetValueData` has a `query` property holding a `ProductSearchPageData`, which is preconfigured with the required filters. To perform the same search query but also filter the results by a facet value, pass `SearchStateData` to the `ProductSearchFacade.textSearch()` method.

Search Breadcrumbs

`ProductSearchPageData` holds the list of search breadcrumbs. Search breadcrumbs are the filters that have been applied to the search in the order in which they were applied.

Each `BreadcrumbData` has a `removeQuery()` method which can be used to run a search with the specific applied filter removed. Each `BreadcrumbData` also has a `truncateQuery()` method which can be used to remove all the filters applied after the specified filter.

Storefront

SearchPageController

The `SearchPageController` controls initiation of free text search and the refinement of a previously-run search. Both of these are handled as GET requests and use bookmarkable URLs that the user can persist. Reloading one of these bookmarked URLs reruns the search.

`SearchPageController` also loads the appropriate CMS page, displaying managed content with the search results.

CategoryPageController

`CategoryPageController` decodes the URL path and identifies the relevant category. It then loads the CMS page for the category, which is used to display managed content on the category page. The controller uses the `ProductSearchFacade` either to initiate a new category search or to refine the existing search, depending on which query parameters are passed. Again, the URLs used are GET requests and are bookmarkable.

Example: Free Text Search URL and Search Refinements

1. Search for **shirt**.

   ```
   /search?text=shirt
   ```

2. Refine by category **Playboard**.

   ```
   /search?q=shirt:relevance:brand:Playboard
   ```

3. Refine by style **black**.

   ```
   /search?q=shirt:relevance:brand:Playboard:style:black
   ```

4. Refine by size **L**.

   ```
   /search?q=shirt:relevance:brand:Playboard:style:black:size:L
   ```

Example: Search Result Paging URLs

1. Search for **shirt**.

   ```
   /search?text=shirt
   ```

2. Navigate to the second page of results.

```
/search?q=shirt:relevance&page=1
```

i Note

The page request parameter is zero based; 0 is the first page of results, 1 is the second.

Example: Category URL and Category Filtering URLs

1. Navigate to the **Shirts** category.

   ```
   /Categories/Clothes/Shirts/c/shirts
   ```

2. Refine by style **red**.

   ```
   /Categories/Clothes/Shirts/c/shirts?q=:relevance:style:red
   ```

3. Refine by size **S**.

   ```
   /Categories/Clothes/Shirts/c/shirts?q=:relevance:style:red:size:S
   ```

SEO Directives

Meta tags provide structured metadata about a storefront page. You can use meta tags to specify various attributes of a page, such as page description and keywords. These tags are placed in the `head` section of the page. This document describes the kinds of pages that are indexed and how meta tags are implemented within SAP Commerce Accelerator.

Robots Attribute

Meta tags are used to help search engines categorize pages correctly. The `robots` attribute controls whether search engine spiders are allowed to index a page or not, and whether they are allowed to follow links from a page or not. For most use cases, you do not want to index pages that are created dynamically, such as shopping carts.

The default rules for the `robots` attribute in the SAP Commerce Accelerator are described in the table below:

Condition	Rules
The page is rendered in response to a`POST` rather than a `GET`.	NoIndex, NoFollow
The page is a secure `GET`.	NoIndex, Follow
The page is an insecure `GET`.	Index, Follow
Other conditions	NoIndex, NoFollow

These rules are configured in `SeoRobotsFollowBeforeViewHandler`, which is in `de.hybris.platform.yacceleratorstorefront.interceptors.beforeview`.

For individual pages, the `robots` attribute value is passed in by the corresponding controller.

The rules implemented in the controllers are described in the following table:

Page Type	Rules
All **My Account**, **My Company**, **Checkout** and order confirmation pages.	NoIndex, NoFollow
Login pages	Index, NoFollow
Category Browse page	Index, Follow

The rules implemented in the controllers for search results are described in the following table:

Search Results	Rules
If a non-default sort is applied, or a facet filter is applied, or it is not the first page of results.	NoIndex, Follow
On the Store Locator search	Index, Follow
If a Store Locator search term has been entered	NoIndex, Follow
On the search results page	NoIndex, Follow

The following code is used to set the `robots` attribute value property in the controller:

```
model.addAttribute("metaRobots", "noindex,nofollow");
```

SEO Content Optimization

Each page has the following attributes set to optimize SEO content:

- Metadata language is set to the current session language
- Page title
- Metadata description
- Metadata keywords

Default Page Titles

The default page titles are set as follows:

- Product Page: <product.name> | <product.primary_category_hierarchy_in_reverse_order> | <storefront.name>
- Category Page: <category.name> | <category.primary_category_hierarchy_in_reverse_order> | <storefront.name>
- Search Results Page: Search <search terms> | <storefront.name>
- Store Search Results Page: Stores near <search terms> | <storefront.name>
- Store Page: <store.name> | <storefront.name>
- Content Page: <contentpage.title> | <storefront.name>

You can refer to the `PageTitleResolver.java` file as an example.

Default Meta Descriptions

The default meta descriptions are set as follows:

- Product Page: <product.summary>
- Category Page: <category.description>
- Search Results Page: Search results for <search terms> on <storefront.name>
- Store Search Results Page: <storefront.name> Store Locations near to <search terms>
- Store Page: <store.description>
- Content Page: <contentPage.description>

You can refer to the `AbstractPageController.java` file as an example.

Default Meta Keywords

The default meta keywords are set as follows:

- Product Page: <productData.keywords>
- Category Page: <category.keywords>
- Search Results Page: <search terms>
- Store Search Results Page: <search terms>
- Store Page: <store.city> <store.postcode> <store.country>
- Content Page: <page.keywords>

You can refer to the `ProductPageController.java` file as an example.

Sanitizing Meta Data

There is a `MetaSanitizerUtil` tool available within the SAP Commerce Accelerator `yacceleratorstorefront` extension and the `yb2bacceleratorstorefront` extension. You can use it to sanitize keywords and descriptions, especially if the descriptions contain HTML code. The HTML is removed; it is not escaped.

Related Information

SEO URLs
SAP Commerce Accelerator SEO URL Tutorial

B2C Spring Security

SAP Commerce Accelerator is based on the Spring Framework. It uses the Inversion of Control (IoC) container to manage its components, as well as other frameworks that are built on top of the Spring Framework.

Spring Security

SAP Commerce Accelerator leverages the built-in Spring Security support of the SAP Commerce Platform. The main goal is to deliver a secure and reliable solution that follows best practices. The existing `CoreAuthenticationProvider` is used to provide a clean separation of the authentication from the authorization by allowing the user to configure access restrictions by means of security interceptors. Instead of relying on session fixation protection, an even more restrictive secure GUID cookie is introduced that offers the following functionality:

- After successful authentication, a GUID is generated and stored in a secure session cookie.
- For every secure request, the presence of this cookie is ensured by means of an interceptor.
- Exclusion URLs are configurable to allow accessing secure pages prior to authentication.

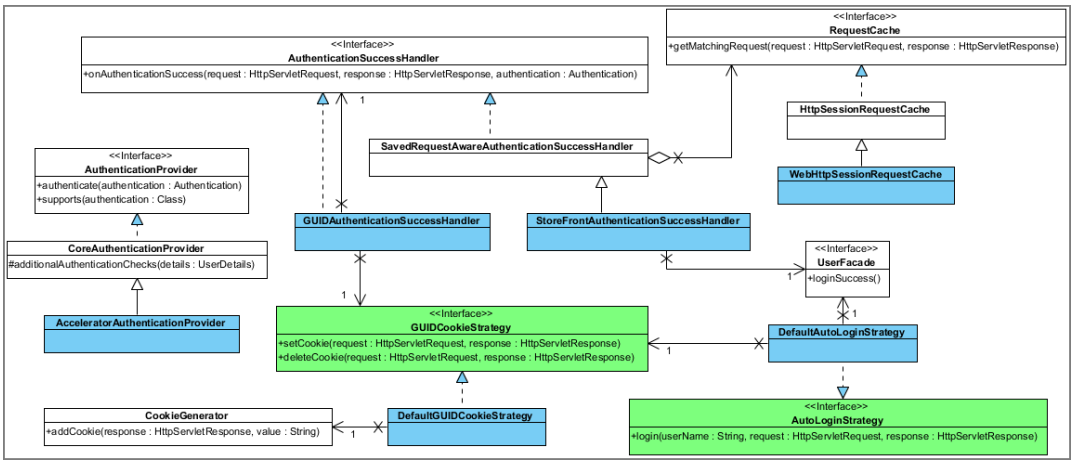


Figure: Classes relevant for the authentication.

During authentication, the following steps are processed:

1. The user is authenticated by the `AcceleratorAuthenticationProvider`, which overrides `additionalAuthenticationChecks`. This has the following purposes:
    - Preventing administrators from authentication in the SAP Commerce Accelerator storefront. This is required because `SearchRestrictions` do not apply for administrators, which leads to catalog version ambiguities.
    - Preventing users that have been created in the CS Cockpit without a password from authenticating.
2. If authentication is successful, the following steps are performed by the system:
    - The secure GUID cookie is added by the `GUIDAuthenticationSuccessHandler`.
    - The `StoreFrontAuthenticationSuccessHandler` is activated, delegating initialization of the JaloSession for the user to the `UserFacade`, and calling for cart restoration or merging, if necessary. For more information on cart merging, see Cart Merging.
    - Deep linking is supported by the `SavedRequestAwareAuthenticationSuccessHandler`.
    - `WebHttpSessionRequestCache` wraps the `SavedRequest` to supply the newly created GUID cookie to the `GUIDInterceptor`.

Automatic login is supported during registration. In this case, reusing the `AuthenticationSuccessHandler` is not possible due to the redirecting responsibility implied by Spring Security.

For more information, see http://static.springsource.org/spring-security/site/ : SpringSource Spring Security.

Configuration

Security-related configuration is contained in the `spring-security-config.xml` file, which is located in *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/config`. All pages are secured with HTTPS.

Redirection to a secure channel, as well as access restrictions, are maintained using the security namespace. The following is an example from the `spring-security-config.xml` file:

```
<security:intercept-url pattern="/my-account/**" access="hasRole('ROLE_CUSTOMERGROUP')" requires-channel="https" />
```

In this example, access to the My Account page and all subpages is granted to users in the `customergroup` only.

Static files should remain unrestricted. The following is an example from the `spring-security-config.xml` file:

```
<security:intercept-url pattern="/_ui/**" security="none" />
```

Disabling HTTPS Everywhere

By default, SAP Commerce Accelerator secures all storefront pages using HTTPS. Previous versions of the Accelerator only secured the Checkout and My Account pages using HTTPS. The following procedure describes how to revert to this "pre-5.5" configuration.

1. Open the `spring-security-config.xml` file, which is located in *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/config`, and go to the checkout section:

    ```
    <security:http disable-url-rewriting="true" pattern="/checkout/**" use-expressions="true">
            ......
            </security:http>
    ```

2. Remove the following security intercept:

    ```
    <security:intercept-url pattern="/**" requires-channel="https" />
    ```

3. In place of the previously removed security intercept, add the following:

    ```
    <security:intercept-url pattern="/events" requires-channel="https" />

    <security:intercept-url pattern="/checkout/j_spring_security_check" requires-channel="https" />

    <security:intercept-url pattern="/checkout*" requires-channel="https" />
    <security:intercept-url pattern="/checkout/multi*" requires-channel="https" />
    ```

```
<security:intercept-url pattern="/checkout/orderConfirmation/*" requires-channel="https" />
<security:intercept-url pattern="/checkout/multi/**" requires-channel="https" />
```

4. Go to the default section of the `spring-security-config.xml` file:

```
<security:http disable-url-rewriting="true"  request-matcher-ref="excludeUrlRequestMatcher" use-expressions="true">
        ......
        </security:http>
```

5. Remove the following security intercept:

```
<security:intercept-url pattern="/**" requires-channel="https" />
```

6. In place of the previously removed security intercept, add the following:

```
<security:intercept-url pattern="/events" requires-channel="https" />

                <!-- SSL / ANONYMOUS pages Login pages need to be SSL, but occur before authentication -->
                <security:intercept-url pattern="/login" requires-channel="https" />
                <security:intercept-url pattern="/login/**" requires-channel="https" />
                <security:intercept-url pattern="/register" requires-channel="https" />
                <security:intercept-url pattern="/register/**" requires-channel="https" />
                <security:intercept-url pattern="/j_spring_security_check" requires-channel="https" />
                <security:intercept-url pattern="/logout" requires-channel="https" />
                <security:intercept-url pattern="/guest/order/**" requires-channel="https" />

                <!-- MiniCart and CartPopup can occur on either secure or insecure pages -->
                <security:intercept-url pattern="/cart/rollover/*" requires-channel="any" />
                <security:intercept-url pattern="/cart/miniCart/*" requires-channel="any" />
                <security:intercept-url pattern="/search/autocomplete/*" requires-channel="any" />

                <security:intercept-url pattern="/search/autocompleteSecure/**" requires-channel="https" />

                <!-- OPEN / ANONYMOUS pages Run all other (public) pages openly. Note that while credentials are secure, the session id can be
                        If this is a security concern, then this line should be re-considered -->
                <security:intercept-url pattern="/**" requires-channel="any" method="POST" /> <!-- Allow posts on either secure or insecure -->
                <security:intercept-url pattern="/**" requires-channel="http" /> <!-- Everything else should be insecure -->
```

Next Steps

The `GUIDInterceptor` is configured in the `spring-mvc-config.xml` file, which is located in *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/config`: The following is an example:

```
<bean class="de.hybris.platform.yacceleratorstorefront.security.GUIDInterceptor" scope="tenant">
  <property name="redirectStrategy" ref="redirectStrategy"/>
  <property name="loginUrl" value="/login"/>
  <property name="excludeUrls">
    <set>
      <value>/</value>
      <value>/login</value>
      <value>/login/register</value>
      <value>/login/checkout</value>
      <value>/login/checkout/register</value>
      <value>/login/pw/request</value>
      <value>/login/pw/change</value>
      <value>/s/slang</value>
      <value>/s/scurrency</value>
      <value>/search</value>
      <value>/rolloverCartPopup/MiniCart</value>
    </set>
  </property>
  <property name="cookieGenerator" ref="guidCookieGenerator"/>
  <property name="urlPathHelper" ref="urlPathHelper"/>
</bean>
```

For paths contained in the `excludeUrls` property, the existence of the GUID cookie is not enforced. All remaining paths trigger a redirect to the `loginUrl` if called on a secure channel without the GUID cookie set.

In keeping with security best practices, secure cookies are implemented in the Accelerator. When reverting to HTTP, you should revert the secure cookies in the `spring-security-config.xml` file, as well as any declared cookie generators defined in the `spring-filter-config.xml` file.

The following is an example of how the `spring-security-config.xml` file should look when reverting to HTTP:

```
<alias name="defaultRememberMeServices" alias="rememberMeServices"/>
        <bean id="defaultRememberMeServices" class="de.hybris.platform.yacceleratorstorefront.security.AcceleratorRememberMeServices" >
                <property name="userDetailsService" ref="originalUidUserDetailsService" />
                <property name="key" value="yacceleratorstorefront" />
                <property name="cookieName" value="yacceleratorstorefrontRememberMe" />
                <property name="alwaysRemember" value="true" />
                <property name="userService" ref="userService"/>
                <property name="useSecureCookie" value="false"/>
                <property name="customerFacade" ref="customerFacade"/>
                <property name="checkoutCustomerStrategy" ref="checkoutCustomerStrategy"/>
                <property name="urlEncoderService" ref="urlEncoderService"/>
                <property name="storeSessionFacade" ref="storeSessionFacade"/>
                <property name="commonI18NService" ref="commonI18NService"/>
                <property name="secureTokenService" ref="secureTokenService"/>
        </bean>
```

The following is an example of how the `spring-filter-config.xml` file should look when reverting to HTTP:

```
<alias name="defaultSessionCookieGenerator" alias="sessionCookieGenerator"/>
        <bean id="defaultSessionCookieGenerator" class="de.hybris.platform.yacceleratorstorefront.security.cookie.EnhancedCookieGenerator" >
                <property name="cookieSecure" value="false"/>
                <property name="cookieName" value="JSESSIONID"/>
                <property name="cookieMaxAge" value="-1"/>
                <property name="useDefaultPath" value="false"/>
                <property name="httpOnly" value="true"/>
        </bean>

        <alias name="defaultCartRestoreCookieGenerator" alias="cartRestoreCookieGenerator"/>
        <bean id="defaultCartRestoreCookieGenerator" class="de.hybris.platform.yacceleratorstorefront.security.cookie.CartRestoreCookieGenerator" >
                <property name="cookieSecure" value="false"/>
                <property name="cookieMaxAge" value="360000000"/>
                <property name="useDefaultPath" value="false"/>
                <property name="httpOnly" value="true"/>
                <property name="baseSiteService" ref="baseSiteService"/>
        </bean>

        <alias name="defaultCustomerLocationCookieGenerator" alias="customerLocationCookieGenerator"/>
        <bean id="defaultCustomerLocationCookieGenerator" class="de.hybris.platform.yacceleratorstorefront.security.cookie.CustomerLocationCookieGen
                <property name="cookieSecure" value="false"/>
                <property name="cookieMaxAge" value="360000000"/>
                <property name="useDefaultPath" value="false"/>
                <property name="httpOnly" value="true"/>
                <property name="baseSiteService" ref="baseSiteService"/>
        </bean>
```

Remember Me

The Accelerator storefronts support Remember Me Authentication (Soft Login), which automatically logs a customer into the storefront based on a cookie. This uses the Spring `TokenBasedRememberMeServices` implementation. Your own implementation only needs to redeclare the `rememberMeServices` bean to be integrated into the process.

Using Spring Remember Me Authentication allows the Accelerator to make use of role-based authentication tags. As a result, certain links can be easily hidden or shown based on the user login state.

ℹ Note

A Remembered (that is, Soft Authenticated) customer needs to provide a password and log in fully to access the Account or to proceed through Checkout.

For more information, see the following:

- Redirecting to a Requested URL After Authentication
- Spring Security Remember-Me Authentication: http://static.springsource.org/.../remember-me.html

Cross-Site Request Forgery

Cross-site request forgery (also known as XSRF, CSRF, and cross-site reference forgery) works by exploiting the trust that a site has for a user. Site tasks are usually linked to specific URLs that allow specific actions to be performed when requested (for example, `http://site/stocks?buy=100&stock=ebay`). If a user is logged into the site and an attacker tricks the browser into making a request to one of these task URLs, then the task is performed and logged as though performed by the logged in user. Typically, an attacker embeds malicious HTML or JavaScript code into an email or website that requests a specific "task URL", which executes without the user's knowledge, either directly or by using a cross-site scripting flaw.

Content Injection Attacks

Add the applicable headers in `project.properties` or `local.properties` to improve security against possible content injection attacks.

Below is a code sample that contains properties for web security settings. The values shown are only examples. You can decide which values work best for improving your security in this case.

```
xss.filter.header.X-Frame-Options=DENY
xss.filter.header.X-XSS-Protection=1; mode=block
xss.filter.header.Strict-Transport-Security=max-age=31536000; includeSubDomains
xss.filter.header.Content-Security-Policy=default-src 'self'
```

Spring Configuration

This implementation of CSRF uses Spring Security's CSRF.

The CSRF token property name is `CSRFToken`.

A bean `csrfTokenRepository` is defined by specifying the `CsrfToken` `headerName` and `parameterName` as `CSRFToken`, as shown in the following example from the `yacceleratorstorefront/web/webroot/WEB-INF/config/spring-security-config.xml` file.

```
<bean id="csrfTokenRepository" class="org.springframework.security.web.csrf.HttpSessionCsrfTokenRepository">
  <property name="headerName" value="CSRFToken" />
  <property name="parameterName" value="CSRFToken" />
</bean>
```

The property names `csrf.allowed.url.patterns` and `csrfAllowedUrlPatternsList` specify URL patterns that allow you to bypass CSRF token validation.

The `de.hybris.platform.yacceleratorstorefront.security.CsrfProtectionMatcher` class supports this functionality by returning `true` for the `POST` method when the request's servlet path does not match the combined list of `csrf.allowed.url.patterns` from the properties file or the `csrfAllowedUrlPatterns` from the Spring configuration. The following is an example from the `yacceleratorstorefront/web/webroot/WEB-INF/config/spring-security-config.xml` file.

```
<alias name="defaultCsrfProtectionMatcher" alias="csrfProtectionMatcher"/>
<bean id="defaultCsrfProtectionMatcher" class="de.hybris.platform.yacceleratorstorefront.security.CsrfProtectionMatcher">
```

```
  <property name="csrfAllowedUrlPatterns" ref="csrfAllowedUrlPatternsList"/>
</bean>
```

The `csrfProtectionMatcher` and `csrfTokenRepository` are passed to the Spring Security configuration to override the default Spring Security CSRF behavior, as shown in the following example from the `yacceleratorstorefront/web/webroot/WEB-INF/config/spring-security-config.xml` file.

```
<security:csrf token-repository-ref="csrfTokenRepository" request-matcher-ref="csrfProtectionMatcher" />
```

As of Spring Security 4.0, CSRF protection is enabled by default with XML configuration. To disable CSRF protection, for example if exclusively non-browser clients use the system, add the following to the corresponding XML configuration.

```
<security:csrf disabled="true"/>
```

Ajax Support

Cross-site request forgery protection is also provided for Ajax POST requests. If you use `$.post`, `$.ajax` or `$.postJSON`, a `CSRFToken` parameter is automatically added to the data submitted to the server.

Redirecting to a Requested URL After Authentication

If a registered user returns to a page they previously visited in your storefront, and they are asked to log in again, you can ensure the requested page is loaded after authentication, rather than loading the default target URL (such as the homepage).

Context

The Accelerator storefront provides Remember Me services based on the Spring Web Framework. When a registered user logs in to the storefront, a cookie is created by the Remember Me services to identify the user. When the user logs out, this cookie is cleared out. However, if a user closes the browser without logging out, the Remember Me cookie that identifies the registered user is not cleared out.

In `yacceleratorstorefront`, the `StorefrontAuthenticationSuccessHandler` is responsible for post-processing after a user successfully authenticates. The `StorefrontAuthenticationSuccessHandler` extends the Spring class `SavedRequestAwareAuthenticationSuccessHandler`. The `SavedRequestAwareAuthenticationSuccessHandler` is an authentication success strategy which can make use of the `DefaultSavedRequest`, which may have been stored in the session by the `ExceptionTranslationFilter`.

When a request is made for a secure resource that requires authentication, Spring Security throws an `AccessDeniedException`. This exception is caught by the `ExceptionTranslationFilter`, which saves the request data (`DefaultSavedRequest`) in the `HttpSessionRequestCache`. By default, the `StorefrontAuthenticationSuccessHandler` makes use of the `DefaultSavedRequest` to redirect the user to the requested URL.

However, if an authenticated user closes their browser without logging out, and the user attempts to visit a specific page in the storefront again, when the request is made for this secure resource (or any other page in the storefront), the `RememberMeAuthenticationFilter` injects the `RememberMeAuthenticationToken` into the session when loading the storefront. This causes issues with the `StorefrontAuthenticationSuccessHandler` because when there is an `AuthenticationToken` present in the session, the `AccessDeniedException` is never thrown, and the request data is not stored in the `HttpSessionRequestCache` by the `ExceptionTranslationFilter`. Since the requested URL is not available, the user is redirected to the default target URL (such as the homepage) by the `StorefrontAuthenticationSuccessHandler`.

You can ensure the user is redirected to the requested URL by saving the request data in the `HttpSessionRequestCache` of the `RequireHardLoginBeforeControllerHandler.beforeController` method. If you are using B2B Accelerator, you also need to modify the `defaultB2BLoginAuthenticationSuccessHandler` bean in `b2bacceleratoraddon-spring-security-config.xml`.

Procedure

1. Open `RequireHardLoginBeforeControllerHandler.java`.
2. Modify lines 17 and 18 of the `RequireHardLoginBeforeControllerHandler.beforeController` method as follows:

```
@Override
                                        public boolean beforeController(final HttpServletRequest request, final HttpServletResponse res
                                        final HandlerMethod handler) throws Exception
                                        {
                                        // We only care if the request is secure
                                        if (request.isSecure())
                                        {
                                        // Check if the handler has our annotation
                                        final RequireHardLogIn annotation = findAnnotation(handler, RequireHardLogIn.class);
                                        if (annotation != null)
                                        {
                                        final boolean redirect = requireHardLoginEvaluator.evaluate(request, response);

                                        if (redirect)
                                        {
                                        // REQUIRED CHANGE START
                                        final RequestCache requestCache = new HttpSessionRequestCache();
                                        requestCache.saveRequest(request, response);
                                        // REQUIRED CHANGE END
                                        LOG.warn("Redirection required");
                                        getRedirectStrategy().sendRedirect(request, response, getRedirectUrl(request));
                                        return false;
                                        }
                                        }
                                        }

                                        return true;
                                        }
```

   Only lines 17 and 18 need to be added to the method, as indicated by the `REQUIRED CHANGE START` and `REQUIRED CHANGE END` comments in the code example.

3. If you are using B2B Accelerator, open `/b2bacceleratoraddon/resources/b2bacceleratoraddon/web/spring/b2bacceleratoraddon-spring-security-config.xml`

Note

If you are using B2C Accelerator, you do not need to follow these final steps.

4. In the `defaultB2BLoginAuthenticationSuccessHandler` Spring bean, set the `useReferer` property to `true`. The following is an example:

```
<alias name="defaultB2BLoginAuthenticationSuccessHandler" alias="loginAuthenticationSuccessHandler"/>
                                                <bean id="defaultB2BLoginAuthenticationSuccessHandler"
                                                class="de.hybris.platform.acceleratorstorefrontcommons.security.StorefrontAuthenticationSuccess
                                                <property name="customerFacade" ref="customerFacade"/>
                                                <property name="defaultTargetUrl" value="#{'responsive' == '${commerceservices.default.desktop.
                                                <property name="useReferer" value="true"/> <!-- REQUIRED CHANGE. Value used to be false. Should
                                                <property name="requestCache" ref="httpSessionRequestCache"/>
                                                <property name="uiExperienceService" ref="uiExperienceService"/>
                                                <property name="cartFacade" ref="cartFacade"/>
                                                <property name="customerConsentDataStrategy" ref="customerConsentDataStrategy"/>
                                                <property name="adminGroup" value="ROLE_ADMINGROUP"/>
                                                <property name="forceDefaultTargetForUiExperienceLevel">
                                                <map key-type="de.hybris.platform.commerceservices.enums.UiExperienceLevel" value-type="java.la
                                                </property>
                                                <property name="restrictedPages" ref="loginSuccessRestrictedPages"/>
                                                <property name="listRedirectUrlsForceDefaultTarget" ref="loginSuccessForceDefaultTarget"/>
                                                <property name="bruteForceAttackCounter" ref="bruteForceAttackCounter"/>
                                                <property name="cartRestorationStrategy" ref="cartRestorationStrategy"/>
                                                </bean>
```

Related Information

B2C Spring Security
https://docs.spring.io/spring-security/site/docs/4.2.4.RELEASE/apidocs/org/springframework/security/web/authentication/SavedRequestAwareAuthenticationSuccessHandler.html
https://docs.spring.io/spring-security/site/docs/4.2.4.RELEASE/apidocs/org/springframework/security/web/savedrequest/DefaultSavedRequest.html
https://docs.spring.io/spring-security/site/docs/4.2.4.RELEASE/apidocs/org/springframework/security/web/access/ExceptionTranslationFilter.html

Spring Usage

The SAP Commerce Accelerator, like the SAP Commerce, is based on the Spring Framework. It not only uses the Inversion of Control(IoC) container to manage its components, but also other frameworks that are built on top of the Spring Framework. This document provides an overview of which frameworks the SAP Commerce Accelerator uses.

Using Spring as IoC Container

The Spring Framework is used to manage components like SAP Commerce services, facades, MVC controllers and so on. They IoC container is responsible for creating these components and wiring the interdependencies. In this regard the SAP Commerce Accelerator is no different than the rest of the SAP Commerce. Please refer to the Spring Framework in the SAP Commerce document for information about how in the application context is set up, where to find the bean definitions and which pattern and practices are used.

For more information, see the following:

- commerceservices Extension - Technical Guide
- commercefacades Extension - Technical Guide
- yacceleratorfacades Extension
- ServiceLayer
- Using Facades and DTOs - Best Practice

Spring MVC

In the **yacceleratorstorefront** extension the front-end logic is implemented using Spring MVC. It follows the patterns explained in the Spring Framework in the SAP Commerce document, section **Using Spring MVC**. The Spring MVC related configuration files are located in the `web/webroot/WEB-INF/config` folder of the **yacceleratorstorefront** extension. The controllers themselves are configured by annotation as per SAP Commerce conventions.

For more information, see the following:

- yacceleratorstorefront Extension
- http://static.springsource.org/spring/.../mvc.html : Web MVC framework

Spring Integration

A hot folder import capability is built into the SAP Commerce Accelerator. It uses the Spring Integration framework to implement Enterprise Integration Patterns, like watching the hot folder and service activators to invoke the necessary ImpEx import infrastructure.

For more information, see the following:

- Data Importing
- Event System
- http://www.springsource.org/spring-integration : Spring Integration

Spring Security

Spring security-related issues are described in B2C Spring Security.

Related Information

Spring Framework in SAP Commerce

Stock Management

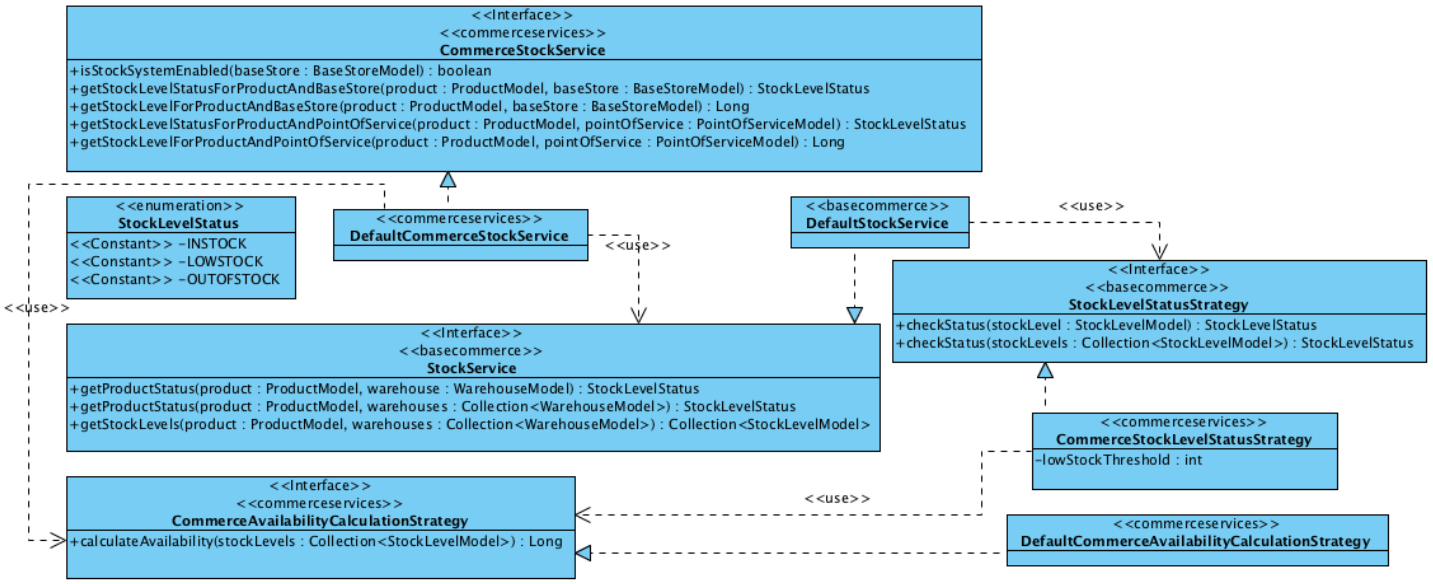
The handling of Product SKU Inventory, often referred to as **stock**, is an important part of the SAP Commerce Accelerator. The Accelerator maintains stock levels at both the BaseStore and PointOfService level. This makes it possible for project implementations to expose both **Delivery** and **Pickup in Store** purchasing options and differentiate stock levels accordingly.

The bulk of the stock management functionality is provided by the **basecommerce** and **commerceservices** extensions and is exposed through the **commerceservices** extension's **CommerceStockService**.

The Accelerator supports the exposure of product stock information on both **Search Result** and **Category** pages, as well as the **Product Detail** page. Stock levels are verified each time a customer adds a quantity of a product to their cart.

Commerce Stock Service

A product's stock level information is exposed through the **CommerceStockService**. Using this service, you can determine a product's Availability To Promise stock level and obtain a more coarse-grained stock status. This service provides stock data for pick up at a **PointOfService** or for delivery.



The **DefaultCommerceStockService** is a default implementation of the **CommerceStockService**. It delegates processing to other services and strategies as follows:

- **StockService**: Resolves stock levels for a SKU and applicable warehouses
- **StockLevelStatusStrategy**: Returns an aggregated **StockStatus** for 0 or more stock levels.
- **CommerceAvailabilityCalculationStrategy**: Calculates availability.

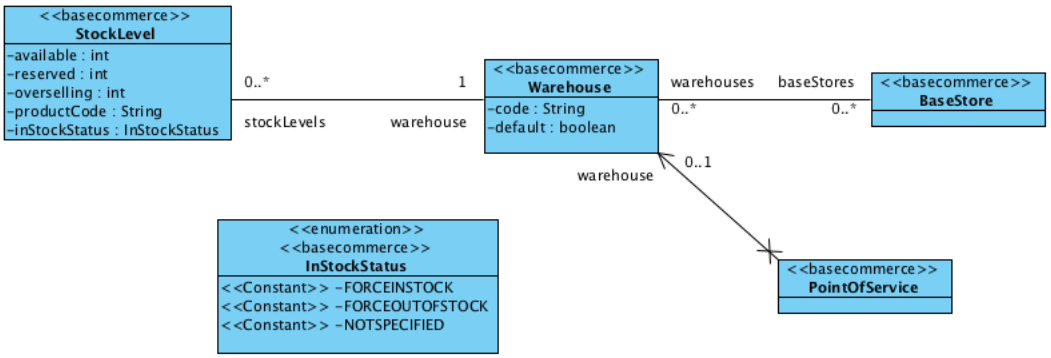
Enabling Stock Functionality

- To enable ***Web Stock,*** link a **Warehouse** to the website's **BaseStore**
- To enable ***In Store Stock,*** link **Warehouses** to the **PointOfService** of type **STORE** that are linked to the website's **BaseStore.**

Stock Level

The **StockLevel** item captures product stock information. It is a generic stock object provided by the **basecommerce** extension. The **StockLevel** provides numerous options. The Accelerator does not currently use all options, therefore some fields are not used or are not populated.

- **StockLevelProductStrategy** retrieves the Stock Keeping Unit (SKU) from the Product. By default this is the **code** of the product.
- **commerceservices DefaultCommerceAvailabilityCalculationStrategy** calculatesavailability.
- **Warehouses** are logical entities used to group stock in a single location.
- **Warehouses** linked directly to the **BaseStore** are used as sources for ***Web Stock***.
- **Warehouses** linked to **PointOfService** are used as sources of ***In Store Stock***.



Stock Status

The **StockLevelStatus** performs an initial check to verify if there is stock for the product. The default indicators are In Stock, Low Stock, and Out Of Stock. The low stock threshold is set in the **CommerceStockLevelStatusStrategy**. If a **StockLevel** linked to an SKU and applicable warehouse has an **InStockStatus** of **FORCEINSTOCK**, the stock status is InStock. If a **StockLevel** has an **InStockStatus** of **FORCEOUTOFSTOCK**, the availability for the stock level is set to 0. The **StockStatus** is used in cases where it is more efficient to have an approximate stock status, than to keep track of the actual value which might change frequently.

Stock in SOLR

Stock is a key facet when a user is browsing for products. Therefore, a series of stock-related attributes and facets can be exported into the SOLR Index using **SolrFacetSearch** Value Providers.

Value Providers

The following table lists the Value Providers provided out-of-the-box to export various stock aspects.

Value Provider	Extension	Usage	Type	Default Attribute Name
ProductStockLevelStatusValueProvider	**commerceservices**	Reduces the need to refresh product data in the Solr index when the Web ATS changes. It is recommended to only store the **StockStatus** in the Solr index. This provider calculates the **StockStatus** for a product and stores it in the index. It uses the **CommerceStockService**.	Property	**stockLevelStatus**
ProductInStockFlagValueProvider	**commerceservices**	Indicates if the product has a Web ATS or not. It uses the **CommerceStockService**.	Property	**inStockFlag**
ProductPickupAvailabilityValueProvider	**acceleratorservices**	Indicates if the product is available for pickup or not. It uses the **PickupAvailabilityStrategy**.	Facet	**pickupAvailableFlag**
ProductStoreStockValueProvider	**commerceservices**	Exports the list of **PointOfService** IDs that have stock. The SAP Commerce Accelerator for B2C uses this value provider to display the **Nearby Stores** facet.	Property	**availableInStores**

Performance Considerations of Maintaining Stock in the SOLR Index

The availability of popular products can fluctuate frequently on high-volume sites. This can result in the volume of index updates for a SKU not being optimal. By default, the SAP Commerce Accelerator only pushes the **StockLevelStatus** and an **InStock** flag to the Solr Index. This is usually sufficient for most projects, since this allows a stock message to be displayed and enables buttons to be enabled or disabled based on the stock level. However, if a project must store an actual availability value for a product in the index, this must be implemented by the project. Currently the **Search Result** and **Category** pages of the SAP Commerce Accelerator do not expect an availability value.

Pushing Stock Updates to the SOLR Index

Stock updates will cause the SOLR Indexer Update job to pick up the product as modified as update the index. This does not pay attention to the "previous" state, so the product will be updated even if the product's availability has not crossed any "threshold" between low stock and in stock, for example.

Nearby Stores Facet

By default, a facet created with the above-mentioned **ProductStoreStockValueProvider** lists all **PointOfService** names that have stock for at least one of the products visible in the current navigational state. However, this can compromise usability, therefore the SAP Commerce Accelerator for B2C provides an additional SOLR Indexed Property Configuration and customized front-end handling of this facet to improve usability.

Some of this functionality is bespoke front-end handling of this facet by the SAP Commerce B2C Accelerator and where this is the case is highlighted.

Activated By Customer Identifying Location

The Accelerator's **yacceleratorstorefront** suppresses the Nearby Stores facet on the front-end until the user has identified their location. Other front-ends display the facet unless similar logic is implemented.

The **CustomerLocationService** is used to save a user's geoposition and the optional search term that was used to identify the geo position. The Nearby Stores facet shows stores if a **UserLocation** is available. If a **UserLocation** is not available, the front-end prompts the user to enter a location or identify the user's position with a geolocation button.

Facet Value Sorting

The **PointOfService** values in the SAP Commerce Accelerator are sorted by distance based on proximity to the user's location. This functionality is provided by the configured **distanceAttributeSortProvider** bean. This provider uses the **LocalStorePreferenceService** to identify the nearest stores to the user's location and sorts the results by distance accordingly.

Stock in the Cart

The **CommerceCartService** performs stock checking when:

- A product is added to a cart.
- A product's quantity in the cart is updated.
- When the Checkout is entered.

If the requested quantity cannot be fulfilled, the **CommerceCartService** adds the remaining available stock returning a **CommerceCartModification** identifying that quantity added vs the quantity requested.

Related Information

basecommerce Extension
Stock Service
Warehouse Integration

Store Locator Configuration

The SAP Commerce Store Locator helps consumers to find stores in the proximity of a postal code, city name, or by using Global Positioning System (GPS) information. The Store Locator can also provide directions to the selected store. From a more technical point of view, the Store Locator functionality provides services that can work with standard online interfaces, as well as with mobile solutions.

Adding Points of Sale (POS) to a Base Store

Each base store object that represents a store in the system contains a collection of the points of sale. A point of sale is a single location, with a geographical representation of the store. The point of sale contains several pieces of information, including descriptions, images, a geographical position, and an address. A point of sale can be added to the base store using the Backoffice Administration Cockpit, which is a quick method to add or edit a small amount of data. For larger amounts of data, using an ImpEx file is a better choice because it is more efficient.

For information on how to use ImpEx, see Using ImpEx with Backoffice or SAP Commerce Administration Console.

Adding a POS Using Backoffice

To add a point of sale using Backoffice:

1. Log into Backoffice (`https://localhost:9002/backoffice`).
2. Navigate to **Base Commerce** > **Point of Service**.
3. Click **+ Point of Service** in the toolbar. The Create New Point of Service form appears.
4. Enter the POS **Name** and select **Store** for the **Type** so that the POS appears in the Store Locator.
5. Click **Done**.
6. To add additional information, click the newly created POS to open its details.
7. In the **Location** tab, enter any required information, such as the **Address**, **Opening hours**, and **Warehouses**.
8. To add the POS to a storefront, select the **Base Store** in the **Administration** tab.
9. Click **Save**.

Adding a POS Using ImpEx

It is possible to add new points of sale by using an ImpEx script that contains all of the required information for creating a point of sale in the system. The following is an example script that imports two sample points of sale, populates them with the addresses, and then assigns them to a specific store with the `basestore` label:

```
INSERT_UPDATE PointOfService;name[unique=true];type(code);address(&addrID);latitude;longitude;geocodeTimestamp[dateformat=dd-MM-yyyy]
;Hybris Electronics - Nakano;STORE;addr1;35,7091;139,6732;29-04-2011
;Hybris Electronics - Shinbashi;STORE;addr2;35,66730;139,75429;29-04-2011
;
INSERT_UPDATE Address;&addrID;streetname;streetnumber;postalcode[unique=true];town[unique=true];country(isocode);owner(PointOfService.name)[unique=t
;addr1;Waseda Dori;13;;Tokio;jp;Hybris Electronics - Nakano
;addr2;Hibiya Dori;20;;Tokio;jp;Hybris Electronics - Shinbashi
;
INSERT_UPDATE PointOfService;name[unique=true];basestore(uid)
;Hybris Electronics - Nakano;electronics
;Hybris Electronics - Shinbashi;electronics
```

If you do not provide geographic coordinates, make sure they are calculated manually, or with a geolocation cron job. Based on the information stored in the system, the locations of the points of service and their distances are calculated, and then displayed on the front end pages. Distance units can also be localized. For example, kilometers can be configured as miles instead.

For more information, see ImpEx Syntax.

Store Content

There are some attributes for each point of sale that are visible on the front end and that can be configured. This section contains a description of these attributes.

Picture

Each point of sale can contain two types of graphics. The first image is represented by the `mapIcon` attribute and is displayed on the Store Locator map of the found results. The second image is a `storeImage` graphic, which is a media container that contains the store images in different formats, and which are displayed on the Search Result List page or on the Store Details page. If no store image is provided, the default **Coming Soon** image is displayed.

Opening Times

A point of sale object also contains the `openingSchedule` attribute. This attribute provides information about the store's specific hours of operation, including special days, such as weekends or holidays. This information is displayed on both the Search Result List page and on the Store Details page.

For more information, see Store Locator Implementation.

The following is a sample ImpEx script that creates weekday openings for Monday and Tuesday:

```
$standardHours=standard-hours

INSERT_UPDATE OpeningSchedule;code[unique=true];
;$standardHours;

INSERT_UPDATE WeekdayOpeningDay;closingTime[dateformat=hh:mm];dayOfWeek(code,itemtype(code))[unique=true];openingTime[dateformat=hh:mm];openingSched
;20:00;MONDAY:WeekDay;08:00;$standardHours;
;20:00;TUESDAY:WeekDay;08:00;$standardHours;
```

The following is a sample ImpEx script that creates a special opening day on 01.01.2011:

```
INSERT_UPDATE  SpecialOpeningDay;date[unique=true,dateformat=dd.MM.yyyy];message[lang=en];closed;openingSchedule(code)[unique=true];closingTime[date
;01.01.2011;;true;$standardHours;;;
```

The following is a sample ImpEx script that assigns a schedule to a `PointOfService`:

```
INSERT_UPDATE PointOfService;name[unique=true];basestore(uid);openingSchedule(code);
;Hybris Electronics - Nakano;electronics;$standardHours;
```

Features

Another visible attribute is `features`, which holds a collection of store features. A store `feature` is a dynamic enumeration value, and you can add a new `feature` to the system. A `feature` holds store-specific information that you can reuse for other stores as well. An example would be a store that is open on Sundays, or a store that has wheelchair access. This information is displayed only on the Store Details page.

Store-Specific Content

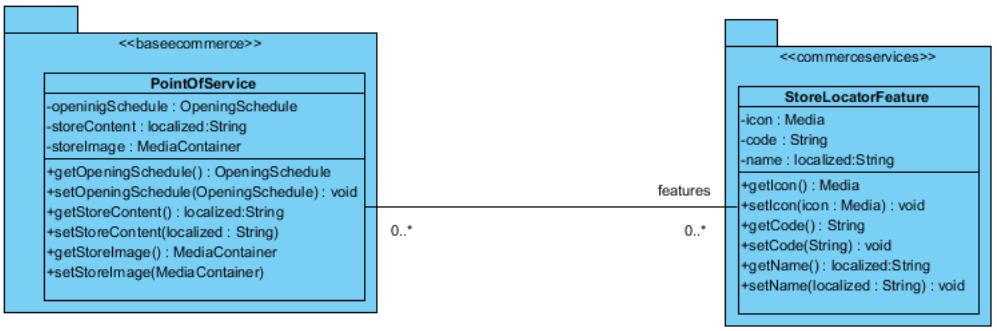
The `storeContent` attribute also holds descriptive content about a specific store. This attribute is localized, and its value is displayed only on the Store Details page.

Address

An important attribute for each point of sale is the store `address`. An address should be formatted in a way that enables the geolocation service to determine the geographical coordinates. It is possible to put latitude and longitude directly in the point of sale using an ImpEx script. In all other cases, these values cannot be edited, and are determined automatically basing on the information that is placed in the point of sale `address`.

Extended Data Model

The following is a data model diagram for the Store Locator:



Storefront API Key Configuration

The SAP Commerce Accelerator project is ready to maintain multiple stores, and each store can operate at a different address. This requires the Google Maps API key to be configurable because each key is valid for only one URL. The project contains a `GoogleAPIKeyInterceptor` function that is able to hold many references to different API keys, and that matches the proper API key for each URL. However, to work properly, it must be configured using the platform's `project.properties` file. This file can contain multiple entries for different registered URL addresses. The data is represented as key to value pairs. The Google API key in the properties file should start with `googleApiKey.`, followed by the URL that this particular key matches. The following are examples of entries in the `project.properties` file for generating Google API keys:

```
googleApiKey.apparel.uk.local=ABQIAAAAv(...)4LmJ7bMEeRQ
googleApiKey.apparel.de.local=ABQIAAAAv(...)Ma6XGgaJvyQ
```

→ Tip

With Google Maps API version 3, it is possible to have one generated Google API key for all URLs. However, it is still supported to have a different key for each specific URL or domain pattern.

You can customize the Google Maps API URL by providing a custom API version: `googleApiVersion`. By default, the value is set up for each storefront as follows:

```
googleApiVersion=3.7
```

For more information, see http://code.google.com/apis/maps/signup.html : Google API keys.

Setting Site Distance Unit

The Store Locator distance unit is held in the `BaseStore` instance as an additional attribute, so no additional property keys are needed.

In the Store Locator Search Results and Store Details pages, the distance between the point of service and the searched location is displayed. Depending on the actual address of the point of sale, the distance can be displayed in different units. Currently, two different unit systems are supported: metric and imperial. You can assign each country ISO code to be displayed in a specific unit system using the `project.properties` file. The following are example entries for setting the site distance unit in the `project.properties` file:

```
acceleratorstorefront.imperial.isocodes=us,gb
acceleratorstorefront.metric.isocodes=de,ja
```

The distances of countries with the ISO codes listed for `acceleratorstorefront.imperial.isocodes` properties are displayed with imperial units, and countries with the ISO codes listed for `acceleratorstorefront.metric.isocodes` properties are displayed with metric units. Distances in the system are always calculated using the metric system, and the actual ratio for an imperial system is hardcoded in the `DistanceHelper` helper class.

CMS Store Locator Pages

The information in the Store Locator Search Results and Store Details pages is placed into the Content Management System (CMS) page context. Aside from the Store Locator-related dynamic content, the displayed page also contain several other elements that need to be configured in the CMS. For both Store Locator-related pages, the CMS page with the `storefinderPage` UID is used. To display Store Locator pages correctly, there must be a page with this UID present in the content catalog version that is used for the current website. The template for this page is called `StoreFinderPageTemplate`, and it contains all the content slots that can be filled with the required data using the `cmscockpit` extension. Most of the content slots are filled by default with data such as `SiteLogo`, `HeaderLinks`, `MiniCart`, `NavigationBar`, and `Footer`, but are easily edited and adapted to your required specifications. Also, the top and side content containing commercial banners can be updated to display the desired content.

For more information, see Setting Up a Basic Web Site in the Backoffice.

Geolocation Cron Job

During system initialization, or the updating of a page, you can select a check box that creates a geocoding cron job. When this cron job is created, it periodically traverses all points of service and tries to determine the actual geographical coordinates for the entered addresses. When the coordinates are determined, they are then used for determining distances between searched locations and points of sale.

For more information, see the **Adjusting Geocoding Cron Job** section of Using Store Locator Service.

Related Information

basecommerce Extension
Store Locator Implementation

Storefront and Catalog Modeling

Learn the basics of how SAP Commerce Accelerator and its sample stores are configured and modeled with respect to the storefront setup and distribution of customer data, as well as product and content catalogs between multiple storefronts. A multiple storefront setup is a single SAP Commerce deployment that serves multiple web sites, rather than separate SAP Commerce deployments for each web site.

This document focuses on the configuration and modeling aspects used by the sample storefronts that ship with the following extensions:

- The `apparelstore` AddOn extension
- The `electronicsstore` AddOn extension

These configuration aspects are the components in the SAP Commerce data model that can be tweaked to alter a storefront content and behavior. As with other SAP Commerce modules, it is possible to customize and adapt all of this, however SAP Commerce Accelerator is designed to meet more typical B2C and B2B storefront requirements.

The `apparelstore` and `electronicsstore` AddOn extensions as well as the `yb2bacceleratorcore` extension contain the scripts and data required to create the sample stores in an empty state. Moreover, they contain all the content and product catalog data, that is products, pages, media and so on, and to provide a great reference for developers.

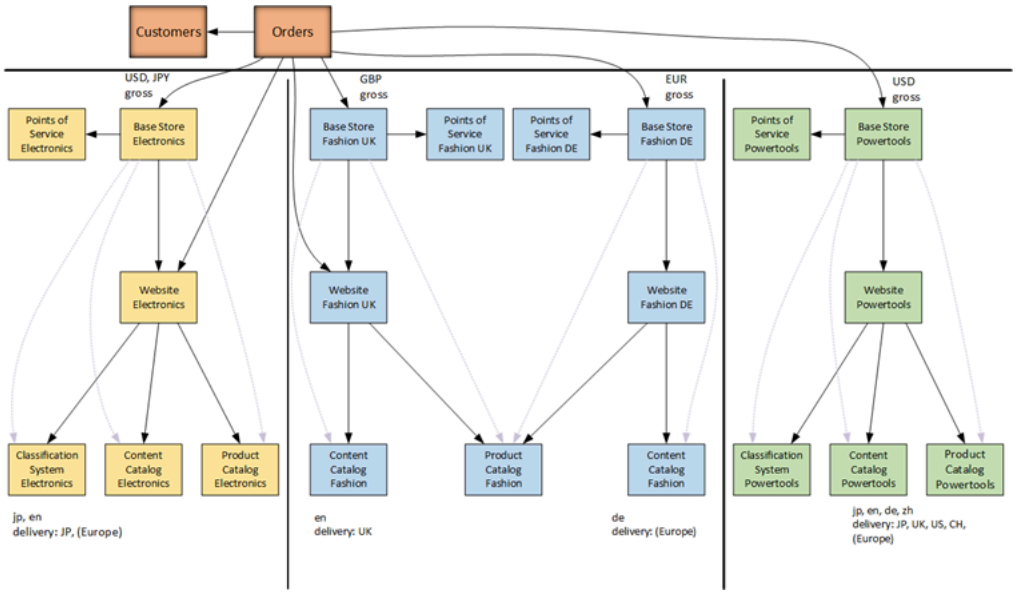
The `yacceleratorinitialdata` and `yb2bacceleratorinitialdata`, after modulegen, hold a preconfigured structure for a sample site. This should be extended rather than copying the ImpEx from the extensions containing sample data.

A storefront provides a view of a particular CMSSite and BaseStore. The URL used to access the site drives the selection of storefront, CMSSite and BaseStore.

For more information, see commerceservices Extension.

Sample Storefronts Diagram

Let us start with a high-level diagram that visualizes how the sample data storefront and customer data is separated.



CMSSite

A CMS site configuration options include:

- The BaseStore for the storefront.
- URL pattern regular expressions that enable the `yacceleratorstorefront` web application to decide which CMSSite and BaseStore to use to fulfill the request.
- The content catalog for the WCMS content. Usually one content catalog serves just one CMSSite, but it is possible to reuse the content catalog for multiple CMSSites.
- The product catalog, again the same catalog of products, categories, promotions reviews, and so on, could be used in multiple base stores or CMSSites.
- The optional classification catalog, it is helpful to have one classification catalog per product catalog if using more than one.
- The theme, which selects the CSS and message bundles for the storefront.
- The homepage for the storefront.
- The corresponding system Java locale when a user selects a particular language, which is important for formatting currencies and numbers.
- A flag to enable or disable the entire storefront.

In addition to the above configuration options, every page within the CMS Site is based on a PageTemplate that can be configured and customized. Each PageTemplate is organized into a variety of CMS slots, and each slot can be filled with a CMS component.

For more information, see the following:

- Adding a New CMS Page Template
- acceleratorcms Extension
- Overview of the CMS2 Data Model
- WCMS Integration
- Overview of the CMS2 Data Model

BaseStore

A BaseStore configuration options include:

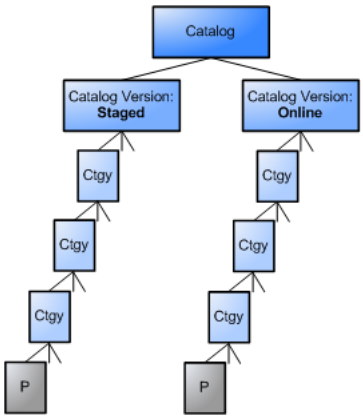
- Currencies that are supported, the first currency in the list is deemed the default.
- Points Of Service, also known as brick and mortar stores, that can be found in the Store Locator.
- Distance unit for the Store Locator (kilometers or miles).
- User tax groups to allow for products to use different sales tax rules per BaseStore.
- Prices shown as net or gross of sales tax.

For more information, see the following:

- Base Store
- Using Store Locator Service

Catalog Data

Catalogs are used to store product, classification, and WCMS content. All sample stores supplied with Accelerator come with the recommended Staged and Online CatalogVersions for each catalog. SynchronisationJobs are created automatically by Accelerator for the sample catalogs.



Content Catalog

A content catalog stores all the WCMS Pages, PageTemplates, components, and media. All content is added first to the Staged CatalogVersion and then published to the Online CatalogVersion either using the WCMS Cockpit or the SynchronisationJobs.

- The languages stored on the Content CatalogVersion identify which languages are available on the storefront.
- The territories stored on the Content CatalogVersion are used to identify which Zoned Delivery Modes, that is delivery options, apply for the storefront.

For more information, see Overview of the CMS2 Data Model.

Product Catalog

A product catalog stores all product and category data and related media. Products and categories are loaded into the Staged CatalogVersion, then published to the Online CatalogVersion either using the Product Cockpit or the SynchronisationJobs.

For more information, see the following:

- Product Cockpit
- Catalog Guide

Products

SAP Commerce Accelerator supports the following approaches to modeling product and categories:

- Vanilla products: Standard SAP Commerce product model.
- Vanilla products with classifications: Standard SAP Commerce product model including classification system.
- Variants: Base products with variant products, `apparelstore` uses the ApparelProduct as an example.
- Variants with classification: Supported, but no example in `apparelstore` or `electronicsstore` AddOns.

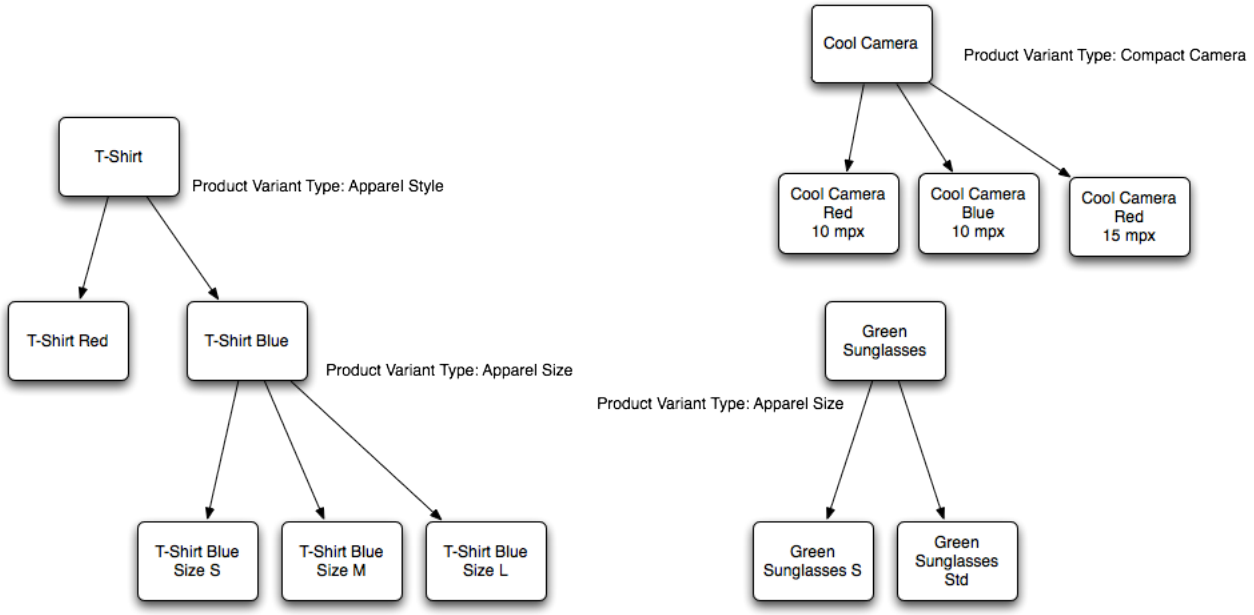
Projects should modify the data model in their own `items.xml` file to fit their requirements, as well as customize and extend the default facades and services provided by Accelerator.

For more information, see the following:

- Product Modeling
- items.xml
- How To Extend the ProductFacade

Variants

Variants are products that differ in some aspects from one another, but are based on the same basic model sharing many attributes. They are represented with a tree-style hierarchy where only the leaf nodes can be purchased, that is a complete SKU. The root, or base, product contains attributes common to the entire hierarchy or branch. Typically, the swtorefront user starts at a base product or a branch product and applies further configuration of the product, such as picking a color and/or size, to arrive at a selected purchasable leaf node, which can then be added to cart.



Accelerator supports the following variant models:

- Single-depth or double-depth Product Variant models are supported. Each depth can have multiple variant attributes.
- The ApparelProduct product variant model is available in the `yacceleratorcore` extension.

Categorization

Categorizing products into a clean hierarchy is fully supported. Adding other category hierarchies and linking either the complete hierarchy or a subset of it to facets in a Solr Search Index is also supported and shown in the sample data. Both AddOns include a taxonomy category hierarchy, for example, Root > Cameras > Digital Cameras > SLR, as well as a brand hierarchy, for example Brands > Nikon.

Categories and brands can be used to trigger merchandised landing pages in the WCMS Cockpit.

For more information, see Structuring Your Collection with Products and Categories.

Pricing

Pricing uses the `europe1` pricing extension. This includes a vast number of configuration options for pricing. For example, the `apparelstore` AddOn has these options:

- Multiple prices can be stored for a product in different currencies.
- If two currencies share the same base currency, it is possible for automatic currency conversion to be performed if a price is unavailable for a particular storefront currency.
- A user tax group can be set on a tax row. This can be matched to the store tax group, so different tax rates can be applied for the same product, but on different storefronts.
- A channel can be set for a price row. The channel-specific price is then only applicable on the specified storefront (e.g., mobile or desktop).

For more information, see Price, Tax, and Discount Calculation.

Classification System

A classification system can be added to enable the capability of assigning dynamic attributes to product categories from the product catalog. Classification systems tend to be good fits for stores that either sell a lot of technical products with a vast number of attributes that would be too much effort to model, or stores with a very disparate taxonomy of products.

Classification attributes can be assigned to a Solr Search Index. This includes facets, attributes displayed in search results, attributes in a free text search, and result sorting. Using classification attributes, it is possible to create a new taxonomy in a Solr Search Index without the need for a system restart.

For more information, see Classification.

Solr Facet Search

Solr Facet Search Indexes are bound to the online product catalog in all of Accelerator sample stores. The system determines which search index to use based on the active product catalog versions for the current site.

Customer Data

Customer accounts are shared across all storefronts. A user who registers in one storefront is registered in all other storefronts. An address book is saved with the customer, and addresses that are not in the supported territories of the current active storefront are filtered out of the box. Saved payment cards are also supported, however, Accelerator uses a Payment Service Provider subscription service to store the full credit card number and only a masked number is saved.

Order Data

Orders can be shared across multiple Storefronts but the CMS Site and Base Store is captured on the order when the order is placed. Services in the `commerceservices` extension are available to return only orders from the current storefront.

The order fulfillment process runs globally. If required, the CMS Site and Base Store are available to affect behaviour of fulfilment tasks.

Stock

It is possible to turn off stock check functionality by simply removing any default warehouse from the system. If you do not turn off stock check functionality, the system assumes all products without stock levels are out of stock.

For more information, see the following:

- Stock Management
- About Stock Service

B2B Storefront and Catalog

Accelerator adds the following B2B features:

- A Powertools Store: The Powertools site is based on the sample electronics site but with some variations in layouts and theme.
- A Powertools Product Catalog: The Powertools product catalog is a classification-based catalog based on a similar data set to the electronics product catalog.

Related Information

Essential and Project Data
yacceleratorcore Extension
apparelstore Extension
electronicsstore Extension

System Context

This document discusses potential extension points for SAP Commerce Accelerator and how Accelerator fits into the wider context of an eCommerce System. This includes pointers to where code should be extended, additional documentation to review, or recommendations for implementation decisions.

Payment Provider Integration

If integrating new payment providers, follow the instructions provide in the Payment Integration document.

You may find it easier to integrate a hosted solution, so that you do not need to be concerned with PCI Requirements.

You may also need to customize the Customer Service Cockpit to support your new payment mechanism.

Warehouse and Stock Integration

Typically, the warehouse or stock management system wants to receive product and order data and manages and provides stock levels to your system. This document does not cover standard ATP and ASN processes, and assumes that your warehouse or stock management system contain the same list of products.

The are two mechanisms for importing information provided by SAP Commerce Accelerator. These are configured through Spring integration:

- `StockImportAdapter`: Imports absolute Stock Levels
- `StockIncrementalImportAdapter`: Imports incremental Stock levels

In order to correctly integrate with your external warehouse or stock management system it is important to use both absolute and incremental feeds, unless you plan on querying the external system directly. The absolute feed should be used once a day to ensure synchronization, the incremental feed should be used intra-day when stock levels change due to deliveries and/or breakages.

Both stock feeds expect delimited files containing rows with something like `ProductCode,15,default`. The first column is the product code, which is a unique identifier, the second column is the stock value, the third is the warehouse name, which by default is `default`. The absolute feed accepts positive integer values or zero for stock levels. The incremental feed accepts positive or negative integer values, as well as zero which is obviously a bit meaningless in this context.

The SAP Commerce Accelerator sample stores show examples of how to configure multiple warehouses, although for most warehouse systems this may not be useful. If your project does not have a requirement to display detailed warehouse information to the end user the recommended approach would be to maintain stock in a single default warehouse only

Note

Currently all warehouses in use on your system must have 'default' set to true, due to the implementation of the `WarehouseService` in SAP Commerce.

For more information, see Stock Management.

Displaying Stock Information on Your Site

In the latest version of SAP Commerce Accelerator a Stock Level Status is extracted to the Solr index in order to provide banded stock levels, for example `InStock`, `LowStock`, on the search result or category browse pages. This is shown by the disabling of the **Add to Basket** button if the product is not in stock. This approach is recommended over, for instance, post-processing the results as it allows you to filter and facet on stock information. Stock levels are additionally shown on the product pages. Both these stock levels are driven off the stock information stored in the hybris-driven systems, but this could be easily changed to query an external system via reimplementation of the `StockService`.

Google Maps Integration

Google Maps integration helps a user to find points of service defined by the system manager that are placed nearby location specified by the front-end user. It requires actual point of service location to be stored in the system and its geographical coordinates to be entered or determined with geocoding cron job. Then a front-end user can specify his location and after sending it to the system it is decoded and actual distance between this location and points of service stored in the system is determined. After this list of nearby points of interest can be presented to the user.

Google Maps integration uses store locator feature that is implemented in the `basecommerce` extension - you can find more information about this feature in the Store Locator Implementation document.

Configuration

Actual configuration of the SAP Commerce Accelerator Google Maps integration required one step: Google API key must be generated and entered to the `project.properties` file. Each API key is generated for a specific address, so it is easy to configure the SAP Commerce Accelerator system to operate with different stores at one running system. The properties file contains data as key to value pairs. Google API key in the property file should start with the `googleApiKey` string followed with a URL that this particular key should match. Here is sample piece of property file for generated Google API keys:

```
googleApiKey.apparel.uk.local=ABQIAAAAv(...)4LmJ7bMEeRQ
                                                googleApiKey.apparel.de.local=ABQIAAAAv(...)Ma6XGgaJvyQ
```

Another parameter that can be configured is distance unit. This can be set for each BaseStore instance using the Backoffice Administration Cockpit or an ImpEx file. You can find more information about the store locator configuration in the Store Locator Configuration document. When there is no Google API key specified for the address that you use, system displays JavaScript alert that informs a client about this fact. No map with positions is displayed.

Controllers

The process of determining nearby points of service starts when a front end user enters his location at the form. Request comes to the controller that uses `StoreLocatorFacade` to decode entered location to actual geographical coordinates and then to find nearest points of service. User can enter his location in any way that can be recognized by Google geocoding API, which can be city name, postal code, full address and more. If entered location could not be decoded to actual position proper message will be sent to the client. Then controller sends list of locations to the view layer.

View Rendering

There are two views assigned to `StoreLocatorController`: search result view and point of service details view. The first one allows user to enter his location and then displays list of search result limited to some amount and ordered ascending by distance. If a user wants to see more results, and if there are more results, there is also proper link provided at the bottom of the page. In addition to the result list there is also a map displayed with applied positions of each point of service that is present in a search result list.

When a user selects a desired point of service, second view is displayed that presents detailed information and also map with only this one point of service displayed.

Google Local Shopping Integration

SAP Commerce Accelerator supports exporting product data, prices and store locations to the Google Shopping / Local services, so that customers can find products in their local stores via Google Search mechanism.

The starting point for this is to creating an account with Google Local Shopping. You can find description how to do it in the http://www.google.com/support/merchants/bin/answer.py?hl=enanswer=187892 document.

The main aspect of the Google Local Shopping export that you need to customize is the conversion beans that convert the hybris items into the appropriate record format for export to Google. For more details, see System Architecture of SAP Commerce Accelerator.

Google Analytics Integration

To begin integrating Google Analytics you should start by uncommenting and completing the following sections from the *<HYBRIS_BIN_DIR>*`/ext-template/yacceleratorstorefront/web/webroot/WEB-INF/tags/template/javaScript.tag` file:

```
<script type="text/javascript">
                                    /*<![CDATA[*/
                                    // uncomment this to include google analytics if required
                                    // var gaJsHost = (("https:" == document.location.protocol) ? "https://ssl." : "http://www.");
                                    // document.write(unescape("%3Cscript src='" + gaJsHost + "google-analytics.com/ga.js' type='text/javascript
                                    /*]]>*/
                                    </script>
                                    <script type="text/javascript">
                                    /*<![CDATA[*/
                                    try
                                    {
                                    // uncomment this to enable google analytics if required
                                    // var pageTracker = _gat._getTracker("XXXXXXXX");
                                    // pageTracker._setDomainName("");
                                    }
                                    catch(err)
                                    {
                                    }
                                    /*]]>*/
                                    </script>
```

Then follow the instructions provided in the Integrating Google Analytics document.

Fraud Detection Integration

Fraud Detection is a complex area, and it is likely that integrating with a third party provider will require significant customizations.

The `fulfilmentprocess` extension has an action that begins the fraud check. The following is an example from the `fulfilmentprocess-spring.xml` file.

```
<bean id="fraudCheckOrder" class="de.hybris.platform.fulfilment.actions.FraudCheckOrder" parent="abstractAction">
                                    <property name="fraudService" ref="fraudService"/>
                                    <property name="providerName" value="Mockup_3rdPartyProvider"/>
                                    </bean>
```

It is recommended that you:

- Provide your own `FraudServiceProvider` encaspsualting the call to the third party.

- Configure the `FraudService` to hold your provider.
- Change the above action to use your `providerName`.
- Store any response information from your third party on the `Order`.
- Extend the `cscockpit` extension to expose this information and allow Orders to be confirmed and/or canceled basing on this information.

For more information, see the following:

- Fraud Detection/
- yacceleratorfulfilmentprocess Extension

Address Verification Lookup Integration

Currently SAP Commerce Accelerator does not provide an address lookup or verification service, but it would not be difficult to integrate a provider such as QAS or Capscan.

Here are some recommendations for integrating such a product:

- You need to change the Register and Checkout processes to integrate an address lookup step
- A good approach is to allow the user to enter a postcode only and use that to look up the list of possible addresses, then allow the user to select from a list.
- Do not expose the address lookup directly through a request-bound controller or via an unsecured URL.

Media Server Integration

If hosting your media externally it is recommended that you make changes to SAP Commerce Accelerator to return URLs instead of media objects when an image is requested.

Top Navigation and Link Management

The SAP Commerce Accelerator offers customizable, top navigation and link management functionality. The customization is done through the WCMS components.

ℹ Note

To keep the navigation perspective and the navigation components in sync, `NavigationBarCollectionComponent` was deprecated. We recommend to use `NavigationComponent` instead.

Top Navigation

The Top Navigation menus on the Accelerator site are controlled though the configuration of the following CMS Components:

- **NavigationComponents** - These components represent sections of the Navigation Bar. Each section can have one **CMSLinkComponent** object and one **CMSNavigationNode** object considered to be the root node.
- **CMSNavigationNodes** - These components represent the structure of the navigation tree and have titles which are rendered in the drop down menu. Each structure can have many **CMSLinkComponent** objects and many child **CMSNavigationNodes** objects.
- **CMSLinkComponents** - These components are rendered on the page and display at each point in the navigation tree.

The process for adding a new **NavigationComponent** object to a site to display the navigation menu is the following:

1. Create a **NavigationComponent** object.
2. Create a **CMSLinkComponent** object, give it a title and a URL, and assign it to the **NavigationComponent** object.
3. Create a root **CMSNavigationNode** object and add it to the **NavigationComponent** object.
4. Create as many children **CMSNavigationNodes** objects as required and add them to the root **CMSNavigationNode** object as children. Each child node creates an additional column in the menu.
5. Create as many **CMSLinkComponents** objects as required and add each one to a child node. These components appear as a list of links in the column.

These additional configuration attributes on the **NavigationComponent** object are available:

- **wrapAfter** - Controls the wrapping of links into additional columns. For example, wrap the link text after seven rows into another column.
- **styleClass** - Enables an additional CSS style to be appended to the navigation bar component. For example, the text can be right-justified or highlighted.
- **dropDownLayout** - Enables configuration of the justification of the **Drop Down Menu**. For example, the menus can appear on the righthand side of the page in order to improve the layout.

Examples

Two examples of customized top navigation are shown in the following figures:

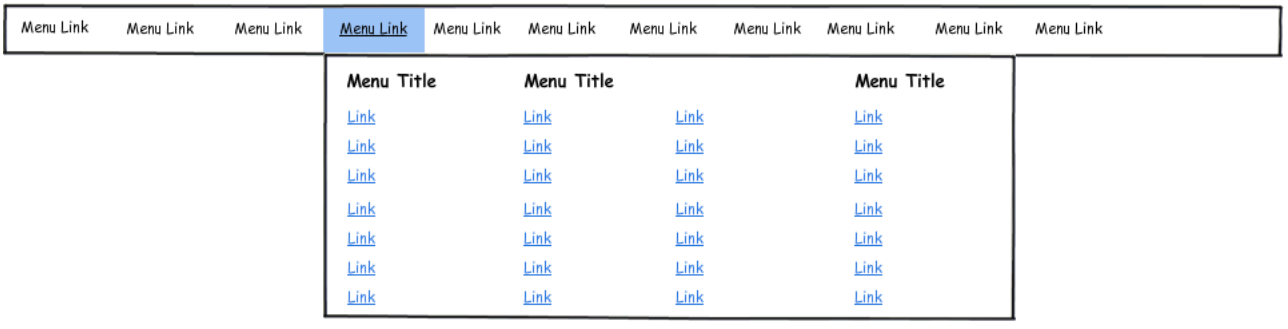


Figure: Example Top Navigation with Drop Down Layout on the Left

- This figure shows eleven **NavigationComponent** objects.
- The highlighted **NavigationComponent** root node has three child nodes.
- The first and third child nodes have seven **CMSLinkComponents** objects, the second child node has 14.
- The **wrapAfter** attribute is set to seven.

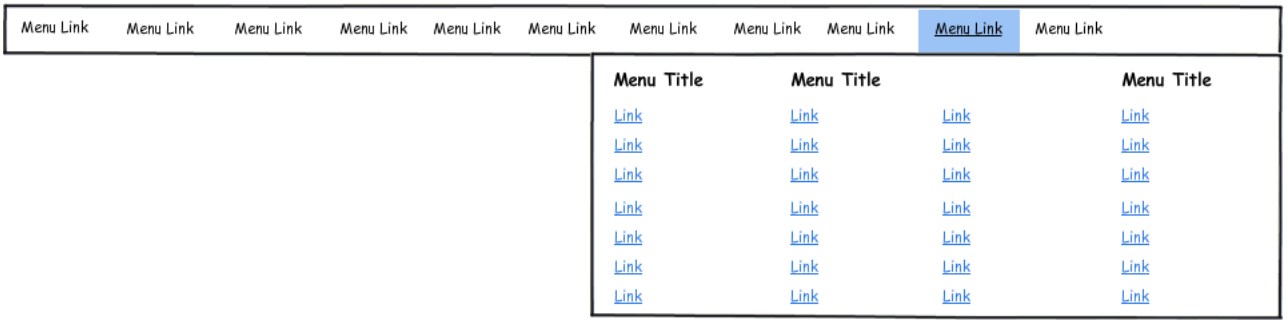


Figure: Example Top Navigation with Drop Down Layout on the Right

This figure shows a similar configuration with **dropDownLayout** set to **Right**.

Link Management

Links in the Top Navigation are **CMSLinkComponents** objects. These can be used to provide links to:

- A URL
- A Content page
- A Category
- A Product

You can use a URL directly to provide any internal links, such as a link to a category. If the **CMSLinkComponent** object has more than one link set, then the order of preference is listed as above. Links can also be configured to open in new tabs or new windows using the **target** attribute.

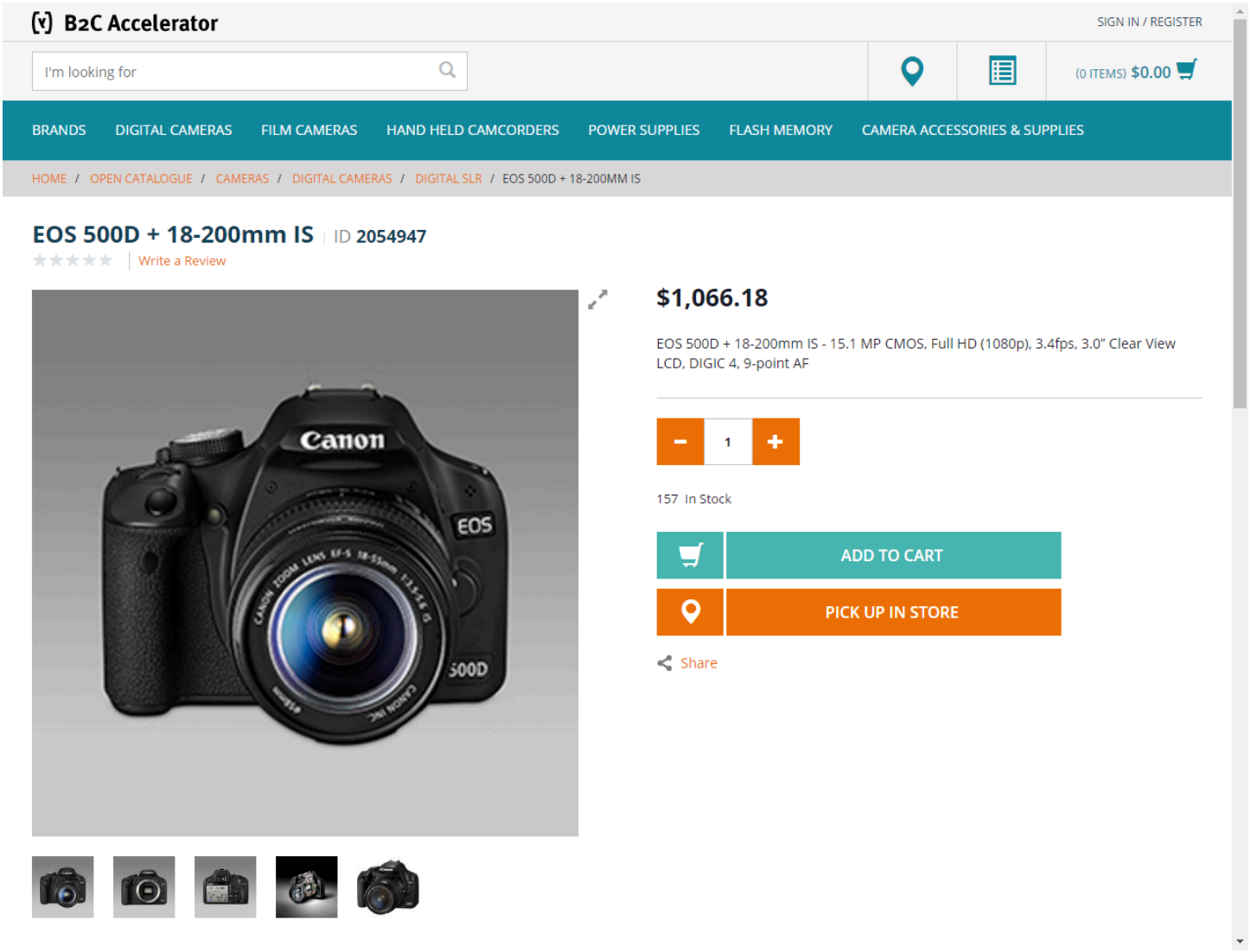
User Interface and Creatives: Responsive Experience

Learn about the interface and design of the default Accelerator responsive storefronts, including how to change the design of the storefronts using themes and CSS.

The Front End

The B2C Accelerator is shipped with the blue theme, which you can use in either the apparel or the electronics storefront.

The following screenshot is an example of the responsive Product Details page:



Design Best Practices

The following sections cover design best practice topics, including color schemes, layout, and accessibility.

Color Schemes

Each of the themes created for the SAP Commerce Accelerator storefronts uses a minimal set of colors to keep customers focused on the important elements of the user interface. A recommended approach to designing a color theme is to select a small range of colors that fit well together, with one or two complementary colors for highlighting purposes.

Adobe offers the following website that can help you in designing your own color scheme: https://color.adobe.com/create/color-wheel/

The following are some suggestions on how you might organize the user interface of your storefront according to your color scheme:

- use one color for all graphical elements that are related to the shopping process, such as the cart, prices, and interface elements related to purchasing items and checking out.
- use one color to highlight elements that are focused, or which the mouse hovers over.
- use one color to show available offers or promotions, such as you might see in banners.
- use one color for the background of navigation elements.

Layout

The most important interface elements are placed above the page fold, based on a 1024x768 pixel screen resolution. The header typically contains those elements that need to appear on every page, such as the mini-cart, the global search area, the breadcrumb navigation bar, the store locator, and the login area.

Figure: Location of exemplary items on the page of the storefront.

The most important design factors for the storefront templates include the following:

- Consistent text sizes, as well as a limited number of text sizes.
- Short product names.
- Short and concise product descriptions.
- Consistent icon styles and sizes.

Dimensions

Every banner has an applied border of one pixel width. Therefore, every banner is 2 pixels smaller than its surrounding content element.

Standardized Components

For information about the default components that ship with Accelerator, see Components

Accessibility

All non-text content presented to the user also has an alternative text that describers the non-text content. The content can be presented in different ways without losing information or the layout structure. For example, you can use a simpler layout without a stylesheet.

It is also possible to navigate the storefront using the Tab key of your keyboard.

For information on Web Content Accessibility Guidelines 2.0, see http://www.w3.org/TR/WCAG20/ .

Technical Structure

The `common` folder, located in the *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/_ui/`*<ui_experience>* directory, provides the basis and underlying structure for each theme for a given UI experience (in this case, responsive), and consists of the following subdirectories:

- The `bootstrap` folder contains files related to the bootstrap framework.
- The `images` folder provides a location for all theme-independent images.
- The `js` folder provides a location for all of the JS libraries that are used globally within B2C Accelerator.

Note

We recommend that all re-branding be placed in your theme directory. The theme directory is discussed in more detail below.

JSTL

The JavaServer Pages Standard Tag Library (JSTL) source is located in the following directories:

- *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/tags`
- *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/views`

Themes are changed in Accelerator using Less. For more information on using Less with the responsive Accelerator, see Building a Responsive Website Front-End and JS, LESS and JSPs: Build Process for Generating Variables.

Tables are avoided for CSS-based styling and Google optimization. Tables have a predefined structure that cannot be changed using CSS, so a DIV-based HTML structure is used instead. The DIV-based HTML structure is more generic and flexible.

You can use the Firefox Firebug extension to explore and analyze the structure of the web front end of Accelerator. This extension allows you to select any item in the DOM tree by clicking on it in the browser. It also displays all of the available styles for this item, and which CSS files contain this information.

As previously noted, the HTML code is located in the *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF` directory. For localization purposes, you can localize the files located in the *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/messages` directory.

Theme Directory

The theme directory, located in the *<HYBRIS_BIN_DIR>*`/emodules/base-accelerator/yacceleratorstorefront/web/webroot/_ui/`*<ui_experience>*`/`*<themeName>* directory, consists of the following subfolders:

- The `css` folder contains all of the CSS files pertaining to your theme. For Accelerator, this consists of at least one file: `changes.css`.

  You can create the desired look and feel of your store by making changes to this style sheet. All images that are located in the *<HYBRIS_DATA_DIR>*`/media/sys_master` directory, such as logos and banners, are not changed by CSS.

- The `fonts` folder contains glyphicons, which are the default fonts used by Accelerator. If you wish to use different fonts, place the fonts in this directory, and update the references to fonts in Less. For more information on using Less with the responsive Accelerator, see Building a Responsive Website Front-End and JS, LESS and JSPs: Build Process for Generating Variables.
- The `images` folder contains theme-dependant images. When defining an image URL in the `changes.css` file, the image should fit the criterion of being theme-dependent.

JavaScript

Accelerator uses the jQuery JavaScript framework. The following JavaScript libraries are currently in place, and are located in the *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/_ui/`*<ui_experience>*`/common/js` directory:

- `jquery-2.1.1.min.js`: Provides the core jQuery library functionality.
- `jquery-ui-1.11.2.custom.min.js`: Provides the interface for the jQuery core.
- `Imager.min.js`: Based on the BBC Imager, which selects the image based on the screen width and the parent element size. For more information, see Responsive Image WCMS Component.
- `owl.carousel.customer.js`: Enables the product carousel component.
- `jquery.colorbox-min.js` and `jquery.colorbox-1.3.16.js` (for B2C and B2B storefronts respectively): Enable all Ajax layers (lightboxes), such as the Ajax Basket Lightbox component.

For more information, see the following:

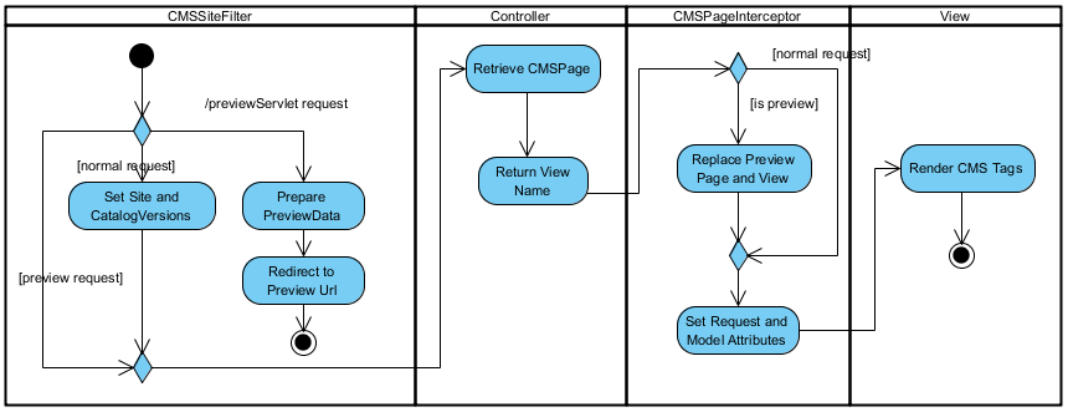
- jQuery framework page: http://jquery.com/
- jQuery user interface page: http://jqueryui.com/
- jQuery plug-ins page: http://plugins.jquery.com/

WCMS Integration

This document describes the WCMS integration of the `yacceleratorstorefront` extension and how to enable the content management and preview functionality based on the `cms2` extension. This integration is implemented by reusing existing classes of the `storetemplate` extension while adding and modifying existing functionality.

WCMS Integration Overview

The general page processing in the `yacceleratorstorefront` extension regarding WCMS is shown in the next diagram:



As shown, there are four major steps during processing:

1. General session setup and preview handling, that is session setup and redirect.
2. Page and view resolution in the Spring MVC Controller.
3. Further population of request and model attributes in the Interceptor with additional page and view replacement functionality during preview.
4. View rendering using CMS tags, for example `<cms:body>` , `<cms:slot>` , or `<cms:component>` .

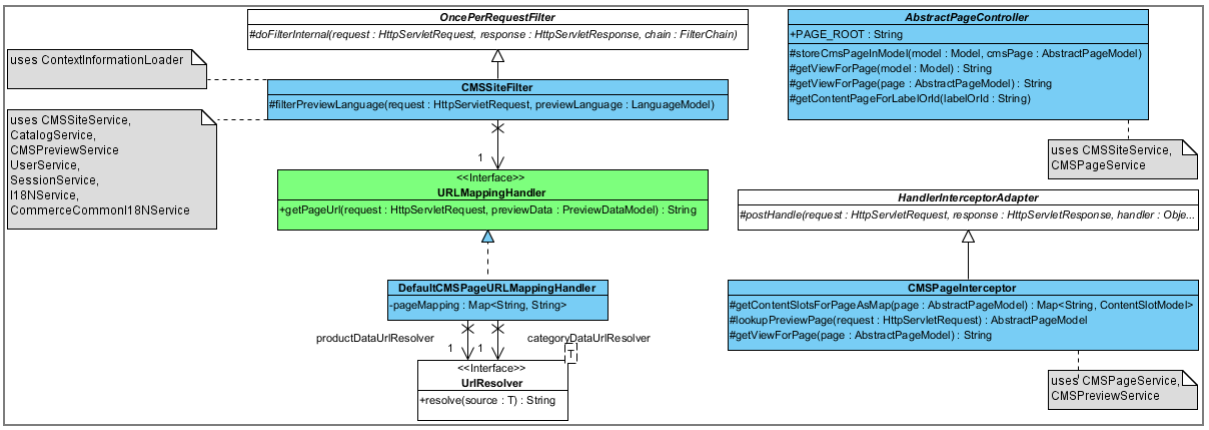


Figure: A diagram of classes for the WCMS integration within the SAP Commerce Accelerator .

CMSSiteFilter

The CMSSiteFilter is responsible for setting up sessions and handling previews. It was reused from the `storetemplate` extension. The changes made to these classes include:

- Using CommerceCommonI18NService instead of CommonI18NService.
- Adding the `filterPreviewLanguages` method to restrict preview languages to languages supported by the CMSSite.
- Replacing the URLMappingHandler with an implementation that supports page mappings and product and category pages.

For initializing, the JaloSession CMSSiteFilter collaborates with the ContextInformationLoader.

DefaultPreviewDataModelUrlResolver

The DefaultPreviewDataModelUrlResolver retrieves a redirect URL during previewing and supports URLs for:

- A previewed Product or Category by delegating to UrlResolver implementations in the `commercefacades` extension.
- Content pages using the DefaultPageController and using the lookup path as UID, for example `/about`
- Content pages using custom controllers configured in a `pageMapping` .

Page mappings are added to the `spring-mvc-config.xml` file as follows:

```
<bean id="defaultPreviewDataModelUrlResolver" class="de.hybris.platform.yacceleratorstorefront.preview.DefaultPreviewDataModelUrlResolver" scope="te
          <property name="productModelUrlResolver" ref="productModelUrlResolver"/>
          <property name="categoryModelUrlResolver" ref="categoryModelUrlResolver"/>
          <property name="pageMapping">
          <map>
          <entry key="homepage" value="/"/>
          <entry key="cartPage" value="/cart"/>
          <entry key="search" value="/search"/>
          <entry key="searchEmpty" value="/search"/>
          <entry key="account" value="/my-account"/>
          <entry key="profile" value="/my-account/profile"/>
          <entry key="address-book" value="/my-account/address-book"/>
          <entry key="add-edit-address" value="/my-account/add-edit-address"/>
          <entry key="payment-details" value="/my-account/payment-details"/>
          <entry key="order" value="/my-account/order"/>
          <entry key="orders" value="/my-account/orders"/>
          <entry key="singleStepCheckoutSummaryPage" value="/checkout/single/summary"/>
          <entry key="multiStepCheckoutSummaryPage" value="/checkout/multi/choose-delivery-address"/>
          <entry key="storefinderPage" value="/store-finder"/>
          <entry key="login" value="/login"/>
          <entry key="checkout-login" value="/login/checkout"/>
          <entry key="forgottenPassword" value="/login/pw/request"/>
          <entry key="updatePassword" value="/login/pw/change"/>
          </map>
          </property>
          </bean>
```

For more information, see commercefacades Extension .

AbstractPageController

The AbstractPageController class is the base class of all controllers in the `yacceleratorstorefront` extension. It provides the following:

- Lookup functionality for CMS pages.
- Lookup functionality for retrieving the view name for controllers supporting switchable views, for example HomePageController.
- Convenient methods to set basic model attributes, that is user, currencies, languages, currentCurrency, user, and request.

Example for Use of Switchable Template

HomePageController is an example of a controller supporting switchable views. The view is manageable in the SmartEdit by setting the `frontendTemplateName` . To support this behavior, the following two lines of code are required:

```
@Controller
    @RequestMapping("/")
    public class HomePageController extends AbstractPageController
    {
      @RequestMapping(method = RequestMethod.GET)
      public String home(final Model model) throws CMSItemNotFoundException
```

```
        {
          storeCmsPageInModel(model, getContentPageForLabelOrId(null));
          return getViewForPage(model);
        }
      }
```

First, the `home` method is mapped to the correct path and request method. Then `getContentPageForLabelOrId` is called, which in turn performs the following calls:

- If the parameter `labelOrId` is null or empty, it retrieves the label for the homepage from CMSPageService.

  If no page is marked as homepage, it retrieves the label for the starting page for the current site.

- It returns the page for the resolved key by calling `getPageForLabelOrId` on CMSPageService.

The retrieved page is stored as the `CMS_PAGE_MODEL` model attribute. In a second step, the view is retrieved by calling `getViewForPage` , which is achieved by receiving the previously set page from the model and accessing the `frontendTemplateName` attribute of master template as shown here in the AbstractPageController:

```
protected String getViewForPage(final AbstractPageModel page)
    {
      if (page != null)
      {
        final PageTemplateModel masterTemplate = page.getMasterTemplate();
        if (masterTemplate != null)
        {
          final String targetPage = cmsPageService.getFrontendTemplateName(masterTemplate);
          if (targetPage != null && !targetPage.isEmpty())
          {
            return PAGE_ROOT + targetPage;
          }
        }
      }
      return null;
    }
```

This allows you to exchange the landing page layout by switching to a different landing page template in SmartEdit . The `frontendTemplateName` could also be updated directly to a different JSP page that supports the specifed template latyout. The recommended approach used in the SAP Commerce Accelerator is to define a new template instead of updating the template name directly.

Example of Using a Static Template

For some controllers, supporting switchable views is not required. For example, the login and account pages do not support multiple layouts and therefore take care of returning the view name directly after defining the CMS page. An example of this is the method to show the page to update the password, as shown here in the AccountPageController:

```
@RequestMapping(value = "/update-password", method = RequestMethod.GET)
    public String updatePassword(final Model model) throws CMSItemNotFoundException
    {
      final UpdatePasswordForm updatePasswordForm = new UpdatePasswordForm();
      model.addAttribute("updatePasswordForm", updatePasswordForm);
      storeCmsPageInModel(model, getContentPageForLabelOrId(PROFILE_PAGE));
      return ACCOUNT_PAGE_ROOT + PASSWORD_PAGE;
    }
```

This simple shows how to perform the following actions:

- Map a controller method to a request by using annotations.
- Populate the model with a form bean.
- Retrieve the CMS page by calling `storeCmsPageInModel` providing a page UID.
- Return the view name to Spring MVC using constants.

CMSPageInterceptor

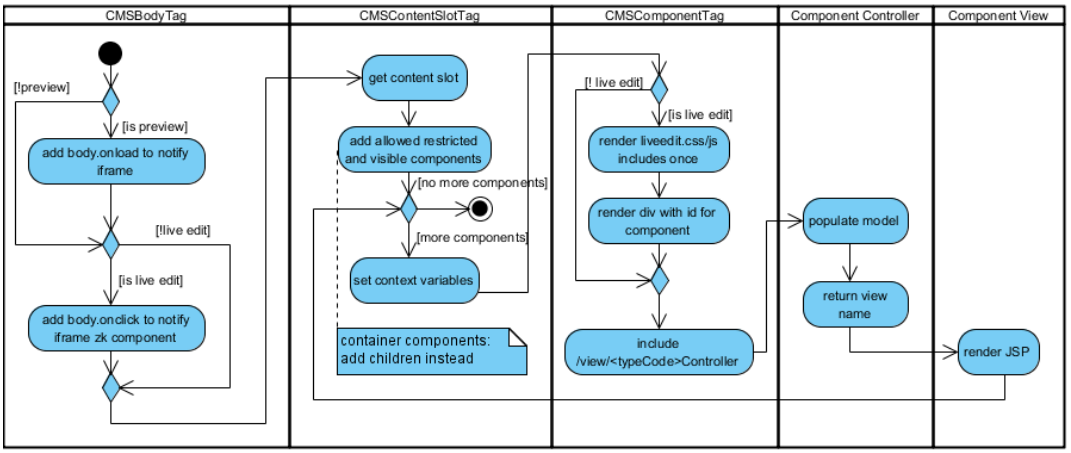
The CMSPageInterceptor is required to keep the controller agnostic of preview-related functionality and to populate the `Model` and `HttpServletRequest` attributes for the view. First, in the presence of the `PREVIEW_TICKET_ID_PARAM` request parameter, the preview context is evaluated. If the page attribute was set in the SmartEdit and differs from the page retrieved by the controller, then the page is replaced in the model or, if there is a view associated with the template and this is different from the view returned by the controller, then the view is replaced in the ModelAndView.

In a second step, `Model` and `HttpServletRequest` attributes are defined. The following provides attributes to be taken care:

Class	Attribute Name	Attribute Type	Description
`HttpServletRequest`	currentPage	AbstractPageModel	The page request attribute required by `CMSBodyTag` .
`Model`	cmsPage	AbstractPageModel	The page model attribute. Currently, only referenced in `master.tag` to retrieve the page title and provide debug information.
`Model`	cmsSite	CMSSiteModel	The current site. Currently, only referenced in `master.tag` to provide debug information.
`Model`	slots	Map<String, ContentSlotModel>	The slots attribute to allow for referencing CMS content slots in JSPs and tags.

View

The view rendered using the `cms2` extension tags and Spring MVC controllers is shown in the following diagram:



The classes involved in this processing are as follows:

Class Name	Description
`CMSBodyTag`	Adds specific scripts for preview and live edit to integrate preview in SmartEdit .
`CMSContentSlotTag`	• Retrieves the content slot either by UID or as a tag attribute. • Initializes a list with all components that are visible or allowed restricted. For restricted components, this decision is delegated to CMSRestrictionService. • Iterates over all components, setting the required page context attributes: `contentSlot` , `elementPos` , `isFirst` / `LastElement` , `numberOfElements` , tag attribute `var` .
`CMSComponentTag`	• Renders the component by calling `pageContext.include("/view/" + controllerName)` using: ◦ < ***componentTypeCode*** > `Controller` as a controller name, if a bean with this name is defined in the Spring application context. ◦ If not, it uses DefaultCMSComponentController.
`Component Controller`	• Populates the model ◦ The implementation of the DefaultCMSComponentController puts all frontend properties of the component into the model. ◦ Returns the view name to use, following the convention to append the component type in lowercase to one of the following: ▪ `WEB-INF/views` for specific components or ▪ `WEB-INF/views/cms` for components using the DefaultCMSComponentController
`Component View`	Finally, renders the component.

You find examples of components using a default and a specific controller above. As an example of a specific controller see also the *Storefront* section of Search and Navigation .

Example of Using the Default Controller

The CMS Paragraph component provides a simple example of using the default controller. This component only contains a content attribute to manage rich text content. To display this component, a JSP is provided in the directory ***$[HYBRIS_BIN_DIR]*** `/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/views/cms` for B2C storefronts, or in the directory ***$[HYBRIS_BIN_DIR]*** `/modules/base-accelerator/yb2bacceleratorstorefront/web/webroot/WEB-INF/views/cms` for the B2B storefront. The following line is from `cmsparagraphcomponent.jsp` :

```
<div class="content">${content}</div>
```

This component is used in the header to display the contact information. To achieve this, the `<cms:slot>` tag is added to the `header.tag` tag file, referencing a previously configured content slot, as follows:

```
<cms:slot var="link" contentSlot="${slots.HeaderLinks}">
      <li><cms:component component="${link}"/></li>
    </cms:slot>
```

This example shows how to:

- Add a JSP view for a simple CMS component.
- Use the CMS tags to render the component in a JSP or tag file.

Example of Using a Custom Controller

You can use more advanced CMS component controllers to call:

- Facade implementations in the `yacceleratorfacades` and `commercefacades` extensions.
- Services in the `commerceservices` extension.
- Other platform services.

An example of this is shown in the ProductFeatureComponentController as follows:

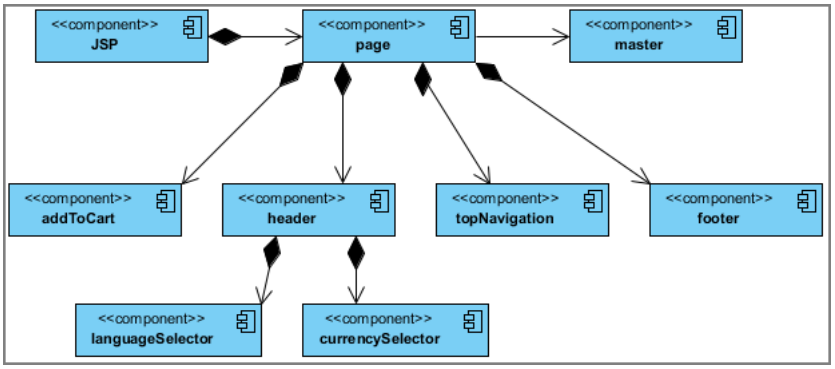
```
@Controller("PurchasedProductReferencesComponentController")
    @RequestMapping(value = ControllerConstants.Actions.Components.PURCHASED_PRODUCT_REFERENCES_COMPONENT)
    public class PurchasedProductReferenceComponentController extends
        AbstractCMSComponentController<PurchasedProductReferencesComponentModel>
    {
      @Autowired
      @Qualifier("simpleSuggestionFacade")
      private SimpleSuggestionFacade simpleSuggestionFacade;

      @Override
      public String renderComponent(final HttpServletRequest request, final HttpServletResponse response, final Model model,
          final PurchasedProductReferencesComponentModel component)
      {
        final List<ProductData> products = simpleSuggestionFacade.getReferencesForPurchasedInCategory(component.getCategory()
            .getCode(), component.getProductReferenceType(), component.getFilterPurchased().booleanValue(), component
            .getMaximumNumberProducts());
        model.addAttribute("title", component.getTitle());
        model.addAttribute("productReferences", products);
        model.addAttribute("comp", component);

        return ControllerConstants.Views.Components.PRODUCT_REFERENCES;
      }
    }
```

This CMS component is used to retrieve and display product references of a product purchased by a customer. These references are returned by calling a SimpleSuggestionFacade in the `yacceleratorfacades` extension. Additionally, component attributes are populated in the Spring Model. Finally, the view name to use is returned using a constant in ControllerConstants. This example shows how to:

- Map a custom controller to a request URL using a Spring MVC RequestMapping annotation.
- Autowire a dependency using Spring annotations.
- Retrieve additional data by calling a facade method.
- Populate the model.
- Return a view name to render the component.

Tag Files

The view is rendered using JSPs and tag files. JSPs include several tag files to impose a general, reusable structure shown in the following diagram:



The following tag components are provided in **`/WEB-INF/tags`** folder:

Component	Filename	Description
page	`template/page.tag`	General page structure
master	`template/master.tag`	Master HTML template
header	`common/header/header.tag`	Header bar
languageSelector	`common/header/languageSelector.tag`	Language selector
currencySelector	`common/header/currencySelector.tag`	Currency selector
topNavigation	`nav/topNavigation.tag`	Top navigation
footer	`common/footer/footer.tag`	Footer
addToCart	`cart/addToCart.tag`	Ajax cart pop-up

For more information, see Storefront Web Application Deconstructed .

Tag Files Example

The electronics storefront landing page uses `landingPage2.jsp`, a dedicated JSP that is referenced in the CMS master template. The following example from `landingPage2.jsp` shows how to:

- Include the page tag in a JSP file.
- Access a content slot by name using the predefined `slots` attribute set up in CMSPageInterceptor.
- Render CMS components within content slots.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %><c:set var="ignore-white-space">
        <%@ taglib tagdir="/WEB-INF/tags/template" prefix="template" %>
        <%@ taglib prefix="cms" uri="/cms2lib/cmstags/cmstags.tld" %>
    </c:set>
```

```
<template:page pageTitle="${title}">
    <cms:slot var="feature" contentSlot="${slots.Section1}">
            <div class="span-24 section1 advert">
                    <cms:component component="${feature}"/>
            </div>
    </cms:slot>
    <div class="span-24 section2">
            <div class="span-6 zone_a thumbnail_detail">
                    <cms:slot var="feature" contentSlot="${slots.Section2A}">
                            <cms:component component="${feature}"/>
                    </cms:slot>
            </div>
            <div class="span-6 zone_b advert">
                    <cms:slot var="feature" contentSlot="${slots.Section2B}">
                            <cms:component component="${feature}"/>
                    </cms:slot>
            </div>
            <div class="span-12 zone_c advert last">
                    <cms:slot var="feature" contentSlot="${slots.Section2C}">
                            <cms:component component="${feature}"/>
                    </cms:slot>
            </div>
    </div>
    <div class="span-24 section3 advert">
            <cms:slot var="feature" contentSlot="${slots.Section3}">
                    <cms:component component="${feature}"/>
            </cms:slot>
    </div>
    <div class="span-24">
            <cms:slot var="feature" contentSlot="${slots.Section4}">
                    <div class="span-6 section4 small_detail ${(elementPos%4 == 3) ? 'last' : ''}">
                            <cms:component component="${feature}"/>
                    </div>
            </cms:slot>
    </div>
    <div class="span-24 section5 advert">
            <cms:slot var="feature" contentSlot="${slots.Section5}">
                    <cms:component component="${feature}"/>
            </cms:slot>
    </div>

</template:page>
```

CMS Page Configuration

You can configure templates and pages in the SmartEdit or by using an ImpEx script. Due to the complexity of the data, the more common approach is to use ImpEx. To do so, execute the following steps:

1. Define the PageTemplate.
2. Define the ContentSlotName.
3. Define the page.
4. Define the ContentSlot.
5. Assign reusable content slots to the template in the ContentSlotsForTemplate.
6. Assign specific content slots to the page in the ContentSlotsForPage.
7. Add a CMS component like text, graphics, media, links, and so on to the page.

For more information, see Overview of the CMS2 Data Model .

CMS Page Configuration Example

As an example, you find the relevant scripts for configuring the homepage on the electronics storefront site. The `yacceleratorcore` extension provides the general setup for steps one to three in `cms-content.impex` :

```
#
    # Import the CMS content for the Electronics site
    #
    $contentCatalog=electronicsContentCatalog
    $contentCV=catalogVersion(CatalogVersion.catalog(Catalog.id[default=$contentCatalog]),CatalogVersion.version[default=Staged])[default=$contentCa
    $jarResource=jar:de.hybris.platform.yacceleratorcore.setup.CoreSystemSetup&
    # These define sets of components that can fit into similar slots
    $wideContent=CMSImageComponent,SimpleBannerComponent,BannerComponent,ImageMapComponent,RotatingImagesComponent,ProductCarouselComponent,CMSParag
    $narrowContent=ProductFeatureComponent,CategoryFeatureComponent,CMSImageComponent,SimpleBannerComponent,BannerComponent,ImageMapComponent,Rotati
```

This defines some macro definitions. Namely, the catalog version to use and the class used for loading resources from the classpath in FileLoaderValueTranslator by calling getResourceAsStream. Additionally, component types that fit into the wide and narrow content slots are assigned.

PageTemplate Definition

For better readability, the PageTemplate definition is split into two steps:

1. Name, UID and frontendTemplateName definition.
2. Velocity script used for displaying the template in SmartEdit .

```
INSERT_UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];name;frontendTemplateName;active[default=true]
    ;;LandingPage2Template;Landing Page 2 Template;landingPage2

    # Add Velocity templates that are in SmartEdit. These give a better layout for editing pages
    # The FileLoaderValueTranslator loads a File into a String property. The templates could also be inserted in-line in this file.
    UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];velocityTemplate[translator=de.hybris.platform.acceleratorcore.setup.FileLoade
    ;;LandingPage2Template;$jarResource/yacceleratorcore/import/cmscockpit/structure-view/structure_landingPage2Template.vm
```

ContentSlotName Definition

Content slot names are defined according to the wireframe definition in Landing Page 2 layout:

```
INSERT_UPDATE ContentSlotName;name[unique=true];template(uid,$contentCV)[unique=true][default='LandingPage2Template'];validComponentTypes(code)
    ;SiteLogo;;CMSImageComponent,BannerComponent
    ;HeaderLinks;;CMSLinkComponent,CMSParagraphComponent
    ;MiniCart;;MiniCartComponent
    ;NavigationBar;;NavigationComponent
    ;Section1;;$wideContent
    ;Section2A;;$narrowContent
    ;Section2B;;$narrowContent
    ;Section2C;;$wideContent
    ;Section3;;$wideContent
    ;Section4;;$narrowContent
    ;Section5;;$wideContent
    ;Footer;;CMSLinkComponent,CMSParagraphComponent,FooterNavigationComponent
```

Page Definition

Finally in the `yacceleratorcore` extension, the CMS page is defined. The same applies to the B2B Accelerator storefront, although if you want to customize it, change the classes belonging to the `yb2bacceleratorcore` extension.

```
INSERT_UPDATE ContentPage;$contentCV[unique=true];uid[unique=true];name;masterTemplate(uid,$contentCV);defaultPage[default='true'];approvalStatus(co
    ;;homepage;Homepage;LandingPage2Template
```

The remaining steps are accomplished in `cms-content.impex` in the `acceleratorsampledata` extension.

ContentSlot Definition

First, content slots are defined using the document ID syntax meaning the referenced components have to be defined in the same file. Alternatively, components could also be referenced by UID.

```
# commonly used content slots
    INSERT_UPDATE ContentSlot;$contentCV[unique=true];uid[unique=true];name;active;cmsComponents(&componentRef)
    ;;SiteLogoSlot;Default Site Logo Slot;true;SiteLogoImage
    ;;HeaderLinksSlot;Header links;true;contactInfo
    ;;MiniCartSlot;Mini Cart;true;MiniCart
    ;;NavigationBarSlot;Navigation Bar;true;DigitalCamerasBarComponent,FilmCamerasBarComponent,HandheldCamcordersBarComponent,PowerSuppliesBarCompon
    ;;FooterSlot;Footer;true;FooterNavigationComponent

    # content slots used for the home page
    INSERT_UPDATE ContentSlot;$contentCV[unique=true];uid[unique=true];name;active;cmsComponents(&componentRef)
    ;;Section1Slot-Homepage;Section1 Slot for Homepage;true;BrandsGaloreBanner
    ;;Section2ASlot-Homepage;Section2A Slot for Homepage;true;CanonPowershotBannerComponent,SonyCybershotBannerComponent,KodakEasyshareBannerCompone
    ;;Section2BSlot-Homepage;Section2B Slot for Homepage;true;CamcordersBanner,LensesBanner
    ;;Section2CSlot-Homepage;Section2C Slot for Homepage;true;ElectronicsHomepageCarouselComponent
    ;;Section3Slot-Homepage;Section3 Slot for Homepage;true;ElectronicsHomepageProductCarouselComponent,PurchasedCategoryCrossSelling
    ;;Section4Slot-Homepage;Section4 Slot for Homepage;true;DigitalCompactCamerasBannerComponent,DigitalSLRCamerasBannerComponent,MemoryBannerCompon
    ;;Section5Slot-Homepage;Section5 Slot for Homepage;true;NextDayDeliveryBanner
```

For more information, see ImpEx Syntax .

ContentSlotForTemplate Definition

In the next step, content slots are assigned to a template in a previously named position. Typically, content slots that can be reused across multiple pages are assigned to the page template:

```
INSERT_UPDATE ContentSlotForTemplate;$contentCV;uid[unique=true];position[unique=true];pageTemplate(uid,$contentCV)[unique=true][default='LandingPag
    ;;SiteLogo-LandingPage2;SiteLogo;;SiteLogoSlot;true
    ;;HomepageLink-LandingPage2;HomepageNavLink;;HomepageNavLinkSlot;true
    ;;NavigationBar-LandingPage2;NavigationBar;;NavigationBarSlot;true
    ;;MiniCart-LandingPage2;MiniCart;;MiniCartSlot;true
    ;;Footer-LandingPage2;Footer;;FooterSlot;true
    ;;HeaderLinks-LandingPage2;HeaderLinks;;HeaderLinksSlot;true
```

ContentSlotForPage Definition

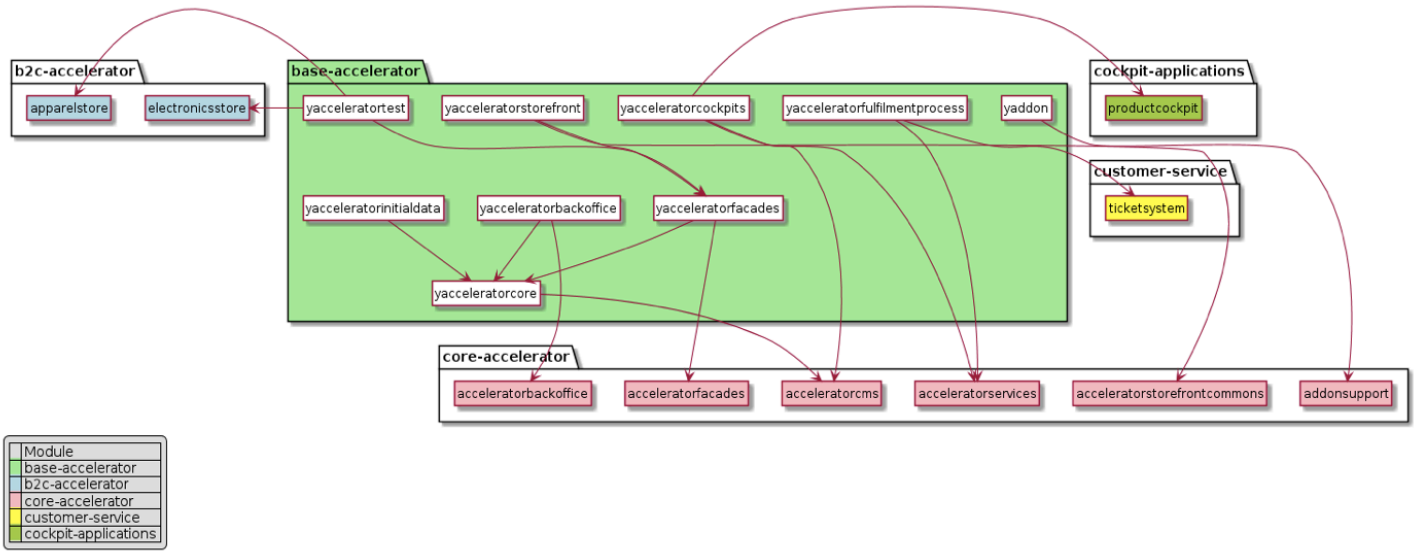
Finally, specific content slots are assigned to the page:

```
INSERT_UPDATE ContentSlotForPage;$contentCV[unique=true];uid[unique=true];position[unique=true];page(uid,$contentCV)[unique=true][default='homepage'
    ;;Section1-Homepage;Section1;;Section1Slot-Homepage
    ;;Section2A-Homepage;Section2A;;Section2ASlot-Homepage
    ;;Section2B-Homepage;Section2B;;Section2BSlot-Homepage
    ;;Section2C-Homepage;Section2C;;Section2CSlot-Homepage
    ;;Section3-Homepage;Section3;;Section3Slot-Homepage
    ;;Section4-Homepage;Section4;;Section4Slot-Homepage
    ;;Section5-Homepage;Section5;;Section5Slot-Homepage
```

Base Accelerator Architecture

The Base Accelerator module is a set of extensions that includes all the template extensions for extending Accelerator services and functionality.

Dependencies



Recipes

For a complete list of SAP Commerce recipes that may include this module, see Installer Recipes.

For a complete list of the SAP Commerce Cloud, integration extension pack recipes that may include this module, see Installer Recipe Reference.

Extensions

The Base Accelerator module consists of the following extensions:

- yacceleratorcockpits Extension
- yacceleratorcore Extension
- yacceleratorfacades Extension
- yacceleratorinitialdata Extension
- yacceleratorfulfilmentprocess Extension
- yacceleratorstorefront Extension
- yacceleratortest Extension
- yaddon Extension
- yacceleratorbackoffice Extension

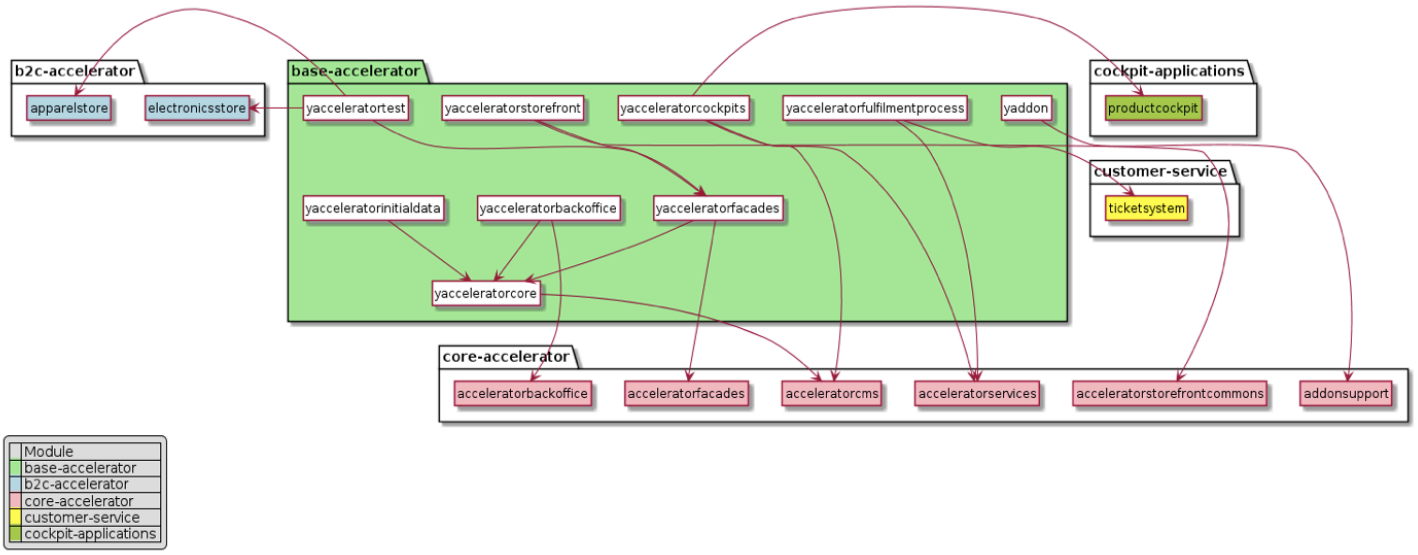
yacceleratorbackoffice Extension

The `yacceleratorbackoffice` extension provides an interface to configure the features found within the Base Accelerator Module.

About the Extension

Name	Directory	Related Module
`yacceleratorbackoffice`	`bin/modules/base-accelerator`	Base Accelerator Module

Dependencies



Related Information

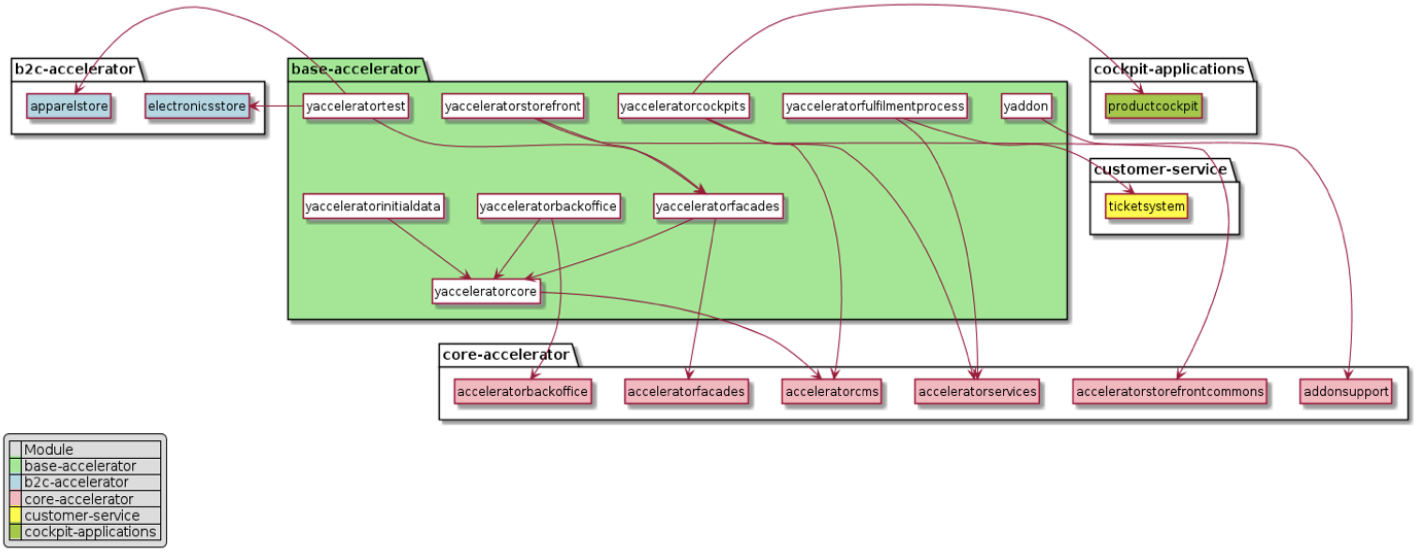
Extensions and AddOns

yacceleratorcockpits Extension

The `yacceleratorcockpits` extension is a template extension that demonstrates how a project can apply software customization to one or more existing cockpits without needing to create a project specific versions of a cockpit using the `ycockpit` template.

> ⚠ **Caution**
> This page refers to deprecated software. For further details, see Deprecation Status.

Dependencies



Spring Beans Redefinition

One of the purposes of the `yacceleratorcockpits` extension is to put all Spring bean redefinitions that are used in the SAP Commerce cockpits in one place. There are two kinds of Spring bean redeclarations in the `yacceleratorcockpits` extension.

Global Spring Context

The first one is in the `yacceleratorcockpits-spring.xml` file. Here you can redefine beans that are then used in all cockpit extensions. A simple example that overrides `calculationService` can be as shown in the excerpt from the `yacceleratorcockpits-spring.xml` file:

```
<alias name="defaultImpersonatingCalculationServiceWrapper" alias="calculationService"/>
                                    <bean id="defaultImpersonatingCalculationServiceWrapper" class="de.hybris.platform.yacceleratorcockpits.csco
                                    <property name="calculationService" ref="defaultCalculationService"/>
                                    <property name="acceleratorImpersonationService" ref="acceleratorImpersonationService"/>
                                    </bean>
```

The name of this file is defined in the `project.properties` file:

```
# Specifies the location of the spring context file put automatically to the global platform application context.
                                    yacceleratorcockpits.application-context=yacceleratorcockpits-spring.xml
```

Web Spring Context

Second kind of Spring beans redeclaration is related to a specific cockpit extension. For example you can redefine product perspective used in the SAP Commerce `productcockpit` extension. Here is a simple example from the `productocockpit-web.xml` file:

```
<bean id="ProductPerspective" class="de.hybris.platform.productcockpit.session.impl.ProductPerspective" scope="session"
                                    parent="BasePerspective">
                                    <property name="uid" value="productcockpit.perspective.product" />
                                    <property name="label" value="perspective.product" />
                                    <property name="customCsaURI" value="/productcockpit/productCSA.zul" />
                                    <property name="infoBoxTimeout" value="2000" />
                                    (...)
                                    </bean>
```

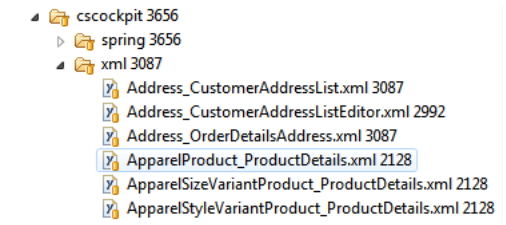
Second step is to put this filename to the `project.properties` file, so it is included to the specified extension Spring configuration:

```
# Inject additional config into the productcockpit
                                    productcockpit.additionalWebSpringConfigs=classpath:/yacceleratorcockpits/productcockpit/spring/productocock
```

Now the redeclared `ProductPerspective` bean is used in the Product Cockpit.

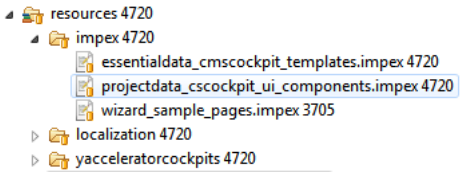
Cockpit UI Configuration Redefinition

In the `yacceleratorcockpits` extension you can also redefine cockpit configurations that are imported to the database and used in cockpit extensions. It is imported in the default way, to keep the clear folder structure in the `yacceleratorcockpits` extension. Find a sample structure of cockpit configuration that you could import here:



For more information, see How to Import Cockpit Configuration.

You need to import the files then. You can use the import by name convention using correctly named and placed ImpEx files:



In the `projectdata_cscockpit_ui_components.impex` file, you need to import all required configuration files. Below is excerpt from the sample ImpEx file, `projectdata_cscockpit_ui_components.impex`:

```
$defaultCatalog=Default
                                    $defaultCatalogVersion=Staged
                                    $catalogVersion=catalogVersion(catalog(id[default='$defaultCatalog']),version[default='$defaultCatalogVersion'])[uni
                                    $media=media(code, $catalogVersion);

                                    $jarResource=jar:de.hybris.platform.yacceleratorcockpits.constants.YAcceleratorCockpitsConstants&/yacceleratorcockpi

                                    #
                                    # Load medias
                                    #
                                    INSERT_UPDATE Media;code[unique=true];$catalogVersion;mime;realfilename;@media[translator=de.hybris.platform.impex.j

                                    ;Acc_Customer_NewCustomerDetails_ui_config;;text/xml;Customer_NewCustomerDetails.xml;$jarResource/xml/Customer_NewCu

                                    #
                                    # Setup cockpit configuration
                                    #
                                    INSERT_UPDATE CockpitUIComponentConfiguration;code[unique=true];factoryBean;objectTemplateCode[unique=true];principa

                                    ;newCustomerDetails;editorConfigurationFactory;Customer;cockpitgroup;Acc_Customer_NewCustomerDetails_ui_config;
                                    ;editCustomerDetails;editorConfigurationFactory;Customer;cockpitgroup;Acc_Customer_EditCustomerDetails_ui_config;
```

For more information, see Impex Syntax

Overriding or Adding Other Files to Cockpits

Another thing that you can do using the SAP Commerce `yacceleratorcockpits` extension is to override or add front-end files that are copied during build process to the destination extensions. To do this, modify the `buildcallbacks.xml` file of the `yacceleratorcockpits` extension. In this file you need to find macro called `yacceleratorcockpits_after_build` and put there things you want to copy. Find a simple example that copies all files from the `/resources/yacceleratorcockpits/cmscockpit/zul` folder to the `cmscockpit/web/webroot` folder:

```
<echo level="info" message="Start copying custom cmscockpit files to cmscockpit..."/>
                                    <copy todir="${ext.cmscockpit.path}/web/webroot">
                                    <fileset dir="${ext.yacceleratorcockpits.path}/resources/yacceleratorcockpits/cmscockpit/zul">
```

```
                                <include name="**/**.*" />
                                </fileset>
                                </copy>
```

After building the system with Ant, the files are copied to the destination folder. The `acceleratorcockpits_after_build` macro is also used to copy or override the `cmscockpit` extension images and icons. The code in the code snippet from the `buildcallbacks.xml` file shown below is an example for such an action:

```
<copy todir="${ext.cmscockpit.path}/web/webroot/cmscockpit/images">
                                <fileset dir="${ext.yacceleratorcockpits.path}/resources/yacceleratorcockpits/cmscockpit/images">
                                <include name="**/**.*" />
                                </fileset>
                                </copy>
```

Customization of the cscockpit Extension

The `yacceleratorcockpits` extension contains customization made to the `cscockpit` extension. Both Spring configuration and cockpit UI configurations were extended and new widgets and controllers as well as Java classes were added.

Cockpit UI Configuration

Cockpit component configuration was added to the **cockpitgroup** user group for several item types. New configuration files are placed in the `/resources/yacceleratorcockpits/cscockpit/xml` folder and are imported in the `projectdata_cscockpit_ui_components.impex` file. This ImpEx file is imported automatically by file name convention.

Spring Beans Customization

There are several new Spring configuration files introduced in the `yacceleratocockpits` extension for the `cscockpit` extension. These files are placed in the `/resources/yacceleratorcockpits/cscockpit/spring` folder and imported by inclusion in the `import.xml` file that is referenced in the `project.properties` file. Here is a short description of new Spring configuration introduced for the `cscockpit` extension:

- `cscockpit-web.xml`: A new `messageSource` bean defined that overrides existing one and joins message sources from the original `cscockpit` extension and in the `yacceleratorcockpits` extension.
- `cscockpit-services.xml`: In this file, only the `addressModelLabelProvider` bean is redefined.
- `cscockpit-controllers.xml`: This file contains several bean redefinitions including redefinition of the `csCallContextController` that is extended to use `baseStoreSelectorStrategy`.
- `cscockpit-widgets.xml`: Several renderer bean implementations were redefined in this file, as well as the `csProductUrlStrategy`.

cmscockpit Extension Customization

Spring Bean Customization

The `/resources/yacceleratorcockpits/cmscockpit/spring/cmscockpit-wizards.xml` file redefines the `cmsSiteWizard` Spring bean. Other beans defined in this file are related to the new defined wizard. This wizard extends default CMS site wizard with new attributes defined for the `cmsSite` type in the `commerceservices` extension, that are `theme`,`defaultLanguage` and `locale`.

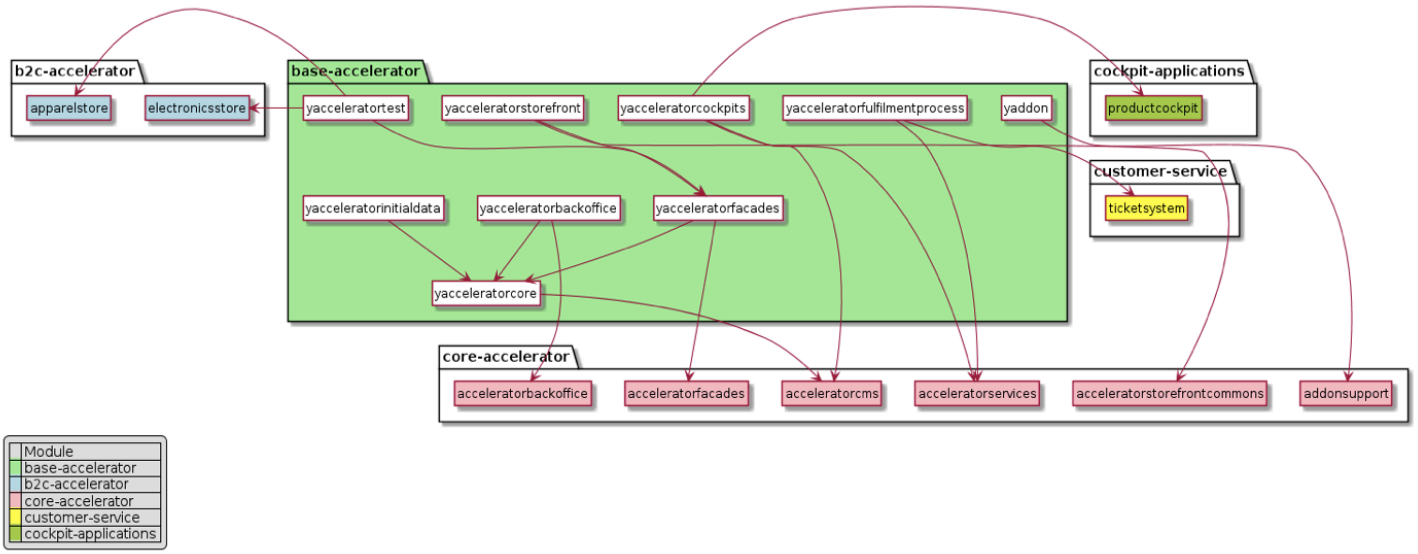
yacceleratorcore Extension

The `yacceleratorcore` extension is the template business layer extension for your project. You can use this extension to extends or add further services, as well as extend the SAP Commerce data model to fit your project's requirements.

A project starts by either using the `extgen` or the `modulegen` tool to create a copy of this extension, with project-specific naming and packaging.

> **Note**
> An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Dependencies

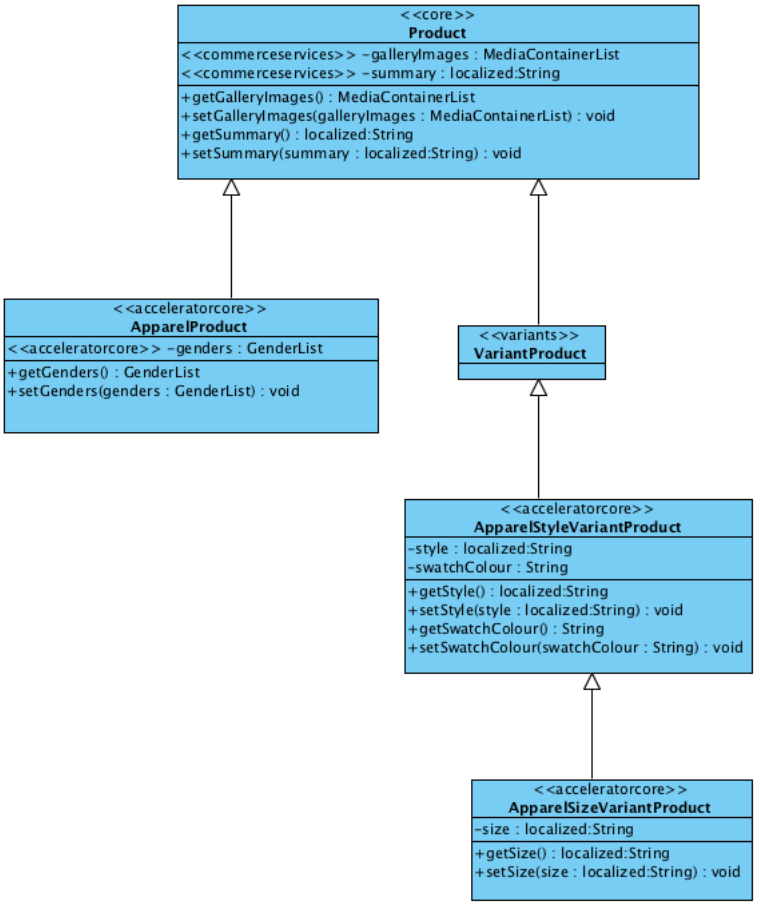


Apparel Products

The `yacceleratorcore` extension is used to extend the product data model. It supports apparel products by adding the concept of variants for style and for size.

The recommended modeling approach is a "variant of a variant" model, which is organized as follows:

- The `ApparelProduct` extends the `Product`, and adds a gender attribute that enables you to indicate if a product is suited only to a certain gender.
- The `ApparelStyleVariantProduct` extends the `VariantProduct`, and allows you to indicate different style variants, such as a t-shirt that comes in red or green.
- The `ApparelSizeVariantProduct` extends the **ApparelStyleVariantProduct**, and allows you to indicate different size variants, such as a red t-shirt that comes in size medium and size large.



This approach allows you to set promotions and cross-sells at the style level rather than having to set them up for every size or color variation. It is also still possible to set up size level promotions and cross-sells. And if a product has just one size, it is possible to create a direct link to the size variant, since it extends the style variant, and all the style attributes are available.

Related Information

Customizing the Accelerator with extgen and modulegen

yacceleratorcore Extension Related Data

The data is imported in two stages, related to the extension they are provided in. Data provided with each extension is of two kinds: essential and project. The first extension that provides the data is the `yacceleratorcore` extension.

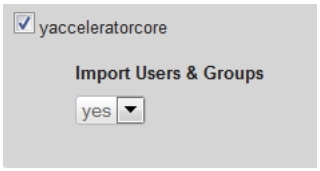
yacceleratorcore Extension Essential Data

Data imported at this stages includes:

Filename	Contents
`essential-data.impex`	• Languages • Currencies • Titles • Vendors • Warehouses • Supported Credit/Debit cards • User Groups • DistanceUnits for Storelocator
`countries.impex`	Full country list conforming to ISO 3166-1 alpha-2
`delivery-modes.impex`	Example delivery modes to be replaced/changed as required
`themes.impex`	CSS themes

yacceleratorcore Extension Project Data

Project data setup for the `yacceleratorcore` extension provides the following option:



These customizations are created in XML configuration files stored in `resources/yacceleratorcore-config` directory.

Import Users and Groups load:

- Users and groups for the WCMS Cockpit and Product Cockpit.
- Access rights for the cockpit users and groups.

yacceleratorfacades Extension

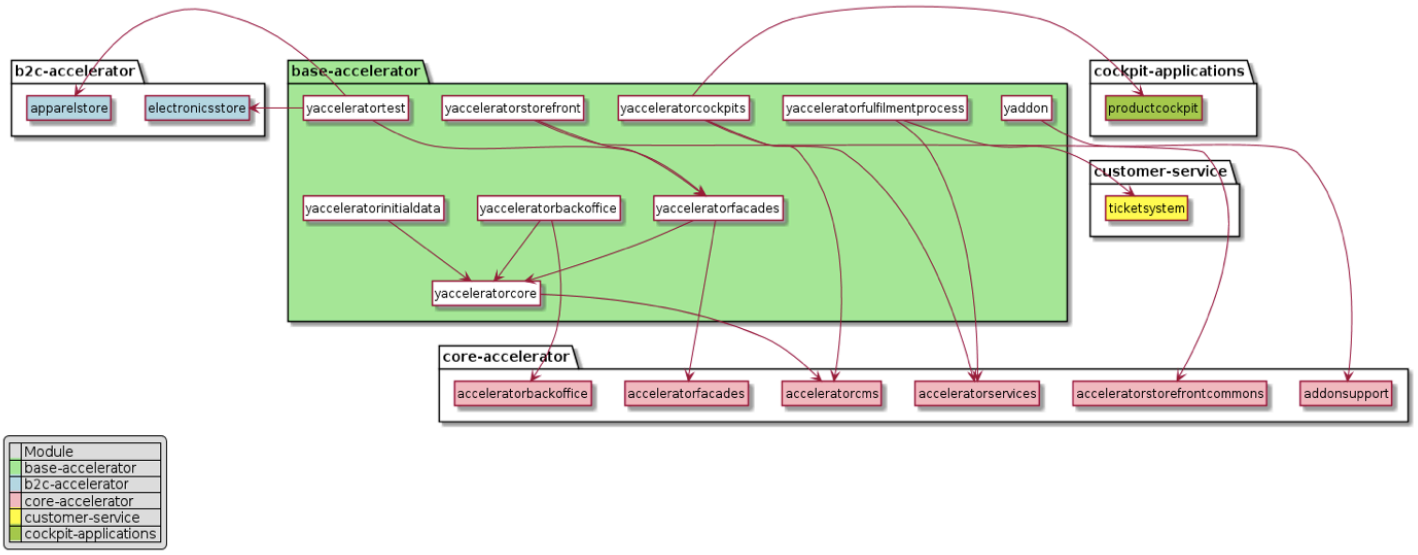
The `yacceleratorfacades` extension is a starting point template for a project to extend the functionality provided by the `commercefacades` extension. It enables you to add additional project-specific facades, as well as extending or adding further data objects, customizing, or adding new converters and adding additional populators.

Out of the box, the `yacceleratorfacades` extension is quite light. It provides some examples of how to extend `commercefacades` with some extended Data Objects, Converters, Populators and a new Facade. Lastly, it provides an extension point for the email generation processes to inject facade data object context variables into the velocity context used when generating emails.

As with all template extensions of the SAP Commerce Accelerator, a project would first generate a copy of the extension using `modulegen`.

ℹ Note
An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Dependencies



Apparel

`ProductFacade` from the `commercefacades` extension is extended to include additional support for the `ApparelProduct` provided by `yacceleratorcore` extension template. This includes:

- Extending and replacing the default `ProductConverter`
- Supporting the handling of Apparel Size and Style variants with an extended `VariantOptionDataConverter`
- Adding a new Data Object type `GenderData` with counterpart converter and Populator.

These serves as providing an example as to how to extend `commercefacades`, as well as providing the adaptations necessary to expose the data required for the `yacceleratorstorefront` Apparel front end storefronts.

For more information on related resources, see:

- yacceleratorcore extension Technical Guide section `ApparelProducts`
- How To Extend the ProductFacade

SimpleSuggestionFacade

The `SimpleSuggestionFacade` is used by the `yacceleratorstorefront` to render the content required for the `PurchasedProductReferences` CMS component. It is an example of how to add an additional standalone facade to complement those provided out of the box with the `commercefacades` extension.

Email

Emails are generated asynchronously using the business process email functionality provided in the `commerceservices` extension. Like a storefront, much of the content of the email will be generated using the facade layer. Prototype beans are set up in the application context to configure which facade data objects are injected into the velocity context for rendering of email content.

For each email Accelerator generates, the `yacceleratorfacades` layer provides a context type extending the `AbstractEmailContext` type. During the email generation process, the `DefaultEmailContextFactory` in `acceleratorservices` retrieves the name of the email context type from the renderer template, which holds the velocity script referencing the email context object, associated with the email template. Then it tries to get the bean instance of the email context type from the application context.

The `CustomerEmailContext` type populates customer data into email context. The `ForgottenPasswordEmailContext` adds details like token in addition to customer data, and the `OrderNotificationEmailContext` populates order data into the email context. The data objects that get generated and injected into the email context is exactly the same data objects generated for storefront. This makes the scripting of velocity template to generate the email text is simpler, and relatively similar to that of a web page that uses the same data objects.

The following code excerpt from the `acceleratorservices-spring.xml` file is the `AbstractEmailContext` prototype bean defined in the `acceleratorservices` extension:

```
<bean id="abstractEmailContext" class="de.hybris.platform.acceleratorservices.process.email.context.AbstractEmailContext" abstract="true" scope="pro
                                <property name="customerEmailResolutionService" ref="customerEmailResolutionService"/>
                                <property name="siteBaseUrlResolutionService" ref="siteBaseUrlResolutionService"/>
                                <property name="configurationService" ref="configurationService"/>
                                </bean>
```

And the three email context prototype beans defined in `yacceleratorfacades`extension are shown below in an excerpt from the `yacceleratorfacades-spring.xml` file:

```
<bean id="customerEmailContext" class="de.hybris.platform.yacceleratorfacades.process.email.context.CustomerEmailContext" parent="abstractEmailConte
                                <property name="customerConverter" ref="customerConverter"/>
                                </bean>

                                <bean id="forgottenPasswordEmailContext" class="de.hybris.platform.yacceleratorfacades.process.email.context.Forgott
                                <property name="customerConverter" ref="customerConverter"/>
                                </bean>

                                <bean id="orderNotificationEmailContext" class="de.hybris.platform.yacceleratorfacades.process.email.context.OrderNo
                                <property name="orderConverter" ref="orderConverter"/>
                                </bean>
```

Related Information

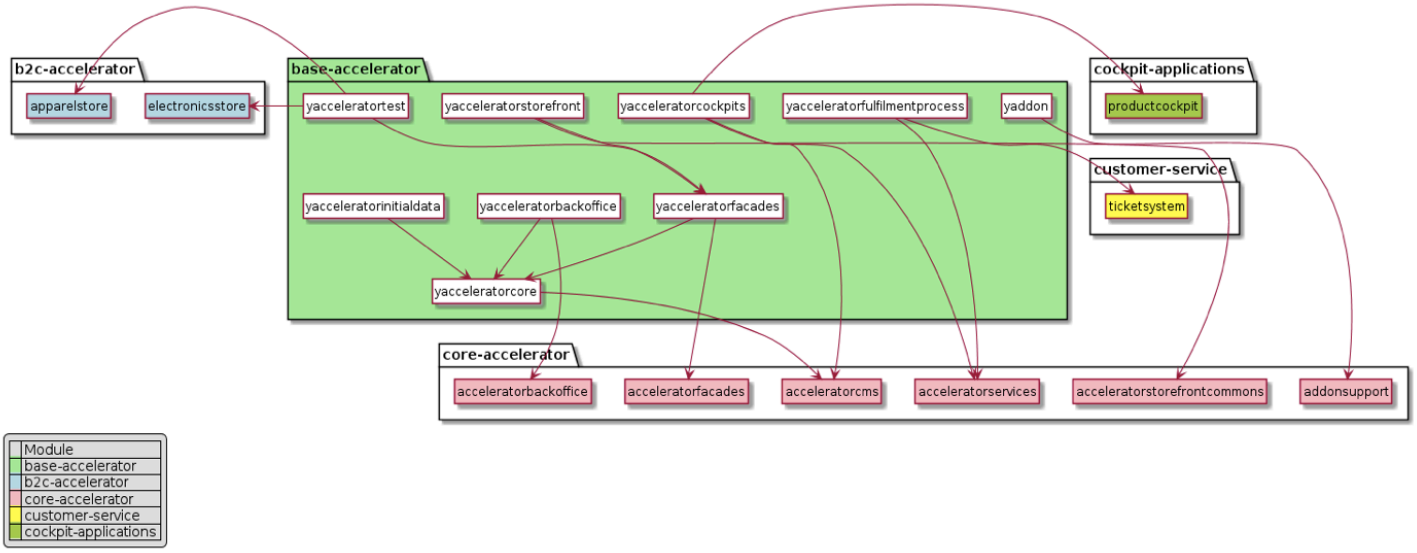
yacceleratorinitialdata Extension

The `yacceleratorinitialdata` extension provides a framework for your B2C store and sites. It also provides an option to import the sample data provided with the `apparelstore` or `electronicsstore` extension.

This `yacceleratorinitialdata` extension is a template extension designed to be used with `modulegen`.

Note
An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Dependencies



Purpose

The `yacceleratorinitialdata` extension provides:

- A location for your customized system setup process.
- A skeleton ImpEx structure for your stores and sites.
- Some handy usergroup configurations in ImpEx files.
- Provides events listeners that allows you to import core and sample data located in sample stores extensions like `electronicsstore` or `apparelstore` extensions when included in your build

After running `modulegen` for your new SAP Commerce Accelerator project, the generated version of this template extension is the first area you customize. You customize the `Setup` class and add your own data to the ImpEx structure.

Impex Data Configuration

The initial data extension contains template ImpEx files for importing your data. The following data is configured:

- User groups and default users for the business tools
  - CMS Cockpit, CS Cockpit, Product Cockpit
- Your Store
  - Product Catalog
  - Products / Variants
    - Media
    - Prices
    - Stock
    - Related Products
    - Product Reviews
  - Categories
    - Classification Categories
  - Promotions
  - Solr Search
  - Store Locations
- Your CMS Site

- CMS Content
- Email Content

The template ImpEx files contain the required macros and ImpEx header lines but contain no data lines. You need to add in the data lines to match your required data.

Relation to Sample Data

The `electronicsstore` and `apparelstore` extensions are based on the initial data ImpEx templates and provides examples of the data lines that you need to add to the initial data ImpEx files. When you are updating your initial data templates you should review the equivalent files in the `electronicsstore` or `apparelstore` extensions.

InitialDataSystemSetup class

This class is provided as a place to add your own custom setup process. This hooks into the SAP Commerce Initialization Process via the annotations described in Hooks for Initialization and Update Process document.

For additional information, see Initializing and Updating SAP Commerce.

Initialization Options

To change the options shown on the SAP Commerce **Init/Update** screen in the SAP Commerce Administration Console, you can edit the following method:

```
/**
	 * Generates the Dropdown and Multi-select boxes for the project data import
	 */
	@Override
	@SystemSetupParameterMethod
	public List<SystemSetupParameter> getInitializationOptions()
	{
		final List<SystemSetupParameter> params = new ArrayList<SystemSetupParameter>();

		params.add(createBooleanSystemSetupParameter(IMPORT_CORE_DATA, "Import Core Data", true));
		params.add(createBooleanSystemSetupParameter(IMPORT_SAMPLE_DATA, "Import Sample Data", true));
		params.add(createBooleanSystemSetupParameter(ACTIVATE_SOLR_CRON_JOBS, "Activate Solr Cron Jobs", true));
		// Add more Parameters here as you require

		return params;
	}
```

Out of the box, this is just a placeholder for adding options to the data process you want to put in place.

The `createBooleanSystemSetupParameter` method is a helper method designed to assist you in creating `Yes/No` options.

Add additional `SystemSetupParameters` to the list to modify the options presented in AdminWeb.

Essential Data

Modify the following method to add actions to the essential data step:

```
 screen in the hybris@SystemSetup(type = Type.ESSENTIAL, process = Process.ALL)
public void createEssentialData(final SystemSetupContext context)
{
	// Add Essential Data here as you require
}
```

These actions should be able to be re-executed safely.

Project Data

Modify the following method to create your own initialization process:

```
/**
	 * Implement this method to create data that is used in your project. This method will be called during the system
	 * initialization.
	 *
	 * @param context the context provides the selected parameters and values
	 */
	@SystemSetup(type = Type.PROJECT, process = Process.ALL)
	public void createProjectData(final SystemSetupContext context)
	{
		/*
		 * Add import data for each site you have configured
		 *
		 * final ImportData sampleImportData = new ImportData();
		 * electronicsImportData.setProductCatalogName(SAMPLE_PRODUCT_CATALOG_NAME);
		 * electronicsImportData.setContentCatalogNames(Arrays.asList(SAMPLE_CONTENT_CATALOG_NAME));
		 * electronicsImportData.setStoreNames(Arrays.asList(SAMPLE_STORE_NAME));

		 * getCoreDataImportService().importData(context, sampleImportData);
		 * getEventService().publishEvent(new CoreDataImportedEvent(context, Arrays.asList(sampleImportData)));

		 * getSampleDataImportService().importData(context, sampleImportData);
		 * getEventService().publishEvent(new SampleDataImportedEvent(context, Arrays.asList(sampleImportData)));
		*/
	}
```

To add a new store with a number of sites, you should add an `ImportData` object. This object holds your store names, your product catalog name, and your content catalog name. Then this object should be sent as an input for `CoreDataImportService` to import core data and `SampleDataImportService` to import sample data.

You also need to place your data into the provided ImpEx skeleton. The following process is recommended:

- Rename the `import/store/storeName` directory to match your new store. For this example we assume that the store is called `megastore`.

    - You should update the ImpEx macros named `$storeName` to `$storeName=megastore`.
    - You should name your product catalog `megastoreProductCatalog` and update the ImpEx macros named `$productCatalog` to `$productCatalog=megastoreProductCatalog`.
- Rename the `import/store/storeName/siteName` directory to match your new site, a suggested name for the UK version of the site would be `megastoreUK`.
    - Copy and rename this directory for each site your store supports.
    - You should name your content catalog for this UK site `megastoreUKContentCatalog` and update the ImpEx macros named `$contentCatalog` to `$contentCatalog=megastoreUKContentCatalog`.
- To import your store and all your site call the `importStoreInitialData()` method passing:
    - `megastoreProductCatalog` as the `productCatalog` string.
    - `megastore` as the `storeName`.
    - A list of `siteNames`, perhaps including `megastoreUK`.
    - A list of `contentCatalogs`, perhaps including `megastoreUKContentCatalog`.

This enables you to import all of the data required and pushes this to the online version for you as well as populating SOLR.

You should now be able to begin adding real product and CMS data to the provided ImpEx skeleton. See Initial Data ImpEx Structure document.

Related Information

Hooks for Initialization and Update Process
Customizing the B2C Accelerator
Commerce B2C Accelerator

Initial Data ImpEx Structure

The initial data ImpEx files are designed to speed up creation of a store and allow you to easily add new sites. The files are based on the supplied sample data ImpEx files retaining macros and header lines only. You should populate only the files you require.

ImpEx Configured per Store

Categories

Add your category list and the structure of your categories to the `categories.impex` file. If you are using the classification system, then populate the `categories-classifications.impex` and `units.impex` files as well.

Add logos and other category-specific media to the `categories-media.impex` file.

Products

Place products in the `products.impex` file. If you are using variant products, place these in `variant-products.impex`.

Place product-specific media in the `products-media.impex` file and ensure that `MediaContainers` are used to hold varying image formats of the same media or images.

Cross sells and up sells and other relations between products should be added to the `products-relations.impex` file.

Stock levels, warehouses, and vendors are added through the `products-stocklevels.impex` file.

Prices for products go in the `products-prices.impex` file. If Products have a price, they are currently set to approved through this ImpEx file, but this can be changed if you require.

If you use the classification system, populate the `products-classifications.impex` file.

For more information, see basecommerce Extension.

Brands/Suppliers

Brands and/or suppliers are stored as categories in the data model, but are separately imported through the `suppliers.impex` file and the `suppliers-media.impex` file.

Promotions

Promotions are configured through the `promotions.impex` file, see promotions extension (Legacy) for more details.

Reviews

If you wish, you can pre-populate reviews, perhaps from an existing site, in the `reviews-impex` file.

Faceted Search Configuration

SAP Commerce Accelerator provides integration with Solr. SAP recommends you configure this through the `solr.impex` file. You can find example configurations in the `apparelstore` or `electronicsstore` extension.

ImpEx Configured per Site

CMS Content

Much of your work in building a CMS-driven site is done in `cms-content.impex` file, where you can add:

- Components that appear on every page, for example:
  - Site Logo
  - Mini Cart
  - Navigation Bar
  - Footer
- Product/Category restricted pages, such as landing pages for certain brands or categories.
- Banners
- Paragraphs
- Carousels
- Feature Components
- `ProductReferencesComponents` for displaying cross sells, accessories, and so on.

Email Content

You can configure the look and feel of emails sent to the client on registration or order placement through the `email-content.impex` file.

Points of Service

Your business locations in the real world can be added in the `store-pos.impex` file and the `store-media.impex` file, to make use of the SAP Commerce Accelerator Store Locator integration with Google Maps.

Language-Specific Files

Where there are localized attributes, these should be held in separate language specific files. English-localized files are provided. You can copy these and change the `$lang` macro to add more languages. Language files are automatically imported if present in the system.

Related Information

Using ImpEx with Backoffice or SAP Commerce Administration Console

yacceleratorfulfilmentprocess Extension

The `yacceleratorfulfilmentprocess` extension is a template extension that provides a customizable fulfillment process and is designed to support order management in SAP Commerce. This extension replaces the `fulfilmentprocess` template extension. It also integrates the features provided by the Commerce Services and the Accelerator.

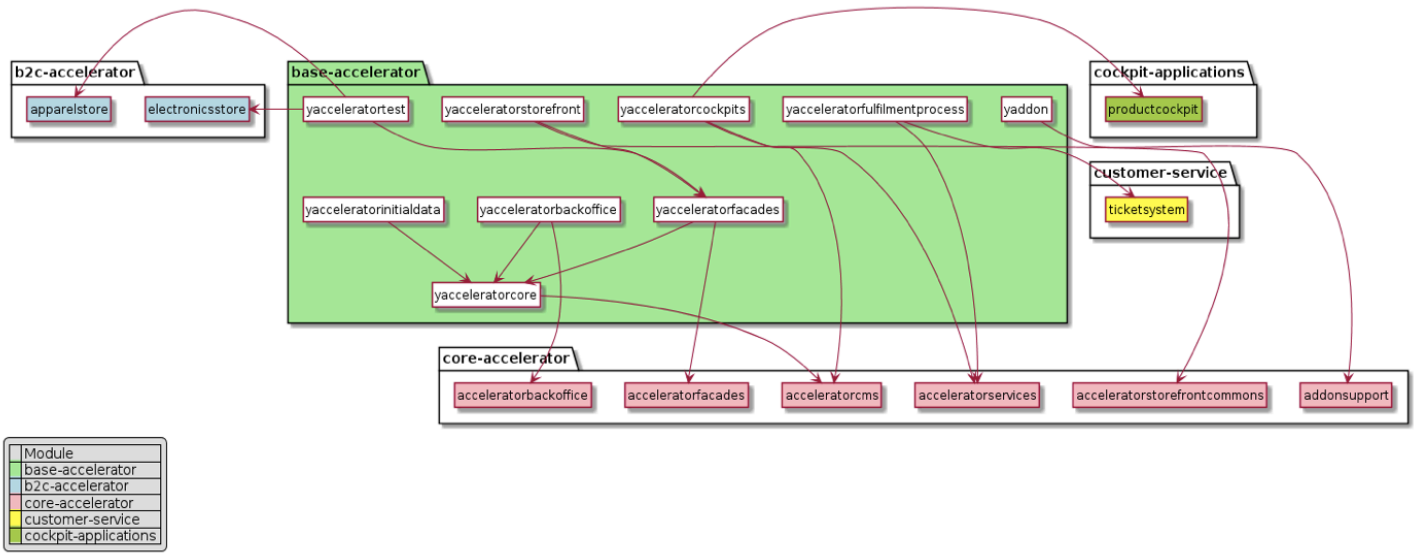
> **Note**
> An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

The `yacceleratorfulfilmentprocess` extension is designed to be used with `modulegen` and `extgen` to create the fulfillment and order management processes required for projects.

For additional information, see:

- Customizing the B2C Accelerator
- processing Extension

Dependencies



Generation

When generating from Accelerator, a specific extension based on the `yacceleratorfulfilmentprocess` extension is created. Begin by modifying the generated extension and not the template.

The `yacceleratorfulfilmentprocess` extension can also be used with `extgen`, as with other template extensions, to create a customized fulfillment process that you can use with or without the rest of Accelerator.

For more information, see:

- Customizing the B2C Accelerator
- Creating a New Extension

Note

The `yacceleratorcore` extension depends on this extension, and after generation the generated core depends on the generated fulfillment process.

Customization

The `yacceleratorfulfilmentprocess` uses the `processing` extension and so can be customized via XML configuration files. Accelerator supports the ability to specify the fulfillment process you wish to use through configuration of the `BaseStore`. In this way, a single process can be used to support all stores or tailored processes can be created for a specific store. For more information, see processing Extension.

By default, two processes are available: `order-process` and `consignment-process`. These processes are skeleton processes representing the high level steps that most fulfillment processes require. It is expected these will be changed before use. The B2B extensions provide additional processes for `Approval` and `Replenishment`.

Default Order Process

The default order process includes the following steps:

- Check the order for validity.
- Authorize and reserve the payment.
- Call fraud check.
- Optionally, allow a Customer Service agent to approve or deny an order.
- Capture the payment.
- Split the order into consignments.
- Start and wait on consignment processes.
- Complete the order.

Default Consignment Process

The default consignment process includes the following steps:

- Notify the Warehouse system of the consignment.
- Wait for updates from the Warehouse system.
- Send a `dispatch` and `ready for pickup` notification.
- Complete the consignment.

Order Splitting

The `yacceleratorfulfilmentprocess` extension uses functionality from the `basecommerce` extension to split an order into a number of consignments based on availability or delivery options.

New `OrderSplittingStrategies` can be added to customize this part of the process.

For additional information, see:

- basecommerce Extension
- Order Splitting

yacceleratorstorefront Extension

The `yacceleratorstorefront` extension is a template for a ready-to-be-adapted web frontend. It uses the Spring MVC.

Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

The `yacceleratorstorefront` template extension unites all the functions typically found in an online shop front end and interacts with the Backoffice Administration Cockpit and Product Cockpit using essential data from the `yacceleratorcore` extension and, optionally, sample store content from the `apparelstore` or the `electronicsstore` extension. It supports multiple store fronts served from the same web application, as well as splits by country, language, and currency. Each storefront has independently managed CMS content and product catalogs, both can be separated or shared between multiple storefronts.

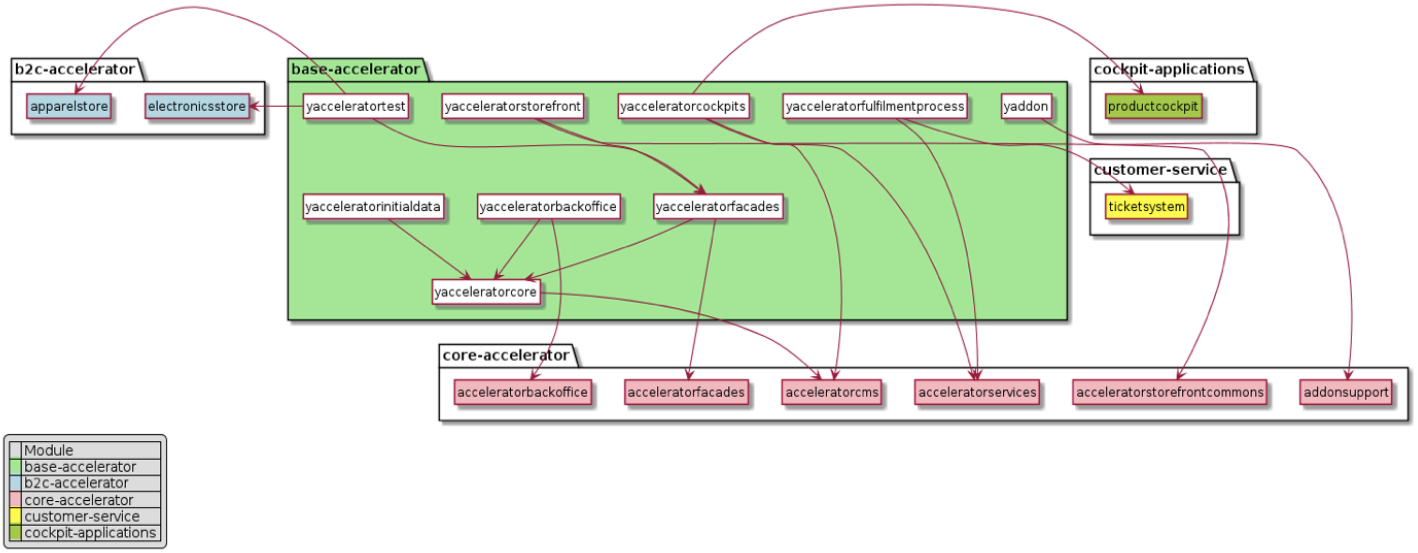
The `yacceleratorstorefront` extension is intended as a starting point to accelerate project development or as a cheat sheet for your own website. If getting live quickly is your aim, then using the site theme feature, advanced CMS site configuration, and flexibility to attach your site to a different search index greatly simplify your efforts. With these three features in place, it is possible to rebrand the current web store, add your own products and CMS content, and go live with few code modifications.

The `yacceleratorstorefront` extension uses the `commercefacades` extension to provide much of the storefront functionality exposed on the front end. This makes it much easier to share business logic and storefront functionality across multiple channels.

i Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Dependencies



Best Practice Storefront

The `yacceleratorstorefront` adopts best practice web application design principles. Like our storetemplate, mobiletemplate, and b2bstore, it is based on Spring MVC. No unproven bleeding edge technologies have been used. This ensures project developers have the greatest chance of immediate familiarity with the frameworks and the architecture when they initially take their first glance at the source code. Controllers are annotation based for fastest development turnaround and form objects use the javax validation framework. Controllers are free of business logic and lightweight due to the inclusion of a facade layer provided in separate extensions that exposes a single storefront API that cleanly separates the storefront from the business layer. This ensures projects have the greatest chance of reusing business logic with different front ends.

JSP and JSTL are used for the view with less HTML, more CSS policy for all UI component mark up to ensure the storefront's can be easily re-skinned with the `yacceleratorstorefronts` theming functionality. All pages have been designed to fit a standard 950 pixel wide Grid-Based canvas using the Blueprint CSS framework. Best practice examples are shown of how to use the SAP Commerce WCMS to dynamically include content all used throughout the site. All UI components and assets are cleanly organized into a logical folder structure. Tags have been used to UI functionality to components. Additionally, UI components are also broken up into Pages, Fragments and WCMS components using simple templating tags and the tags provided by the `cms2lib` extension. Message Resource Bundles have also been used to internationalize the storefront with an extra dimension of adapting content by theme and site. All content is internationalized, UTF-8 encoded, and has been tested end-to-end. The front end has also been accessibility tested.

The storefront is made secure using Spring Security and also has had a level of XSS testing applied. A Secure GUID cookie function as well as limited time link support has been added for extra security.

The storefront also provides Automated Smoke Test support with tools such as Selenium by introducing a tag that enables component ids to be output when the Smoke Test feature is enabled.

The storefront also demonstrates best practice integration with other SAP Commerce B2C commerce features not directly available through a business facade layer such as WCMS (pages, components, and preview).

Getting Started

Before starting development, you should first start your project by making a copy of this `yacceleratorstorefront` extension together with counterpart extensions from the `yaccelerator` module by using our extgen and modulegen functionality. For more information about extgen and modulegen tools, see Customizing the B2C Accelerator.

Related Information

Storefront Web Application Deconstructed
User Interface and Creatives: Responsive Experience
WCMS Integration
Search and Navigation
Store Locator Configuration
Top Navigation and Link Management
SEO URLs
Stock Management

Upgrading 3rd Party libs

It is highly recommended that you upgrade the used 3rd party libs to the latest version to eliminate vulnerabilities after generating your storefront from `yacceleratorstorefront`.

The list of used js 3rd libs are under `yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/responsive/lib/common`:

- enquire
- jquery
- jquery-ui
- jquery.blockUI
- jquery.colorbox
- jquery.currencies
- jquery.form
- jquery.hoverIntent
- jquery.waitforimages
- purify

And the 3rd js lib under `yacceleratorstorefront/web/webroot/WEB-INF/lib`:

- less-rhino
- lessc-rhino

The list of java 3rd party libs are under `yacceleratorstorefront/web/webroot/WEB-INF/lib`:

- jakarta.servlet.jsp.jstl
- jsoup
- rhino
- wro4j-core

Configuring Content Security Policy

This document instructs you on how to configure a Content Security Policy in `yacceleratorstorefront` and sort out all of the 3rd party resources that will be loaded by `yacceleratorstorefront` and all of the UI related functions in commerce. For example, Add-Ons. It is strongly recommended to configure this security header to prevent from XSS, code injection and click-jacking attacks.

Configuring Content-Security-Policy in `yacceleratorstorefront`

Add a property named `yacceleratorstorefront.xss.filter.header.Content-Security-Policy` in the `project.properties` file of `yacceleratorstorefront`. Listed below is an example that contains all of the resources that `yacceleratorstorefront` will load. If your storefront has not been integrated with any other UI related functions, the following property can work for you directly.

```
yacceleratorstorefront.xss.filter.header.Content-Security-Policy=default-src 'self';script-src 'self' 'unsafe-inline' 'unsafe-eval' *.google-analyti
```

If your storefront has been integrated with other UI related functions, the resources that will be loaded by the integrated UI related functions should be configured as well. Note that if the storefront and UI related functions need to load the same resources, the duplicate resources need to be de-duplicated. List below is an example that `yacceleratorstorefront` integrates with `captchaaddon`.

```
yacceleratorstorefront.xss.filter.header.Content-Security-Policy=default-src 'self';script-src 'self' 'unsafe-inline' 'unsafe-eval' *.google-analyti
```

In addition, if your storefront is generated based on the template `yacceleratorstorefront` and renamed, replace the property name to `<yourStorefrontApplicationName>.xss.filter.header.Content-Security-Policy`, then the Content Security Policy will take affect on your storefront.

Collection of the 3rd party resources

The following table sorts out all of the 3rd party resources that `yacceleratorstorefront` and all of the UI related functions in commerce will load. Refer to the content to configure your property `yacceleratorstorefront.xss.filter.header.Content-Security-Policy`.

Extensions	Directive Name	Directive Value
yacceleratorstorefront	default-src	'self'
	script-src	'self' 'unsafe-inline' 'unsafe-eval' *.google-analytics.com
	img-src	'self' *.google-analytics.com
	style-src	'self' 'unsafe-inline' https://fonts.googleapis.com
	font-src	'self' data: https://fonts.gstatic.com
captchaaddon	script-src	• https://www.google.com/recaptcha/api.js • https://www..gstatic.com/recaptcha/releases/a9s0j4pCVT6gaTEkLiFbtZPH/recaptcha__en.js
chinesestoreaddon	script-src	https://api.map.baidu.com/api
assistedservicestorefront	script-src	https://maps.googleapis.com/maps/api

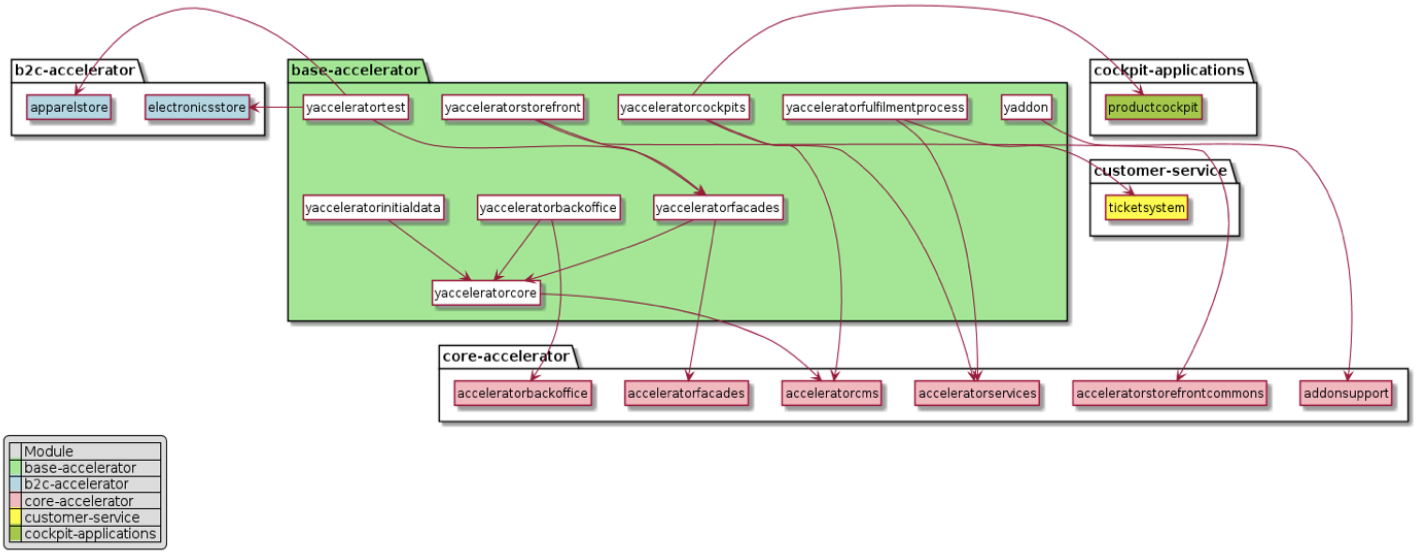
Extensions	Directive Name	Directive Value
	style-src	self' 'unsafe-inline' https://fonts.googleapis.com
sap-qualtrics	script-src	'self' 'unsafe-inline' 'unsafe-eval' *.qualtrics.com
sap-subscription-billing	script-src	'self' 'unsafe-inline' 'unsafe-eval' https://code.jquery.com
sap-customer-data-cloud	script-src	'self' 'unsafe-inline' 'unsafe-eval' *.gigya.com
smartedit	script-src	https://zn3fda1zkixglosts-sapinsights.siteintercept.qualtrics.com/
merchandisingaddon	connect-src	*.context.cloud.sap

yacceleratortest Extension

The `yacceleratortest` extension provides testing tools, configuration, and data for the SAP Commerce Accelerator.

The `yacceleratortest` extension is a template extension that is intended to be used with `modulegen`.

Dependencies



Test Data

By default, the system setup tasks for the `yacceleratortest` extension create some sample data that may be useful while testing your SAP Commerce Accelerator site.

```
@SystemSetup(type = Type.PROJECT, process = Process.ALL)
public void createProjectData(final SystemSetupContext context)
{
	if (getBooleanSystemSetupParameter(context, CREATE_TEST_DATA))
	{
		LOG.info("Creating Test Payment Subscriptions...");
		getAcceleratorTestOrderData().createPaymentInfos();

		LOG.info("Creating Test Orders...");
		getAcceleratorTestOrderData().createSampleOrders();
	}
}
```

Selecting Yes from the options creates sample `PaymentInfos` and sample `Orders`. This is selected by default. These are then accessible via your storefront extension, the Backoffice Administration Cockpit, and the Backoffice Customer Support Cockpit.

Add additional test data by extending the `AcceleratorTestOrderData` class and adding additional setup options.

Performance Test Scripts

The `yacceleratortest` extension offers a suite of JMeter test scripts that you can use for performance testing of your new storefront.

The scripts are used for internal performance testing of SAP Commerce Accelerator. The out-of-the-box sample data is used for testing.

Before you can use these to test your own sites you need to:

- Set up the test data, such as which categories, products, and users are used.
- Configure the environment settings so that you make requests using your load balancer, and so on.
- Rework any scripts where your processes were customized, such as having different steps to select a variant, for example.

Then you are able to proceed with the normal processes of baselining, testing,tweaking, and re-testing.

Configuring JMeter Test Scripts

The test plans are located in `modules/base-accelerator/yacceleratortest/resources/jmeter/`. The test plan for the B2C stores is `AcceleratorTestPlan.jmx`. The test plan for the B2B store is `B2BAcceleratorTestPlan.jmx`. The test results file is located in `base-accelerator/yacceleratortest/resources/jmeter/ <store> /results`.

You can configure the test data used, the relative priorities of individual tests, and the length and intensity of the test runs in the `user.properties` file.

You can use SAP Commerce sample data (included with the electronics, apparel, and powertools stores) to get to know the `yacceleratortest` extension, but it is also recommended to work with sample data that meets your specific performance testing needs.

The following may be useful in guiding your testing:

- Ensure you are using JMeter version 3.1 or higher for compatibility with the test plan.
- To run the test plan using JMeter, you need to use mySQL instead of hsqldb.
- The default configuration in `user.properties` is to run tests on the electronics store. You can modify `user.properties` to test other stores, or to update the thread amount, ramp-up time, and so on.
- The GUI mode can be used for updating the test plan, but tests should not be run in GUI mode. You can run tests from the command line using the following command: `jmeter -q user.properties -n -t AcceleratorTestPlan.jmx`
- You can start the GUI from the command line by removing `-n` from the command listed in the previous bullet.
- You should not aim to use more than 100 threads in each JMeter instance.
- If the server seems to be responding to a browser request much faster than the JMeter client is reporting, check that the client has enough resources.
- If you have issues running a test, check the `jmeter.log` file for error messages.

Configuring the user.properties File

The `user.properties` file has a lot of properties that allow you to configure the performance tests for your specific needs. Apart from generic JMeter configurations, such as number of threads, ramp up, and duration time, the following configurations are specific to SAP Commerce:

- sop/hop configuration
- number of pages to browse
- number of items to add to the cart
- percentage of users that browse only
- percentage of users that browse, and then proceed to the cart
- percentage of users that browse, proceed to the cart, and then check out

These properties are loaded in the jmx plan, and can be used to simulate how the site will behave with the expected conversion rates, for example.

One important property to be aware of is the `inputDataFolder`. It indicates where your data will be coming from. For each distinct scenario that requires input, there is a `.csv` file to simulate the user input. For example, `userInputData.csv` contains possible values that a user would input into the registration form. The `initializationSetup.csv` file is particularly relevant, as it sets the host, ports and path that JMeter threads will call. In a basic scenario, you have it pointing to a SAP Commerce server that simulates your expected production environment, while your JMeter sits on another server in the same data center, to avoid connection delays.

Related Information

http://jmeter.apache.org/

yaddon Extension

The `yaddon` extension template is a predefined extension that serves as a starting point for creating a new AddOn, with the most typical use-case being for customer-specific implementations.

⚠ Caution

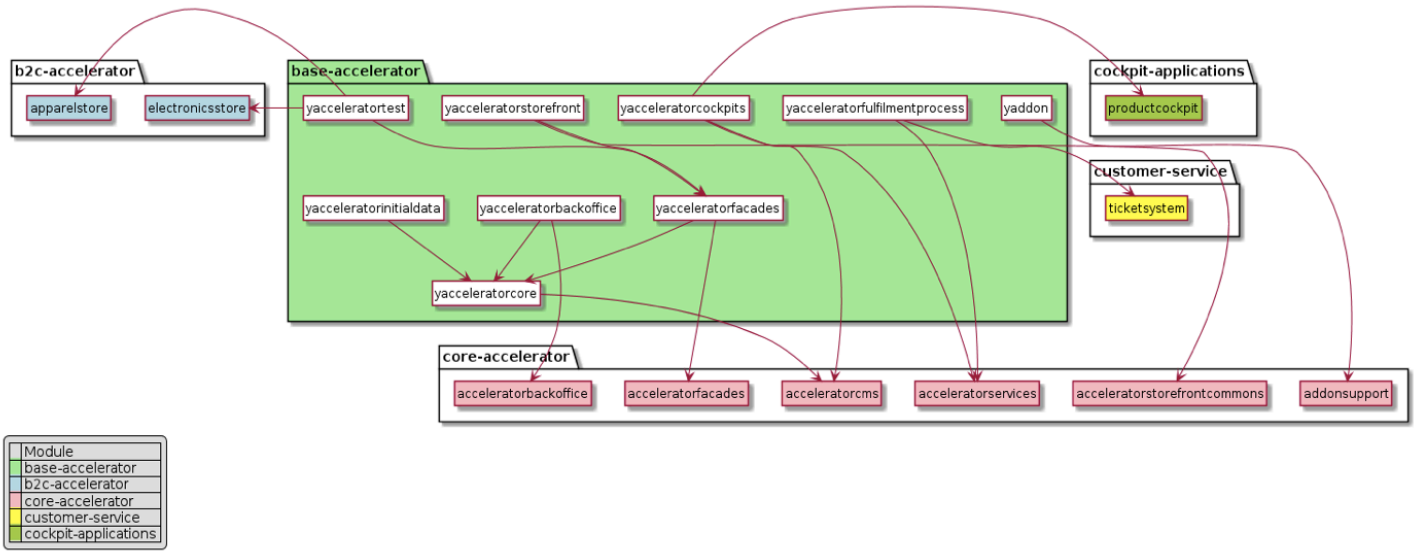
This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

The `yaddon` extension is essentially an empty extension with minimal implementations needed to create an AddOn. We recommend making a copy of the `yaddon` extension to use as your starting point.

i Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Dependencies



Overview

The `yaddon` extension acts as a template used for creating SAP Commerce AddOns. It is normally used in conjunction with the `yacceleratorstorefront` extension to extend storefront functionality. What makes this extension different from the `yempty` extension is the existence of the *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yaddon/acceleratoraddon/` folder, which contains resources that get copied over to its associated storefront extensions during build time.

Contents of the yaddon Extension

The following sections describe the contents of the `yaddon` extension.

Impex Files

The `yaddon` extension contains the following empty impex files:

- essentialdata_yaddon.impex
- projectdata_yaddon.impex

These files are copied to the generated AddOn and have the following convention:

```
essentialdata_{addonname}.impex and projectdata_{addonname}.impex
```

The files are automatically imported during update and initialization.

Template File for Properties

The `yaddon` extension also contains the `project.properties.template` file that is necessary for AddOns.

The only property that is present in this file is the following:

```
yaddon.application-context=yaddon-spring.xml
```

This is the property that describes which additional Spring file needs to be loaded in the Spring application context when the AddOn is activated.

Responsive Support

The `yaddon` extension includes the *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/yaddon/acceleratoraddon/web/webroot/WEB-INF/_ui-src` folder that contains LESS files used to generate responsive storefront pages. The `yaddon.less` file (located in `_ui-src\responsive\less`) is used to store relative import references to LESS files used by the AddOn (for example: `@import "buttons.less"`). The relative import references must be added manually in this file.

For more information on how responsive storefront pages are generated using LESS, see Responsive Website Build Process.

Creating a New yaddon Using the Extgen Tool

The following procedure describes how to create a new AddOn extension using the `extgen` tool. For more information about the `extgen` tool, see Creating a New Extension.

To create a new `yaddon` using the `extgen` tool:

1. Locate the `yaddon` template, which can be found in the *<HYBRIS_BIN_DIR>*`/modules/base-accelerator/` directory.
2. In the Terminal or Command Prompt window, navigate to the *<HYBRIS_BIN_DIR>*`/platform/extgen` directory, and invoke the `ant` command. If the `ant` executable is not placed in the system path, you need to add it there or invoke `ant` with the whole path to the `ant` executable.
3. When the `ant` script is started, you are prompted to input the following information:

- The name of your extension, which is the name of your newly created AddOn. Because you are creating a new AddOn-related extension, we recommend that you include `addon` at the end of the name, such as `testaddon`.
- The package of your extension, which is the package where all your classes are placed.
- The template for generation, which is `yaddon` in this case, since you want to create an AddOn-related extension. The following is an example:

```
[echo] Next steps:
[echo]
[echo] 1) Add your extension to your hybris/config/localextensions.xml
[echo]
[echo]      <extension dir="hybris/bin/custom/testaddon"/>
[echo]
[echo] 2) Please remove all template extensions (again) before you proceed.
[echo]
[echo] 3) Make sure the applicationserver is stopped before you build the extension the first time.
[echo]
[echo] 4) Perform 'ant' in your ${HYBRIS_BIN_DIR}/platform directory.
[echo]
[echo] 5) Restart the application server.
```

After the `extgen` tool has finished running, a new extension is generated, and you are then informed about the next steps that you can perform.

Building and Running the Platform with a New Extension

The following procedure describes how to make your new AddOn extension work.

1. Open your `localextensions.xml` file inside the `config` directory and add your new extension.
2. Stop the application server if it was running, and build the whole platform by invoking the `ant all` command inside the platform directory.
3. Install the new AddOn using the following command:

```
ant addoninstall -Daddonnames="<your_addon_name>" -DaddonStorefront.yacceleratorstorefront="yacceleratorstorefront"
```

4. Run `ant clean all`.
5. Start the application server.

AddOn Automatic Setup Services

The `yaddon` extension comes preconfigured with hooks into the system initialization and update process. A generated AddOn based on the `yaddon` template has hooks into the Core Data Setup as well as the Sample Data Setup.

The hook is made by listening to `CoreDataImportedEvent` events, which are fired when core data is finished importing, as well as `SampleDataImportedEvent` events, which are fired when sample data for the Accelerator has finished loading.

Hook into the Core Data Import

For core data, the following impex files are loaded in order:

```
/testaddon/import/productCatalogs/template/catalog.impex
/testaddon/import/contentCatalogs/template/catalog.impex
/testaddon/import/contentCatalogs/template/cms-content.impex
/testaddon/import/contentCatalogs/template/cms-content-mobile.impex
/testaddon/import/contentCatalogs/template/email-content.impex
/testaddon/import/stores/template/store.impex
/testaddon/import/stores/template/site.impex
/testaddon/import/solr/template/solr.impex
/testaddon/import/solr/template/solrtrigger.impex
```

ℹ Note

All scripts are parameterized with the following attributes, which are defined in `de.hybris.platform.commerceservices.impex.data.ImpexMacroParameterData`:

- $contentCatalog: content catalog ID
- $productCatalog: product catalog ID
- $siteUid: site ID
- $storeUid: IDof the store
- $channel: site channel (B2B or B2C)
- $ solrIndexedType: solr index type name
- $configExtensionName: name of the AddOn extension
- $addonExtensionName: name of the AddOn

Hook into the Sample Data Import

The following impex is loaded after a sample data event is triggered by the Accelerator system setup. All the impex will be created for you by the `extgen ant` command, based on the `yaddon` template.

The following example is for the `electronicsProductCatalog` and the `electronicsContentCatalog`.

```
/testaddon/import/common/user-groups.impex
/testaddon/import/cockpits/cmscockpit/cmscockpit-users.impex
/testaddon/import/cockpits/productcockpit/productcockpit-users.impex
/testaddon/import/cockpits/cscockpit/cscockpit-users.impex
/testaddon/import/common/common-addon-extra.impex

/testaddon/import/productCatalogs/electronicsProductCatalog/classifications-units.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/categories.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/categories-classifications.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/suppliers.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/suppliers-media.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/categories-media.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-media.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-classifications.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-relations.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-fixup.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-prices.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-stocklevels.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-pos-stocklevels.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-tax.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/users.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/products-addon-extra.impex
/testaddon/import/productCatalogs/electronicsProductCatalog/reviews.impex

/testaddon/import/contentCatalogs/electronicsContentCatalog/cms-content.impex
/testaddon/import/contentCatalogs/electronicsContentCatalog/cms-mobile-content.impex
/testaddon/import/contentCatalogs/electronicsContentCatalog/email-content.impex
/testaddon/import/contentCatalogs/electronicsContentCatalog/cms-addon-extra.impex

/testaddon/import/stores/electronics/store.impex
/testaddon/import/stores/electronics/site.impex
/testaddon/import/stores/electronics/points-of-service-media.impex
/testaddon/import/stores/electronics/points-of-service.impex
/testaddon/import/stores/electronics/points-of-service-addon-extra.impex

/testaddon/import/stores/electronics/btg.impex
/testaddon/import/stores/electronics/warehouses.impex

/testaddon/import/stores/electronics/solr.impex
/testaddon/import/stores/electronics/promotions.impex
```

Related Information

Customizing the B2C Accelerator
addonsupport Extension

Base Accelerator Implementation

You can create a custom Accelerator with the modulegen and extgen tools, and also extend and optimize Accelerator by configuring the sitemap, managing strategies for multi step checkout, adding new CMS pages, and much more.

Adding a New CMS Page Template

SmartEdit provides a central facility for managing content. CMS pages are laid out according to their page template. You can create a page template in SAP Commerce Accelerator.

How To Extend the ProductFacade

This document describes how to extend the `ProductFacade` functionality to customize it to fit the needs of the storefront.

JS, LESS and JSPs: Build Process for Generating Variables

The `yacceleratorstorefront` template provides a process to generate shared variables among JSPs, JavaScript and LESS files, based on one common properties file.

JSP Tag Scripts: Installing in an Accelerator Storefront

This document describes the mechanism for installing JSP tag scripts through an AddOn.

Just One Storefront

Modify your configuration to support a development installation for a project with one storefront.

Customizing Configurable Checkout

SAP Commerce Accelerator allows you to customize the checkout flow. The checkout framework enables each checkout step to be split into its respective controller, while the validation logic and the logic for transitioning to other checkout steps has been externalized.

Managing Multi-Step Checkout Strategies

The following is a brief overview of the different checkout strategies offered for customers to complete their purchases of the item they have added to their shopping cart. It mainly looks at how these strategies should be managed for the implementation required by your project. The expectation is that any project implemented using the SAP Commerce Accelerator removes the ability for the customer to decide their own checkout flow, and instead adapts the Checkout Strategy provided according to the Merchant's requirements.

SEO URLs

The SAP Commerce Accelerator allows you to create search-engine-friendly URLs for your website pages, which can help your website pages rank higher in search-engine results. It also produces URLs that are easy for your customers to read. This document provides instructions on implementing and customizing SEO URL generation.

Sitemap Configuration

Sitemaps allow webmasters to inform search engines about pages on their sites that are available for indexing. Accelerator supports sitemaps for different page types (such as product pages and category pages), as well as for different languages and currencies.

Storefront Web Application Deconstructed

SAP Commerce Accelerator provides reference storefronts that offer best-practice features for typical B2C and B2B uses.

Adding a New CMS Page Template

SmartEdit provides a central facility for managing content. CMS pages are laid out according to their page template. You can create a page template in SAP Commerce Accelerator.

In an SAP Commerce Accelerator project, a page template consists of the following:

- A list of content slots predefined for the template.
- A FrontendTemplateName that corresponds to the name of a JSP that drives the layout of the page.

- A list of PageTypes that the template is restricted to.
- A VelocityTemplate to specify the layout in SmartEdit.

To create a page template, you must do the following:

- Define a page template.
- Define content slots.
- Create a JSP for the page template.
- Create a velocity template.

You can create a Page Template using ImpEx files.

This trail also shows how to create a content Page Template, a slight variation on the current ContentPage1Template item.

You can also add a page template using the Backoffice Administration Cockpit (WCMS > Page Template).

Defining a Page Template

The Page Template is configured using the following ImpEx script:

```
INSERT_UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];name;frontendTemplateName;restrictedPageTypes(code);active[default=true]
;;ContentPage2Template;Content Page 2 Template;layout/contentLayout2Page;ContentPage
```

The presented Page Template is restricted to ContentPages and is bound to the JSP in the `WEB-INF/views/pages/layout/contentLayout2Page.jsp` file.

Define Content Slots

Set up the ContentSlotNames for the template:

```
INSERT_UPDATE ContentSlotName;name[unique=true];template(uid,$contentCV)[unique=true][default='ContentPage2Template'];validComponentTypes(code)
;SiteLogo;;CMSImageComponent,BannerComponent
;HeaderLinks;;CMSLinkComponent,CMSParagraphComponent
;MiniCart;;MiniCartComponent
;NavigationBar;;NavigationComponent
;Section1;;$wideContent
;Section2;;$wideContent
;Section3;;$narrowContent
;Footer;;CMSLinkComponent,CMSParagraphComponent,FooterNavigationComponent
```

Each ContentSlotName in the template has a list of permitted CMSComponent Types. These are restricted to only one component type for the header slots.

The list of ContentSlotForTemplate bindings for the PageTemplate is added as follows:

```
INSERT_UPDATE ContentSlotForTemplate;$contentCV;uid[unique=true];position[unique=true];pageTemplate(uid,$contentCV)[unique=true][default='ContentPag
;;SiteLogo-ContentPage2;SiteLogo;;SiteLogoSlot;true
;;HomepageLink-ContentPage2;HomepageNavLink;;HomepageNavLinkSlot;true
;;NavigationBar-ContentPage2;NavigationBar;;NavigationBarSlot;true
;;MiniCart-ContentPage2;MiniCart;;MiniCartSlot;true
;;Footer-ContentPage2;Footer;;FooterSlot;true
;;HeaderLinks-ContentPage2;HeaderLinks;;HeaderLinksSlot;true
```

In the code above, some slots are defined as shared content slots. `AllowOverwrite` controls the behavior of how the page is displayed in SmartEdit.

For details on configuring the Content Slots, see WCMS Integration.

Create PageTemplate JSP

Create the following JSP in the `acceleratorstorefront/web/webroot/WEB-INF/views/pages/layout/contentLayout2Page.jsp` file:

```
<%@ page trimDirectiveWhitespaces="true" %>
<%@ taglib prefix="template" tagdir="/WEB-INF/tags/template" %>
<%@ taglib prefix="cms" uri="/cms2lib/cmstags/cmstags.tld" %>
<%@ taglib prefix="breadcrumb" tagdir="/WEB-INF/tags/nav/breadcrumb" %>

<template:page pageTitle="${pageTitle}">
        <breadcrumb:breadcrumb breadcrumbs="${breadcrumbs}" />
        <cms:slot var="feature" contentSlot="${slots.Section1}">
                <div class="span-24 section1 advert">
                        <cms:component component="${feature}"/>
                </div>
        </cms:slot>
        <div class="span-20 section2 advert">
                <cms:slot var="feature" contentSlot="${slots.Section2}">
                        <cms:component component="${feature}"/>
                </cms:slot>
        </div>
        <div class="span-4 section3 advert last">
                <cms:slot var="feature" contentSlot="${slots.Section3}">
                        <cms:component component="${feature}"/>
                </cms:slot>
        </div>
</template:page>
```

Create Velocity Template

Create a Velocity Template that controls the layout of the template in SmartEdit. This step is not mandatory.

```
UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];velocityTemplate[translator=de.hybris.platform.yacceleratorcore.setup.FileLoaderValueTr
;;ContentPage2Template;$jarResource/yacceleratorcore/import/cmscockpit/structure-view/structure_contentPage2Template.vm
```

The following file, `structure_contentPage2Template.vm`, is located in the `yacceleratorstorefront/resources/yacceleratorcore/import/cmscockpit/structure-view/` directory:

```
<div>
    <table width="100%" cellspacing="0" style="margin:0;padding:0;border:1px solid #1E4EBF;">
        <tbody>
            <tr>
                <td height="125px" width="25%" colspan="2" rowspan="2" class="structureViewSection">
                    <cockpit code="SiteLogo"/>
                </td>
                <td colspan="2" class="structureViewSection">
                    <cockpit code="HeaderLinks"/>
                </td>
                <td width="20%" rowspan="2" class="structureViewSection">
                    <cockpit code="MiniCart"/>
                </td>
            </tr>
            <tr>
                <td colspan="2" height="89px" class="structureViewSection">
                    <div>Header</div>
                </td>
            </tr>
            <tr>
                <td colspan="5" class="structureViewSection">
                    <cockpit code="NavigationBar"/>
                </td>
            </tr>
            <tr>
                <td colspan="5" height="58px" style="vertical-align:middle;" class="structureViewSection">
                    <div>Breadcrumb</div>
                </td>
            </tr>
            <tr>
                <td colspan="5" style="vertical-align:top;" class="structureViewSection">
                    <cockpit code="Section1"/>
                </td>
            </tr>
            <tr>
                <td width="80%" colspan="4" style="vertical-align:top;" class="structureViewSection">
                    <cockpit code="Section2"/>
                </td>
                <td width="20%" style="vertical-align:top;" class="structureViewSection">
                    <cockpit code="Section3"/>
                </td>
            </tr>
            <tr>
                <td height="270px" colspan="5" class="structureViewSection">
                    <cockpit code="Footer"/>
                </td>
            </tr>
        </tbody>
    </table>
    <div style="width:100%; border-top: 2px solid #bbb">
        <cockpit code="editor"/>
    </div>
</div>
```

How To Extend the ProductFacade

This document describes how to extend the `ProductFacade` functionality to customize it to fit the needs of the storefront.

The `ProductFacade` functionality can be customized to fit the needs of the storefront. For example, to configure the `gender` attribute that is defined for the type of `ApparelProduct` component, you need to extend the `ProductFacade` functionality. These steps are described in the following tutorial.

To extend the `ProductFacade`:

1. Create a data container that holds a representation of the `gender` attribute for the `ApparelProduct` object as shown in this example from the `GenderData.java` file:

   ```
   public class GenderData
   {
           private String code;
           private String name;

           // public getters and setters
   }
   ```

2. Create a new class in the `ApparelProductData.java` file that extends the existing `ProductData` object so it can hold your gender data:

   ```
   public class ApparelProductData extends ProductData
   {
           List<GenderData> genders;

           // public getter and setter
   }
   ```

3. In the `GenderConverter.java` file, create a converter for the gender that takes the gender as an enumeration and returns the `GenderData`:

   ```
   public class GenderConverter extends AbstractPopulatingConverter<Gender, GenderData>
   {
           private TypeService typeService;

           protected TypeService getTypeService()
           {
                   return typeService;
           }

           @Required
           public void setTypeService(final TypeService typeService)
   ```

```
		{
			this.typeService = typeService;
		}

		@Override
		protected GenderData createTarget(final Gender source)
		{
			return new GenderData();
		}

		@Override
		public void populate(final Gender source, final GenderData target)
		{
			target.setCode(source.getCode());
			target.setName(getTypeService().getEnumerationValue(source).getName());

			super.populate(source, target);
		}
	}
```

4. Implement a populator that populates the `ApparelProductData` object with the `gender` attributes as shown in the example from the `ApparelProductPopulator.java` file. Based on the type of passed arguments, you need to decide where to get the gender attributes from:

```
	public class ApparelProductPopulator implements Populator<ProductModel, ProductData>
	{
		private GenderConverter genderConverter;

		protected GenderConverter getGenderConverter()
		{
			return genderConverter;
		}

		@Required
		public void setGenderConverter(final GenderConverter genderConverter)
		{
			this.genderConverter = genderConverter;
		}

		@Override
		public void populate(final ProductModel source, final ProductData target) throws ConversionException
		{
			ApparelProductModel apparelProductModel = null;
			if (source instanceof ApparelProductModel)
			{
				apparelProductModel = (ApparelProductModel) source;
			}
			else if (source instanceof VariantProductModel)
			{
				if (((VariantProductModel) source).getBaseProduct() instanceof ApparelProductModel)
				{
					apparelProductModel = (ApparelProductModel) ((VariantProductModel) source).getBaseProduct();
				}
				else if (((VariantProductModel) source).getBaseProduct() instanceof ApparelStyleVariantProductModel)
				{
					apparelProductModel = (ApparelProductModel) ((ApparelStyleVariantProductModel) ((VariantProductModel) source)
							.getBaseProduct()).getBaseProduct();
				}
			}

			if (target instanceof ApparelProductData && apparelProductModel != null)
			{
				if (CollectionUtils.isNotEmpty(apparelProductModel.getGenders()))
				{
					final List<GenderData> genders = new ArrayList<GenderData>();
					for (final Gender gender : apparelProductModel.getGenders())
					{
						genders.add(getGenderConverter().convert(gender));
					}
					((ApparelProductData) target).setGenders(genders);
				}
			}
		}

	}
```

5. Having all the required classes, you can create the `AcceleratorProductConverter` object. It creates the proper data holder object of the required type, and then populates it with all the required data. The following code excerpt is from the `AcceleratorProductConverter.java` file:

```
	public class AcceleratorProductConverter extends ProductConverter
	{
		private Populator<ProductModel, ProductData> apparelProductPopulator;

		protected Populator<ProductModel, ProductData> getApparelProductPopulator()
		{
			return apparelProductPopulator;
		}

		@Required
		public void setApparelProductPopulator(final Populator<ProductModel, ProductData> apparelProductPopulator)
		{
			this.apparelProductPopulator = apparelProductPopulator;
		}

		@Override
		protected ProductData createTarget(final ProductModel source)
		{
			if (source instanceof ApparelProductModel)
			{
				return new ApparelProductData();
			}
			else if (source instanceof VariantProductModel)
			{
				if (((VariantProductModel) source).getBaseProduct() instanceof ApparelProductModel)
				{
					return new ApparelProductData();
				}
				else if (((VariantProductModel) source).getBaseProduct() instanceof ApparelStyleVariantProductModel)
				{
					return new ApparelProductData();
				}
```

```
                }
                return super.createTarget(source);
        }

        @Override
        public void populate(final ProductModel source, final ProductData target)
        {
                getApparelProductPopulator().populate(source, target);

                super.populate(source, target);
        }
}
```

6. To connect all of these pieces together, reference all the classes as Spring beans and use the Spring configuration to make them work. The following is an example of a Spring configuration file, `yacceleratorfacades-spring.xml`:

```
<alias name="acceleratorGenderConverter" alias="genderConverter"/>
        <bean id="acceleratorGenderConverter" class="de.hybris.platform.yacceleratorfacades.converters.GenderConverter" >
                <property name="typeService" ref="typeService"/>
        </bean>

        <bean id="apparelProductPopulator" class="de.hybris.platform.yacceleratorfacades.converters.populator.ApparelProductPopulator" >
                <property name="genderConverter" ref="genderConverter"/>
        </bean>

        <alias name="acceleratorProductConverter" alias="productConverter"/>
        <bean id="acceleratorProductConverter" class="de.hybris.platform.yacceleratorfacades.converters.AcceleratorProductConverter" parent="de
                <property name="apparelProductPopulator" ref="apparelProductPopulator"/>
        </bean>
```

The `Accelerator ProductConverter` function is aliased with the `productConverter` function, so it overrides the existing product converter in the system and assumes all of its responsibilities. Afterwards, all product conversions are performed by this new converter that checks if a product is related to an Apparel variant and populates it with the proper attributes.

JS, LESS and JSPs: Build Process for Generating Variables

The `yacceleratorstorefront` template provides a process to generate shared variables among JSPs, JavaScript and LESS files, based on one common properties file.

Overview

The `yacceleratorstorefront` template provides an extra step in the build process (defined in the `buildcallbacks.xml` file of the storefront) to generate variables for use in different frameworks (JSP/tag, LESS and JS). This provides an easy way to share values across the different frameworks. To accomplish this, a user can add a property in the `yacceleratorstorefront/resources/generateVariables.properties` file and then run `ant all`. This will update the following three files:

- `/yacceleratorstorefront/web/webroot/WEB-INF/tags/shared/variables/generatedVariables.tag`
- `/yacceleratorstorefront/web/webroot/_ui/shared/js/generatedVariables.js`
- `/yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/shared/less/generatedVariables.less`

After this is run, the variables are ready to use.

⚠ Caution

JavaScript Variable Names

Do not use a hyphen (-)' in the variable names of the properties file if you intend to use it in JavaScript, because JS does not accept the hyphen as a valid character in a variable name.

It is not recommended to manually modify or delete the following generated files:

- `generatedVariables.tag`
- `generatedVariables.js`
- `generatedVariables.less`

The way to modify these files is to modify the `generateVariables.properties` file first, then run `ant build` to regenerate these three files. If you do not want to use these generated variables, you must remove the references from the UI experience-specific `master.tag`, `javaScriptVariables.tag` files, as well as the theme-specific `variables.less` file.

The generation or update of the files during the build process only occurs if the `generateVariables.properties` file is newer than any of the three `generatedVariables` files. Running `ant clean` will not remove these generated files.

LESS Integration

If you want to use the variables in LESS, the LESS must be recompiled (as described in Building a Responsive Website Front-End), by using Grunt or the `ant compileuisrc` command.

Another caveat when using LESS and Bootstrap is the variable names. Since the hyphen is not allowed in a variable name in JavaScript, but certain variables that are integrated into Bootstrap require this specific naming convention, a `variableMapping.less` file is provided. This can be used to override Bootstrap-specific variables. A common use-case would be the breakpoints, predefined (as of Bootstrap v. 3.2.0) as screen-sm etc. The `theme-variable.less` file is loaded last and can also be used to override the screen mapping values which is declared in the base store. An example of a breakpoint definition is shown in the excerpt from the `generateVariables.properties` file:

```
screenSmMin=640px
```

After setting the breakpoint definition, you can define the variable definition for Bootstrap in the `/yacceleratorstorefront/web/webroot/WEB-INF/_ui-src/shared/less/variableMapping.less` file, as follows:

```
@screen-sm-min:@screenSmMin;
```

This `variableMapping` should be loaded in the theme specific `variables.less` file, which is imported in the theme-specific `style.less` file.

JSP Tag Scripts: Installing in an Accelerator Storefront

This document describes the mechanism for installing JSP tag scripts through an AddOn.

Overview

The **yacceleratorstorefront** extension has various JSP tag files that are included in every Accelerator storefront page. For example, the **`analytics.tag`** file is included in every page of the storefront to collect analytics. This is acheived by having the **`master.tag`** refer to the **`analytics.tag`**, which in turn includes the Google and Jirafe analytics tag scripts. This is also possible because the tag script file is in the same extension as the storefront.

However, there are scenarios where you need your AddOn to install its JSP tag file in various pages of the storefront, in a way that is independent of the **yacceleratorstorefront** extension. For example, the **hybrisAnalytics** AddOn needs to install the **piwikAnalytics.tag** script file in various pages of the storefront, even though the AddOn and the tag script file are both external to the **yacceleratorstorefront** extension.

The following sections describe how this is achieved.

For more information on the **hybrisAnalytics** AddOn, see hybrisAnalytics AddOn.

Configuring the PlaceholderContentSlot in the Accelerator Storefront

A CMS slot called the **PlaceholderContentSlot** is configured for all the Accelerator page templates. Any page that is created from one of these templates automatically inherits the **PlaceholderContentSlot**, including every page in the **yacceleratorstorefront**. A WCMS component can therefore be mapped to this slot, where the WCMS component's JSP file includes a reference to the JSP tag script file that you want to install in the storefront.

The following example shows how the **hybrisAnalytics** AddOn installs the **`piwikAnalytics.tag`** script file into the **yacceleratorstorefront** in order to collect analytics information.

First of all, a new CMS component is created in the **hybrisAnalytics** AddOn, and this CMS component is mapped to the **`PlaceholderContentSlot`** in the Accelerator, as follows:

Mapping an AddOn's tag script CMS component to the PlaceholderContentSlot

```
$contentCatalog=electronicsContentCatalog
                                $contentCV=catalogVersion(CatalogVersion.catalog(Catalog.id[default=$contentCatalog]),CatalogVersion.version[default
                                $jarResourceCms=jar:de.hybris.platform.hybrisanalyticsaddon.constants.HybrisanalyticsaddonConstants&/hybrisanalytics

                                INSERT_UPDATE HybrisAnalyticsTagScriptComponent;$contentCV[unique=true];uid[unique=true];name;
                                ;;HybrisAnalyticsTagScriptComponent;Hybris analytics tag file;HybrisAnalyticsTagScript;;

                                INSERT_UPDATE ContentSlot;$contentCV[unique=true];uid[unique=true];active;cmsComponents(uid,$contentCV)
                                ;;PlaceholderContentSlot;true;HybrisAnalyticsTagScriptComponent
```

Each CMS component can have a JSP file, and in this case, we have the **`hybrisAnalyticsTagScriptComponent.jsp`** file in the AddOn that includes the tag file, as follows.

```
<%@ taglib prefix="hybrisanalyticsaddon" tagdir="/WEB-INF/tags/addons/hybrisanalyticsaddon/shared/analytics" %>

                                <hybrisanalyticsaddon:piwikAnalytics/>
```

There is also an **`addonScripts.tag`** file that is included in the `master.tag` file in the storefront. The **`addonScripts.tag`** file includes the **`PlaceholderContentSlot`**, which in turns pulls up all the CMS components that are mapped against it.

The result is that all the tag scripts from various AddOns are included in the **yacceleratorstorefront**.

Just One Storefront

Modify your configuration to support a development installation for a project with one storefront.

If you have multiple storefronts, you can apply the changes described here to the storefront you want to appear as the default storefront when the user's URL does not provide sufficient information to discriminate between different storefronts.

SAP Commerce Accelerator provides the functionality to support multiple storefronts; a request to the servlet at its root context path directly results in a 500 error. You need to use `hosts` file mappings or use the site-ID parameter to avoid a 500 error.

The CMS site has an attribute called `urlPatterns`. This is a list of regular expressions that the website CMS filters match to determine which storefront a user is trying to access.

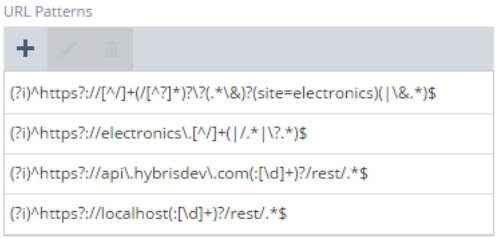
To resolve this, add an additional regular expression to the CMS site such as the following:

```
(?i)^https?://[^/]*/yacceleratorstorefront((?![\?\&]site=).)*
```

The regular expression can be added in the Backoffice Administration Cockpit.

Setting URL Patterns in Backoffice

1. Log into Backoffice.
2. Navigate to WCMS > Website. A list of storefronts appears in the main pane.
3. Click the storefront you want to set up.
4. In the Properties tab, make sure that the text field in the URL Patterns section looks like the one in the example below:



5. To make the change permanent, add the following code to the CMS site setup ImpEx script run during the project data phase of initialization.

```
# CMS Site
UPDATE CMSSite;uid[unique=true];urlPatterns;
;electronics;(?i)^https?://[^/]+(/[^?]*)?\?(.*\&)?(site=electronics)(|\&.*)$,(?i)^https?://electronics\.[^/]+(|/.*|\?.*)$,(?i)^https?://api\.hy
```

For more information on setting up scripts that run during initialization, see Essential and Project Data.

Related Information

B2C and B2B Accelerator Implementation

Customizing Configurable Checkout

SAP Commerce Accelerator allows you to customize the checkout flow. The checkout framework enables each checkout step to be split into its respective controller, while the validation logic and the logic for transitioning to other checkout steps has been externalized.

Checkout flows are aggregated or classified into `Checkout` groups, and each `Checkout` group is mapped to a base store. Each `Checkout` group can have different checkout steps, and each checkout step can have different behavior for different base stores. The configurable checkout enables Spring configuration to determine the next and previous steps in the checkout flow, and also allows you to configure different validations. Finally, this new framework provides a standardized way of configuring URLs for checkout steps and their operations.

i Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Checkout Groups

Checkout groups are the starting point for creating a customized checkout flow. The `CheckoutGroup` contains a unique group ID that is mapped to a base store. The `CheckoutGroup` contains a mapping between the different checkout step keys and the `checkoutStep` Spring configuration of possible combinations. The `CheckoutGroup` also contains the mapping between the `ValidationResult` and the redirect link in the case of `Validation` results.

The following is an example of the default Checkout group:

```
<!---Default checkout group-->
        <alias name="defaultMultiStepCheckoutGroup" alias="defaultCheckoutGroup" />
        <bean id="defaultMultiStepCheckoutGroup"  class="de.hybris.platform.acceleratorstorefrontcommons.checkout.steps.CheckoutGroup">
                <property name="groupId"  value="defaultGroup"/>
                <property name="checkoutStepMap">
                        <map merge="true">
                                <entry key="multi" value-ref="defaultMultiStepCheckout"/>
                                <entry key="delivery-address" value-ref="deliveryAddressCheckoutStep"/>
                                <entry key="delivery-method" value-ref="deliveryMethodCheckoutStep"/>
                                <entry key="pickup-location" value-ref="pickupLocationCheckoutStep"/>
                                <entry key="payment-method" value-ref="paymentMethodCheckoutStep"/>
                                <entry key="summary" value-ref="summaryCheckoutStep"/>
                        </map>
                </property>
                <property name="validationResultsMap">
                        <map merge="true">
                                <entry key="FAILED" value-ref="REDIRECT_TO_CART"/>
                                <entry key="REDIRECT_TO_DELIVERY_ADDRESS" value-ref="REDIRECT_TO_DELIVERY_ADDRESS"/>
                                <entry key="REDIRECT_TO_PICKUP_LOCATION" value-ref="REDIRECT_TO_PICKUP_LOCATION"/>
                                <entry key="REDIRECT_TO_CART" value-ref="REDIRECT_TO_CART"/>
                                <entry key="REDIRECT_TO_PAYMENT_METHOD" value-ref="REDIRECT_TO_PAYMENT_METHOD"/>
                                <entry key="REDIRECT_TO_DELIVERY_METHOD" value-ref="REDIRECT_TO_DELIVERY_METHOD"/>
                        </map>
                </property>
                <property name="checkoutProgressBar">
                        <map merge="true">
                                <entry key="1" value-ref="deliveryAddressCheckoutStep"/>
                                <entry key="2" value-ref="deliveryMethodCheckoutStep"/>
                                <entry key="3" value-ref="paymentMethodCheckoutStep"/>
                                <entry key="4" value-ref="defaultSummaryCheckoutStep"/>
                        </map>
                </property>
        </bean>
```

CheckoutStep

The `CheckoutStep` holds information about the next, previous, and current links or redirect URLs. It also holds the information about the `Validator` class that has to be used. The checkout steps are created with help of Spring bean configurations and these can be overridden by AddOns. Also, you can use different checkout steps for the same checkout flow by changing the `Validator` or any attribute part of the `CheckoutStep`.

The following is an example of a `checkoutStep` in the Spring configuration.

```
<alias name="defaultDeliveryMethodCheckoutStep" alias="deliveryMethodCheckoutStep" />
	<bean id="defaultDeliveryMethodCheckoutStep"  parent="checkoutStep">
		<property name="checkoutGroup" ref="defaultCheckoutGroup"/>
		<property name="checkoutStepValidator" ref="deliveryMethodCheckoutValidator"/>
		<property name="transitions">
			<map merge="true">
				<entry key="previous" value-ref="REDIRECT_TO_DELIVERY_ADDRESS"/>
				<entry key="current" value-ref="REDIRECT_TO_DELIVERY_METHOD"/>
				<entry key="next" value-ref="REDIRECT_TO_PAYMENT_METHOD"/>
			</map>
		</property>
	</bean>
```

Validation

Each `CheckoutStep` is mapped to a `Validator` and refers to a `Validator` bean. Usually, the `enterStep()` method of the `CheckoutController` is annotated with `PreValidateStep` annotation. The point cut is mapped to this annotation, and with help from this aspect, the required `Validator` is called from the `checkouStep` bean and validation is applied. If the validation succeeds, then it proceeds to the `enterStep()` method to continue with that step, or else it is redirected to the link fetched from the `ValidationresultsMap` configured against the Checkout group. Based on the validation results, the checkout step looks for the next steps based on the XML configuration for the action.

Multiple Path

To change the checkout path, or to override the next configured checkout step based on user action or on the contents of the cart, the `validateOnExit()` method should be used. To do so, create or override the behavior of the `validateOnExit()` method and check for cart contents or for user action. According to the result, redirect to the appropriate step, or else return SUCCESS to make sure that it goes to the step as configured, as shown in the following example.

```
@Override
	public ValidationResults validateOnExit()
	{
		final String dummyStepSelection = getSessionService().getAttribute("DUMMY_STEP");
		if(dummyStepSelection != null && dummyStepSelection.equals("YES"))
		{
			return ValidationResults.REDIRECT_TO_DUMMY_STEP;
		}
		return ValidationResults.SUCCESS;
	}
```

Configuring Multiple Flows for Base Stores

You can assign a different checkout flow for each base store with the help of `CheckoutGroup`. For example, the electronics store can be configured with a checkout flow that is different from the apparel stores. By using an AddOn, each checkout flow can be further modified to include additional steps. The `BaseStoreModel` is modified with an additional attribute that holds the name of the Checkout group Spring bean that defines the checkout flows.

The following is an example of the default Checkout group:

```
<!---Default checkout group-->
	<alias name="defaultMultiStepCheckoutGroup" alias="defaultCheckoutGroup" />
	<bean id="defaultMultiStepCheckoutGroup"  class="de.hybris.platform.acceleratorstorefrontcommons.checkout.steps.CheckoutGroup">
		<property name="groupId"  value="defaultGroup"/>
		<property name="checkoutStepMap">
			<map merge="true">
				<entry key="multi" value-ref="defaultMultiStepCheckout"/>
				<entry key="delivery-address" value-ref="deliveryAddressCheckoutStep"/>
				<entry key="delivery-method" value-ref="deliveryMethodCheckoutStep"/>
				<entry key="pickup-location" value-ref="pickupLocationCheckoutStep"/>
				<entry key="payment-method" value-ref="paymentMethodCheckoutStep"/>
				<entry key="summary" value-ref="summaryCheckoutStep"/>
			</map>
		</property>
		<property name="validationResultsMap">
			<map merge="true">
				<entry key="FAILED" value-ref="REDIRECT_TO_CART"/>
				<entry key="REDIRECT_TO_DELIVERY_ADDRESS" value-ref="REDIRECT_TO_DELIVERY_ADDRESS"/>
				<entry key="REDIRECT_TO_PICKUP_LOCATION" value-ref="REDIRECT_TO_PICKUP_LOCATION"/>
				<entry key="REDIRECT_TO_CART" value-ref="REDIRECT_TO_CART"/>
				<entry key="REDIRECT_TO_PAYMENT_METHOD" value-ref="REDIRECT_TO_PAYMENT_METHOD"/>
				<entry key="REDIRECT_TO_DELIVERY_METHOD" value-ref="REDIRECT_TO_DELIVERY_METHOD"/>
			</map>
		</property>
		<property name="checkoutProgressBar">
			<map merge="true">
				<entry key="1" value-ref="deliveryAddressCheckoutStep"/>
				<entry key="2" value-ref="deliveryMethodCheckoutStep"/>
				<entry key="3" value-ref="paymentMethodCheckoutStep"/>
				<entry key="4" value-ref="defaultSummaryCheckoutStep"/>
			</map>
		</property>
	</bean>
```

The following is an example of the mapping of the `BaseStore` to the `CheckoutGroup`:

```
# Base Store
UPDATE BaseStore;uid[unique=true];checkoutFlowGroup;
;$storeUid;apparelCheckoutGroup;
```

Configuring the Progress Bar for Configurable Checkout

With the configurable checkout framework, it may be necessary to configure the progress bar to reflect the checkout flow, as illustrated by the following scenarios:

- You might have different checkout progress bars for different storefronts because each storefront can have different checkout steps with the Checkout groups.
- The storefront might have a multiple path checkout flow, where there are different child checkout steps based on user selection. In this case, the child steps should be highlighted under the same step as their parent step in the checkout progress bar.
- The storefront might have additional steps, and the checkout progress bar should be able to render all these steps to the user.

The following is an example of configuring the progress bar in the Checkout group:

Configuring the Progress Bar in the CheckoutGroup

```
<alias name="defaultMultiStepCheckoutGroupApparel" alias="apparelCheckoutGroup" />
	<bean id="defaultMultiStepCheckoutGroupApparel"  class="de.hybris.platform.acceleratorstorefrontcommons.checkout.steps.CheckoutGroup">
		<property name="groupId"  value="apparelCheckoutGroup"/>
		<property name="checkoutStepMap">
			<map merge="true">
				<entry key="multi" value-ref="defaultMultiStepCheckout"/>
				<entry key="delivery-address" value-ref="deliveryAddressCheckoutStep"/>
				<entry key="delivery-method" value-ref="deliveryMethodCheckoutStep"/>
				<entry key="pickup-location" value-ref="pickupLocationCheckoutStep"/>
				<entry key="payment-method" value-ref="paymentMethodCheckoutStep"/>
				<entry key="summary" value-ref="summaryCheckoutStep"/>
			</map>
		</property>
		<property name="validationResultsMap">
			<map merge="true">
				<entry key="FAILED" value-ref="REDIRECT_TO_CART"/>
				<entry key="REDIRECT_TO_DELIVERY_ADDRESS" value-ref="REDIRECT_TO_DELIVERY_ADDRESS"/>
				<entry key="REDIRECT_TO_PICKUP_LOCATION" value-ref="REDIRECT_TO_PICKUP_LOCATION"/>
				<entry key="REDIRECT_TO_CART" value-ref="REDIRECT_TO_CART"/>
				<entry key="REDIRECT_TO_PAYMENT_METHOD" value-ref="REDIRECT_TO_PAYMENT_METHOD"/>
				<entry key="REDIRECT_TO_DELIVERY_METHOD" value-ref="REDIRECT_TO_DELIVERY_METHOD"/>
			</map>
		</property>
		<property name="checkoutProgressBar">
			<map merge="true">
				<entry key="1" value-ref="deliveryAddressCheckoutStep"/>
				<entry key="2" value-ref="deliveryMethodCheckoutStep"/>
				<entry key="3" value-ref="paymentMethodCheckoutStep"/>
				<entry key="4" value-ref="summaryCheckoutStep"/>
			</map>
		</property>
	</bean>
```

The `checkoutProgressBar` property in the above XML snippet shows how the steps for the progress bar are configured. The entry key with numbers indicates the order in which the checkout is populated in the progress bar. Each `CheckoutStep` has a `progressBarId` attribute that determines whether it is part of some other checkout step or whether it is a step by itself.

The following is an example of a multiple path configuration:

Example of a Progress Bar Configuration in a Multiple Path Configuration

```
<alias name="defaultPickupLocationCheckoutStep" alias="pickupLocationCheckoutStep" />
	<bean id="defaultPickupLocationCheckoutStep" parent="checkoutStep">
		<property name="checkoutGroup" ref="defaultCheckoutGroup"/>
		<property name="checkoutStepValidator" ref="defaultPickupCheckoutValidator"/>
		<property name="transitions">
			<---Entries-->
		</property>
		<property name="progressBarId" value="deliveryMethod"/>
	</bean>
```

In the above XML snippet, `pickupLocationCheckoutStep` is part of `deliveryMethod`, so `progressBarId` has the value of `deliveryMethod`.

The following XML snippet shows how to configure the `progressBarId` when there is a new standalone step `dummyCheckoutStepForApparelGroup` that exists along with other steps, and therefore needs a separate place in the progress bar. The progress bar can be reconfigured with a new step, and can be reordered in the Checkout group Spring configuration as shown below for `apparelCheckoutgroup`.

Example of Progress Bar Configuration in a Multiple Flow Configuration

```
<alias name="defaultDummyCheckoutStepForApparelGroup" alias="dummyCheckoutStepForApparelGroup"/>
	<bean id="defaultDummyCheckoutStepForApparelGroup" parent="checkoutStep">
		<property name="checkoutGroup" ref="apparelCheckoutGroup"/>
		<property name="transitions">
			<map merge="true">
				<entry key="previous" value-ref="REDIRECT_TO_DELIVERY_METHOD"/>
				<entry key="current" value-ref="REDIRECT_TO_DUMMY_STEP"/>
				<entry key="next" value-ref="REDIRECT_TO_PAYMENT_METHOD"/>
			</map>
		</property>
		<property name="progressBarId" value="dummy"/>
	</bean>

	<alias name="defaultMultiStepCheckoutGroupApparelWithDummy" alias="apparelCheckoutGroup"/>
	<bean id="defaultMultiStepCheckoutGroupApparelWithDummy" class="de.hybris.platform.acceleratorstorefrontcommons.checkout.steps.CheckoutGroup
		<property name="checkoutStepMap">
			<--Entries-->
		</property>
		<property name="checkoutProgressBar">
			<map merge="true">
				<entry key="1" value-ref="deliveryAddressCheckoutStep"/>
				<entry key="2" value-ref="deliveryMethodCheckoutStepForApparelGroup"/>
				<entry key="3" value-ref="dummyCheckoutStepForApparelGroup"/>
				<entry key="4" value-ref="paymentMethodCheckoutStep"/>
				<entry key="5" value-ref="summaryCheckoutStep"/>
			</map>
		</property>
	</bean>
```

Related Information

Configurable Checkout

Managing Multi-Step Checkout Strategies

The following is a brief overview of the different checkout strategies offered for customers to complete their purchases of the item they have added to their shopping cart. It mainly looks at how these strategies should be managed for the implementation required by your project. The expectation is that any project implemented using the SAP Commerce Accelerator removes the ability for the customer to decide their own checkout flow, and instead adapts the Checkout Strategy provided according to the Merchant's requirements.

Checkout Page Flow Overview

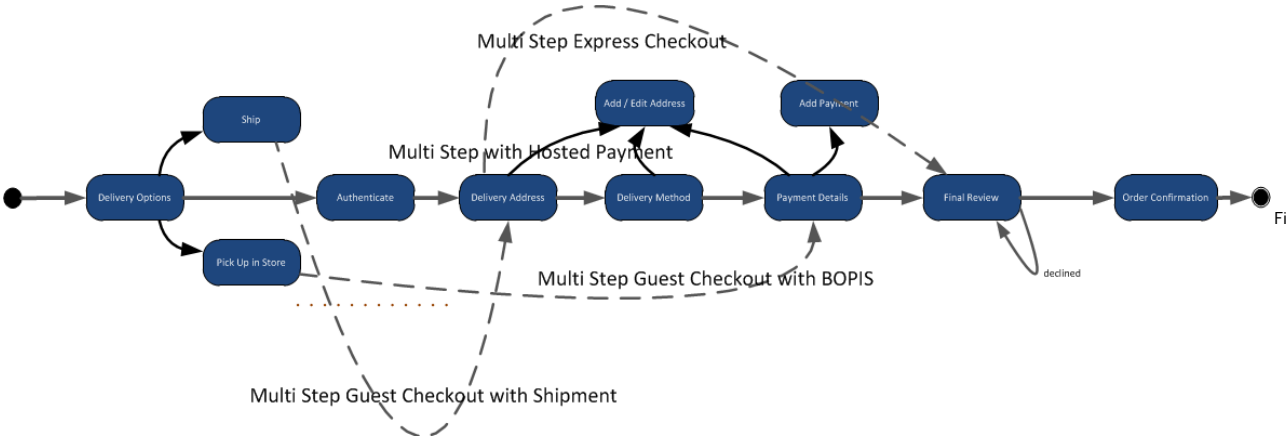


Figure: Checkout Page flow.

Before the multi-step checkout flow is initiated, following conditions have to be fulfilled:

- Customer adds items to cart.
- Customer clicks the **Checkout** button.
- The customers decide how the product items they will be proceed:
  - The items can be delivered to the address provided by a customer.
  - The items are purchased to the offline store selected by a customer. To learn more about this feature, see the Pickup In Store document.
- If the users are not already logged in, they can follow one of the checkout flow:
  - The customer can choose the guest customer checkout where there is no need to be logged in. To find more about guest customer checkout go to the Guest Checkout document.
  - The customers want to authenticate and then follow the multi-step checkout procedure.

Checkout page flow starts. Depending on the storefront and the device being used by the customer, one or more checkout page flows are available and the customer may have the option to let the system decide the most appropriate flow or choose for themselves.

In the multi-step checkout pages flow, the order of data entry is dictated by the checkout journey. At each stage the customer has the option to return to a previous step in the journey.

- If the customer has no delivery address, the **Add/Edit Address** page is shown for a new address to be entered. If however the customer has one or more addresses, he is taken to the **Select Delivery Address** page where he can select one of the addresses listed as the delivery address.
- On submitting the delivery address details the customer is taken to the **Payment Method** page for choosing the desired available delivery method.
- The customer is then taken to the **Payment Method** page where he can either add or select the payment details.
- The user is taken to the **Checkout Summary** page on submitting the **Payment Method** page.
- Once the customer is satisfied with the details provided, he can click the **Place Order** button to submit payment for the items in the cart.
- Following successful payment, the customer is directed to the **Order Confirmation** page.

In the multi-step checkout pages flow the customer can select also the **Express Checkout** option where the customer starts the checkout process on **Proceed To Checkout Page** and next they are taken to the **Checkout Summary** page. Therefore, the customer skips the pages in-between where the required fields are filled within the values saved in the account profile.

Implemented Checkout Strategies

The table below shows the implemented checkout strategies used for each B2C storefront in the SAP Commerce Accelerator:

Storefront	Strategy Checkout Flow	Available Checkout Flows
Desktop B2C Electronics	Multi Step: If customer does not have full data available for checkout, that is customer account, delivery address, card details and so on.	Multi Step
Desktop B2C Apparel UK	Multi Step: always	Multi Step
Desktop B2C Apparel DE	Multi Step: always	Multi Step
Mobile B2C (all storefronts)	Multi Step: always	Multi Step

Checkout Strategy Configuration

The checkout strategies described in the Implemented Checkout Strategies section can be easily configured by simply making some changes to several Spring configuration files. This section describes which extension and related files require configuration.

The table below provides definitions of strategy and selection strategy terms:

Term	Description
Checkout Page Flow Strategy	This describes the strategy that is responsible for redirecting the customer to the appropriate checkout page. The landing page determines whether they have an express checkout experience or multiple pages wizard style checkout experience.

Term	Description
Checkout Page Flow Selection Strategy	This describes the strategy used to determine which Checkout Page Flow Strategy the customer can experience depending on the storefront and the device being used by the customer.

yacceleratorfacades Extension

This extension provides a checkout flow faade to the storefronts for choosing the checkout page flow strategy to use. If the entire proposed default checkout flow strategy needs to be replaced or removed this is where it is done. This is the highest level where you can implement the configuration. Finer grained configurations can be done using the `acceleratorservices` and `yacceleratorcore` extensions.

For more information, see the following:

- yacceleratorfacades Extension
- yacceleratorcore Extension

Configuration

The `checkoutFlowFacade` bean is defined in the `yacceleratorfacades/resources/yacceleratorfacades-spring.xml` file. This bean has a `checkoutFlowStrategy` property which contains the logic for selecting a checkout page flow to use during checkout.

```
<alias name="defaultCheckoutFlowFacade" alias="checkoutFlowFacade"/>
                        <bean id="defaultCheckoutFlowFacade" class="de.hybris.platform.yacceleratorfacades.flow.impl.DefaultCheckoutFlowFacade"
                        parent="defaultAcceleratorCheckoutFacade" >
                        <property name="checkoutFlowStrategy" ref="checkoutFlowStrategy"/>
                        <property name="checkoutPciStrategy" ref="checkoutPciStrategy"/>
                        </bean>
```

acceleratorservices Extension

This extension provides the definition of the checkout page flow strategy.

Configuration

The important configuration properties can be found in the `acceleratoreservices/resources/acceleratorservices-items.xml` file.

The enumerations for the checkout page flow strategies should be defined in the `acceleratorservices` extension. Here additional enum values can be defined for any extra checkout flow that cannot be implemented using the defaults:

```
<enumtypes>
                        <enumtype generate="true" code="CheckoutFlowEnum" autocreate="true" dynamic="true" >
                        <value code="MULTISTEP"/>
                        </enumtype>
                        </enumtypes>
```

yacceleratorcore Extension

This extension provides the definition the actual checkout page flow strategies implemented by the Accelerator storefronts. Here finer grained configuration can be done depending on your project requirements.

Configuration

The important configuration properties can be found in the `yacceleratorcore/resources/yacceleratorcore-spring.xml` file.

Page Flow Implementation Configuration

The main strategy that implement the checkout page flows is `multiStepCheckoutFlowStrategy`. This strategy is identified by the `CheckoutFlowEnum` defined in its `checkoutFlow` property.

The following code extract from the `yacceleratorcore-spring.xml` file contains the bean definition for the checkout page flow strategies. Here additional strategies can be defined for any extra checkout flow that cannot be implemented using the defaults:

```
<bean id="multiStepCheckoutFlowStrategy" class="de.hybris.platform.yacceleratorcore.checkout.flow.impl.FixedCheckoutFlowStrategy"  >
                        <property name="checkoutFlow" value="MULTISTEP" />
                        </bean>
```

Express Checkout

The Express Checkout feature in SAP Commerce Accelerator allows customers to speed up their checkout experience by allowing them to skip most checkout steps by using default values saved in their account.

After adding products to a cart, customers can select the Express Checkout option, which automatically redirects them to the checkout summary page in the storefront. The following settings are set with values saved in the customer's account:

- Delivery address
- Delivery method: this feature is available for shipping and pick up in store delivery method. If any of the products are shipped, the cheapest delivery method is selected by default.
- Payment details

The Express Checkout is also supported by SAP Commerce Accelerator on mobile devices.

If the customer does not have default values, they need to follow all steps in the multi-step checkout process.

Overview and Features

Express Checkout is designed to provide functionality that enables customers to jump to the last step of the checkout flow, and includes these main features:

- The ability for the customer to jump directly to the Multistep Checkout Summary Page.
- The Express Checkout feature is optional and can be activated or deactivated on a per store level.
- This feature is only available to logged in customers.
- Logged out customers also have the option to use Express Checkout when logging in to complete their order. In this scenario, if customers do not have all the delivery and payment details to complete the Express Checkout, they are redirected to the checkout step that prevented Express Checkout from completing.
- Account details such as Address and Payment Details are automatically attached to the order. There is no need for customer to interact.
- If errors appear, it falls back to the default multi-step checkout behavior.

Technical Overview

Standard Behavior

The Express Checkout feature can be enabled and disabled on the Base Store properties. To see how to configure this feature go to the Configuration section of this document. Depending on those, the info boxes and checkout options will be shown on the page.

The default behavior of the Express Checkout is just available to registered and authenticated customers:

- If a customer is hard authenticated, a checkbox with the option for **Express Checkout** will appear on the Cart Page.
- For soft- and non-authenticated customers, there is an option for the customer to decide for Express Checkout flow on the Multistep Checkout - Proceed To Checkout Page.

Facade Layer

All the checkout-specific facades are leveraging the `CheckoutFacade` and `AcceleratorCheckoutFacade` implementation to determine the current user to map the address and other payment details during checkout. There are methods added in the *<HYBRIS_BIN_DIR>*`/commercefacades` and *<HYBRIS_BIN_DIR>*`/acceleratorfacades` folders.

MVC Layer

Controllers

Controller changes were made to the `CheckoutLoginController` and the `CartPageController` to allow for enabling or disabling of the Express Checkout as well as checking the eligibility to do Express Checkout.

View Configuration

By default, there are multiple places in the storefront where the Express Checkout option is offered to the customer:

- On the Cart Page
- On the Multistep Checkout - Proceed To Checkout Page

Cart Page Configuration

The Cart Page uses the `cartPotentialExpressCheckout.tag` file to display the info box. For the checkbox itself, the `cartExpressCheckoutEnabled.tag` file is used. So, to use it, the page should contain:

```
<cart:cartPotentialExpressCheckoutInfoBox/>
<cart:cartExpressCheckoutEnabled/>
```

Proceed to Checkout Page Configuration

On the Multistep Checkout - Proceed To Checkout Page the necessary information is already included. The decision whether or not to display the Express Checkout option can be realized with access expressions:

```
<sec:authorize access="isFullyAuthenticated()">/<sec:authorize>
```

How to Enable Express Checkout

To enable the Express Checkout feature you need to have the proper configuration and set up the Express Checkout option in the Backoffice Administration Cockpit.

Configuration

The `expressCheckoutEnabled` property is defined on the `BaseStore` level in the `acceleratorservices-items.xml` file as follows:

```
<typegroup name="BaseCommerce">
                        <itemtype code="BaseStore" autocreate="false" generate="false">
                                <description>Extending BaseStore type with additional attributes.</description>
                                <attributes>
                                        <attribute type="java.lang.Boolean" qualifier="expressCheckoutEnabled">
                                                <description>Determines if a site has the express checkout option.</description>
                                                <persistence type="property" />
                                                <modifiers optional="false" />
                                                <defaultvalue>java.lang.Boolean.FALSE</defaultvalue>
                                        </attribute>
                                </attributes>
                        </itemtype>
                </typegroup>
```

For the sample stores this property is set to true for the following storefronts:

- B2C UK Apparel Store
- B2C Japanese Electronics Store

Toggling Through the Backoffice Administration Cockpit

Business users can enable or disable the **Express Checkout** feature in the Backoffice Administration Cockpit.

1. Access the Backoffice Administration Cockpit (`https://localhost:9002/backoffice`).
2. Navigate to Base Commerce > Base Store.
3. Select the storefront where you want to enable Express Checkout.
4. In the Properties tab in the Editor area, select True or False under Enable Express Checkout as required.
5. Click Save.

Guest Checkout

The **Guest Checkout** feature in the Accelerator enables a user to complete the checkout as a guest without creating an account for login. The reference storefronts provide the **Guest Checkout** integrated with the multistep checkout flow.

Overview and Features

The **Guest Checkout** feature is designed to provide a functionality that lets customers complete the checkout flow without having to create an account in the storefront. The main features around the implementation are as follows:

- Ability to complete checkout without creating an account for login.
- Guest user is never logged into the system and remains anonymous, but still leverages secure channel and GUID authentication to ensure security.
- For every **Guest Checkout**, the system generates a unique temporary account, and maps it to the session cart. The guest user will always be an anonymous user.
- The address and payment details are saved against the session cart rather than the anonymous customer in a session.
- The guest user should re-enter address and payment information related to the checkout every time a new guest checkout is started. This ensures that checkout information is safe if a guest abruptly deviates from checkout or timeout. Removing address and payment data for abandoned checkouts is done using cron jobs.
- Provide an option for the guest user to create an account by just entering values for password fields after finishing the checkout.
- Account details such as **Address** and **Payment Details** are automatically populated once the guest user creates an account.
- Enhanced security with use of **Order GUID** to access order details and checkout confirmation page after checkout.
- Easy customization.
- A flagged guest customer entry in the Customer Service Cockpit in case of a customer puzzling circumstance with the **Guest Checkout** flow.

ServiceLayer

The **CheckoutCustomerStrategy** implementation exists to centralize the logic to check if a checkout is a **Guest Checkout** or not. Also, this implementation helps in fetching the temporary account user from the session cart for the entire checkout flow. The regular checkout flow maps the information against the logged in session user, whereas the **Guest Checkout** maps all the information against this temporary system generated account that is mapped to the cart.

Facade Layer

The facades include changes to create a unique temporary customer account for **Guest Checkout** and maps it to the cart.

All the checkout-specific facades are leveraging the **CheckoutCustomerStrategy** implementation to determine the current user to map the address and other payment details during checkout.

MVC Layer

Controllers

Controller changes were made to map the **Guest Checkout** URL and register the guest user as a regular customer when the guest opts for the conversion.

Filter Changes

The **AnonymousCheckoutFilter** removes any information about the checkout, if the guest user deviates from checkout or abandons the checkout.

View Configuration

To enable **Guest Checkout** for any of the checkout flows, the **`guestCheckout.tag`** file, which contains the view for **Guest Checkout**, should be included in the **Checkout Login** page of that particular flow and UI experience.

For example, if the **Guest Checkout** has to be enabled for the multistep checkout, then the **`.tag`** file that contains the **Guest Checkout** view has to be included in the **`checkoutLoginPage.jsp`** file of the multistep checkout flow, as follows:

```
<sec:authorize ifAnyGranted="ROLE_ANONYMOUS">
                                    <div class="${spanStyling} last">
                                    <c:url value="/login/checkout/guest" var="guestCheckoutUrl" />
                                    <user:guestCheckout actionNameKey="checkout.login.guestCheckout" action="${guestCheckoutUrl}"/>
                                    </div>
                                    </sec:authorize>
```

To control other UI components for **Guest Checkout**, the **<sec:authorize ifAnyGranted="ROLE_ANONYMOUS">** tag can be used.

Spring Security Configuration

The current https links can only be accessed by a logged in customer. To support a guest customer for checkout links, the following Spring security configuration has to be made to allow that particular link accessed by anonymous users.

```
<security:intercept-url pattern="/checkout*" requires-channel="https" />
                                        <security:intercept-url pattern="/checkout/multi*" requires-channel="https" />
                                        <security:intercept-url pattern="/checkout/multi/**" requires-channel="https" />
```

In the above configuration, the restriction of **ROLE_CUSTOMERGROUP** has been removed from the multistep checkout URL patterns.

SEO URLs

The SAP Commerce Accelerator allows you to create search-engine-friendly URLs for your website pages, which can help your website pages rank higher in search-engine results. It also produces URLs that are easy for your customers to read. This document provides instructions on implementing and customizing SEO URL generation.

SEO URL Implementation

In the SAP Commerce Accelerator storefront, SEO URLs for products and product categories are generated by resolvers configured using Spring.

The following code excerpt from the `commercefacades-spring.xml` file provides an example of this:

```
<alias name="defaultCategoryDataUrlResolver" alias="categoryDataUrlResolver" />
                                <bean id="defaultCategoryDataUrlResolver" class="de.hybris.platform.commercefacades.url.impl.DefaultCategoryDataUrlR
                                <property name="commerceCategoryService" ref="commerceCategoryService"/>
                                <property name="pattern" value="/{category-path}/c/{category-code}"/>
                                </bean>
                                <alias name="defaultProductDataUrlResolver" alias="productDataUrlResolver" />
                                <bean id="defaultProductDataUrlResolver" class="de.hybris.platform.commercefacades.url.impl.DefaultProductDataUrlRes
                                <property name="commerceCategoryService" ref="commerceCategoryService"/>
                                <property name="productService" ref="productService"/>
                                <property name="pattern" value="/{category-path}/{product-name}/p/{product-code}"/>
                                </bean>
```

These resolver classes are used to build internal links and for SEO. To change the generated URLS, you must:

- Extend the `DefaultProductDataUrlResolver` or `DefaultCategoryDataUrlResolver` .
- Change the configured pattern.
- Change the annotated binding in the corresponding page controller, that is either `ProductPageController` or `CategoryPageController` .

In the default configuration shown above, a product URL is configured that consists of the `supercategorystructure` , `productname` , and `productcode` variables; the `DefaultProductDataUrlResolver` populates these variables when it resolves the URL of a product. A category URL is also configured that consists of the `supercategorystructure` and `categorycode` variables; the `DefaultCategoryDataUrlResolver` populates these variables when it resolves the URL of a category.

ℹ Note

If you have customized the `DefaultCategoryDataUrlResolver` class, please ensure that special characters '<' or '>' are not present in the resulting category URL.

During a page request, based on the RequestMapping of controllers, a controller handles the request where the URL parameters are parsed, the appropriate product or category is identified, and a response is generated rendering a view specified in the controller with the identified product or category.

The `ProductPageController` is configured to handle any URL that matches the pattern, `/**/p/{productcode}` , whereby the product URL generated by the `DefaultProductDataUrlResolver` is handled. Similarly, the `CategoryPageController` is configured to handle any URL that matches the pattern, `/**/c/{categoryCode}` , whereby the category URL generated by the `DefaultCategoryDataUrlResolver` is handled.

In the SAP Commerce Accelerator , storefront requests belong to the `yacceleratorstorefront` extension and are bound to controllers using the configured `RequestMapping` annotation. The following excerpt from the `ProductPageController.java` file provides an example of this.

```
/**
                                * Controller for product details page
                                */
                                @Controller
                                @RequestMapping(value = "/**/p/{productCode}")
                                public class ProductPageController extends AbstractPageController
                                {
                                private static final Logger LOG = Logger.getLogger(ProductPageController.class);
                                ...
```

SEO URLs are also used by Advanced Personalization, therefore, if you make any changes to the SEO URL implementation you would also have to make in the Advanced Personalization filter configuration.

Example SEO URLs

Type of Target Data	Name	URL	Controller	URL Resolver
Product	Cyber-shot W55	`/yacceleratorstorefront/Cameras/Digital-Cameras/Digital-Compacts/Cyber-shot-W55/p/676442`	`ProductPageController`	`DefaultProductDataUrlResolver`
Category	Digital Compacts	`/yacceleratorstorefront/Cameras/Digital-Cameras/Digital-Compacts/c/576`	`CategoryPageController`	`DefaultCategoryDataUrlResolver`
Content Page	FAQ	`/yacceleratorstorefront/faq`	`DefaultPageController`	n/a

Type of Target Data	Name	URL	Controller	URL Resolver
Store Finder	tokyo	/yacceleratorstorefront/store-finder?q=Tokyo	StoreLocatorController	n/a

URL Encoding Attributes

The SAP Commerce Accelerator supports `UrlEncodingAttributes` that can be configured for a customer's site and that are to be encoded into all URLs on the site. This is supported on the storefronts through the `UrlEncoderFilter` . The `CMSSiteModel` provides an attribute named `urlEncodingAttributes` ; this attribute holds a collection of strings. Each attribute must be supported by the `UrlEncodingAttributeManager` implementation that knows how to provide the current value for each attribute.

By default, each sample site encodes the following attributes:

- Language
- Storefront

```
/**
 * Default implementation for language attribute handler. This changes the store language if language is included as
 * of the encoding attribute.
 */
public class DefaultLanguageAttributeManager extends AbstractUrlEncodingAttributeManager
{
@Override
public Collection<String> getAllAvailableValues()
{
getSessionService().setAttribute(AcceleratorFacadesConstants.LANGUAGE_ENCODING,Boolean.TRUE);
return CollectionUtils.collect(getStoreSessionFacade().getAllLanguages(), new Transformer()
{
@Override
public Object transform(final Object object)
{
return ((LanguageData) object).getIsocode();
}
});
}
@Override
public void updateAndSyncForAttrChange(final String value)
{
if (isValid(value))
{
getStoreSessionFacade().setCurrentLanguage(value);
}
}
@Override
public String getDefaultValue()
{
return getStoreSessionFacade().getDefaultLanguage().getIsocode();
}
@Override
public String getCurrentValue()
{
return getStoreSessionFacade().getCurrentLanguage().getIsocode();
}
}
```

This then inserts the language ISO code in the URLs.

Related Information

SEO Directives

SEO URL Tutorial

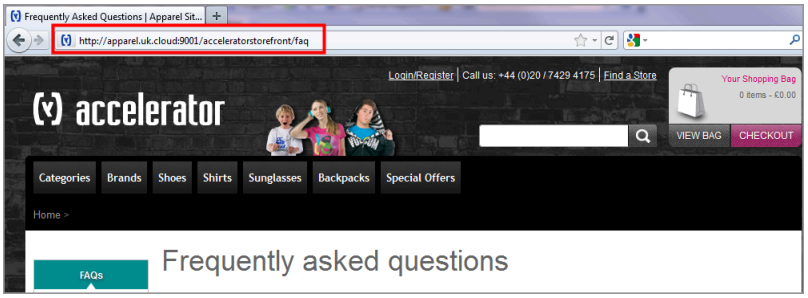
The SAP Commerce Accelerator supports Search Engine Optimization (SEO) URLs, which help pages rank higher in search engine results. SEO URLs are also easier to read for your customers. You can use the SmartEdit and the Product Cockpit to modify the URLs of your storefront for Search Engine Optimization (SEO).

For more technical information about SEO URL construction in the SAP Commerce Accelerator, see SEO URLs.

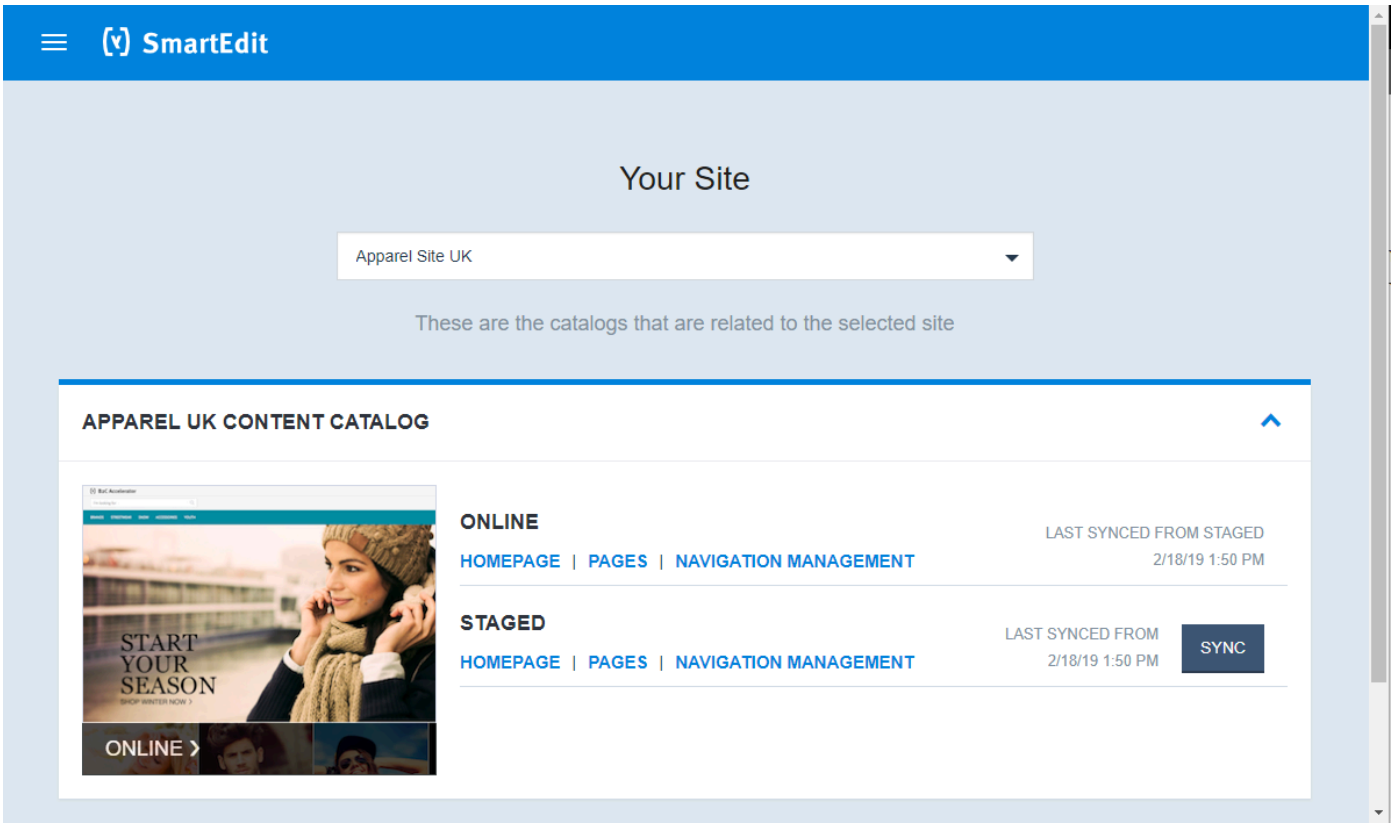
Changing the URL of a Content Page

A WCMS content page can be accessed in the front end via its UID or its label. Labels are more human readable and can also be used to bundle multiple content pages into one address.

To change the URL of a content page, follow the steps below. The procedure outlines changing the URL of the Frequently Asked Questions page, where the current URL is:
`http://apparel.uk.cloud:9001/acceleratorstorefront/faq`



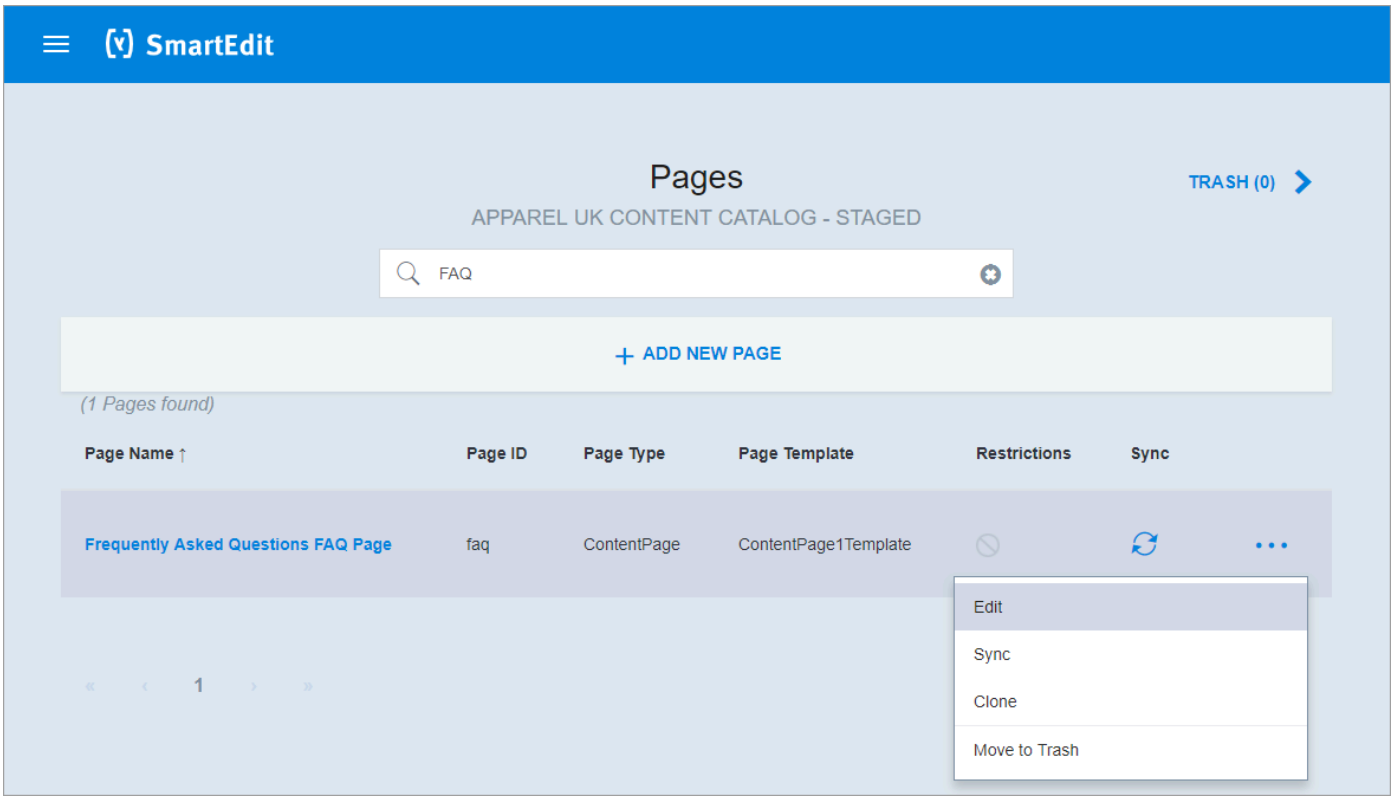
1. In SmartEdit, click the Pages link in the staged version of the target content catalog.



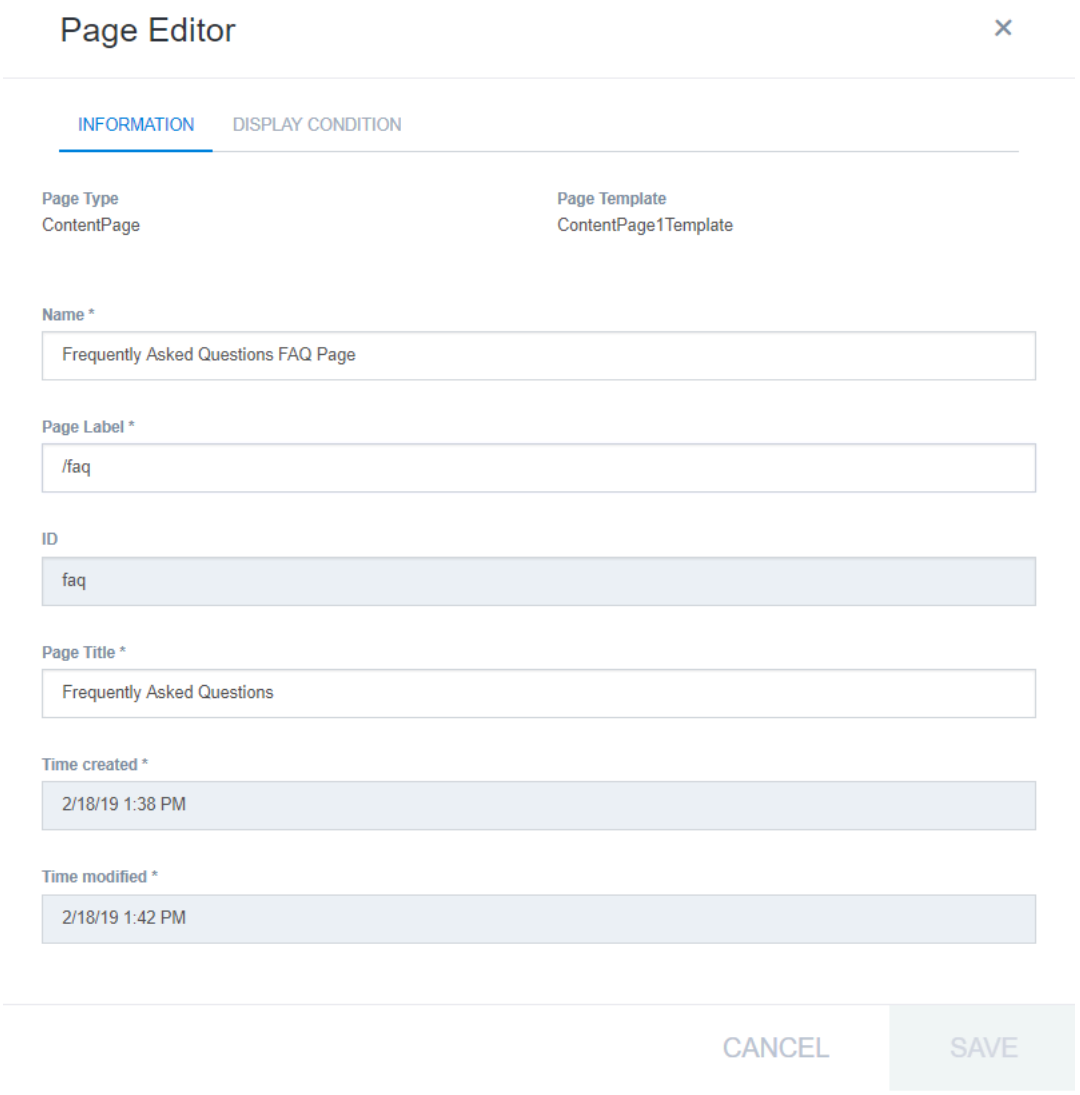
2. In the Pages view, go to the page that you want to edit. For our trail, go to the FAQ page.

→ **Tip**
Specify the name of the page in the search field to filter the list.

3. Edit the page.

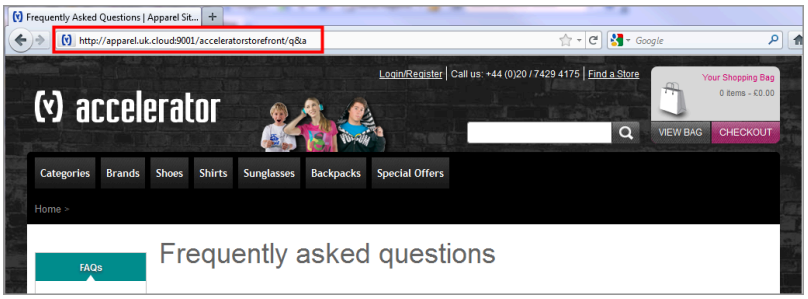


4. Specify the new label for the page. For our example, specify `/q&a`.



5. Synchronize the page with its online version.

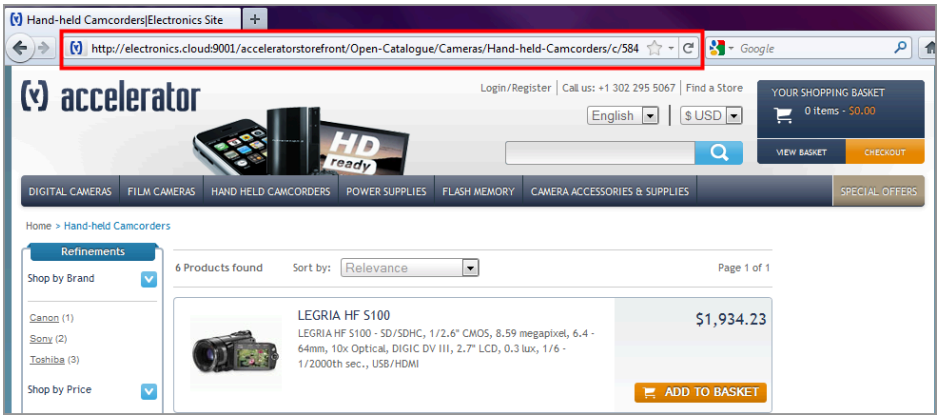
6. In the front end, type the new URL of the Frequently Asked Questions page: `http://apparel.uk.cloud:9001/acceleratorstorefront/q&a`



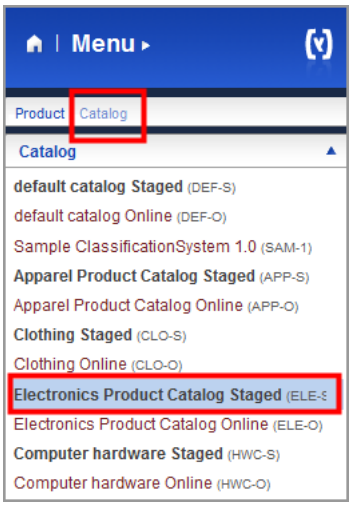
Changing the URL of a Category Page

The URL of a category page reflects the category structure constructed in the Product Cockpit. If you change the name of a category, the URL changes as well.

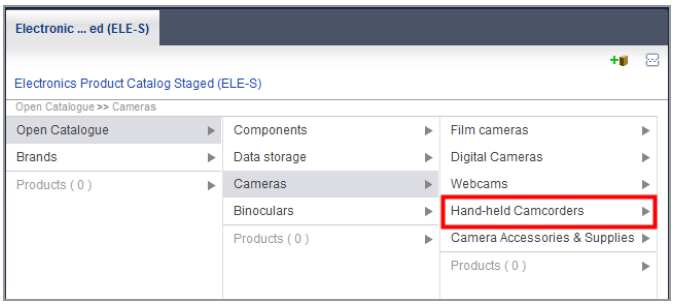
The example below outlines changing the URL of the **Handheld Camcorders** page, where the current URL is: `http://electronics.cloud:9001/acceleratorstorefront/Open-Catalogue/Cameras/Hand-held-Camcorders/c/584`



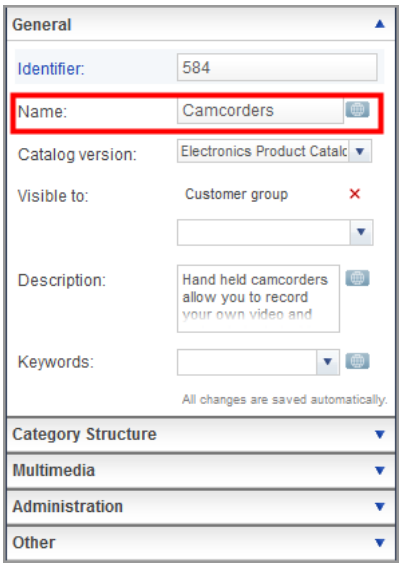
1. Go to the Product Cockpit, select **Electronics Product Catalog Staged**, and then switch to the **Catalog** perspective.



2. Browse through the category structure displayed in the browser area to the **Hand-held Camcorders** category.

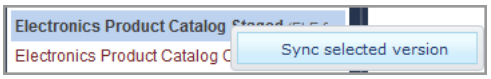


3. Double click the **Hand-held Camcorders** category. The editor area displays.

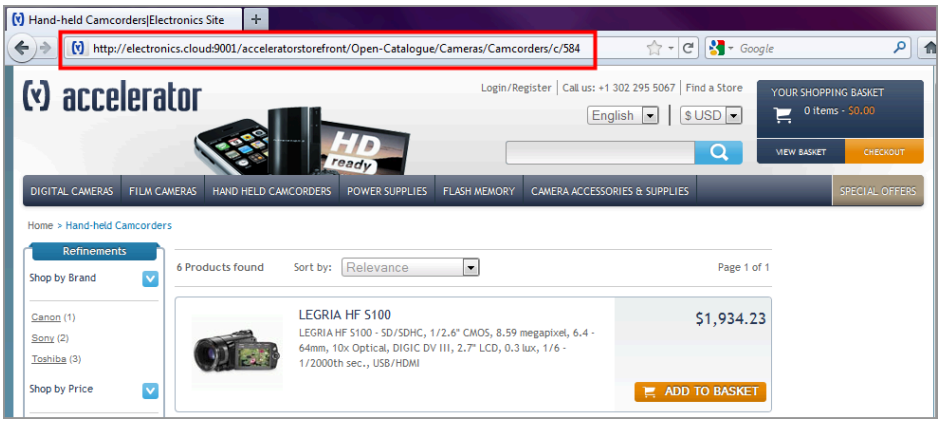


4. Change the name to **Camcorders**.

5. Right click the catalog version and synchronize it.



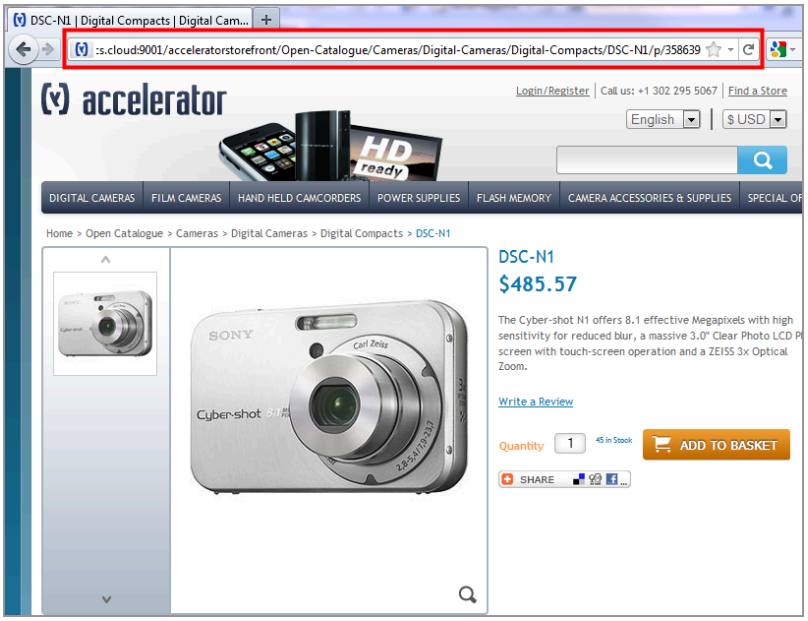
6. In the front end, type the new URL of the **Camcorders** page: `http://electronics.cloud:9001/acceleratorstorefront/Open-Catalogue/Cameras/Camcorders/c/584`



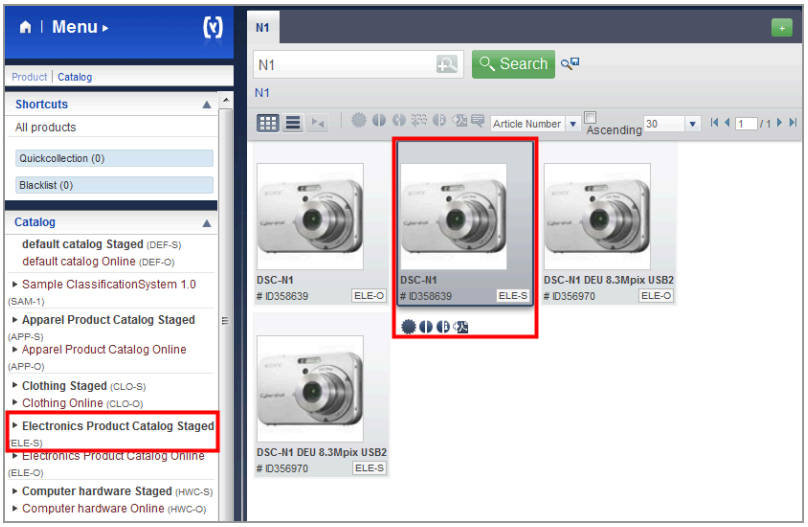
Changing the URL of a Product Page

The URL of a product page reflects the category structure constructed in the Product Cockpit. If you change the name of a product, the URL changes as well.

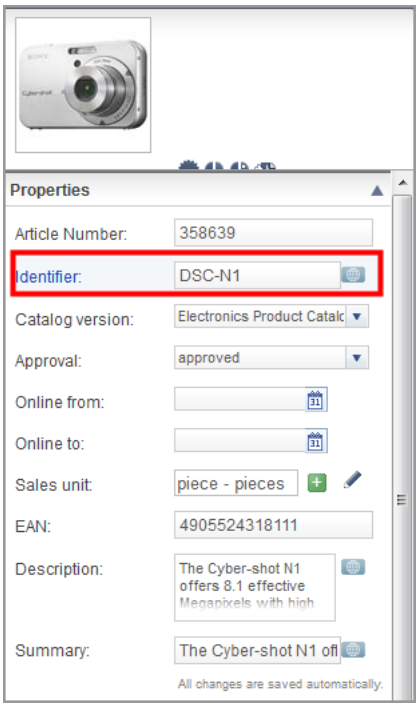
The example below outlines changing the URL of the **DSC-N1** product page, where the current URL is: `http://electronics.cloud:9001/acceleratorstorefront/Open-Catalogue/Cameras/Digital-Cameras/Digital-Compacts/DSC-N1/p/358639`



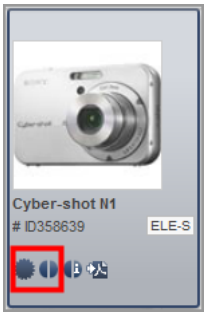
1. Go to the Product Cockpit, select **Electronics Product Catalog Staged**, and then search for **N1**.



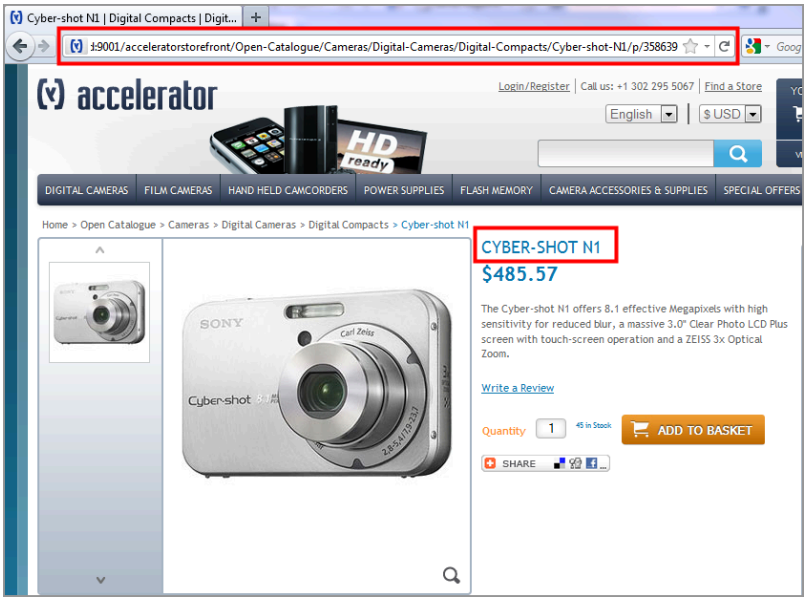
2. Double click the **DSC-N1** product. The editor area displays.



3. Change the name to **Cyber-shot N1**.
4. Click the **Approved** icon, and then synchronize the product with its on-line version.



5. Refresh the page in the front end. The URL for the **Cyber-shot N1** camera is updated with the new product name:
`http://electronics.cloud:9001/acceleratorstorefront/Open-Catalogue/Cameras/Digital-Cameras/Digital-Compacts/Cyber-shot-N1/p/358639`



Related Information

SEO URLs
SEO Directives
Catalog Perspective
Product Perspective

Sitemap Configuration

Sitemaps allow webmasters to inform search engines about pages on their sites that are available for indexing. Accelerator supports sitemaps for different page types (such as product pages and category pages), as well as for different languages and currencies.

In its simplest form, a sitemap is an XML file that lists the URLs for a site, along with additional metadata about each URL, so that search engines can more intelligently index the site. Examples of metadata include information about when a URL was last updated, how often it changes, how important it is relative to other URLs in the site, and so on.

The sitemap is based on the protocol defined by `sitemaps.org`.

Storefront URL to the Sitemap

The sitemap is exposed in Accelerator through the following URL: `http://electronics.local:9001/yacceleratorstorefront/sitemap.xml`.

The code excerpt is an example sitemap index.

```
<?xml version="1.0" encoding="UTF-8"?>
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
        <sitemap>
                        <loc>http://electronics.local:9001/medias/Homepage-ja-JPY-3422021852412885281.xml?context=bWFzdGVyfHJvb3R8MzQwfHRleHQveG1sfG
                </sitemap>

        </sitemapindex>
```

Note

For information on configuration properties for URLs, see acceleratorservices Extension.

Web Robots Pages

The web robots page under `/robots.txt` includes the URL to the sitemap URL:

```
# Allow search crawlers to discover the sitemap
Sitemap: /yacceleratorstorefront/electronics/en/sitemap.xml
```

Supported Sitemap Page Types

The types of supported sitemap pages are defined though the `SiteMapPageEnum` enumeration in the `acceleratorservices` extension, as follows:

SiteMapPageEnum	Comment
Homepage	Homepage URL
Product	Product detail page URL
Category	Category page URL
CategoryLanding	Category landing pages which are category pages with restrictions
Store	Store finder detail page URL
Content	Content pages URL
Custom	A list of URLs configurable on SiteMapConfig.customUrls

Sitemap URL Change Frequency

The available change frequency values for the sitemap URLs are defined in the `acceleratorservices` extension by the enumeration `SiteMapChangeFrequencyEnum`, as follows:

SiteMapChangeFrequencyEnum values
always
hourly
daily
weekly
monthly
yearly
never

Generators for Sitemap Pages

Each generator implementation must extend an abstract generator `de.hybris.platform.acceleratorservices.sitemap.generator.impl.AbstractSiteMapGenerator`, which implements the interface `de.hybris.platform.acceleratorservices.sitemap.generator.SiteMapGenerator`.

```
public interface SiteMapGenerator<T>
{
 public File render(final CMSSiteModel site, final CurrencyModel currencyModel, final LanguageModel languageModel,
 final RendererTemplateModel rendererTemplateModel, final List<T> models, final String filePrefix, final Integer index)
 throws IOException;
 public List<T> getData(final CMSSiteModel site);
```

```
 public SiteMapPageEnum getSiteMapPageEnum();
}
```

The following table lists the available generators.

SiteMapPageEnum	Bean ID	Bean Class
HOMEPAGE	homePageSiteMapGenerator	de.hybris.platform.acceleratorservices.sitemap.generator.impl.HomePageSiteMapGenerator
PRODUCT	productPageSiteMapGenerator	de.hybris.platform.acceleratorservices.sitemap.generator.impl.ProductPageSiteMapGenerator
CATEGORY	categoryPageSiteMapGenerator	de.hybris.platform.acceleratorservices.sitemap.generator.impl.CategoryPageSiteMapGenerator
CATEGORYLANDING	categoryLandingPageSiteMapGenerator	de.hybris.platform.acceleratorservices.sitemap.generator.impl.CategoryLandingPageSiteMapGenerator
STORE	pointOfServicePageSiteMapGenerator	de.hybris.platform.acceleratorservices.sitemap.generator.impl.PointOfServicePageSiteMapGenerator
CONTENT	contentPageModelSiteMapGenerator	de.hybris.platform.acceleratorservices.sitemap.generator.impl.ContentPageModelSiteMapGenerator
CUSTOM	customPageSiteMapGenerator	de.hybris.platform.acceleratorservices.sitemap.generator.impl.CustomPageSiteMapGenerator

Populators for Generator Beans

Each generator needs to know how to create a sitemap URL for a given sitemap page type. There are populators provided for each page type, as follows:

Generator Bean ID	Populator Bean ID	Populator Bean Class
homePageSiteMapGenerator	homepageToSiteMapUrlDataPopulator	de.hybris.platform.acceleratorservices.sitemap.populators.ContentPageModelToSiteMapUrlDataPopulator
productPageSiteMapGenerator	productModelToSiteMapUrlDataPopulator	de.hybris.platform.acceleratorservices.sitemap.populators.ProductModelToSiteMapUrlDataPopulator
categoryPageSiteMapGenerator	categoryModelToSiteMapUrlDataPopulator	de.hybris.platform.acceleratorservices.sitemap.populators.CategoryModelToSiteMapUrlDataPopulator
categoryLandingPageSiteMapGenerator	categoryModelToSiteMapUrlDataPopulator	de.hybris.platform.acceleratorservices.sitemap.populators.CategoryModelToSiteMapUrlDataPopulator
pointOfServicePageSiteMapGenerator	pointOfServiceModelIToSiteMapUrlDataPopulator	de.hybris.platform.acceleratorservices.sitemap.populators.PointOfServiceModelIToSiteMapUrlDataPopulator
contentPageModelSiteMapGenerator	contentPageModelToSiteMapUrlDataPopulator	de.hybris.platform.acceleratorservices.sitemap.populators.ContentPageModelToSiteMapUrlDataPopulator
customPageSiteMapGenerator	customPageToSiteMapUrlDataPopulator	de.hybris.platform.acceleratorservices.sitemap.populators.CustomPageToSiteMapUrlDataPopulator

For each populator, there is an `abstractPopulatingConverter` converter that is configured as a Spring bean. An example configuration is as follows:

```
<bean id="siteMapUrlData" class="de.hybris.platform.acceleratorservices.sitemap.data.SiteMapUrlData" scope="prototype"/>
        <alias name="defaultProductModelToSiteMapUrlDataConverter" alias="productModelToSiteMapUrlDataConverter"/>
        <bean id="defaultProductModelToSiteMapUrlDataConverter" parent="abstractPopulatingConverter">
                <lookup-method name="createTarget" bean="siteMapUrlData"/>
                <property name="populators">
                        <list>
                                <ref bean="productModelToSiteMapUrlDataPopulator"/>
                        </list>
                </property>
        </bean>
```

URL Resolvers

The URL resolvers are defined in the `commerceservices-spring.xml` file of the `commerceservices` extension, as follows:

SiteMapPageEnum	Bean ID	Bean Class
HOMEPAGE	homepageContentPageUrlResolver	de.hybris.platform.commerceservices.url.impl.DefaultHomepageContentPageUrlResolver
PRODUCT	productModelUrlResolver	de.hybris.platform.commerceservices.url.impl.DefaultProductModelUrlResolver
CATEGORY	categoryModelUrlResolver	de.hybris.platform.commerceservices.url.impl.DefaultCategoryModelUrlResolver
CATEGORYLANDING	categoryModelUrlResolver	de.hybris.platform.commerceservices.url.impl.DefaultCategoryModelUrlResolver
STORE	pointOfServiceUrlResolver	de.hybris.platform.commerceservices.url.impl.DefaultPointOfServiceUrlResolver
CONTENT	contentPageModelSiteMapGenerator	de.hybris.platform.commerceservices.url.impl.DefaultContentPageUrlResolver
CUSTOM	contentPageUrlResolver	de.hybris.platform.commerceservices.url.impl.DefaulCustomPageUrlResolver

Cron Job for Generating Sitemap Media

The `de.hybris.platform.acceleratorservices.cronjob.SiteMapMediaJob` cron job is responsible for generating sitemap files and storing them as media on the CMSSite. A cron job bean gets configured with a list of the generators it will use to generate the sitemaps, as follows:

```
<bean id="siteMapMediaJob"
 class="de.hybris.platform.acceleratorservices.cronjob.SiteMapMediaJob"
 parent="abstractJobPerformable">
 <property name="generators">
 <list>
 <ref bean="pointOfServicePageSiteMapGenerator"/>
 <ref bean="productPageSiteMapGenerator"/>
 <ref bean="categoryPageSiteMapGenerator"/>
 <ref bean="homePageSiteMapGenerator"/>
 <ref bean="contentPageModelSiteMapGenerator"/>
 <ref bean="categoryLandingPageSiteMapGenerator"/>
```

```
  <ref bean="customPageSiteMapGenerator"/>
  </list>
  </property>
...
  </bean>
```

Example Spring Configuration

The following is an example of a complete Spring configuration for the `productPageSiteMapGenerator` generator and its dependencies:

```
<alias name="defaultProductModelUrlResolver" alias="productModelUrlResolver"/>
        <bean id="defaultProductModelUrlResolver" class="de.hybris.platform.commerceservices.url.impl.DefaultProductModelUrlResolver">
                <property name="threadContextService" ref="threadContextService"/>
                <property name="commerceCategoryService" ref="commerceCategoryService"/>
                <property name="baseSiteService" ref="baseSiteService"/>
                <property name="defaultPattern" value="/{category-path}/{product-name}/p/{product-code}"/>
        </bean>
        <alias name="defaultProductPageSiteMapGenerator" alias="productPageSiteMapGenerator"/>
        <bean id="defaultProductPageSiteMapGenerator"
              class="de.hybris.platform.acceleratorservices.sitemap.generator.impl.ProductPageSiteMapGenerator" parent="abstractSiteMapGenerator">
                <property name="siteMapUrlDataConverter" ref="productModelToSiteMapUrlDataConverter"/>
                <property name="siteMapPageEnum" value="PRODUCT"/>
        </bean>

        <alias name="defaultProductModelToSiteMapUrlDataPopulator" alias="productModelToSiteMapUrlDataPopulator"/>
        <bean id="defaultProductModelToSiteMapUrlDataPopulator" class="de.hybris.platform.acceleratorservices.sitemap.populators.ProductModelToSiteMa
                <property name="urlResolver" ref="productModelUrlResolver"/>
        </bean>

        <alias name="defaultProductModelToSiteMapUrlDataConverter" alias="productModelToSiteMapUrlDataConverter"/>
        <bean id="defaultProductModelToSiteMapUrlDataConverter" parent="abstractPopulatingConverter">
                <lookup-method name="createTarget" bean="siteMapUrlData"/>
                <property name="populators">
                        <list>
                                <ref bean="productModelToSiteMapUrlDataPopulator"/>
                        </list>
                </property>
        </bean>
```

Sitemap Configuration

The following table provides descriptions for various configuration items:

Item	Description
`SiteMapLanguageCurrency`	A holder for the language and currency combination for a sitemap type.
`SiteMapPage`	A holder for the mapping between a sitemap type and its change frequency and priority.
`SiteMapConfig`	A holder for `SiteMapLanguageCurrency`, `SiteMapPage` and a list of custom URLs for generating a custom sitemap file.
`RendererTemplate`	A velocity renderer configuration for Google sitemap.

The CMSSite is configured to support sitemap generation through impex, as follows:

```
#
# Import the CMS Site configuration for the Electronics store
#
$productCatalog=electronicsProductCatalog
$contentCatalog=electronicsContentCatalog
$contentCV=catalogVersion(CatalogVersion.catalog(Catalog.id[default=$contentCatalog]),CatalogVersion.version[default=Staged])[default=$contentCatalo
$defaultLanguage=en
$storeUid=electronics
$siteUid=electronics
$webServiceSiteUid=$siteUidWS
$siteMapUrlLimitPerFile=50000
$jarResource=jar:de.hybris.platform.yacceleratorcore.setup.CoreSystemSetup&/yacceleratorcore/import/common/
$siteMapLangCur=jaJpy,jaUsd,enJpy,enUsd,deJpy,deUsd,zhJpy,zhUsd
$siteMapPage=Homepage,Product,CategoryLanding,Category,Store,Content,Custom
$customSiteMapUrls=/Open-Catalogue/c/1?q=%3AtopRated%3AallPromotions%3ABOGOFElectronics%3AallPromotions%3A10DiscountCanonEOS450D%3AallPromotions%3AM

# SiteMap Configuration
INSERT_UPDATE SiteMapLanguageCurrency;&siteMapLanguageCurrency;language(isoCode)[unique=true];currency(isocode)[unique=true];
;enUsd;en;USD
;enJpy;en;JPY
;jaUsd;ja;USD
;jaJpy;ja;JPY
;deUsd;de;USD
;deJpy;de;JPY
;zhUsd;zh;USD
;zhJpy;zh;JPY

INSERT_UPDATE CatalogUnawareMedia;&siteMapMediaId;code[unique=true];realfilename;@media[translator=de.hybris.platform.impex.jalo.media.MediaDataTrans
;$siteUid-siteMapMedia;$siteUid-siteMapMedia;siteMapTemplate.vm;$jarResource/site-siteMapTemplate.vm;

INSERT_UPDATE RendererTemplate;&siteMapRenderer;code[unique=true];content(&siteMapMediaId);contextClass;rendererType(code)[default='velocity'];
;$siteUid-siteMapTemplate;$siteUid-siteMapTemplate;$siteUid-siteMapMedia;de.hybris.platform.acceleratorservices.sitemap.renderer.SiteMapContext;

INSERT_UPDATE SiteMapPage;&siteMapPage;code(code)[unique=true];frequency(code)[unique=true];priority[unique=true];active[default=true]
;Homepage;Homepage;daily;1.0;;
;Product;Product;weekly;0.6;;
;CategoryLanding;CategoryLanding;daily;0.9;;
;Category;Category;daily;0.8;;
;Store;Store;weekly;0.6;;
;Content;Content;monthly;0.4;;
;Custom;Custom;daily;1.0;;

INSERT_UPDATE SiteMapConfig;&siteMapConfigId;configId[unique=true];siteMapLanguageCurrencies(&siteMapLanguageCurrency);siteMapPages(&siteMapPage);si
;$siteUidSiteMapConfig;$siteUidSiteMapConfig;$siteMapLangCur;$siteMapPage;$siteUid-siteMapTemplate;$customSiteMapUrls;
# CMS Site
INSERT_UPDATE CMSSite;uid[unique=true];theme(code);channel(code);stores(uid);contentCatalogs(id);defaultCatalog(id);defaultLanguage(isoCode);siteMap
```

```
;$siteUid;blue;B2C;$storeUid;$contentCatalog;$productCatalog;$defaultLanguage;$siteUidSiteMapConfig;(?i)^https?://[^/]+(/[^?]*)?\?(.*\&)?(site=$siteU

# Sitemap Generation CronJobs
INSERT_UPDATE SiteMapMediaCronJob;code[unique=true];job(code)[default=siteMapMediaJob];contentSite(uid)[default=$siteUid];sessionLanguage(isoCode)[de
;$siteUid-SiteMapMediaJob;;;;$siteMapUrlLimitPerFile;
```

Adding a New Sitemap Page to Accelerator

To add a new sitemap page to Accelerator:

1. Add your new page type (such as REVIEW, for example) to the `SiteMapPageEnum` enumeration.
2. Add REVIEW to the `SiteMapConfig` of your site. Define a `SiteMapPage` for review and add it to the `SiteMapConfig.siteMapPages`, as demonstrated in the following example:

```
INSERT_UPDATE SiteMapPage;&siteMapPage;code(code)[unique=true];frequency(code)[unique=true];priority[unique=true];active[default=true]
;Review;Review;daily;1.0;;
INSERT_UPDATE SiteMapConfig;&siteMapConfigId;configId[unique=true];siteMapLanguageCurrencies(&siteMapLanguageCurrency);siteMapPages(&siteMapPag
;$siteUidSiteMapConfig;$siteUidSiteMapConfig;$siteMapLangCur;Review,Homepage,Product,CategoryLanding,Category,Store,Content,Custom;$siteUid-sit
```

3. Provide the implementation of a review URL resolver as follows:

```
public class DefaultReviewPageUrlResolver extends AbstractUrlResolver<ReviewModel>
{
	private final String CACHE_KEY = DefaultReviewPageUrlResolver.class.getName();
	private String pattern;
	protected String getPattern()
	{
		return pattern;
	}
	@Required
	public void setPattern(final String pattern)
	{
		this.pattern = pattern;
	}
	@Override
	protected String getKey(final String source)
	{
		return CACHE_KEY + "." + source.hashCode();
	}
	@Override
	protected String resolveInternal(final ReviewModel review)
	{
		String url = getPattern(); // /review/{product-name}/{product-code}/{review_code}
		if (url.contains("{product-name}"))
		{
			url = url.replace("{product-name}", urlSafe(review.getProduct.getName()));
		}
		if (url.contains("{product-code}"))
		{
			url = url.replace("{product-code}", review.getProduct.getCode());
		}
		if (url.contains("{review_code}"))
		{
			url = url.replace("{review_code}", review.getCode());
		}
		// example url: /review/sonyX1/2342232/12345
		return url;
	}
}
```

The following is an example bean definition of the review URL resolver:

```
<alias name="defaultReviewPageUrlResolver" alias="reviewPageUrlResolver"/>
	<bean id="defaultReviewPageUrlResolver" class="...DefaultReviewPageUrlResolver">
		<property name="threadContextService" ref="threadContextService"/>
		<property name="pattern" value="/review/{product-name}/{product-code}/{review_code}"/>
	</bean>
```

4. Implement a populator for URLs identified by the review URL resolver as follows:

```
public class ReviewModelToSiteMapUrlDataPopulator implements Populator<ReviewModel, SiteMapUrlData>
{
	private UrlResolver<ReviewModel> urlResolver;
	@Override
	public void populate(final ReviewModel model, final SiteMapUrlData siteMapUrlData) throws ConversionException
	{
		final String relUrl = StringEscapeUtils.escapeXml(getUrlResolver().resolve(model));
		siteMapUrlData.setLoc(relUrl);
	}
	public UrlResolver<ContentPageModel> getUrlResolver()
	{
		return urlResolver;
	}
	public void setUrlResolver(final UrlResolver<ReviewModel> urlResolver)
	{
		this.urlResolver = urlResolver;
	}
}
```

The following is an example Spring configuration for the populator and converter:

```
<alias name="defaultReviewModelToSiteMapUrlDataPopulator" alias="reviewModelToSiteMapUrlDataPopulator"/>
	<bean id="defaultReviewModelToSiteMapUrlDataPopulator" class="mypackage.ReviewModelToSiteMapUrlDataPopulator">
		<property name="urlResolver" ref="reviewPageUrlResolver"/>
	</bean>
	<alias name="defaultReviewModelToSiteMapUrlDataConverter" alias="reviewModelToSiteMapUrlDataConverter"/>
	<bean id="defaultReviewModelToSiteMapUrlDataConverter" parent="abstractPopulatingConverter">
		<lookup-method name="createTarget" bean="siteMapUrlData"/>
		<property name="populators">
			<list>
				<ref bean="reviewModelToSiteMapUrlDataPopulator"/>
			</list>
		</property>
	</bean>
```

5. Implement a generator for your page as follows:

```
public class ReviewPageSiteMapGenerator extends AbstractSiteMapGenerator<ReviewModel>
{
	private ReviewService reviewService;

	@Override
	public List<SiteMapUrlData> getSiteMapUrlData(final List<String> models)
	{
		return Converters.convertAll(models, getSiteMapUrlDataConverter());
	}
	@Override
	protected List<ReviewModel> getDataInternal(final CMSSiteModel siteModel)
	{
		return (List<ReviewModel>) reviewService.getReviewsForSite(siteModel);
	}
}
```

The following is an example of the Spring configuration for the generator bean:

```
<alias name="defaultReviewPageSiteMapGenerator" alias="reviewPageSiteMapGenerator"/>
	<bean id="defaultReviewPageSiteMapGenerator"
	      class="mypackage.ReviewPageSiteMapGenerator" parent="abstractSiteMapGenerator">
	        <property name="siteMapUrlDataConverter" ref="reviewModelToSiteMapUrlDataConverter"/>
	        <property name="siteMapPageEnum" value="REVIEW"/>
	        <property name="reviewService" ref="reviewService"/>
	</bean>
```

6. Add the generator bean to the `siteMapMediaJob`. You also need to overwrite the bean definition in your extension and add a new generator to the list as follows:

```
<alias name="extendedSiteMapMediaJob" alias="siteMapMediaJob"/>

	<bean id="extendedSiteMapMediaJob"
	      class="de.hybris.platform.acceleratorservices.cronjob.SiteMapMediaJob"
	      parent="abstractJobPerformable">
	        <property name="generators">
	                <list>
	                        <ref bean="pointOfServicePageSiteMapGenerator"/>
	                        <ref bean="productPageSiteMapGenerator"/>
	                        <ref bean="categoryPageSiteMapGenerator"/>
	                        <ref bean="homePageSiteMapGenerator"/>
	                        <ref bean="contentPageModelSiteMapGenerator"/>
	                        <ref bean="categoryLandingPageSiteMapGenerator"/>
	                        <ref bean="customPageSiteMapGenerator"/>
	                        <!-- your review generated is added to the list -->
	                        <ref bean="reviewPageSiteMapGenerator"/>
	                </list>
	        </property>
	        <property name="mediaService" ref="mediaService"/>
	        <property name="activateBaseSiteInSession" ref="activateBaseSiteInSessionStrategy"/>
	        <property name="cmsSiteService" ref="cmsSiteService"/>
	</bean>
```

Configuration in Backoffice

This section provides configuration instructions using Backoffice.

Cron Job for Generating Sitemap Media

To create a cron job for generating sitemap media:

1. In Backoffice, navigate to System > Background Processes > Cronjobs. A list of cron jobs appears in the main pane.
2. Use the Search bar to locate the sitemapMediaJob cron jobs. The results display one cron job instance for each of the sample stores that come with Accelerator. For example, for the electronics site, the cron job is electronics-SiteMapMediaJob.
3. Click the cron job for the required storefront. The details for that cron job load in the main pane.
4. To run the cron job, click the Run Cronjob icon ( ) in the toolbar. A pop-up message, `CronJob performed`, appears when the job is completed.
5. Now validate that your sitemap media has been properly assigned to the CMSSite. To find your CMSSite, navigate to WCMS > Website, and then click the storefront you want to verify.
6. In the Properties tab and scroll down to see the Site Map Configurations section, which displays a table of all the sitemap media generated by the cron job.

Sitemap Configuration

The sitemap configuration is stored on the CMSSite. The following procedure describes how to view and edit the configuration.

1. In Backoffice, navigate to WCMS > Website, and then click the storefront you want to modify.
2. Double-click the entry in the Site Map Configuration. The Edit form appears.
3. You can perform the following actions in the edit form:
   - Add or remove custom URLs used for the generated sitemap page of type CUSTOM.
   - Manage the languages and currencies for generating sitemap media for each of the supported sitemap page types (double-click an entry to edit it).
   - Manage what sitemap pages are supported, including deactivating a given sitemap page type (double-click an entry to edit it).
   - Access the Site Map Template, which holds the velocity template definition used to generate the sitemaps (double-click the entry to edit it).

Storefront Web Application Deconstructed

SAP Commerce Accelerator provides reference storefronts that offer best-practice features for typical B2C and B2B uses.

The web application behind the storefront is designed to make the checkout journey as easy as possible for the customer, while also enabling web developers to adjust the reference storefront to a specific project's needs.

This document focuses on a description of the features and technologies.

Web Application Configuration

All the files related to a storefront definition are placed in the *<$HYBRIS_BIN_DIR>*/modules/base-accelerator/yacceleratorstorefront directory. The structure of the directories and files is as follows:

- web/src Contains the Java source for the storefront. This includes page and Content Management System (CMS) controllers, interceptors, various custom implementations of classes required for Spring security, custom store front components, and so on.
- web/webroot
  - _ui: Contains the JavaScript and CSS styling for the current theme.

    Note

    The desktop folder contains the JS and CSS styling for desktop screens, but is not used in B2C or B2B storefronts. B2C and B2B storefronts are rendered as responsive web pages that use the styling definitions in the responsive folder.

    - shared/js: Shared JavaScript used by desktop and responsive pages.
    - responsive/common: Commonly used style sheets, JavaScript libraries and images.
    - responsive/theme-alpha: The blue theme definition.
    - responsive/theme-lambda: The black theme definition.
  - WEB-INF
    - _ui-src: Contains JS testing, full libraries, and the Less files used to generate the CSS for a theme. To learn how to generate the required UI files from this folder, see Responsive Website Build Process
    - common/tld: The tag library descriptor files for the CMS and ycommerce tags.
    - config: Spring application context files.
    - lib: The libraries required by the storefront.
    - messages: The localization files.
    - tags: The tags that are used within views.
    - views: The JSP pages, fragments and CMS components.

For more information, see About Extensions.

Structure of a Web Application

Web application structure contains several elements that define the request flow from the client to the server and back to the client. The most important elements are servlet and servlet mappings. In addition, any filter or listener can be introduced to the request flow. Short descriptions of key elements are provided below.

web.xml

The web.xml file sets the default values for the various web applications deployed in an application server.

Filter Chain

The filters are in a chain and are called in the following order. The request is delivered to the servlet.

The filter chain has three main entry points in the web.xml file:

- resourceFilter

```
<filter>
                <description>
                        ResourceFilter
                        Filter used to server file resources by bypassing the other filters.
                </description>
                <filter-name>resourceFilter</filter-name>
                <filter-class>de.hybris.platform.yb2bacceleratorstorefront.servlets.ResourceFilter</filter-class>
        </filter>

<filter-mapping><filter-name>resourceFilter</filter-name><url-pattern>/_ui/*</url-pattern></filter-mapping>
        <filter-mapping><filter-name>resourceFilter</filter-name><url-pattern>/js/*</url-pattern></filter-mapping>
        <filter-mapping><filter-name>resourceFilter</filter-name><url-pattern>/stylesheets/*</url-pattern></filter-mapping>
```

- storefrontFilterChain

```
<filter>
                <description>
                        Spring configured based chain of the spring configurable filter beans
                </description>
                <filter-name>storefrontFilterChain</filter-name>
                <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
        </filter>

        <filter-mapping><filter-name>storefrontFilterChain</filter-name><servlet-name>DispatcherServlet</servlet-name></filter-mapping>
```

- springSecurityFilterChain

```
<filter>
                <description>
```

```
                SpringSecurityFilterChain
                Supports delegating to a chain of spring configured filters. The filter name
                must match the bean name.
        </description>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping><filter-name>springSecurityFilterChain</filter-name><servlet-name>DispatcherServlet</servlet-name></filter-mapping><fil
        <description>
                Spring configured based chain of the spring configurable filter beans
        </description>
        <filter-name>storefrontFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping><filter-name>storefrontFilterChain</filter-name><servlet-name>DispatcherServlet</servlet-name></filter-mapping>
```

Filter	Description	Documentation
`resourceFilter`	The filter is used by server file resources by bypassing the other filters. It is only mapped to paths that contain files on the disk and should be served unaltered. This is used to serve the JavaScript, CSS, images, and theme files.	-
`storefrontFilterChain`	This is a reference to the chain of Spring-based filters.	Platform Filters You can find details on the `storefrontFilterChain` in the storefrontFilterChain Filters section below.
`Spring Security Filter`	<This filter is used to enable Spring security support in the application. This filter is of type `DelegatingFilterProxy`, but the filter name is `springSecurityFilterChain`. The filter name has to match the bean ID defined by Spring security.	• http://static.springsource.org/.../security-filter-chain.html : The security filter chain

storefrontFilterChain Filters

Filter	Description	See Also
`log4jFilter`	Spring-based core platform filter, configures LOG4J.	Platform Filters
`dynamicTenantActivationFilter`	Spring-based core Platform filter, deals with tenant switching while handling the `HttpRequest`.	Platform Filters
`sessionFilter`	Spring-based core Platform filter, handles JALO session attachment and deattachment from the `HttpSession`.	Platform Filters
`RequestLoggerFilter`	The filter logs each HTTP request received by the web application. This is a useful filter during development, but is not recommended for use in production systems. It logs each request along with the time it took to process the request and generate the response.	-
`characterEncodingFilter`	The filter ensures proper encoding of the frontend page content.	-
`CMSSiteFilter`	This filter sets up the CMS integration for the application. It uses the requested URL to select the current site and sets up the session catalog versions. It also handles the preview data and redirecting to specific pages based on deep preview links.	-
`StoreFrontFilter`	This is the storefront application specific filter that initializes new sessions. It is responsible for calling the `StoreSessionFacade` to initialize each new session. The `StoreSessionFacade` sets up the initial language and currency for the session. This filter is also responsible for recording each request in the `BreadcrumbBuilder`, which is used when building a historical breadcrumb trail.	-

Listeners

These are servlet context listeners.

Listener	Description	See Also
`HybrisContextLoaderListener`	The SAP Commerce provides a `HybrisContextLoaderListener` for setting the global `ApplicationContext` automatically as a parent of your `WebApplicationContext`. You can use the global bean definitions, for example, to implement them into your `WebApplicationContext` beans. If you make a `getBean` call to your `WebApplicationContext`, it is checked whether there is a definition available. If not, the parent `ApplicationContext` is used.	Spring Framework in the SAP Commerce
`RequestContextListener`	If you want to enable the usage of the web application-specific scope: `session` and `request`, you must add the Spring `RequestContextListener`.	Spring Framework in SAP Commerce

Servlet Mapping

Servlet mapping specifies the web container of which a Java servlet should be invoked from a URL given by a client.

Servlet	Description	See Also
`DispatcherServlet`	The storefront web application uses the Spring MVC framework to handle requests. The core of the Spring MVC framework is the `DispatcherServlet`. It has to be configured in your `web.xml` file for activating the dispatching of incoming requests to the special MVC controller.	Using Spring MVC section ofSpring Framework in SAP Commerce

AddOn Extension Filter

The SAP Commerce AddOn concept adds extension possibilities to the storefront extensions. To avoid building the SAP Commerce Platform each time you add some storefront content, such as JSPs, CSS, images, and so on, there is a servlet `addonfilter` which copies over the required files to the target storefront extension.

Each of the resources you add using the AddOn extensions are associated with an URL. The process of copying the resources occurs when the browser sends a request for such resources. It checks if there is a newer version of the target item, and, if it exists, copies it over to the target storefront extension. Then, it passes control on to the next item in the chain.

This filter copies the front-end related resources. It does not copy any Java source code. If you add any logic-related items, you have to build the SAP Commerce Platform.

For more information, see AddOn Concept.

Disabling the AddOn Filter

The filter should not be used in the production environment as it reduces the performance of the system. You can disable the filter in the `local.properties` file, by setting the property value to *<false>*:

```
addonfilter.active=false
```

After adding or changing the property, you do not need to build your system. You can use the SAP Commerce Administration Console to change the property value.

Authentication and Security

SAP Commerce Accelerator uses the Spring Security library for authentication and security. The `SavedRequestAwareAuthenticationSuccessHandler` provided by Spring is extended to provide additional logic, particularly with cart restoration and cart merging that can occur when a customer logs into their account. For more on storefront security, see Spring Security.

Cart Merging

The process initiated by the `StorefrontAuthenticationSuccessHandler` makes a call to the commerce layer to merge or restore the cart. In the case of cart merging, the user's saved cart is merged into their current session cart. If there are any modifications to the cart, these will be displayed once on the cart page. If a user is logging in during checkout, they are redirected to the cart page to review their entire cart, which allows the customer to be aware of any changes. For more information on how cart merge is implemented in the commerce layer, see Cart Merging.

MVC

The SAP Commerce Accelerator storefront is based on the Spring Model View Controller (MVC) framework. The MVC pattern provides a clean separation of objects into three categories, models for maintaining data, controllers for handling events, and views for rendering data.

Annotation-based Page Controllers

Page Controllers provide data that will be rendered on the views. These are Spring MVC 3 controllers that use annotations for URL mapping both at the controller and method level. Further reading for how to provide SEO URLs.

Constants defined in `ControllerConstants` provide the common view mappings for various pages, CMS components, and fragments. The following is an example from the `Controller Constants Interfaces`.

```
ControllerConstants.Actions.Cms
ControlelrConstants.Views.Cms
ControllerConstants.Views.Pages.Account
ControllerConstants.Views.Pages.Password
ControllerConstants.Views.Pages.Checkout
ControllerConstants.Views.Pages.Error
ControllerConstants.Views.Pages.Cart
ControllerConstants.Views.Pages.StoreFinder
ControllerConstants.Views.Pages.Misc
ControllerConstants.Views.Fragments.Cart
ControllerConstants.Views.Fragments.Checkout
ControllerConstants.Views.Fragments.Product
```

`AbstractPageController` is used as the super class of all page controllers, it inserts the request, languages, currencies, `currentLanguage`, `currentCurrency`, and user variables into all page models. These are the most common objects used by various pages, fragments, and CMS components. For example, the header makes use of all of them for the language and currency selector as well as to display the welcome message for the user.

Annotation-based CMS Component Controllers

All functionality is provided by the `DefaultCMSComponentController` unless a specific functionality is desired. The job `CMSComponentController` is used to populate the page model with component properties, which is accomplished in the `fillModel` method. Much like the page controllers a CMS controller return a view. The `getView` method from the `AbstractCMSComponentController` can be overridden to return the view specific to the new component. The CMS component controllers are always invoked by the CMS component tag.

For more information, see WCMS Integration.

Validation

Form validation in the SAP Commerce Accelerator storefront is handled by Spring Forms. Two custom validators are `PaymentDetailsValidator` and the `EqualAttributesValidator`. Additional tags have been added to simplify error display for form elements in *<$HYBRIS_BIN_DIR>*`/ext-template/yacceleratorstorefront/web/webroot/WEB-INF/tags/form` directory. If the provided tags are not sufficient Spring form tags can also be wrapped in `errorSpanField` elements as in `paymentDetailsForm.tag` tag file:

```
<template:errorSpanField path="expiryMonth">
        <form:select id="ExpiryMonth" path="expiryMonth" cssClass="card_date">
                <option value="" label="<spring:theme code='payment.month'/>"/>
                <form:options items="${months}" itemValue="code" itemLabel="name"/>
        </form:select>
</template:errorSpanField>
```

JSON Binding for Servicing Ajax Requests

Some controllers, for example, the `CheckoutController` return Java Script Object Notation (JSON) data for Asynchronous JavaScript (Ajax) requests. This is handled by the Jackson message converter `org.springframework.http.converter.json.MappingJacksonHttpMessageConverter`, defined in the `spring-mvc-config.xml` file. The actual string is eval()'ed into a JSON object by JQuery, by specifying the `dataType` parameter as JSON.

```
$.ajax({
   url: "/accelerator/url/",
   dataType: "json",
   success: function(data){

   }
});
```

In the `@RequestMapping` for these controller methods .JSON is appended to the value just as a clue to the developer.

Messages

Error, status, and information messages for the `yacceleratorstorefront` extension are located in `base.properties` files, with translations in `base_de.properties` and `base_ja.properties` files. The Spring message tags can be used to display the messages.

```
<spring:message code="message.code"/>
```

bean.xml Files

Spring context files in the SAP Commerce Accelerator storefront define the beans used within the Spring environment.

Filename	Description
`web-application-config.xml`	Main entry point that imports all other configuration files.
`spring-mvc-config.xml`	Contains beans for Spring MVC functionality.
`spring-security-config.xml`	Spring security related beans, URL interceptors.
`spring-filter-config.xml`	Provides bean definitions for all Spring-based filters.

Interceptors

There are several configured in the `spring-mvc-config.xml` file, which are used for:

- Setting the current locale (`de.hybris.platform.yacceleratorstorefront.interceptors.LocaleInterceptor`).
- Putting the Google API Key into the request (`de.hybris.platform.yacceleratorstorefront.interceptors.GoogleAPIKeyInterceptor`).
- Loading the appropriate CMS page slots into the model (`de.hybris.platform.acceleratorstorefront.interceptors.CmsPageInterceptor`)
- Additional security for HTTPS requests (`de.hybris.platform.yacceleratorstorefront.security.GUIDInterceptor`)

You can add more interceptors if additional low-level functionality is required.

View

The storefront uses JSP 2.1 as its view-rendering technology. The Spring MVC controllers specify a view name to use and spring is configured to look in the *<$HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/views` folder for a JSP file that matches the view name.

The JSP files are all located under the web applications *<$HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF` folder, which means they are not directly accessible from the browser. The only way that the JSPs are used is as the view renderer.

Tag files are used to provide common functionality for the JSPs. Also, tag files are used to build a simple common master template that removes the need for a third-party tool like SiteMesh.

The JSP and tag files predominantly use JSP Expression Language (EL) and the JSP Standard Tag Library (JSTL). The use of standard technologies in the view layer should simplify adoption of the Accelerator storefront.

JSPs

The JSP files are organized into three groups under the *<$HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/views/` folder.

- `pages`: Contains JSPs that are used to render a complete HTML web page that can be returned to the client browser.
- `fragments`: Contains JSPs that are used to render parts of HTML web pages that are requested or updated asynchronously. This typically happens using an Ajax request from the client browser.
- `cms`: Contains JSPs that are used to render the view for specific CMS components.

Page JSPs

The Page JSPs are used to render the view for a complete HTML web page. These JSPs are organized into subfolders based on their main area of functionality.

The following example page JSP that is used to render CMS content pages is shown below:

```
<%@ page trimDirectiveWhitespaces="true" %>
<%@ taglib prefix="template" tagdir="/WEB-INF/tags/template" %>
<%@ taglib prefix="cms" uri="/cms2lib/cmstags/cmstags.tld" %>
<%@ taglib prefix="breadcrumb" tagdir="/WEB-INF/tags/nav/breadcrumb" %>
```

```
<template:page pageTitle="${pageTitle}">
	<div id="breadcrumb" class="breadcrumb">
		<breadcrumb:simpleBreadcrumbBar title="${title}" />
	</div>
	<cms:slot var="feature" contentSlot="${slots.Section1}">
		<div class="span-24 section1 advert">
			<cms:component component="${feature}"/>
		</div>
	</cms:slot>
	<div class="span-24 section2">
		<div class="span-4 zone_a advert">
			<cms:slot var="feature" contentSlot="${slots.Section2A}">
				<cms:component component="${feature}"/>
			</cms:slot>
		</div>
		<div class="span-20 zone_b last">
			<cms:slot var="feature" contentSlot="${slots.Section2B}">
				<cms:component component="${feature}"/>
			</cms:slot>
		</div>
	</div>
	<div class="span-24 section3 advert">
		<cms:slot var="feature" contentSlot="${slots.Section3}">
			<cms:component component="${feature}"/>
		</cms:slot>
	</div>
</template:page>
```

This page uses the `<template:page>` tag to build the HTML page. The `<template:page>` ensures that the correct HTML directives are included, it includes all the required Javascript and CSS resources, adds in the page header, navigation bar, and footer. This allows the page to concentrate on the main body of the page.

The page uses the `<breadcrumb:simpleBreadcrumbBar>` to populate the breadcrumb bar, then uses the `<cms:slot>` and `<cms:component>` tags to output the CMS configured components into the appropriate positions on the page.

Find more information on JSP tags in the Template Tags section below.

Fragment JSPs

The Fragment JSPs are used to render a view that is only part of the HTML web page and are typically integrated into the web page on the client side. This is used to handle functionality like the product quick view pop-up. The product quick view pop-up is requested by the client browser using an Ajax request. The view is rendered for a specific product and returned to the client browser. The client browser then inserts this HTML into the web page.

This means that the fragment JSPs are not responsible for rendering the whole HTML page and, therefore, do not need to use the <template:page> tag to generate a whole page.

CMS JSPs

CMS is built using different types of components. Each of these components needs to be rendered to HTML and included in the response. As you have seen above, a custom Spring MVC controller can be used to handle any logic associated with rendering the CMS component, but ultimately the component is rendered by a JSP view found in the cms folder. The JSP file name is looked up using the lowercase type code of the component.

The following is an example from the `cmsimagecomponent.jsp` file:

```
<%@ page trimDirectiveWhitespaces="true" %>

<div class="cmsimage">
	<img title="${media.altText}" src="${media.url}" alt="${media.altText}"/>
</div>
```

This is the `cmsimagecomponent.jsp` component, which is used to render the view for the `CMSImageComponent`.

This is a relatively simple component to render, but you can apply the same pattern to more complex components.

Tag Files

The JSPs use tags to provide common functionality or just to simplify the JSP and improve readability. Tags cannot be rendered directly, but must be included in a JSP file or in another tag file. The tag files are organized into several different folders under the *<$HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/tags/` folder. Tag files were introduced in the JSP 2.0 standard.

Tags are split roughly into two separate groups:

- Template tags: Are used as a master template to include all the required layout and structural elements on each page
- Component tags: Are used to encapsulate simple functionality into a reusable component.

Template Tags

The example use of the `<template:page>` tag was described above. This tag is responsible for defining the common layout required for each page.

This is the content of the `page.tag` file:

```
<%@ tag body-content="scriptless" trimDirectiveWhitespaces="true" %>
                                    <%@ attribute name="pageTitle" required="false" rtexprvalue="true" %>
                                    <%@ attribute name="pageCss" required="false" fragment="true" %>
                                    <%@ attribute name="pageScripts" required="false" fragment="true" %>
                                    <%@ taglib prefix="template" tagdir="/WEB-INF/tags/template" %>
                                    <%@ taglib prefix="header" tagdir="/WEB-INF/tags/common/header" %>
                                    <%@ taglib prefix="footer" tagdir="/WEB-INF/tags/common/footer" %>
                                    <%@ taglib prefix="cart" tagdir="/WEB-INF/tags/cart" %>
                                    <%@ taglib prefix="nav" tagdir="/WEB-INF/tags/nav" %>

                                    <template:master pageTitle="${pageTitle}">
                                    <jsp:attribute name="pageCss">
                                    <jsp:invoke fragment="pageCss"/>
```

```
</jsp:attribute>
<jsp:attribute name="pageScripts">
<jsp:invoke fragment="pageScripts"/>
</jsp:attribute>
<jsp:body>
<div id="wrapper">
<div id="page">
<a href="#skip-to-content" class="skiptocontent">skip to content</a>
<a href="#skiptonavigation" class="skiptonavigation">skip to main navigation</a>
<header:header/>
<a name="skiptonavigation"></a>
<nav:topNavigation/>
<cart:addToCart/>
<div id="content">
<a name="skip-to-content"></a>
<jsp:doBody/>
</div>
<footer:footer/>
</div>
</div>
</jsp:body>
</template:master>
```

This tag uses the `<template:master>` tag to generate the actual HTML directives, `HTML` element, `head` element, `body` element, include all scripts and CSS resources. The main function of the `page.tag` is to include the `header` and `top navigation` sections followed by the main body of the page, and finally the standard footer.

You can customize the `page.tag` tag if you need to change the main layout elements of the page.

The `master.tag` is used to generate the required HTML to make the compliant HTML web page. The advantage of putting this all in one tag is that it only needs to be done and maintained in one place and is then shared by all the pages that need it.

Component Tags

The rest of the tag files are reusable components that are included on JSP pages. For example, the `/template:master<>format:price>`/div tag is used to render a> price. Here is an example of the tag being used: <footer:footer/> </div> </div

```
jsp:attribute name="pageScripts"<>format:price priceData="${entry.basePrice}" displayFreeForZero="true"/
                ><jsp:invoke fragment="pageScripts"/>
        </jsp:attribute>
        <jsp:body>
                <div id="wrapper">
                        <div id="page">
                                <a href="#skip-to-content" class="skiptocontent">skip to content</a>
                                <a href="#skiptonavigation" class="skiptonavigation">skip to main navigation<%@ tag body-content="scriptless" trimDi
<%@ attribute name="pageTitle" required="false" rtexprvalue="true" %></a>
                                <header:header/>
<%@ attribute name="pageCss" required="false" fragment="true" %>

<<a name="skiptonavigation"><%@ attribute name="pageScripts" required="false" fragment="true" %/a>>

                                <%@ taglib prefix="template" tagdir="/WEB-INF/tags/template" %>
<<%@ taglib prefix="header" tagdir="/WEB-INF/tags/common/header" %nav:topNavigation/>
><%@ taglib prefix="footer" tagdir="/WEB-
                                INF/tags/common/footer" %<>cart:addToCart/
><
                                %@ taglib prefix="cart" tagdir="/WEB-INF/tags/cart" %nullnullnullnullnullnullnullnullnullnullnullnullnullnullnullnul
null<null%@ taglib prefix="nav" tagdir="/WEB-INF/tags/nav" %null>
```

In the above example, the tag is being used to render the base price for a cart entry product. In this case, if the price is zero, then we want to display the Free promotional text instead of the numerical value.

Find an example implementation of the tag below:

```
<%@ tag body-content="empty" trimDirectiveWhitespaces="false" %>
<%@ attribute name="priceData" required="true" type="de.hybris.platform.commercefacades.product.data.PriceData" %>
<%@ attribute name="displayFreeForZero" required="false" type="java.lang.Boolean" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<c:choose>
        <c:when test="${priceData.value > 0}">
                ${priceData.formattedValue}
        </c:when>
        <c:otherwise>
                <c:if test="${displayFreeForZero}">
                        <spring:theme code="text.free" text="FREE"/>
                </c:if>
                <c:if test="${not displayFreeForZero}">
                        ${priceData.formattedValue}
                </c:if>
        </c:otherwise>
</c:choose>
```

Theme Support

The SAP Commerce Accelerator storefront extends the theme support provided by Spring MVC to load resources based on both the current CMS site as well as the theme set on the site. The resource bundles are located in the *<$HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/messages/` folder. Resources are loaded in the following order (highest priority first):

1. Site, for example: `site-electronics.properties`.
2. Theme, for example: `theme-blue.properties`.
3. Base, for example: `base.properties`.

As these are standard Java resource bundles, each properties file can be specialized for each language.

Each theme has a separate directory of resources under the *<$HYBRIS_BIN_DIR>*/modules/base-accelerator/yacceleratorstorefront/web/webroot/_ui/responsive/*<theme name>*/ path, for example *<$HYBRIS_BIN_DIR>*/modules/base-accelerator/yacceleratorstorefront/web/webroot/_ui/responsive/theme-blue. You can use this resource directory to hold CSS files and images included from the CSS. Theme resource look-up is done using the <spring:theme> tag, for example:

```
<spring:theme code="basket.page.total"/>
```

This looks up the resource with the key basket.page.total in the resource bundle for the current site. If it cannot be found there, it falls back to the resource bundle for the current theme, and if not found there, it looks in the base resource bundle.

In order to customize the behavior of the Spring MVC theme support the yacceleratorstorefront extension redefines the themeSource and themeResolver beans.

For more information, see http://static.springsource.org/spring/docs/(...)/ThemeResolver.html : Theme Resolver documentation.

Resources

The SAP Commerce Accelerator storefront exposes simple resource files under the *<$HYBRIS_BIN_DIR>*/modules/base-accelerator/yacceleratorstorefront/web/webroot/_ui/ web application path. These resources include CSS files, JavaScript files, related to CSS images, and theme-specific versions of the same. These files are not interpreted by the web application and are served directly from disk. The *<$HYBRIS_BIN_DIR>*/modules/base-accelerator/yacceleratorstorefront/web/webroot/_ui/ path prefix is used as this is unlikely to interfere with any of the content or SEO URLs required by the site itself. In a production environment, it is recommended that these static file resources are served directly by the front end web server.

CSS and JavaScript Compression with WRO4J

SAP Commerce Accelerator storefronts use wro4j for compressing CSS style files and JavaScript source files.

Configuring Wro4j

To use wro4j, you must enable it in *<$HYBRIS_BIN_DIR>*/modules/base-accelerator/yacceleratorstorefront/project.properties, as follows:

```
storefront.wro4j.enabled=true
```

You can enable logging with Log4j 2 by setting the following properties in *<$HYBRIS_BIN_DIR>*/platform/project.properties:

```
log4j2.logger.wro4j.name=ro.isdc
log4j2.logger.wro4j.level=debug
log4j2.logger.wro4j.appenderRef.stdout.ref=STDOUT
```

If you are using the older version of Log4j, you can enable it by setting the following properties in *<$HYBRIS_BIN_DIR>*/platform/project.properties:

```
log4j.logger.ro.isdc=DEBUG
log4j.logger.ro.isdc.wro.model.resource.locator.support=DEBUG
```

The first one is a general property, and the second one is for information about compressing files.

A standard set of configuration properties for wro4j is located in *<$HYBRIS_BIN_DIR>*/modules/base-accelerator/yacceleratorstorefront/web/webroot/WEB-INF/wro.properties. The following notes are important to consider with regards to these configuration properties:

- When Accelerator is deployed into production, you must set debug=false.
- The only mandatory configuration properties are the listing of pre-processors and post-processors, and it is very important to note that the order of the listed processors matters.
- To make wro4j work correctly when the AcceleratorAddOnFilter is enabled, you must set either resourceWatcherUpdatePeriod or cacheUpdatePeriod to a value greater than zero (0).

Developing with Wro4j

The wro.xml file contains all the .CSS and .JS files from the storefront extension that are processed by wro4j (minified, and so on). You need to update this file each time new .CSS or .JS is introduced in the storefront extension. In the case of AddOn resources, a different mechanism is used that makes resources installed from AddOns available automatically after running certain ant commands, as described in the **AddOns Support** section below.

Resources inside wro.xml are gathered in groups. The group name is referenced inside jsp or tag files using single files, such as *<group_name>*.js and *<group_name>*.css.

ℹ Note

The order of files in the list is important, because they are processed in the order of appearance. For example, jquery libraries should be listed first in order to use jquery in other .JS files. If you have in-line javascript between includes of .JS files, you should consider splitting files among a few groups, in order to maintain the original order of the includes and the in-line scripts.

Resource Import from CSS

Wro4j does not support importing resources directly from CSS (using the @import url) when there is no protocol specified. Such imports should be done in the tag files instead. The following is an example where a line is added to the relevant UIExperience styleSheet.tag:

```
...
<link rel="stylesheet" type="text/css" href="//fonts.googleapis.com/css?family=Open+Sans:400,300,300italic,400italic,600,600italic,700,700italic,800
...
```

Supporting AddOns

Some customizations for `ant` were introduced to support AddOns with wro4j.

The `addoninstall` command inserts the following property into an AddOn's `project.properties` file:

```
<storefrontTemplateName>.wro4jconfigscan.<addonname>=true
```

Also, when you run the standard `ant` command (on the storefront or platform), the `generate_addons_wroxml` macro is invoked, which generates the `wro_addons.xml` file with wro4j groups definitions for the installed AddOns on the storefront. In this process, the property created by the `addoninstall` command is used. The `project.properties` file from the AddOn extension is checked to detect if a given AddOn should apply for a given storefront extension. If this is the case, the `project.properties` file is processed to gather information about which .JS and .CSS files should be included.

wro_addons_template.xml

The `wro_addons_template.xml` template file is used for AddOn groups definitions. It is located in the `/yacceleratorstorefront` folder. This file cannot be changed because it is used as a template for the generation of `wro_addons.xml`.

wro_addons.xml

The `wro_addons.xml` file is generated by `buildcallbacks.xml` from `wro_addons_template.xml`. This file contains all of an AddOns .JS and .CSS files, gathered into groups, and which are compressed if wro4j is enabled. If no AddOns are installed, the file contains empty groups for responsive and desktop.

buildcallbacks.xml

New macros for `wro_addons.xml` file generation have been added, as follows:

Macro	Description
create_empty_wro_addons_xml	Copies `wro_addons_template.xml` into `wro_addons.xml` so it exists even if no AddOns are installed. This macro is invoked by `yacceleratorstorefront_after_build` and `create_wro_addons_xml`.
fix_root_node_for_wro_addons_xml	Inserts correct namespaces into `<groups>` root node in `wro_addons.xml` so it's properly validated by wro4j. This macro is invoked by `yacceleratorstorefront_after_build`.
delete_wro_addons_xml	Removes `wro_addons.xml`. This macro is invoked by `yacceleratorstorefront_after_clean` and `yacceleratorstorefront_before_build`.
generate_wro_addons_xml	Reads the `project.properties` file and gathers CSS and JS locations. This macro is invoked by `yacceleratorstorefront_after_build`.
create_wro_addons_xml	Completes groups in `wro_addons.xml` with gathered CSS and JS locations. This macro is invoked by `generate_wro_addons_xml`.

acc-ant-addons.xml

The `generate_addon_properties_file` macro was extended to insert into an AddOn's `project.properties` file the following new key:

```
<storefrontTemplateName>.wro4jconfigscan.<addonname>=true
```

This key informs the system that the AddOn is used by wro4j.

Additional Configuration Options

Wro4j can use many more pre-processors and post-processors, but the decision about which ones to use will depend on the requirements of your specific implementation project. The same can be said regarding all of the available properties that can be used.

B2C and B2B Configurations

If you are using extgen to create two storefronts that are running together, the `buildcallbacks.xml` file of the `yacceleratorstorefront` extension needs to be updated before installing or running a recipe, so that the `generate_wro_addons_xml` is unique for both storefronts. To do this simply change the following lines and references in `buildcallbacks.xml`:

```
-   <macrodef name="generate_wro_addons_xml">
-   <macrodef name="create_empty_wro_addons_xml">
-   <macrodef name="fix_root_node_for_wro_addons_xml"/>
-   <macrodef name="delete_wro_addons_xml" />
+   <macrodef name="yacceleratorstorefront_generate_wro_addons_xml">
+   <macrodef name="yacceleratorstorefront_create_empty_wro_addons_xml" />
+   <macrodef name="yacceleratorstorefront_fix_root_node_for_wro_addons_xml" />
+   <macrodef name="yacceleratorstorefront_delete_wro_addons_xml" />
```

This will ensure that the call to wro4j is unique to both storefronts.

Limitations

The following is a list of limitations that may be encountered when working with wro4j:

- In the `wro.xml` file, there is no configuration support for wildcard directories (such as `addons/*/desktop/common/...`), so each AddOn that is used should be explicitly listed. Also note, some AddOns (such as the `assistedservicestorefront` AddOn) use only one UI directory (that is, the `responsive` UI directory) for all UI versions.

- There is no elegant way to enforce ordering without listing the individual scripts in the order required. This is required because `acc.skiplinks.js` needs to be last.
- For AddOns, it is not possible to disable resource minification. In other words, you cannot set `sattribute minimize="false"`.

JavaScript

The SAP Commerce Accelerator storefront uses a number of different JavaScript libraries to provide dynamic functionality. The JavaScript files are located in the *<$HYBRIS_BIN_DIR>*`/modules/base-accelerator/yacceleratorstorefront/web/webroot/_ui/responsive/common/js/` folder.

The main library used is jQuery. jQuery is a fast and concise JavaScript library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for web development that has become very popular. The jQuery user interface library is used to provide basic user interface components and behaviors.

Several other jQuery plug-ins are used to provide additional behavior, including image carousel, client side HTML templating, blocking user interactions during Ajax updates, pop-up windows, review stars, form elements, and many more. Review the files in the *<$HYBRIS_BIN_DIR>*`/ext-template/yacceleratorstorefront/web/webroot/_ui/responsive/common/js/` folder for more details.

For additional information, see http://jquery.com/ : jQuery project.

Breadcrumb Support

The `yacceleratorstorefront` extension uses simple design to display breadcrumb information. Breadcrumb are represented in the system with a list of object of type `Breadcrumb`. Each breadcrumb element contains three pieces of information:

- `url`: Is relative URL address to where the breadcrumb piece points.
- `name`: Is the displayed name of a breadcrumb piece.
- `linkClass`: Represents the CSS class with which the single piece of a breadcrumb is decorated.

A page that is required to display breadcrumb uses a `breadcrumb` tag that requires single attributes that are a list of `Breadcrumb` elements. Default breadcrumb tag displays at the beginning one link pointing to homepage, and thereafter links all elements of a provided list are displayed.

There are a number of sample implementations of `BreadcrumbBuilder` interface in the `yacceleratorstorefront` module that can be used to build breadcrumbs for some common purposes like for product, category, search result page or simple breadcrumb with one single link at the end. These breadcrumb builders are defined as Spring beans and used in the MVC controllers. Each breadcrumb builder includes a list of `Breadcrumb` objects that can be directly passed to the breadcrumb tag at the displayed page.

Here is a simple example of breadcrumb-related pieces of code for a product details page:

```
// Somewhere in product page controller
(...)
model.addAttribute("breadcrumbs", productBreadcrumbBuilder.getBreadcrumbs(productData));
(...)


// Somewhere in product page
(...)
<div id="breadcrumb" class="breadcrumb">
        <breadcrumb:breadcrumb breadcrumbs="${breadcrumbs}"/>
</div>
(...)
```

Related Information

yacceleratorstorefront Extension

B2C Accelerator Module

The B2C Accelerator module lets you set up fully-functional apparel and electronics storefronts.

Features	Architecture	Implementation
		
Multi-Dimensional Products	apparelstore Extension electronicsstore Extension	Removing Apparel and the Apparel Stores

B2C Accelerator Features

The B2C Accelerator module provides the multi-dimenional products feature, which allows you to configure products to include multiple attributes.

Multi-Dimensional Products
This page describes how to build and and expand multi-dimensional products.

Multi-Dimensional Products

This page describes how to build and and expand multi-dimensional products.

Overview

Multi-dimensional products have multiple attributes. Multi-dimensional products are described using multiple `Categorys` (dimensions) and multiple `VariantValueCategorys` (attributes).

To understand the difference between mult-dimensional products and variants, consider the following:

- Products in Powertools are multi-dimensional because they include a group products that can be indirectly related. For example, you can list safety footwear a being related to powertools.
- Products in Electronics are multi-dimensional because they include a group of products that match that cateory. For example, electronics can include digital cameras, music players, GPS devices, etc.
- Products in the Apparel store contain variants, which are the same basic model of a product, but that differ in aspects (color, size, pattern, etc.).

ℹ Note

For more information, see the following topics:

- B2B Multi-Dimensional Products

Building Product Variants by Adding Dimensions

A product variant is a product that varies from the base product in some way that creates uniqueness but not independence. The VariantCategory and VariantValueCategory Object Diagram below shows how the `VariantCategory` dimensions color, size, and fit are used to describe the different aspects of a product. A product can have as many as three dimensions in the out-of-the-box version of SAP Commerce Accelerator. However, they do not have to be color, size, and fit. As you define a product, you can define your attributes as weight, length, or width for example or any other three dimensions. You define the dimensions in the Backoffice Administration Cockpit or through ImpEx when you define the product; see Creating Multi-Dimensional Products. If you need more than three attributes, see Creating Additional Dimensions for Multi-Dimensional Products for a description of how to create and add them.

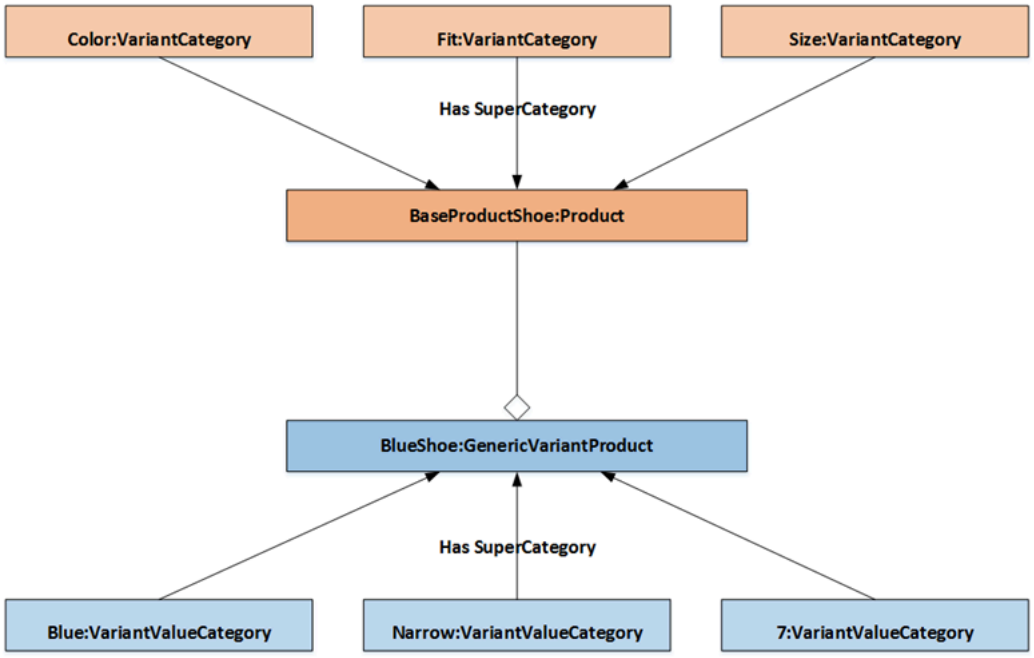


Figure: VariantCategory and VariantValueCategory Object Diagram

The following table describes the elements within the diagram.

Element	Description
`Product`	This is the base product, such as a shoe. Base products begin with no `VariantValueCategorys` (attributes).
`GenericVariantProduct`	A `GenericVariantProduct` describes a product with at least one `VariantValueCategory` (attribute), such as red.
`SuperCategory`	A `SuperCategory` is a root category that serves as a container for other subordinate categorys. The subordinate categorys all share something in common with the `SuperCategory`.
`VariantCategory`	A `VariantCategory` is equivalent to one of the dimensions, such as size, color, or fit. `VariantCategorys` are containers for `VariantValueCategorys`.
`VariantValueCategory`	A `VariantValueCategory` is a product attribute that describes something about the product, such as red, green, large, small, narrow, or wide.

Example

You can use ImpEx or the Backoffice to create a multi-dimensional product. Use the following steps:

1. Create your `VariantCategorys` (dimensions). For a shoe, this might be color, fit, and size. For a drill, this might be power source and chuck size. For a contact lens, this might be power, base curve, diameter, cylinder, axis, add power, color, and brand. Link these to their `SuperCategory`.

> **Note**
>
> This contact lens example requires extending the base implementation of Accelerator to support additional dimensions. See Creating Additional Dimensions for Multi-Dimensional Products.

2. Create your `VariantValueCategorys`. `VariantValueCategorys` (attributes) are the values of the dimensions. For a shoe, they might be color - brown, fit - narrow, and size - 7, or they might be color - black, fit - medium, and size - 7.

> **Note**
>
> Even though both examples are size 7, the shoes have other unique characteristics, which define them as separate products.

3. Create your product and add its `SuperCategorys`.

For additional information, see also: B2B Multi-Dimensional Products.

Loading a Multi-Dimensional Product Using ImpEx

You can use ImpEx to load `GenericVariantProducts`, `VariantCategorys`, and `VariantValueCategorys`. For examples, see Loading a Multi-Dimensional Product Using ImpEx.

Indexing Solr for Multi-Dimensional Products

Product search services are provided by the Apache Solr search engine. All the product information must be indexed for Apache Solr to provide search results for multi-dimensional products.

All the parameters used for the indexing process are defined in the `powertoolsstore` extension located in the `/resources/powertoolsstore/import/coredata/store/powertools/solr.impex` file.

Indexing all of the dataset's products is a two step process. First, you must index the base products, which are products with no dimensions, and then you index all of the multi-dimensional products.

1. The following query is run against the platform database and selects only base products by filtering out all the products whose type is not `GenericVariantProduct`. Only base products fit this description.

```
SELECT {PK} FROM {Product} WHERE {*code*} NOT IN(   {{ SELECT {*code*} FROM {GenericVariantProduct} }})
```

2. You now need to capture all the multi-dimensional products and index them. To do this, `de.hybris.platform.commerceservices.search.solrfacetsearch.provider.impl.VariantCategorySource` was modified to collect `VariantValueCategorys` and uses the `de.hybris.platform.commerceservices.search.solrfacetsearch.provider.impl.VariantProductSource`, which was modified to collect `VariantCategorys`, to retrieve all the multi-dimensional products and make them available for indexing.

Validation Rules for the Multi-Dimensional Structure

Validation rules are used by the Backoffice, any cockpit, and properly configured ImpExto ensure the integrity of the multi-dimensional data structure. The following rules are enforced by the validators within the `Service Layer`:

- A `GenericVariantProduct` can only have supercategories of type `VariantValueCategory`.
- A `GenericVariantProduct` must have at least one `VariantValueCategory`.
- A `VariantCategory` is composed of `VariantValueCategorys`. While a `VariantCategory` represents a single product attribute, many kinds of this attribute can exist. For example, color is a single product attribute and is represented by a `VariantCategory`, but the product might come in a wide variety of colors. Each of these colors is represented by a single `VariantValueCategory`.
- The attributes defined by a `VariantCategory` have an ordering. This ordering is defined by the order the `VariantCategory` objects are chained. Each `VariantCategory`, if it is not the first in the ordering has another `VariantCategory` in its supercategories. For reordering of the attributes you need to redefine the supercategories of the `VariantCategory` objects.
- When a `VariantCategory` has `VariantValueCategorys`, the `VariantValueCategorys` appear in a specific order. This order is defined by a `VariantValueCategory` Sequence attribute. If your `VariantValueCategorys` contain five colors, for example red, blue, purple, black, and green, then these colors always appear in that Sequence, and the Sequence cannot be changed when you define the product.
- If a `GenericVariantProduct` refers to several `VariantValueCategory` entries, it is validated against its base product and the `VariantCategory` entries of the base product. The verification prevents you from assigning a `GenericVariantProduct` entry into a wrong set of `VariantValueCategory` entries.
- The ordering of each `VariantCategory` entry is validated as chained. This is true for all `VariantCategorys` of the `BaseProduct` and exists within one setup of a `GenericVariantProduct` entry.

Validators in the Service Layer

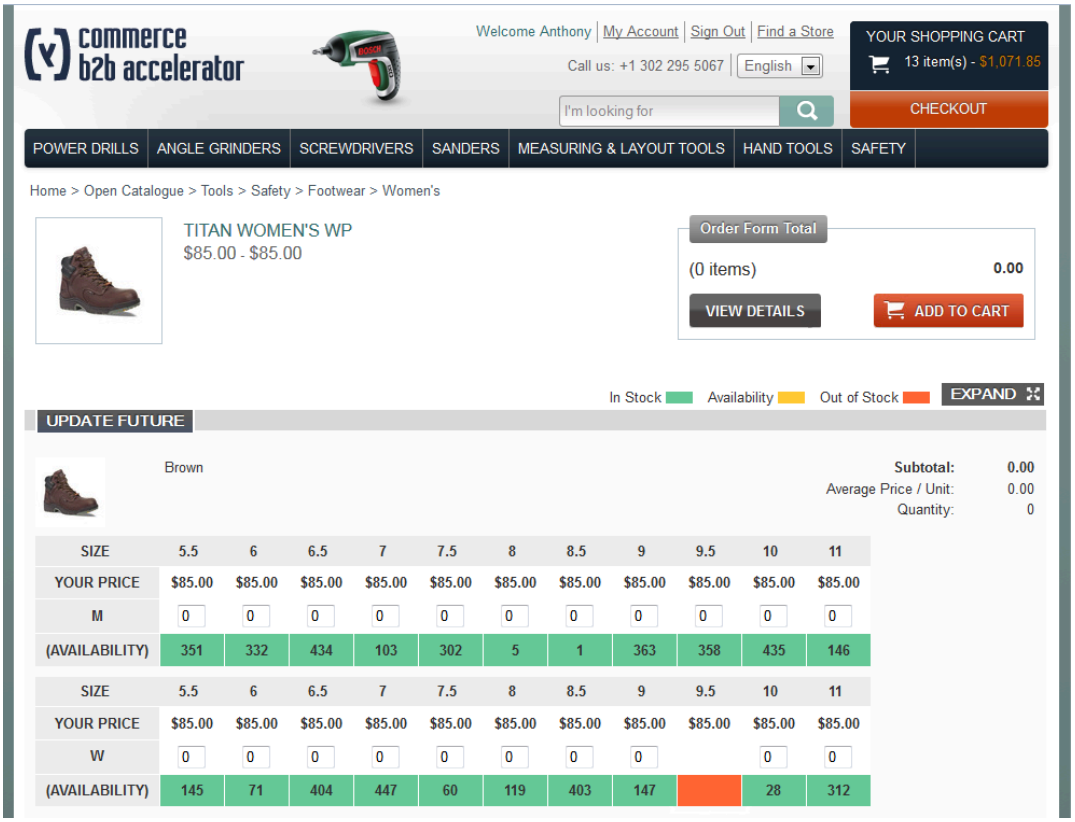
The validation rules are only used when using the Backoffice or any Cockpit to save the models through the Service Layer. If you are importing data through ImpEx, it is strongly recommended that the ImpEx validation be enabled as well. To enable ImpEx validation, you must deactivate the legacy ImpEx mode by setting the following property to false:

```
impex.legacy.mode=false
```

With this deactivated, ImpEx imports using the SAP Service Layer, which triggers the multi-dimensional product validators.

Product Detail Order Grid

The Product Detail Order Grid is used to display a multi-dimensional product. This grid is used within the Product Detail Page for a multi-dimensional product and makes it easy to place an order for more than one SKU at a time. As shown below, you simply enter the quantity for each product into the grid and then click **Add to Cart**.



Note

To view the order form, the user must be logged in to the site.

The `ProductPageController` has been modified to render the configured product into an Product Detail Order Grid based on its Backoffice definition.

Support for Any Product

The **Add to Cart** button was changed to support the selection of single or multiple Stock Keeping Units (SKUs). The required changes were made to the `CartFacade` and the `AddToCartController`. The modifications were made because the addition of Product Detail Order Grid required an Add to Cart functionality that could handle multiple products with one **Add to Cart** event.

See also:

- Add to Cart in the Product Detail Order Grid
- commercefacades Extension

Support for more than Three Dimensions

While Accelerator supports three dimensions out-of-the-box, it can support addition dimensions. Using ImpEx or the Backoffice, you can extend the model to support any number of dimensions. This also requires refactoring of the Product detail page, the Order Form page, and the Shopping Cart.

For more information about adding dimensions to your model, see Creating Additional Dimensions for Multi-Dimensional Products.

Related Information

Modeling Product Variants
Styles and Size Variants in the SAP Commerce Product Cockpit
Catalog Guide

Loading a Multi-Dimensional Product Using ImpEx

This document provides three sample scripts using the sample data from SAP Commerce Accelerator that can be used to load `VariantValueCategory`s or `GenericVariantProduct`s using ImpEx.

Creating a VariantCategory Using ImpEx

The following code snippet demonstrates how to load a `VariantCategory` using ImpEx.

```
# Macros
                        $productCatalog=powertoolsProductCatalog
                        $catalogVersion=catalogversion(catalog(id[default=$productCatalog]),version[default='Staged'])[unique=true,default=$
                        $supercategories=supercategories(code, $catalogVersion)
                        $baseProduct=baseProduct(code,$catalogVersion)
                        $approved=approvalstatus(code)[default='approved']
                        $priority=variantCategoryPriority(code)
```

```
# Language
$lang=en

# Insert Variant Category
INSERT_UPDATE VariantCategory;code[unique=true];name;$supercategories;hasImage;$catalogVersion;allowedPrincipals(uid
;B2B_Color;Color;;true
;B2B_Fit;Fit;B2B_Color;false
;B2B_Size;Size;B2B_Fit;false
```

The *<hasImage>* property: Include hasimage if there is a category image to display.

Note

The ordering of the `VariantCategory` objects is defined by chaining them. In the sample above the order is Color, Fit, and Size. The `VariantCategory` of B2B_Fit has B2B_Color as a supercategory and B2B_Size has B2B_Fit as supercategory. This interconnection of the `VariantCategory` objects defines the order for display of the selectors on the front-end and also the priority of the dimensions for the grid view.

Creating VariantValueCategorys Using ImpEx

The following code snippet demonstrates how to load a `VariantValueCategory` using ImpEx.

```
# Macros
$productCatalog=powertoolsProductCatalog
$catalogVersion=catalogversion(catalog(id[default=$productCatalog]),version[default='Staged'])[unique=true,default=$
$supercategories=supercategories(code, $catalogVersion)
$baseProduct=baseProduct(code,$catalogVersion)
$approved=approvalstatus(code)[default='approved']
$priority=variantCategoryPriority(code)

# Language
$lang=en

# Insert variant value category
INSERT_UPDATE VariantValueCategory;code[unique=true];name;$supercategories;sequence;$catalogVersion;allowedPrincipals
;B2B_3_5;3.5;B2B_Size;1
;B2B_4;4;B2B_Size;2
;B2B_4_5;4.5;B2B_Size;3
;B2B_M;M;B2B_Fit;1
;B2B_W;W;B2B_Fit;2
;B2B_Black;Black;B2B_Color;1
;B2B_Brown;Brown;B2B_Color;2
```

The *<sequence>* property: This is the sequence or order of appearance of the `VariantValueCategory`s.

Creating GenericVariantProducts Using ImpEx

The following code snippet demonstrates how to load a `GenericVariantProduct` using ImpEx.

```
# Macros
$productCatalog=powertoolsProductCatalog
$catalogVersion=catalogversion(catalog(id[default=$productCatalog]),version[default='Staged'])[unique=true,default=$
$supercategories=supercategories(code, $catalogVersion)
$baseProduct=baseProduct(code,$catalogVersion)
$approved=approvalstatus(code)[default='approved']
$priority=variantCategoryPriority(code)

# Language
$lang=en

# Insert Products
INSERT_UPDATE Product;code[unique=true];$supercategories;manufacturerName;variantType(code);unit(code)[default=piece
;24097000;1805,B2B_Color,B2B_Fit,B2B_Size;TITAN 6" SOFT TOE;GenericVariantProduct

# Update product name and description
UPDATE Product;code[unique=true];name[lang=$lang];description[lang=$lang];$approved;$catalogVersion
;24097000;TITAN 6" SOFT TOE;TITAN 6" SOFT TOE;approved

# Update generic variant product information
INSERT_UPDATE GenericVariantProduct;code[unique=true];$baseProduct;name;summary;ean;$supercategories;description;$ca
;24097000_1;24097000;TITAN 6" SOFT TOE BROWN  4  M;TITAN 6" SOFT TOE BROWN  4  M;829024859728;B2B_4,B2B_M,B2B_Black;
```

The *<variantType>*: The variantType needs to be of type `GenericVariantProduct`.

Related Information

ImpEx

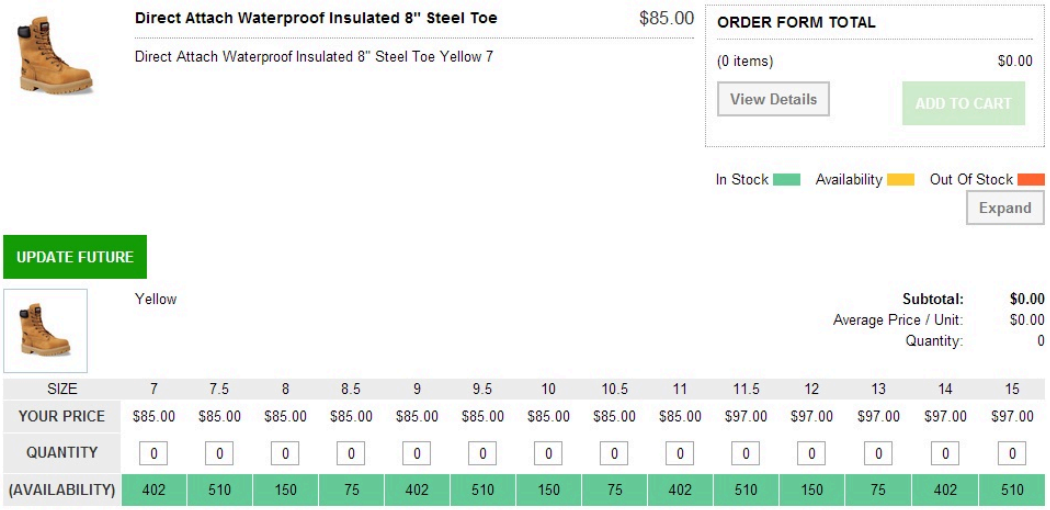
Add to Cart in the Product Detail Order Grid

This document describes the implementation of the **Add to Cart** button in the Product Detail Order Grid, its behavior, and the main components that interact to fulfill its functionality.

Overview and Features

The Add to Cart feature was redesigned because the Product Detail Order Grid can push multiple product order requests into a single Add to Cart event.

The following example is from the B2B Accelerator implementation of the order grid.



Multi-dimensional products are grouped into one line in the mini-cart and the cart page. For example, four size 5 shoes and three size 6 shoes, all variants of the same base product, would appear as seven shoes in one line.

Standard Behavior

In the standard process, a user adds one item to the cart at a time. However, multiple SKUs can be added to the cart on the Product Detail Order Grid page, and this is done the same way single SKUs are added in the Order Form page.

Facade Layer

The implementation `DefaultCartFacade` of the interface `CartFacade` defined in the commercefacades is used for both multi-dimensional and non-dimensional items. The Controller takes care of the action `Add to Cart` in the Product Details page and the Order Form page.

This default behavior can be overridden by creating a new implementation of the `CartFacade` and assigning the alias `cartFacade` in your extension `spring.xml` definition, as follows:

```
<alias name="myCartFacade" alias="cartFacade"/>
                                <bean id="myCartFacade" class="de.hybris.platform.commercefacades.order.impl.MyCartFacade">
```

MVC Layer

Controllers

Changes were made to the controller, `AddToCartController`, to allow querying a non-dimensional, single SKU or multi-dimensional, multiple SKUs when called from the web page. This controller reuses the logic for a single SKU and applies it to multiple SKUs.

Related Information

commercefacades Extension
Using Order Forms

Adding Multi-Dimensional Products to a Cart Using an Order Form

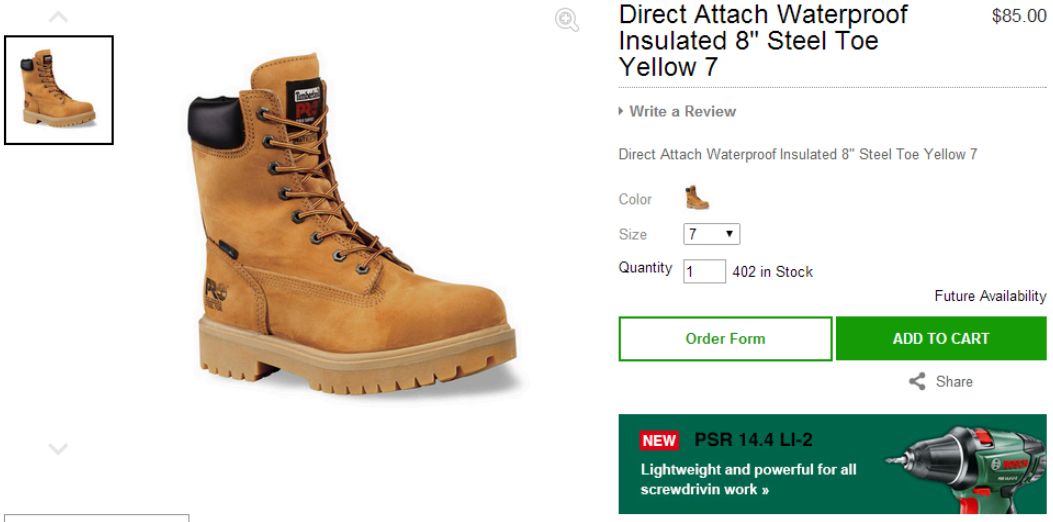
This document describes how to add multi-dimensional products to a cart using an order form.

When adding a multi-dimensional product to the cart, you can create orders that specify different quantities of each product variant. For example, you can display an order form for a particular boot and then order different quantities of size 7, 7.5, 8, and so on. The order form can be accessed from a product page or category listing.

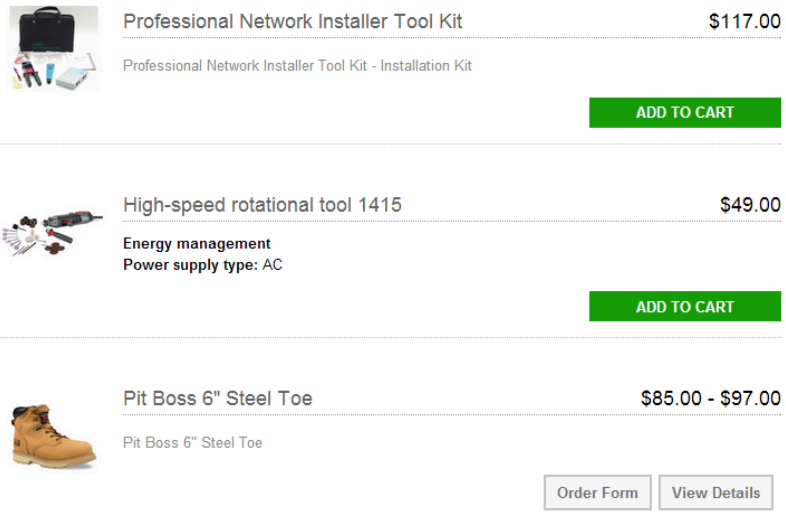
Accessing the Order Form feature

i Note
To view the order form, the user must be logged in to the site.

The following example is of a product page for a multi-dimensional product. The **Order Form** button is visible next to the **Add to Cart** button.

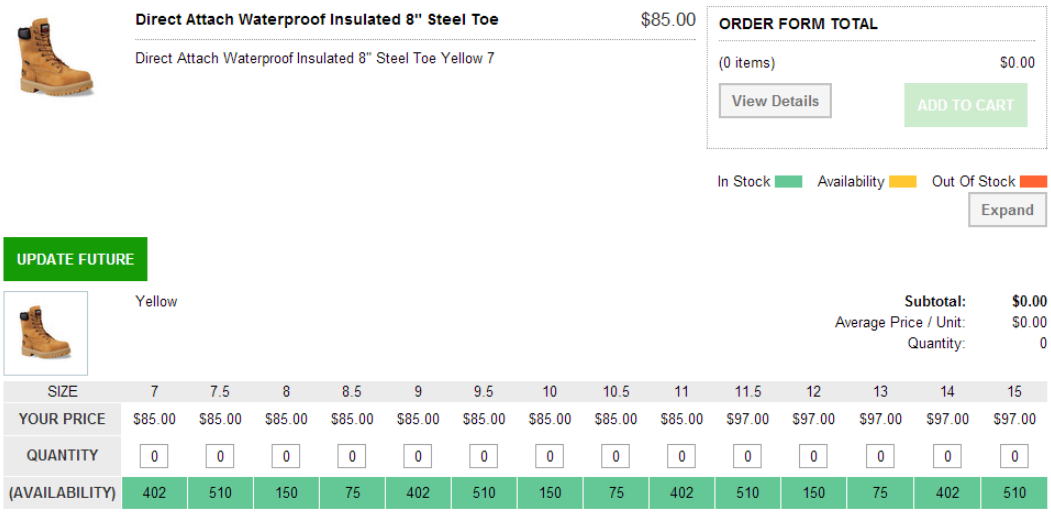


The following is an example of search results containing a multi-dimensional product. The Order Form button is only visible for the last product because it is the only multi-dimensional product in the list.



Adding Variants of a Product using the Order Form

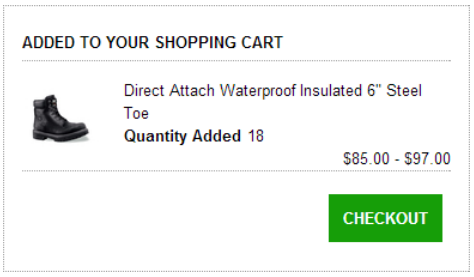
Clicking Order Form displays the Order Form page for that product, which is used to specify quantities for all the product's variants.



SIZE	7	7.5	8	8.5	9	9.5	10	10.5	11	11.5	12	13	14	15
YOUR PRICE	$85.00	$85.00	$85.00	$85.00	$85.00	$85.00	$85.00	$85.00	$85.00	$97.00	$97.00	$97.00	$97.00	$97.00
QUANTITY	0	0	0	0	0	0	0	0	0	0	0	0	0	0
(AVAILABILITY)	402	510	150	75	402	510	150	75	402	510	150	75	402	510

After specifying quantities for each variant, the customer clicks Add to Cart to add the items to the cart. The total quantity of all variants for the product is displayed in a popup window at the cart area at top-right of the page.

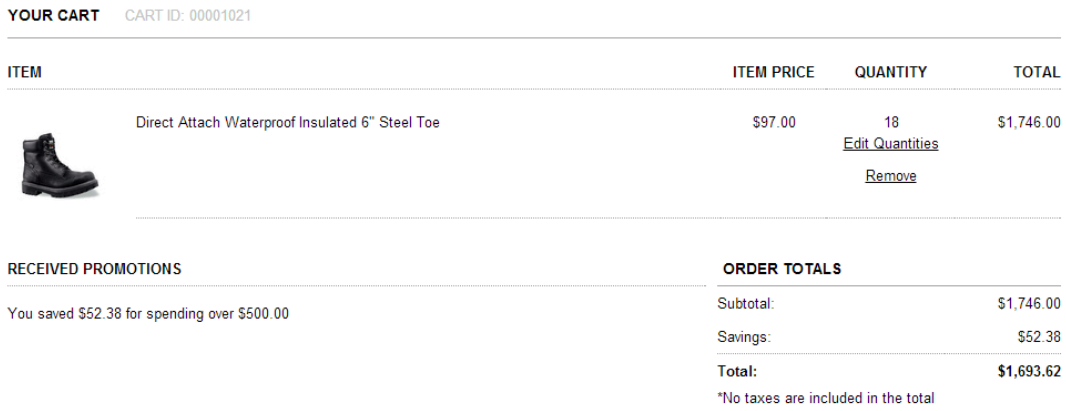
In the following example, 18 variants of the steel toe boot were added to the cart. The price range for all the boots is also displayed.



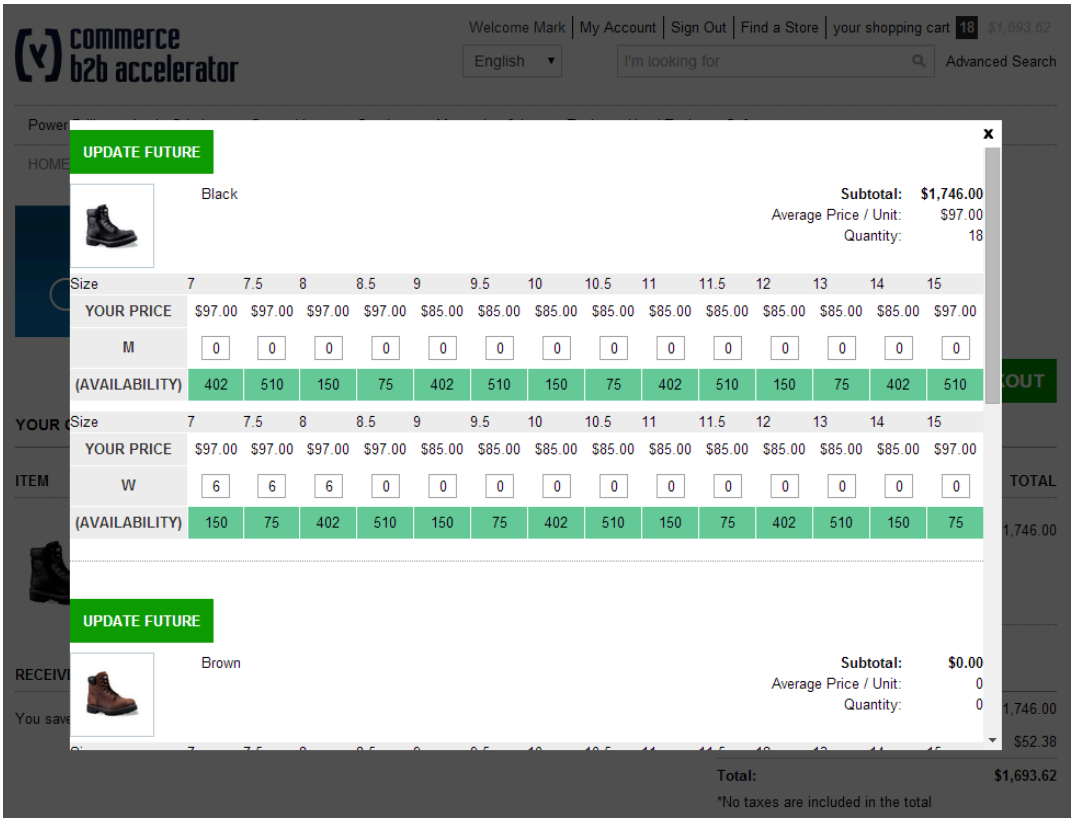
i Note

Variants of a multi-dimensional product appear as a single group in the mini-cart and in the cart.

The following is an example of the cart with a multi-dimensional product.

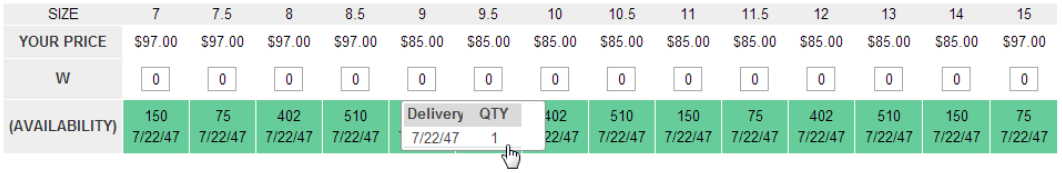


To change the quantity of a multi-dimensional product in the cart, the user clicks Edit Quantities, which displays the order form grid. The order form displays all variants, even if the variants did not have any quantities ordered, in case the user would like to add others.



Displaying Future Stock Availability

Clicking Update Future from the Order Form page fetches the quantities that will be available by a certain date. Pointing at a date displays the future stock for that date.



The color of the Availability row changes depending on availability:

- Green: In stock
- Yellow: Future stock only (no current stock)
- Red: Out of stock

Stock levels will only appear in yellow if current stock is red, future stock exists, and the user clicks Update Future.

Creating Additional Dimensions for Multi-Dimensional Products

Learn how to create additional dimensions for multi-dimensional products.

By default, SAP Commerce Accelerator supports up to three dimensions, but you can add any number of dimensions. The structure can be extended to support four, five, six or however many dimensions needed to describe the product. The steps to add more dimensions to the product definition are shown below.

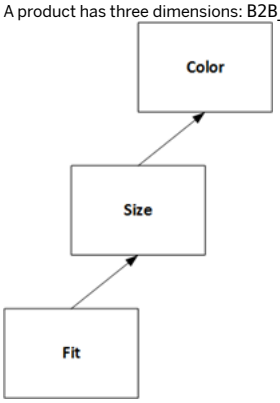
Technical Overview

Multi-dimensional products consist of one, two, three, or more `VariantCategory`s. Each category (dimension) is a characteristic of the product such as color, size, fit, weight, texture, or material. Each category is also a container for product attributes (`VariantValueCategory`), such as red, green, medium, large, coarse, fine, leather, or suede, which define the details of the category. Multi-dimensional products can be fully defined through dimensions and attributes.

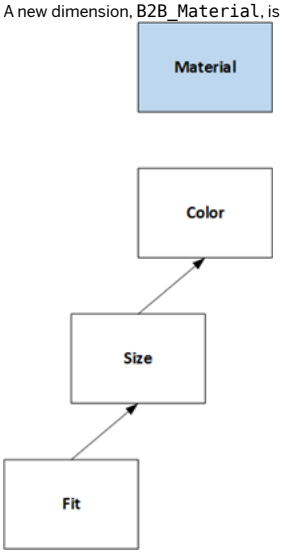
Adding Another Dimension

Commerce Accelerator supports three different dimensions (`category`s) out-of-the-box. These dimensions are chained together with each one serving as a supercategory for the next lower category in the chain. A `SuperCategory` is a root category for subordinate `category`s. By adding an additional category, the number of product dimensions can be extended. The example below begins with a product that has three dimensions, and then an additional dimension is added using either ImpEx or the Backoffice.

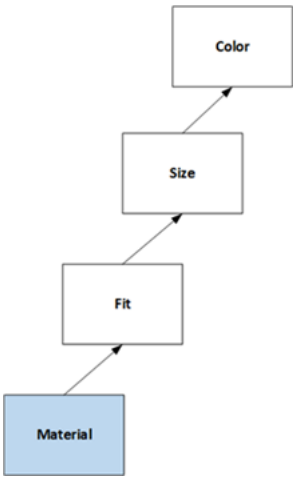
1. A product has three dimensions: `B2B_Color` < `B2B_Size` < `B2B_Fit`.



2. A new dimension, `B2B_Material`, is created.



3. `B2B_Material` is then linked to `B2B_Fit` and `B2B_Fit` is set as `B2B_Material`'s `SuperCategory`.



4. Through the Backoffice or ImpEx, create the required `VariantValueCategory`s. You would create as many of these as is needed for the product descriptors, such as red, brown, gray, and tan.
5. The Accelerator storefront supports up to three attributes by default, so the Product Detail page, the Order Form page, and the Shopping Cart have to be refactored to support more attributes. Specific files to be edited are:
    - `/yacceleratorstorefront/web/webroot/WEB-INF/tags/desktop/product/productVariantSelector.tag`
    - `/yacceleratorstorefront/web/webroot/WEB-INF/tags/desktop/grid` folder, which contains these files:
        - `coreTable.tag`
        - `coreTableHeader.tag`
        - `grid1dimension.tag`
        - `grid2dimension.tag`
        - `grid3dimension.tag`
        - Add a `grid4dimensions.tag` file when adding your fourth dimension.

Related Information

Creating Variant Categories
Creating Variant Value Categories
Creating Multi-Dimensional Products

Multi-Dimensional Product Grouping

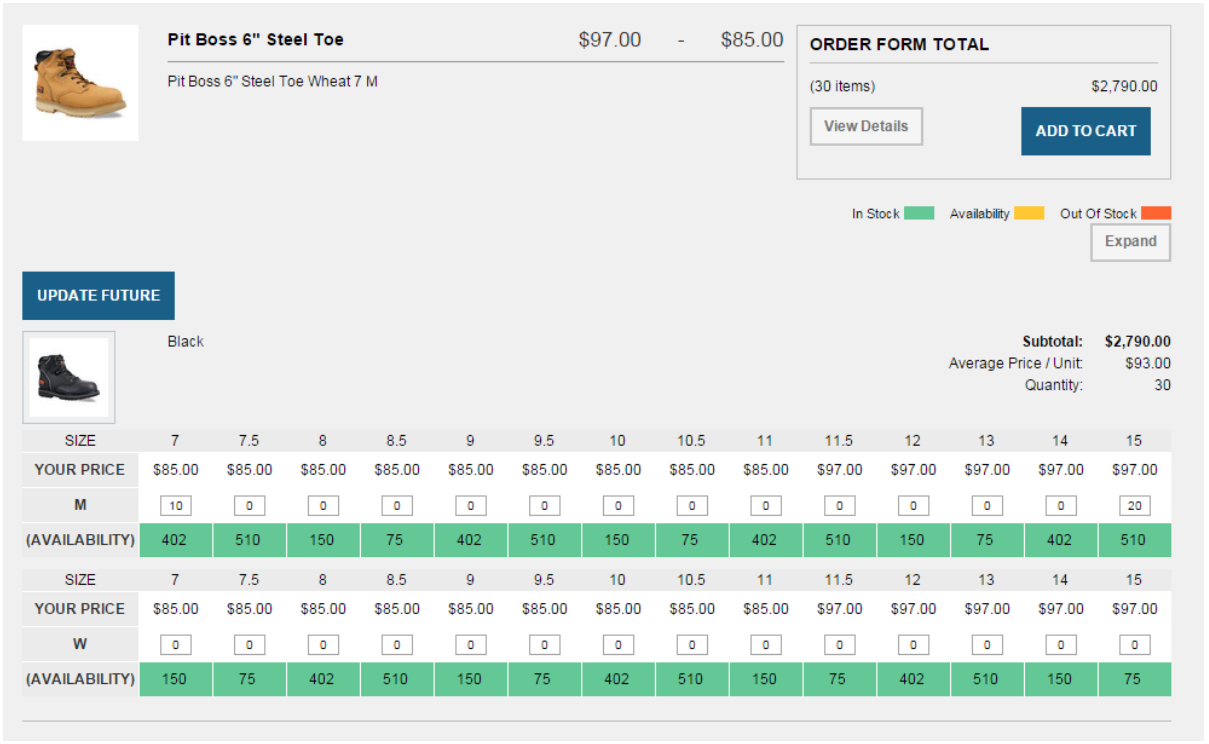
This document provides a technical overview of how multi-dimensional products are grouped in the cart and mini-cart.

The Multi-Dimensional Product feature of SAP Commerce Accelerator allows buyers to order variants of the same base product using an order form. For example, a buyer can order sizes 7, 8, and 9 of a particular boot at the same time. To keep the cart and mini-cart organized, variants of a multi-dimensional product are grouped under the base product in one line. Users can edit the quantities by displaying the order form from the cart.

Feature Overview

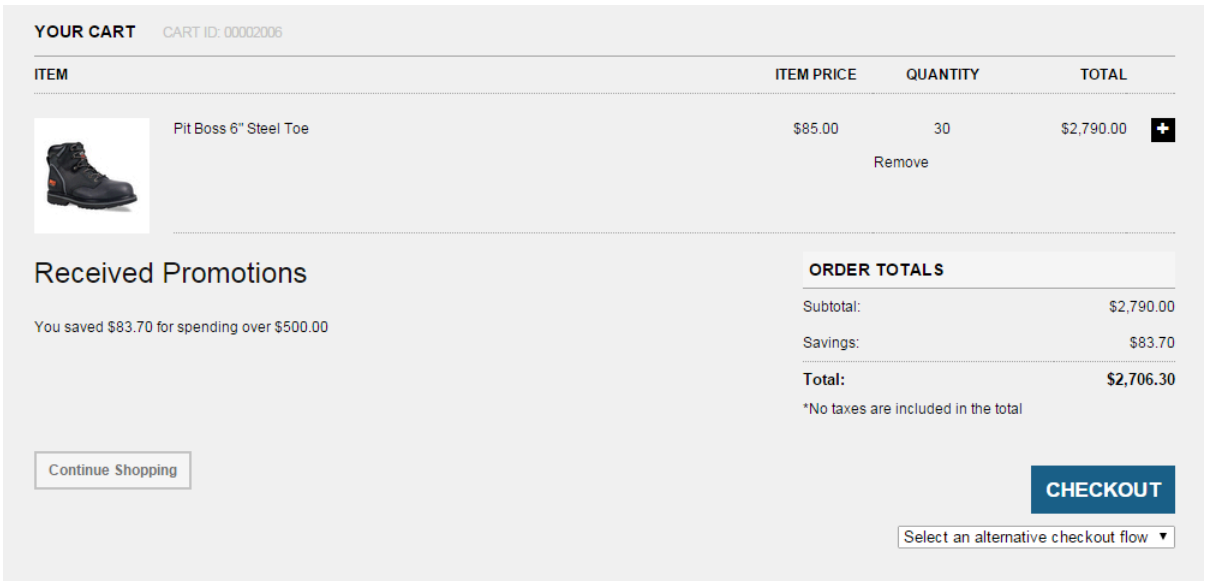
Cart

Multi-dimensional products are added to the cart using the order form available in Commerce B2B Accelerator, as shown in the screenshot below.

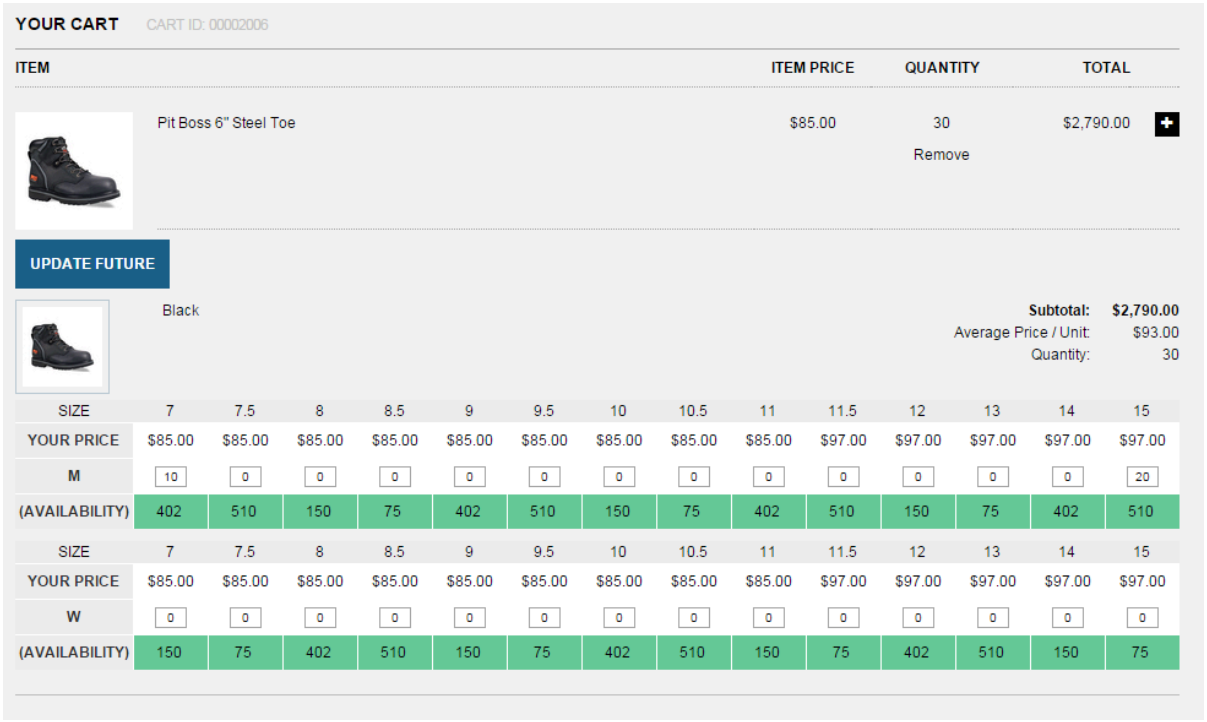


All variants are grouped into one line in the cart. The unit price shows the price range if the variants in the cart have different prices.

The following example shows the cart; users can modify the quantities of each variant by clicking ✚.



Clicking ✚ displays the variant grid below the item, as shown in the following example.



Modifying the Grid Quantities

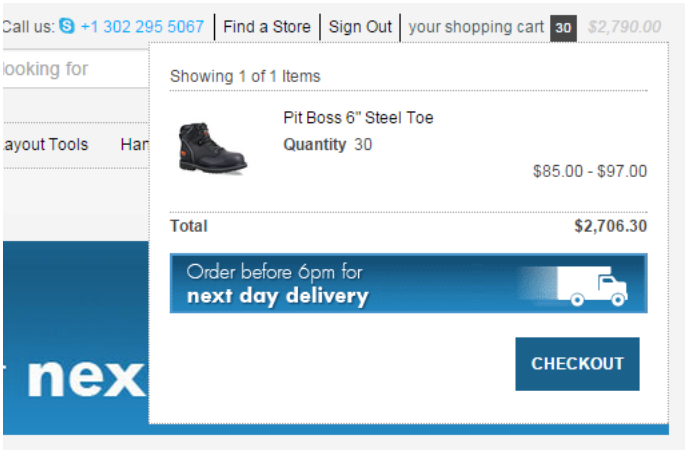
The grid displays the quantities ordered for each variant. Users can update the quantities of each variant in the grid, and all cart data is dynamically updated and refreshed when the quantity in the input field is changed. For example, if the quantity of the size 7 boot is changed from 10 to 0, the quantity, subtotal, average price per unit, and order totals are all automatically updated in the cart.

Checkout Page

The Checkout page is similar to the Cart page, except that the grid appears in a pop-up window and the quantities cannot be edited.

Mini-Cart

Similarly, variants are grouped in the mini cart.



Technical Overview

By default, the `convert` function in the `ycommercewebservices` extension converts `cartModel` to `cartData` as follows:

```
public CartData getSessionCart()
                                {
                                final CartData cartData;
                                final CartModel cart = getCartService().getSessionCart();
                                cartData = getCartConverter().convert(cart);
                                return cartData;
                                }
```

A list that is used to store the sub-entries of each entry is added to `OrderEntryData` in the `ycommercewebservices-beans.xml` file.

```
<bean class="de.hybris.platform.ycommercewebservices.order.data.OrderEntryDataList">
                                <property name="orderEntries" type="java.util.List&lt;de.hybris.platform.commercefacades.order.data.OrderEntryData>",
                                </bean>
```

Variants are put into the sub-entries when they are grouped into one item.

Grouping multi-dimensional products occurs when `cartModel` is converted to `cartData`. The default converters (`cartConverter` and `orderConverter`) are replaced by `groupedCartConverter` and `groupedOrderConverter`, and a `groupOrderEntryPopulator` is added to their populator list. The bean definition to achieve product grouping is located in

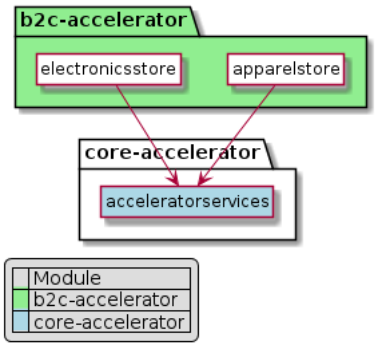
the `commercefacades-spring.xml` file:

```
...

                                <bean id="groupOrderEntryPopulator"
                                class="de.hybris.platform.b2bacceleratorfacades.order.converters.populator.GroupOrderEntryPopulator">
                                <property name="priceDataFactory" ref="priceDataFactory"/>
                                <property name="productService" ref="productService" />
                                </bean>

                                <alias name="groupedOrderConverter" alias="orderConverter"/>
                                <bean id="groupedOrderConverter" parent="defaultOrderConverter">
                                <property name="populators">
                                <list merge="true">
                                <ref bean="orderPopulator"/>
                                <ref bean="groupOrderEntryPopulator"/>
                                <ref bean="orderConsignmentPopulator"/>
                                </list>
                                </property>
                                </bean>

                                <alias name="groupedCartConverter" alias="cartConverter"/>
                                <bean id="groupedCartConverter" parent="defaultCartConverter">
                                <property name="populators">
                                <list merge="true">
                                <ref bean="cartPopulator"/>
                                <ref bean="groupOrderEntryPopulator"/>
                                </list>
                                </property>
                                </bean>

                                ...
```

B2C Accelerator Architecture

The B2C Accelerator module is a set of extensions providing all the sample data necessary to set up fully functioning storefronts. The included data sets are for an apparel storefront and an electronics storefront.

Dependencies



Recipes

For a complete list of SAP Commerce recipes that may include this module, see Installer Recipes.

For a complete list of the SAP Commerce Cloud, integration extension pack recipes that may include this module, see Installer Recipe Reference.

Extensions

The B2C Accelerator module consists of the following extensions:

apparelstore Extension

The `apparelstore` extension adds the necessary dataset for the Accelerator reference sites Apparel DE and Apparel UK. This dataset was previously present in the `acceleratorsampledata` extension. This extension was deprecated in favor of smaller extensions representing only one store.

electronicsstore Extension

The `electronicsstore` extension adds the necessary dataset for the Accelerator reference Japanese Electronics site. This dataset was previously present in the `acceleratosampledata` extension. This extension was deprecated in favor of smaller extensions representing only one store.

apparelstore Extension

The `apparelstore` extension adds the necessary dataset for the Accelerator reference sites Apparel DE and Apparel UK. This dataset was previously present in the `acceleratorsampledata` extension. This extension was deprecated in favor of smaller extensions representing only one store.

The Apparel sites demonstrate the use of Variants and how to set up multiple sites that support different currencies and languages on a single storefront. They offer shipping to multiple countries and have separate tax rules set at the store point of sale country.

→ **Tip**

This Extension contains sample data for the B2C Apparel storefronts. You can find the description of:

- B2C Electronics sample data in the electronicsstore Extension document
- B2B sample data in the powertoolsstore Extension document

i Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Extension Definition

Name	apparelstore
Description	It loads sample and core data for reference apparel storefronts.
Requires	The `apparelstore` Extension depends on the acceleratorservices Extension.
Author	SAP

Supported Markets and Channels

The **apparelstore** Extension is specifically designed for B2C market. It can be used for both, desktop and mobile, channels.

Supported	B2C Commerce	B2B Commerce	Telco Commerce
Market	✅	⛔	⛔
Channel	Desktop: ✅ Mobile: ✅	Desktop: ⛔ Mobile: ⛔	Desktop: ⛔ Mobile: ⛔

Hooks for System Initialization

The **apparelstore** extension implements the **AbstractSystemSetup** class in the **ApparelStoreSystemSetup**. When initialization is triggered, the **createProjectData** method will be called. This method will then call the **CoreDataImportService** and **SampleDataImportService** (in the **yacceleratorinitialdata** extension) to trigger the import of the different ImpEx files.

```
/**
* This method will be called during the system initialization.
*
* @param context the context provides the selected parameters and values
*/
@SystemSetup(type = SystemSetup.Type.PROJECT, process = SystemSetup.Process.ALL)
public void createProjectData(final SystemSetupContext context)
{
final ImportData apparelImportData = new ImportData();
apparelImportData.setProductCatalogName(APPAREL);
apparelImportData.setContentCatalogNames(Arrays.asList(APPAREL_UK, APPAREL_DE));
apparelImportData.setStoreNames(Arrays.asList(APPAREL_UK, APPAREL_DE));

getCoreDataImportService().importData(context, apparelImportData);
getEventService().publishEvent(new CoreDataImportedEvent(context, Arrays.asList(apparelImportData)));

getSampleDataImportService().importData(context, apparelImportData);
getEventService().publishEvent(new SampleDataImportedEvent(context, Arrays.asList(apparelImportData)));
}
```

UK Apparel Store

The following is a list of data loaded by the **apparelstore** extension for the sample UK Apparel store:

- Core data:
  - Essential data like languages, currencies, titles
  - Empty catalogs
  - CMS components
  - Email templates
  - Tax rows
  - Cart removal jobs
  - Store
  - Delivery costs
- Sample data:
  - Shared product catalog (with DE Apparel) with content in two languages:
    - Snowboard and surf
  - Product data model:
    - Variants
  - Exclusive content catalog with content in one language
  - Languages:
    - English GB
  - Currencies:
    - GBP

- Gross prices (using decoupled price rows, as described in Decoupling PDTRows from Product)
- Shipping:
  - UK territory only
- Site theme:
  - Black
- Enhanced SOLR configuration (shared with DE Apparel)
- Some promotions

DE Apparel Store

The following data is loaded by the **apparelstore** extension for the sample DE Apparel Store store:

- Core data:
  - Essential data like languages, currencies, titles
  - Empty catalogs
  - CMS components
  - Email templates
  - Tax rows
  - Cart removal jobs
  - Store
  - Delivery costs
- Sample data:
  - Shared product catalog (with UK Apparel) with content in two languages:
    - Snowboard and Surf
  - Product data model:
    - Variants
  - Exclusive content catalog with content in one language
  - Language:
  - German (default)
  - Currencies:
    - EUR
  - Gross prices (using decoupled price rows, as described in Decoupling PDTRows from Product)
  - Shipping:
    - Continental Europe
  - Site theme:
    - Apparel
  - Enhanced Solr configuration (shared with UK Apparel)
  - Some promotions

Modifications Checklist

The modifications that this Extension makes to the Accelerator are listed below:

Impex Configuration Scripts	⛔
Core Data Listeners	⛔
Model Layer	⛔
Model Interceptors	⛔
Cockpit Configuration	⛔
Cockpit Beans	⛔
Validation Rules	⛔

Service Layer	⛔
Facade DTO	⛔
Facade Layer	⛔
CMS Components	⛔
Page Templates	⛔
JavaScript	⛔
CSS	⛔

Page Controllers	⛔

Tags	⊟
TLD	⊟
Filters	⊟
MVC Interceptors	⊟
Spring Security	⊟
Message Resources	⊟

Responsive Sample Data Loading Mechanism

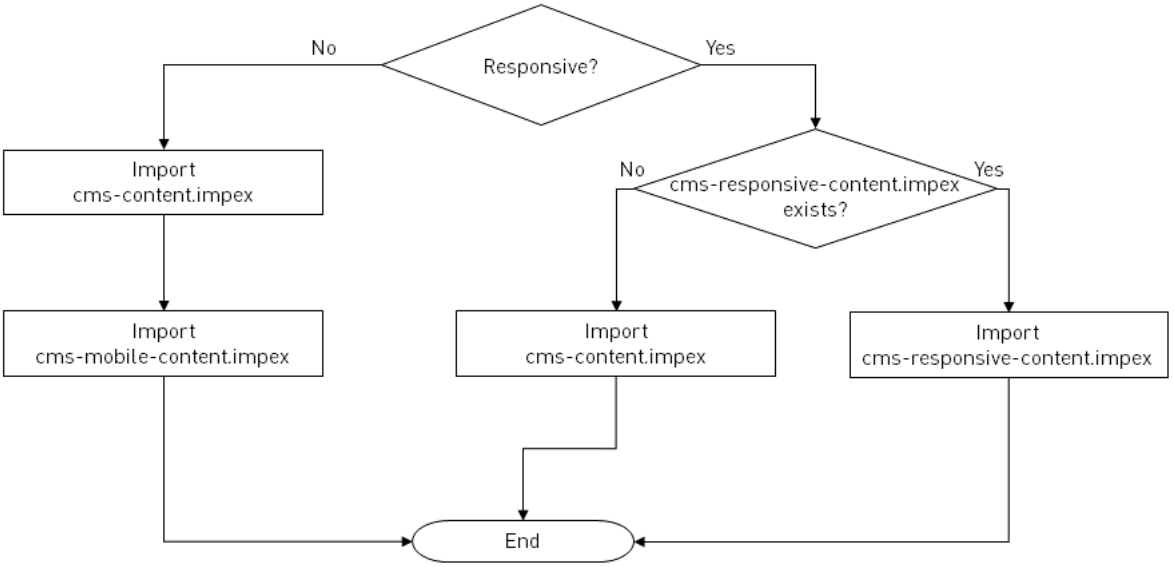
The apparel responsive sample data is self-contained resulting in faster initialization and cleaner sample data. This approach forms the basis for future improvements in features and tools such as SAP Commerce SmartEdit.

ⓘ Note

The changes described in this topic apply to the apparel store sample data only (both UK and German). The electronics storefront uses the same loading mechanism as before.

To streamline loading and initiation, the responsive sample data is self-contained and the import functions of the sample store's `contentCatalog` are modified. Desktop, responsive, and mobile all use the same `productCatalog` data, common data, store data, and Solr index data.

The loading function works as follows:



Localization

All the required `cms-responsive-content.vt`, `cms-responsive-content_en.properties` (for the apparel-uk storefront), and `cms-responsive-content_de.properties` (for the apparel-de storefront) are added in `resource-lang`, so all the localized impex files are auto-generated.

electronicsstore Extension

The `electronicsstore` extension adds the necessary dataset for the Accelerator reference Japanese Electronics site. This dataset was previously present in the `acceleratosampledata` extension. This extension was deprecated in favor of smaller extensions representing only one store.

The electronics storefront offers shipping to multiple countries, it has a tax rule set at the store point of sale country.

→ Tip

This Extension loads in sample data for the B2C Japanese Electronic storefront. Some of this sample data resides in dependent extensions in other modules.

- For a description of B2C Apparel sample data, see apparelstore Extension.
- For a description of B2B sample data, see powertoolsstore Extension.

ⓘ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Extension Definition

Name	electronicsstore
Description	It loads sample and core data for reference electronic storefront.

Requires	The **electronicsstore** Extension depends on the acceleratorservices Extension.
Author	SAP

Supported Markets and Channels

The **electronicsstore** Extension is specifically designed for B2C market. It can be used for both, desktop and mobile, channels.

Supported	B2C Commerce	B2B Commerce	Telco Commerce
Market	✅	⛔	⛔
Channel	Desktop: ✅ Mobile: ✅	Desktop: ⛔ Mobile: ⛔	Desktop: ⛔ Mobile: ⛔

Hooks for System Initialization

The **electronicsstore** extension implements the **AbstractSystemSetup** class in the **ElectronicsStoreSystemSetup**. When initialization is triggered **createProjectData** method will be called. This method will then call the **CoreDataImportService** and **SampleDataImportService** (in the **yacceleratorinitialdata** extension) to trigger the import of the ImpEx files.

```
/**
* This method will be called during the system initialization.
*
* @param context the context provides the selected parameters and values
*/
@SystemSetup(type = SystemSetup.Type.PROJECT, process = SystemSetup.Process.ALL)
public void createProjectData(final SystemSetupContext context)
{
final ImportData electronicsImportData = new ImportData();
electronicsImportData.setProductCatalogName(ELECTRONICS);
electronicsImportData.setContentCatalogNames(Arrays.asList(ELECTRONICS));
electronicsImportData.setStoreNames(Arrays.asList(ELECTRONICS));

getCoreDataImportService().importData(context, electronicsImportData);
getEventService().publishEvent(new CoreDataImportedEvent(context, Arrays.asList(electronicsImportData)));

getSampleDataImportService().importData(context, electronicsImportData);
getEventService().publishEvent(new SampleDataImportedEvent(context, Arrays.asList(electronicsImportData)));
}
```

Data for Japanese Electronics Store

The following is a list of data loaded by the **electronicsstore** extension for the sample Japanese Electronics store:

- Core data:
  - Essential data like languages, currencies, titles
  - Empty catalogs
  - CMS components
  - Email templates
  - Tax rows
  - Cart removal jobs
  - Store
  - Delivery costs
- Sample data:
  - Exclusive Product Catalog with content in three languages:
    - Electronics
  - Product Data Model :
    - Vanilla with classification system
  - Exclusive WCMS content catalog with content in three languages
  - Languages:
    - Japanese
    - English (default)
    - German
  - Currencies:
    - JPY
    - USD (default)
  - Gross prices (using decoupled price rows, as described in Decoupling PDTRows from Product)
  - Shipping:
    - Japan
    - USA
    - Europe
  - Enhanced Solr configuration
  - Advanced Personalization samples
  - Promotions

- Customer review samples
- Cross-sells

Modifications Checklist

The modifications that this Extension makes to the Accelerator are listed below:

Impex Configuration Scripts	⛔
Core Data Listeners	⛔
Model Layer	⛔
Model Interceptors	⛔
Cockpit Configuration	⛔
Cockpit Beans	⛔
Validation Rules	⛔

Service Layer	⛔
Facade DTO	⛔
Facade Layer	⛔
CMS Components	⛔
Page Templates	⛔
JavaScript	⛔
CSS	⛔

Page Controllers	⛔
Tags	⛔
TLD	⛔
Filters	⛔
MVC Interceptors	⛔
Spring Security	⛔
Message Resources	⛔

B2C Accelerator Implementation

The B2C Accelerator module allows you to remove apparel store functionality from your installation of SAP Commerce Accelerator.

Customizing the B2C Accelerator

This procedure describes how to create a customized B2C Accelerator using the B2C Accelerator installation recipe.

B2C Store Extensions Related Data

The other extensions providing data for the project are the `apparelstore` and `electronicsstore` AddOn extensions and `yacceleratorinitaldata` extension. They come with a set of only Project data.

Removing Apparel and the Apparel Stores

You can remove apparel support from your SAP Commerce Accelerator.

Changing the Default Remember Me Timeout

Learn how to update the default remained logged in time.

Customizing the B2C Accelerator

This procedure describes how to create a customized B2C Accelerator using the B2C Accelerator installation recipe.

Prerequisites

Before running the `modulegen` tool, download the latest version of the SAP Commerce solution. For more details, see Download.

Once you have downloaded SAP Commerce, extract the SAP Commerce ZIP file in *<BASE_DIR>*.

→ **Tip**

If you are in a training class, the SAP Commerce ZIP file will be provided by your trainer.

Procedure

1. Open a command line and go to the installer directory of your SAP Commerce instance.
   - Windows: `cd <HYBRIS_HOME_DIR>\installer`
   - Unix/MacOS: `cd <HYBRIS_HOME_DIR>/installer`
2. Build SAP Commerce using the `cx` recipe.
   - Windows: `install.bat -r cx -A initAdminPassword=<your_password>`
   - Unix/MacOS: `./install.sh -r cx -A initAdminPassword=<your_password>`
3. Navigate to the `<HYBRIS_BIN_DIR>/platform` directory and set your `ant` environment by entering the following command.
   - Windows: `setantenv.bat`
   - Unix/MacOS: `. ./setantenv.sh`
4. Remove the `yb2bacceleratorstorefront` extension from the `<HYBRIS_CONFIG_DIR>/localextensions.xml` file.
5. Delete the `yb2bacceleratorstorefront` extension directory under `<HYBRIS_BIN_DIR>/custom`.

   Windows:

   ```
   rm <HYBRIS_HOME_DIR>\hybris\bin\custom\yb2bacceleratorstorefront -rf
   ```

   Unix/MacOS:

   ```
   rm -r <HYBRIS_HOME_DIR>/hybris/bin/custom/yb2bacceleratorstorefront
   ```

6. Run the `ant modulegen` command and modify the `input.name` and `input.package` parameters as required.

   ```
   ant modulegen -Dinput.module=accelerator -Dinput.name=training -Dinput.package=de.hybris.training -Dinput.template=develop
   ```

   Where:

   - `input.module` parameter: Configures the **modulegen** task to use the `accelerator` module.
   - `input.name` value: Indicates the prefix that is added to the new extensions generated by **modulegen**.
   - `input.package` value: Defines the default Java package prefix.
   - `input.template` parameter: Defines how to use the default SAP Commerce configuration.

   Note

   The `training` variable can be substituted for any string value.

7. Remove all the `yaccelerator*` extensions from the `<HYBRIS_CONFIG_DIR>/localextensions.xml` file.
8. Add the new extensions to the `<HYBRIS_CONFIG_DIR>/localextensions.xml` file.

   ```
   <extension dir="${HYBRIS_BIN_DIR}/custom/training/trainingbackoffice"/>
           <extension dir="${HYBRIS_BIN_DIR}/custom/training/trainingcore"/>
           <extension dir="${HYBRIS_BIN_DIR}/custom/training/traininginitialdata"/>
           <extension dir="${HYBRIS_BIN_DIR}/custom/training/trainingstorefront"/>
           <extension dir="${HYBRIS_BIN_DIR}/custom/training/trainingtest"/>
   ```

   Note

   If you include the `trainingtest` extension, it imports test data, which can be inaccurate in a real-world scenario. Only include the `trainingtest` extension if you intend to perform tests with this Accelerator.

9. Re-install the B2C Accelerator AddOns on the training storefront by running the following `ant` command from the `<HYBRIS_BIN_DIR>/platform` folder.

   ```
   ant reinstall_addons -Dtarget.storefront=trainingstorefront
   ```

10. Add the following properties to `<HYBRIS_CONFIG_DIR>/local.properties` file.

    ```
    website.electronics.http=http://electronics.local:9001/trainingstorefront
    website.electronics.https=https://electronics.local:9002/trainingstorefront
    website.apparel-uk.http=http://apparel-uk.local:9001/trainingstorefront
    website.apparel-uk.https=https://apparel-uk.local:9002/trainingstorefront
    website.apparel-de.http=http://apparel-de.local:9001/trainingstorefront
    website.apparel-de.https=https://apparel-de.local:9002/trainingstorefront
    piwik.tracker.url.electronics=http\://electronics.local\:9002/trainingstorefront/events
    piwik.tracker.https.url.electronics=https\://electronics.local\:9002/trainingstorefront/events
    piwik.tracker.url.apparel-uk=http\://apparel-uk.local\:9002/trainingstorefront/events
    piwik.tracker.https.url.apparel-uk=https\://apparel-uk.local\:9002/trainingstorefront/events
    piwik.tracker.url.apparel-de=http\://apparel-de.local\:9002/trainingstorefront/events
    piwik.tracker.https.url.apparel-de=https\://apparel-de.local\:9002/trainingstorefront/events
    ```

11. Run `ant clean all` and `ant initialize` from the `<HYBRIS_BIN_DIR>/platform` directory.

    Windows:

    ```
    cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
    ant initialize
    ```

    Unix/MacOS

    ```
    cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
    ant initialize
    ```

12. Start the SAP Commerce server by running the following command from the `<HYBRIS_BIN_DIR>/platform` folder.
    - Windows: `hybrisserver.bat`
    - Unix/MacOS: `./hybrisserver.sh`
13. If necessary, allow blocked libraries to run in MacOS Catalina. For further information, see Allowing Apps on MacOS.
14. Add the following to your hosts file.

```
127.0.0.1   localhost electronics.local apparel-uk.local apparel-de.local
```

You can access the B2C Accelerator electronics and apparel storefronts with the following URLs:

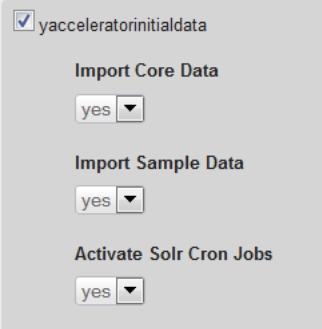
- `https://electronics.local:9002/trainingstorefront/`
- `https://apparel-uk.local:9002/trainingstorefront/`
- `https://apparel-de.local:9002/trainingstorefront/`

B2C Store Extensions Related Data

The other extensions providing data for the project are the `apparelstore` and `electronicsstore` AddOn extensions and `yacceleratorinitaldata` extension. They come with a set of only Project data.

B2C Store Extensions Project Data

Project data setup for the `yacceleratorinitialdata` extension provides the following options:



Import Core Data loads:

- Essential data: warehouses, delivery modes, and currencies
- CMS templates
- Empty catalogs
- Email templates

Import Sample Data loads:

- Products
- Cockpits users
- CMS content
- Promotions
- Points of service

Activate Solr Cron Jobs runs the Solr cron jobs.

Project data setup for the sample data extensions contain:

- Site and store configuration: catalogs, classifications, countries, languages, currencies, delivery options, themes, media formats, and tax groups.
- Base CMS content: page templates, page types, pages, components, and cockpit configurations.
- Solr search configuration: index configurations, indexed types and properties, ranges, queries, and sorts.
- Products
- Product categories
- Promotions
- Product and category media
- Product stock levels
- CMS content, including sample components, pages, images, and paragraphs.
- Email content
- Points of Service (POS): stores and associated media.
- Cockpit users
- BTG rules
- Sample product reviews

Resource Folder Structure

The resource folders of the `apparelstore` and `electronicsstore` AddOn extensions are divided into two parts :

1. Core data: create empty sites, empty catalogs, and essential data
2. Sample data: populate catalogs with products, import users, and user rights

Products

The sample data provided with the SAP Commerce Accelerator demonstrates two approaches to defining a product catalog in the system:

- Using Classifications in **electronics** shop
- Using Variants in **apparel** shop

CMS Content

The sample CMS content for both sites is almost identical but using different media and themes in the two sites makes them look totally different.

Points of Sale

Points of Sale are loaded into the Electronics shop, Apparel UK and Apparel DE separately, so searching in one Apparel site, for example, does not return results from the other.

Promotions

Sample promotions are included that apply for specific products, specific categories and across entire orders.

Related Information

yacceleratorcore Extension
acceleratorservices Extension
apparelstore Extension
electronicsstore Extension

Removing Apparel and the Apparel Stores

You can remove apparel support from your SAP Commerce Accelerator.

The SAP Commerce Accelerator offers a reference storefront with an Apparel functionality that supports product variants out of the box. If your project does not require apparel support, you can remove this functionality. This topic explains how to remove Apparel from the Commerce Accelerator source code, which includes the Apparel data model, any Apparel services and facades, as well as the Apparel sample storefronts configuration.

Before starting the workflow described here, see Customizing the Accelerator with extgen and modulegen to get an overview of working with template extensions.

The procedure for removing the Apparel features depends on the extensions they are related to. The following sections provide the procedure for each extension.

yacceleratorcore Extension

Removing Apparel features related to the `yacceleratorcore` extension is executed in several steps and in a specific order. Remove Apparel features in the following order:

1. Remove the data model.
2. Hot folder batch import
3. Remove beans related to Solr Search

Remove the Data Model

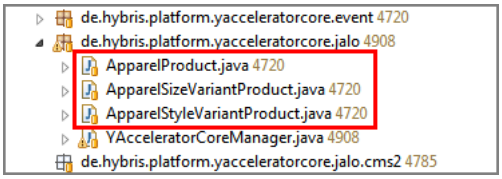
To remove the data model, you start by removing the data model changes and any associated configurations for the cockpits as follows.

1. In the `yacceleratorcore/resources/yacceleratorcore-items.xml` file, remove everything inside the **Apparel** typegroup :

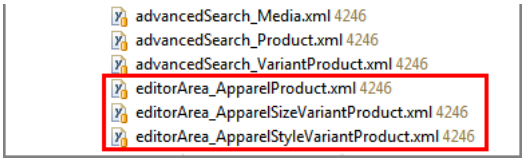
```
<typegroup name="Apparel">
                                        <itemtype code="ApparelProduct" extends="Product"
                                        autocreate="true" generate="true"
                                        jaloclass="de.hybris.platform.yacceleratorcore.jalo.ApparelProduct">
                                        <description>Base apparel product extension that contains additional attributes.</description>
                                        <attributes>
                                        <attribute qualifier="genders" type="GenderList">
                                        <description>List of genders that the ApparelProduct is designed for</description>
                                        <modifiers />
                                        <persistence type="property" />
                                        </attribute>
                                        </attributes>
                                        </itemtype>

                                        <itemtype code="ApparelStyleVariantProduct" extends="VariantProduct"
                                        autocreate="true" generate="true"
                                        jaloclass="de.hybris.platform.yacceleratorcore.jalo.ApparelStyleVariantProduct">
                                        <description>Apparel style variant type that contains additional attribute describing variant s
                                        <attributes>
                                        <attribute qualifier="style" type="localized:java.lang.String"
                                        metatype="VariantAttributeDescriptor">
                                        <description>Colour/Pattern of the product.</description>
                                        <modifiers />
                                        <persistence type="property" />
                                        </attribute>

                                        <attribute qualifier="swatchColour" type="java.lang.String">
                                        <description>A normalised colour mapping to a standardised front-end navigable name.</descripti
                                        <modifiers />
                                        <persistence type="property" />
                                        </attribute>
                                        </attributes>

                                        </itemtype>

                                        <itemtype code="ApparelSizeVariantProduct" extends="ApparelStyleVariantProduct"
                                        autocreate="true" generate="true"
                                        jaloclass="de.hybris.platform.yacceleratorcore.jalo.ApparelSizeVariantProduct">
                                        <description>Apparel size variant type that contains additional attribute describing variant si
```

```
<attributes>
<attribute qualifier="size" type="localized:java.lang.String"
metatype="VariantAttributeDescriptor">
<description>Size of the product.</description>
<modifiers />
<persistence type="property" />
</attribute>
</attributes>

</itemtype>
</typegroup>
```

2. Remove the **ApparelProduct**, **ApparelSizeVariantProduct**, and **ApparelStyleVariantProduct** classes from `yacceleratorcore/src/de/hybris/platform/yacceleratorcore/jalo`.



3. Remove the following Apparel-specific Product Cockpit editor configuration XML files from the `yacceleratorcockpits/resources/yacceleratorcockpits-config/cockpitgroup` folder:
   - `editorArea_ApparelProduct.xml`
   - `editorArea_ApparelSizeVariantProduct.xml`
   - `editorArea_ApparelStyleVariantProduct.xml`



4. Remove the **BatchIntegrationTest.testVariant** test method from the `yacceleratorcore` extension.
5. Execute the `ant clean all` command in the `yacceleratorcore` extension to ensure the extension still compiles.

Hot Folder Batch Import

You must remove the back import instance for the Apparel store. To do so, remove the `yacceleratorcore/resources/yacceleratorcore/integration/hot-folder-store-apparel-spring.xml` file.

Solr Search

When you delete the store content, the scripts that create the Solr index are removed. You must also remove specific value-provider Spring beans from the `yacceleratorcore` extension application context.

1. Remove the **CategorySource** beans from the `yacceleratorcore/resources/yacceleratorcore-spring.xml` file:

```
<bean id="apparelCategorySource" parent="abstractCategorySource">
<property name="rootCategory" value="categories"/> <!-- 'categories' is the root apparel catego
</bean>
<bean id="apparelBrandCategorySource" parent="abstractCategorySource">
<property name="rootCategory" value="brands"/> <!-- 'brands' is the root of the brands hierarch
</bean>
```

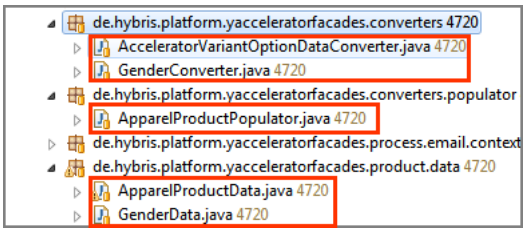
2. Remove the **ValueProvider** beans from the `yacceleratorcore/resources/yacceleratorcore-spring.xml`:

```
<bean id="apparelCategoryCodeValueProvider" parent="abstractCategoryCodeValueProvider">
<property name="categorySource" ref="apparelCategorySource"/>
</bean>
<bean id="apparelBrandCategoryCodeValueProvider" parent="abstractCategoryCodeValueProvider">
<property name="categorySource" ref="apparelBrandCategorySource"/>
</bean>
<bean id="apparelCategoryNameValueProvider" parent="abstractCategoryNameValueProvider">
<property name="categorySource" ref="apparelCategorySource"/>
</bean>
<bean id="apparelBrandCategoryNameValueProvider" parent="abstractCategoryNameValueProvider">
<property name="categorySource" ref="apparelBrandCategorySource"/>
</bean>
```

yacceleratorfacades Extension

Most of the code in the `yacceleratorfacades` extension template is provided to support Apparel functionality. This procedure assumes that you have not adapted any of the code you plan to delete for purposes other than Apparel.

1. Remove the **AcceleratorVariantOptionDataConverter** and **GenderConverter** classes from the `yacceleratorfacades/src/de/hybris/platform/yacceleratorfacades/converters` folder of the `yacceleratorfacades` extension.
2. Remove the **ApparelProductPopulator** class from the `yacceleratorfacades/src/de/hybris/platform/yacceleratorfacades/converters/populator` folder of the `yacceleratorfacades` extension.
3. Remove the **ApparelProductData** and **GenderData** classes from the `yacceleratorfacades/src/de/hybris/platform/yacceleratorfacades/product/data` folder of the `yacceleratorfacades` extension:



4. Remove the Spring bean configuration for all of these classes in the `yacceleratorfacades/resources/yacceleratorfacades-spring.xml` file.

```
<alias name="acceleratorVariantOptionDataConverter" alias="variantOptionDataConverter"/>
                                                <bean id="acceleratorVariantOptionDataConverter" class="de.hybris.platform.yacceleratorfacades.
                                                <property name="mediaService" ref="mediaService"/>
                                                <property name="mediaContainerService" ref="mediaContainerService"/>
                                                <property name="typeService" ref="typeService"/>
                                                <property name="imageFormatMapping" ref="imageFormatMapping"/>
                                                <property name="variantAttributeMapping">
                                                <map>
                                                <entry key="ApparelStyleVariantProduct.style" value="styleSwatch"/>
                                                </map>
                                                </property>
                                                </bean>

                                                <alias name="acceleratorGenderConverter" alias="genderConverter"/>
                                                <bean id="acceleratorGenderConverter" class="de.hybris.platform.yacceleratorfacades.converters.
                                                <property name="typeService" ref="typeService"/>
                                                </bean>

                                                <bean id="apparelProductPopulator" class="de.hybris.platform.yacceleratorfacades.converters.pop
                                                <property name="genderConverter" ref="genderConverter"/>
                                                </bean>

                                                <alias name="acceleratorProductConverter" alias="productConverter"/>
                                                <bean id="acceleratorProductConverter" class="de.hybris.platform.yacceleratorfacades.converters
                                                <property name="apparelProductPopulator" ref="apparelProductPopulator"/>
                                                </bean>
```

5. Execute the command `ant clean all` in the `yacceleratorfacades` extension to ensure the extension still compiles.

yacceleratorstorefront Extension

The storefront controllers are agnostic of apparel, therefore few changes are required here.

Spring Application Context

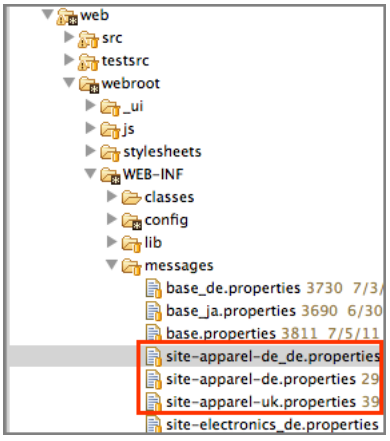
In the `yacceleratorstorefront/web/webroot/WEB-INF/config/spring-mvc-config.xml` file of the `yacceleratorstorefront` extension, remove the following line:

```
<entry key="size" value-ref="sizeAttributeComparator"/>
```

Site Resource Files

Remove the following apparel site resource files, which are located in the `yacceleratorfacades/web/webroot/WEB-INF/messages` folder:

- `site-apparel-de_de.properties`
- `site-apparel-de_en.properties`
- `site-apparel-uk_en.properties`



apparelstore Extension

Remove the `apparelstore` extension from the `localextensions.xml` file.

Recompiling

Execute the command `ant clean all` in the `apparelstore` extension to ensure the extension still compiles.

Reinitializing

Reinitialize the SAP Commerce Platform database to remove the apparel data model from the type system and any orphaned data no longer added by the scripts. For more information on initialization, see the **Initialize System** section of the Initialization and Update of the SAP Commerce document.

Related Information

yacceleratorcore Extension
acceleratorfacades Extension
yacceleratorstorefront Extension
apparelstore Extension
electronicsstore Extension

Changing the Default Remember Me Timeout

Learn how to update the default remained logged in time.

Context

The SAP Commerce Accelerator Remember Me functionality is enabled by default, and has a default validity of two weeks.

Procedure

1. Navigate to `spring-security-config.xml` under `{root path of customer storefront}\web\webroot\WEB-INF\config`.
2. Locate the bean `defaultRememberMeServices` in this XML.
3. Add property `tokenValiditySeconds`, and override the default two weeks with the desired time.
4. Restart the server to make the modification take effect.

Related Information

Remember Me Feature

B2C Accelerator AddOns Module

The B2C Accelerator AddOns module is a set of special extensions, called AddOns, that allow you to extend the functionality of your storefront without having to edit the core code base.

Features

Business Events in the Commerce Accelerator
Google Analytics
User Enumeration Prevention
Login with Verification Token

Architecture

captchaaddon AddOn
hybrisAnalytics AddOn
wishlist Extension

Implementation

Updating captchaaddon to Support reCAPTCHA v2.0
Using Wishlist2Service API

B2C Accelerator AddOns Features

The B2C Accelerator AddOns module provides a number of AddOn extensions that allow you to add functionality to your storefront, such as business event tracking, monitoring web traffic with Google Analytics, and optimizing your storefront for mobile devices.

Business Events in the Commerce Accelerator

The Commerce Accelerator sends business events for a variety of user actions in the storefront. The `hybrisAnalytics` AddOn, which contains the Piwik scripts that are used to track user events, must be installed in the storefront in order to track user actions and send business events to the event system that collects the analytics data.

Google Analytics

This document discusses what is added to the SAP Commerce Accelerator code to enable the monitoring. Using this information, you can also remove Google Analytics from your system, if necessary.

User Enumeration Prevention

The B2C Accelerator storefront provides an user enumeration prevention feature for customer registration.

Login with Verification Token

The B2C Accelerator storefront provides a login with verification token feature where a customer can log in with a one-time verification token besides email address and password.

Business Events in the Commerce Accelerator

The Commerce Accelerator sends business events for a variety of user actions in the storefront. The `hybrisAnalytics` AddOn, which contains the Piwik scripts that are used to track user events, must be installed in the storefront in order to track user actions and send business events to the event system that collects the analytics data.

⚠ Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

The `hybrisanalyticsaddon` AddOn does this by sending events through JavaScript to the `eventtrackingws` AddOn. The events consist of user actions in the storefront, such as viewing a page, or adding an item to the cart. The `hybrisanalyticsaddon` AddOn provides the tag files and necessary JavaScripts that can be installed in the storefront.

For more information, see eventtrackingwsaddon AddOn, Business Events in the Commerce Accelerator, and Event Tracking Module.

i Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Prerequisites

In order to capture and analyze business events in the Commerce Accelerator, the following requirements must be met:

- Accelerator must be installed and running a storefront.
- the `hybrisAnalytics` AddOn must be installed with the Accelerator storefront. For more information, see hybrisAnalytics AddOn.
- the `eventstrackingws` AddOn must be installed with the Accelerator storefront. For more information, see eventtrackingwsaddon AddOn.
- the Business Event extensions must be installed. For more information, see hybris Business Event Extensions - Technical Guide.

View Page Events

There are several View Page events captured by the Piwik script when the `hybrisAnalytics` AddOn is installed in the Accelerator storefront. View Page events are generally sent to the event system whenever a user views any page. Each View Page event includes different attributes, depending on the type of page that is viewed by the user. For example, when a user views a Product Page, there are Product Page-specific events that are captured so that you have access to this information in the analytics system.

Commerce Accelerator generally sets the `pagetype` model attribute in its controllers so that we can check the page type and send the appropriate View Page event.

Generic Page View Events

If any of the View Page events do not fall under a specific category, or do not require additional attributes to be captured, then we use the `trackPageView` method from the Piwik JS client.

Product Page View Events

Product Page View events are sent whenever a Product Page is viewed. Piwik provides the `setEcommerceView` method to set the product-related details. This method is used to capture all the product-related attributes from the model attributes set in the `ProductPageController` Java class. Once `setEcommerceView` is pushed, we call `trackPageView` with the appropriate `ViewPage` name (`ViewProduct`), according to the Piwik recommendation.

Category Page View Event

The Category Page View event is sent when a user navigates to a product category from the navigation bar or through the breadcrumb. In the Accelerator, the Category Page is treated like a Search Result page. For this reason, the Category Page View event is considered a Search event, because we get the product results for the category from the SOLR/indexing engine. In this case, we call the `trackSiteSearch` method that has the required parameters. Some custom data is also added using the `setCustomData` method. In this custom data we send the `categoryName`.

Search Page Events

Search Page events track the usage of the Search Box component in the Accelerator. The Search Box component is a WCMS component that allows users to input a search term, and also provides auto-search results when users enter a search term. The Search event is sent when a user enters a search term, clicks the search button, and gets back a search result that is non-empty. In this case, the information that is sent are the search term and the number of results. Here, we also call the `trackSiteSearch` method to send search-related information.

No Search Result Event

The No Search Result event is similar to the Search Page event, except that it is sent when there are no search results.

Banner Click Event

A Banner Click event is sent when any of the banners in the Accelerator site are clicked by a user. Banners in the Accelerator are WCMS components that have their own view (JSP). To capture the banner click, we need to make sure that the `simple-banner` class element is present in the component view, because the jquery selection is based on this class being present.

In this case, we use the `trackLink` method provided by Piwik. Also, the `setCustomVariable` method is called to pass the banner ID, which is the URL the banner is pointing to.

ProceedToCheckout Event

The ProceedToCheckout event is sent when a user decides to check out the item in their cart page. In this case, the `Checkout` button in the cart page is attached to a click event in the `hybrisAnalytics` AddOn. Piwik provides the `trackEvent` method to track certain events. In terms of parameters, the endpoint expects `checkout` as a `category` and `proceed` as an `action` when calling the `trackEvent` method. The endpoint also expects `cartCode` or `cartId` as one of the custom variables included with this event.

Successful CheckoutEvent

The Successful Checkout event is sent when a user has placed an order and then views the Order Confirmation page.

The order confirmation sends the following requests : `pageview`, `trackevent`, and `ecommerce`.

Cart Modification Events

The Accelerator already provides some events that you can subscribe to regarding cart modifications. The `hybrisAnalytics` AddOn subscribes to these events and then sends the proper information through Piwik when any of these cart modifications is triggered.

AddToCart Event

The AddtoCart event is sent when a user clicks the `Add to Cart` button to add an item to their shopping cart. The Accelerator has `Add To Cart` buttons on various pages, including the Cart Page, the Product Details Page, and the Category Page, as well as in the Suggestions component. Whenever a user clicks the `Add to Cart` button, JS captures the button click and a back-end call is made to the server. In return, we receive the `CartModificationData` response to display the updated information. This data is then used to populate the Piwik event, and the `trackEcommerceCartUpdate` methodis used.

UpdateCart Event

The UpdateCart event in the Accelerator is sent when a user updates the quantity of a particular order entry on the Cart Page. This event is identical to the AddtoCart event. The UpdateCart event is captured from JavaScript where click events are bonded to a specific button.

RemoveCart Event

The RemoveCart event in the Accelerator is sent when a user removes a cart item on the Cart Page. This event is identical to the Add toCart and UpdateCart events except that it sends a quantity of 0.

Google Analytics

This document discusses what is added to the SAP Commerce Accelerator code to enable the monitoring. Using this information, you can also remove Google Analytics from your system, if necessary.

You can monitor detailed statistics about the visitors to your website using the Google Analytics service.

i Note

Monitoring of statistics requires signing in to Google services, and some of the features that are offered must be purchased before they can be used.

Google Analytics Configuration

Google Analytics-specific properties are added to the **`yacceleratorstorefront/project.properties`** and **`yb2bacceleratorstorefront/project.properties`** files. In the **`yacceleratorstorefront`**, these properties are loaded into the page model in the **AnalyticsPropertiesBeforeViewHandler**. In the **yb2bacceleratorstorefront**, these properties are loaded into the page model in the **AbstractPageController**.

It is necessary to add definitions of the storefronts, which are monitored, to the relevant `project.properties` files of your storefronts. The reference version of the SAP Commerce Accelerator provides several storefronts, which are defined in the **yacceleratorstorefront** and **yb2bacceleratorstorefront** extensions. The following are examples:

The following example is from the **`yacceleratorstorefront/project.properties`** file:

```
google.analytics.tracking.id
                              google.analytics.tracking.id.electronics.local
                              google.analytics.tracking.id.apparel-uk.local
                              google.analytics.tracking.id.apparel-de.local
```

The following example is from the **`yb2bacceleratorstorefront/project.properties`** file:

```
google.analytics.tracking.id.powertools.local
```

In order to have all the Google Analytics-specific JavaScript in one tag file, the controllers pass a **PageType** to views where specific tracking is required. In order to track certain calls to Ajax, calls to JavaScript functions defined in the **`web/webroot/WEB-INF/tags/analytics/googleAnalytics.tag/googleAnalytics.tag`** file have been added in the **`addToCart.tag`** and **`cartItems.tag`** files. These functions are only called if they have been defined, so removing all the Google Analytics JavaScript does not cause an error.

Creating a Google Analytics Account

For information on setting up a Google Analytics account, see Google Analytics - Installation Guide.

Related Information

Google Analytics - Installation Guide

User Enumeration Prevention

The B2C Accelerator storefront provides an user enumeration prevention feature for customer registration.

Feature Overview

Customer registration flow including account creation during guest checkout is improved not to share any information about the existence of customer accounts to prevent any possible vulnerability.

Login becomes being required no matter it's a successful registration or failed registration due to duplicated email address.

ℹ Note

This feature is disabled by default. To enable the feature, you need to set the property `toggle.secure.customer.registration.enabled` as true.

For more information, see Patch 2205.26 in Patch Upgrade Notes.

Login with Verification Token

The B2C Accelerator storefront provides a login with verification token feature where a customer can log in with a one-time verification token besides email address and password.

Feature Overview

Customer can request a one-time verification token by providing email address and password. The verification token is delivered through provided email address. By entering the verification token, the customer can continue the login process. Once the verification token validation is passed, the customer will be logged into the site.

ℹ Note

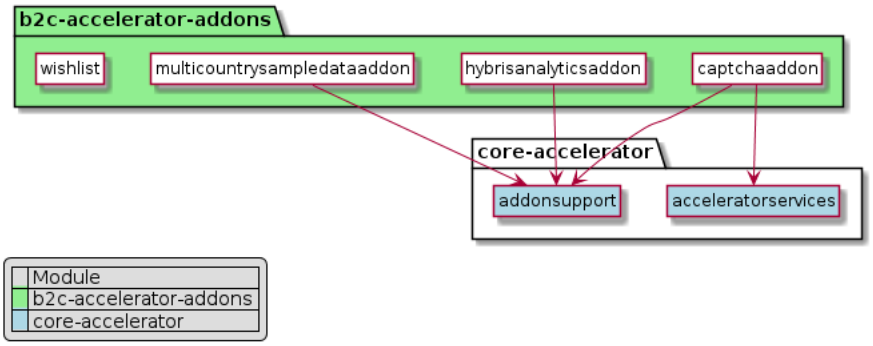
This feature is disabled by default. To enable the feature, you need to set the property `otp.customer.login.enabled` as true.

For more information, see Patch 2205.26 in Patch Upgrade Notes.

B2C Accelerator AddOns Architecture

The B2C Accelerator AddOns module is a set of extensions that allow you to extend the functionality of B2C Accelerator without having to edit the core code base. For example, you can add a reCAPTCHA widget to your storefront with the `captchaaddon` AddOn, or integrate business event tracking functionality with `hybrisanalyticsaddon`.

Dependencies



Recipes

For a complete list of SAP Commerce recipes that may include this module, see Installer Recipes.

For a complete list of the SAP Commerce Cloud, integration extension pack recipes that may include this module, see Installer Recipe Reference.

Extensions

The B2C Accelerator AddOns module consists of the following extensions:

- captchaaddon AddOn
- hybrisAnalytics AddOn
- multicountrysampledataaddon AddOn
- wishlist Extension

captchaaddon AddOn

The `captchaaddon` AddOn introduces a reCAPTCHA widget to the **New Customer Registration** form of the Accelerator storefronts. The technology is used to block spammers and bots that try to automatically harvest e-mail addresses, or that try to automatically sign up for (or make use of) websites, blogs, or forums.

⚠ Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

ℹ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

AddOn Definition

Name	`captchaaddon`
Description	CAPTCHA is a program that protects websites against bots by generating and grading tests that human users can pass but current computer programs cannot. The `captchaaddon` AddOn adds a reCAPTCHA widget to the registration form in the Accelerator storefronts using a free reCAPTCHA service from Google.
Requires	The `captchaaddon` AddOn depends on: • addonsupport Extension
Author	SAP Commerce

Spring and JavaScript Configuration

The `captchaaddon` extension uses Spring AOP in order to intercept the controller methods used for registration-form rendering and submission. It also checks the challenge answer from the reCAPTCHA widget, which is rendered using JavaScript.

The aspect is configured using Spring in the `/hybris/bin/modules/b2c-accelerator-addons/captchaaddon/resources/captchaaddon/web/spring/captchaaddon-web-spring.xml` file.

JavaScript is used to inject the reCAPTCHA widget into the registration form in the storefront. The `captchaaddon.js` file that contains the code `/hybris/bin/modules/b2c-accelerator-addons/captchaaddon/acceleratoraddon/web/webroot/_ui/responsive/common/js/captchaaddon.js`.

All CSS properties used by the `captchaaddon` AddOn are located in `/hybris/bin/modules/b2c-accelerator-addons/captchaaddon/acceleratoraddon/web/webroot/WEB-INF/_ui-src/responsive/less/captchaaddon.less`.

Enabling or Disabling CAPTCHA

You can enable or disable the CAPTCHA feature in the Backoffice Administration Cockpit.

1. Log into the Backoffice and navigate to Base Commerce > Base Store.
2. Click the Search button and select a storefront.
3. In the Properties tab, select the True or False radio buttons under Captcha Widget Enabled to enable or disable the CAPTCHA widget.

Installing the `captchaaddon` AddOn

Note

The `captchaaddon` AddOn is disabled by default. You must enable it after you install it.

1. Add the `captchaaddon` AddOn to your `localextensions.xml` file, ensuring the listed required extensions are also included.

```
<extension dir="${HYBRIS_BIN_DIR}/ext-addon/captchaaddon"/>
<extension dir="${HYBRIS_BIN_DIR}/ext-addon/addonsupport"/>
```

2. Call the `addoninstall` ant command.

```
ant addoninstall -Daddonnames="captchaaddon" -DaddonStorefront.yacceleratorstorefront="yacceleratorstorefront"
```

   This generates the correct properties in the `project.properties` file of the `captchaaddon` AddOn and adds a dependency from your storefront to the `captchaaddon` AddOn.

3. Register your domain, which will allow you to obtain a reCAPTCHA key. You can do this at https://www.google.com/recaptcha/intro/index.html. When you receive your public and private keys, add them to the `local.properties` file.

```
recaptcha.publickey=myGeneratedSiteKey
recaptcha.privatekey=myGeneratedSecretKey
```

   If you wish to have store-specific keys, you can add your store name to the properties. For the following example, change `electronics` to the name of your storefront.

```
recaptcha.publickey.electronics=myGeneratedSiteKey
recaptcha.privatekey.electronics=myGeneratedSecretKey
```

4. Add the Spring configuration property of the `captchaaddon` AddOn to the `project.properties` file.

```
yacceleratorstorefront.additionalWebSpringConfigs.captchaaddon=classpath:/captchaaddon/web/spring/captchaaddon-web-spring.xml
```

Note

If you initialized your system before installing the `captchaaddon` AddOn, you have to run `ant clean all` and update the system.

Modifications Checklist

The modifications that this AddOn makes to Accelerator are listed below:

Impex Configuration Scripts	⛔
Core Data Listeners	⛔
Model Layer	⛔
Model Interceptors	⛔
Cockpit Configuration	⛔

Cockpit Beans	⛔
Validation Rules	⛔

Service Layer	⛔
Facade DTO	⛔
Facade Layer	⛔
CMS Components	⛔
Page Templates	⛔
JavaScript	✅
CSS	✅

Page Controllers	✅
Tags	✅
TLD	✅
Filters	⛔
MVC Interceptors	⛔
Spring Security	⛔
Message Resources	✅

Related Information

yacceleratorstorefront Extension
http://www.google.com/recaptcha
https://www.google.com/recaptcha/admin/create

hybrisAnalytics AddOn

The purpose of the `hybrisanalyticsaddon` AddOn is to provide an integration point between the SAP Commerce Accelerator and the `eventtrackingws` AddOn.

⚠ Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

The `hybrisanalyticsaddon` AddOn does this by sending events through JavaScript to the `eventtrackingws` AddOn. The events consist of user actions in the storefront, such as viewing a page, or adding an item to the cart. The `hybrisanalyticsaddon` AddOn provides the tag files and necessary JavaScripts that can be installed in the storefront.

For more information, see eventtrackingwsaddon AddOn, Business Events in the Commerce Accelerator, and Event Tracking Module.

ℹ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

AddOn Definition

Name	hybrisanalyticsaddon
Description	The purpose of the `hybrisanalyticsaddon` AddOn is to provide an integration point between the Accelerator and the `eventtrackingws` AddOn by sending events through JavaScript to this endpoint.
Requires	Business Events extensions
Author	SAP Commerce

Supported Markets and Channels

The `hybrisanalyticsaddon` AddOn supports the B2C and B2B markets. It can only be used for the desktop channel.

Supported	B2C Commerce	B2B Commerce	Telco Commerce
Market	✅	✅	⛔
Channel	Desktop: ✅ Mobile: ⛔	Desktop: ✅ Mobile: ⛔	Desktop: ⛔ Mobile: ⛔

JS and Tag Components

The `hybrisanalyticsaddon` AddOn uses a Piwik library to send asynchronous JavaScript events to the `eventtrackingws` AddOn. The `piwikAnalytics.tag` file contains the JavaScript code that calls the Piwik JavaScript, which then sends the events to the `eventtrackingws` AddOn.

The `base.js.properties` file can be used to install key-value properties as JavaScript variables in the storefronts. The `base.js.properties` file is located in `/hybrisSrc/acceleratoraddons/hybrisanalyticsaddon/acceleratoraddon/web/webroot/WEB-INF/messages/`.

For more information, see Business Events in the Commerce Accelerator.

The HybrisAnalyticsBeforeViewHandler

The `HybrisAnalyticsBeforeViewHandler` gets executed every time before the view is rendered. The `HybrisAnalyticsBeforeViewHandler` is responsible for reading the `base.js.properties` file in the AddOn, and sets these key-values as model attributes.

The `HybrisAnalyticsBeforeViewHandler` also calls the `SitesConfigService` to set the attributes in the model object, such as the `piwik site id` and other attributes.

Installation

The following procedure describes how to install the `hybrisanalyticsaddon` AddOn.

1. Add the `hybrisanalyticsaddon` AddOn to your `localextensions.xml` file, along with any required extensions, as follows:

   ```
   <extension dir="${HYBRIS_BIN_DIR}/ext-addon/hybrisanalyticsaddon"/>
   <extension dir="${HYBRIS_BIN_DIR}/ext-addon/addonsupport"/>
   ```

2. Install the AddOn by executing the `addoninstall` ant command.

   ```
   ant addoninstall -Daddonnames="hybrisanalyticsaddon" -DaddonStorefront.yacceleratorstorefront="yacceleratorstorefront"
   ```

   This generates the correct properties in the `project.properties` file of the `hybrisanalyticsaddon` AddOn, and adds a dependency from your storefront to the `hybrisanalyticsaddon` AddOn.

The `yacceleratorstorefront`, or the storefront generated from `yacceleratorstorefront`, has CSRF filters that expect a CSRF token during the `POST` request. To ensure that the Ajax `POST` request from the Piwik JavaScript doesn't get filtered by the CSRF filter, add the events servlet path in the list of URLs where the CSRF is allowed to ignore POST requests.

The following is an example from the `project.properties` of your storefront:

```
csrf.allowed.url.patterns=/[^/]+(/[^?]*)+(sop/response)$,/[^/]+(/[^?]*)+(merchant_callback)$,/[^/]+(/[^?]*)+(hop/response)$,/[^/]+(/[^?]*)+(language
```

In this example, `/(events)$` is added to this property to ensure that the CSRF filter allows `POST` requests without the CSRF tokens.

Configuring the Site IDs

Piwik expects site IDs to be included with each event. These site IDs must be integers and they're configurable. You can have a site ID for your entire SAP Commerce setup, or you can configure them per site. This is done through the `local.properties` file.

The following is the global site ID:

```
piwik.tracker.siteid=<your-site-id>
```

The following is the site ID per site :

```
piwik.tracker.siteid.<storeuid>=<your-site-id>

For example:
piwik.tracker.siteid.electronics=123456789
```

Configuring the Tracking URLs

You can define the URLs to send the events to. You can define two different URLs that depend on which channel you're sending the events on (for example, HTTP and HTTPS). By default, the AddOn sends the events to the localhost.

```
piwik.tracker.url=http://electronics.local:9002/yacceleratorstorefront/events
piwik.tracker.https.url=https://electronics.local:9002/yacceleratorstorefront/events
```

Integration with the Endpoint

For information on installing the event tracking endpoint, see eventtrackingwsaddon AddOn.

Verifying the Integration

The following procedure describes how to verify the integration of the `hybrisanalyticsaddon` AddOn.

1. Start the SAP Commerce server.
2. Access the Accelerator storefront home page.

   The following link is an example: `http://electronics.local:9001/yacceleratorstorefront`

3. Browse to a product page to trigger event tracking.

4. Using the Chrome browser, right-click on a page and select `Inspect Element`.
5. In the window that appears, select the `Network` header tab and find the `POST` method with the following URL: `/yacceleratorstorefront/events`.

   By adding two properties, you can enable the logging of the events, as follows:

   ```
   log4j.logger.de.hybris.eventtracking.publisher=DEBUG
   spring.profiles.active=eventtrackingpublisher_develop
   ```

   Note

   These properties aren't specific to the `hybrisanalyticsaddon` AddOn but to the `eventtrackingws` AddOn.

6. The end-point URL is HTTPS, and for a development environment the browser may block the Ajax calls due to an invalid SSL certificate. If this is the case, open the browser, type in the endpoint URL `https://electronics.local:9002/events` and force the browser to allow this HTTPS URL even though the certificate may not be valid.

Modifications Checklist

The following table lists the modifications that the AddOn makes to the Accelerator:

Impex Configuration Scripts	⛔
Core Data Listeners	⛔
Model Layer	⛔
Model Interceptors	⛔
Cockpit Configuration	⛔
Cockpit Beans	⛔
Validation Rules	⛔

Service Layer	⛔
Facade DTO	⛔
Facade Layer	⛔
CMS Components	✅
Page Templates	⛔
JavaScript	⛔
CSS	⛔

Page Controllers	⛔
Tags	⛔
TLD	⛔
Filters	⛔
MVC Interceptors	⛔
Spring Security	⛔
Message Resources	⛔

wishlist Extension

The `wishlist` extension provides functionality for listing and maintaining items a customer would like to have, for example products intended for buying or desired as gifts.

Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Related Information

Using Wishlist2Service API

multicountrysampledataaddon AddOn

The multicountrysampledataaddon is the AddOn that is responsible for creating sample data for the electronics storefront to demonstrate the multi-country features in the Accelerator web application. The AddOn creates additional electronics sites to simulate parent-child catalog hierarchies.

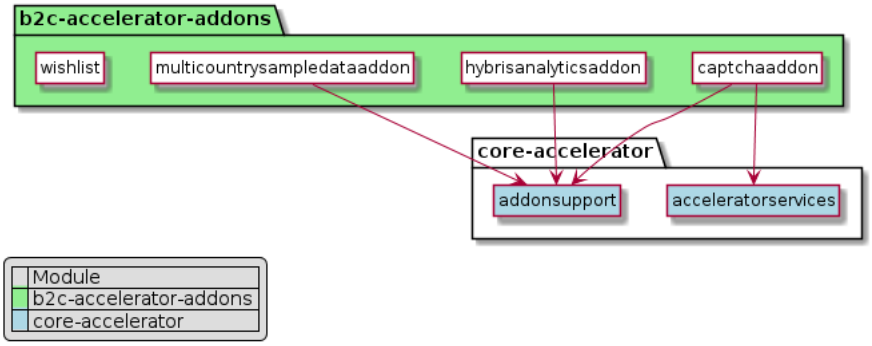
Caution

This page refers to software that has been deprecated as part of the Accelerator UI and older OCC template extensions deprecation. For more information, see Deprecation of Accelerator UIs and Older OCC Template Extensions.

About the Extension

Name	Directory	Related Module
`multicountrysampledataaddon`	`hybris/bin/modules/b2c-accerlator-addons`	B2C Accelerator AddOns Architecture

Dependencies



B2C Accelerator AddOns Implementation

You can extend your B2C Accelerator storefront with wishlist functionality, add a reCAPTCHA widget, and optimize your storefront for mobile devices.

Updating captchaaddon to Support reCAPTCHA v2.0

Google no longer provides new keys for reCAPTCHA v1.0. Existing implementations of the `captchaaadon` AddOn will continue to work, but new implementations must use reCAPTCHA v2.0. The `captchaaadon` AddOn can be modified to support reCAPTCHA v2.0.

Using Wishlist2Service API

The `wishlist` extension is based on the ServiceLayer and is service-oriented. Use the Wishlist2Service API to manage wishlist.

Updating captchaaddon to Support reCAPTCHA v2.0

Google no longer provides new keys for reCAPTCHA v1.0. Existing implementations of the `captchaaadon` AddOn will continue to work, but new implementations must use reCAPTCHA v2.0. The `captchaaadon` AddOn can be modified to support reCAPTCHA v2.0.

Context

Google's reCAPTCHA v2.0 no longer supports public and private keys. Instead, reCAPTCHA v2.0 uses a site key instead of a public key, and a secret key instead of a private key. These keys are used to render the captcha widget and validate the response.

Procedure

1. Open the following URL: https://www.google.com/recaptcha/intro/index.html
2. Register your website and retrieve a new site key and secret key.
3. Update the `config/local.properties` file to include your two new keys, as follows:

```
...
recaptcha.publickey=YOUR SITE KEY
recaptcha.privatekey=YOUR SECRET KEY
...
```

4. Update `captchaaddon/acceleratoraddon/web/webroot/WEB-INF/views/responsive/pages/widget/recaptcha.jsp` to include an element with the appropriate reCAPTCHA class and site key, as follows:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@ taglib prefix="theme" tagdir="/WEB-INF/tags/shared/theme" %>
<%@ taglib prefix="ycommerce" uri="http://hybris.com/tld/ycommercetags" %>

<c:if test="${requestScope.captchaEnabledForCurrentStore}">
   <div id="g-recaptcha_incorrect"><spring:theme code="recaptcha.challenge.field.invalid"/></div>
   <div id="g-recaptcha_widget" class="g-recaptcha" data-sitekey="${requestScope.recaptchaPublicKey}"></div>
</c:if>
```

   The `g-recaptcha` class indicates which `div` holds the reCAPTCHA widget. The `data-sitekey` is used for validating to the reCAPTCHA widget. In the preceding example, there is an additional `div` to show user validation in cases where a user tries to register without completing the captcha.

5. You may want to update the CSS. The following is an example from `captchaaddon/acceleratoraddon/web/webroot/WEB-INF/_ui-src/responsive/less/captchaaddon.less`.

```
#registerForm .form_field_error {
  width:auto;
  float: none;
```

```
  padding-right:0;
  clear:both;
}

.js-recaptcha-captchaaddon {
  margin: 20px 0;
}

#g-recaptcha_incorrect {
  color: red;
  display: none;
}
```

6. Update `captchaaddon/acceleratoraddon/web/webroot/_ui/responsive/common/js/captchaaddon.js` as follows:

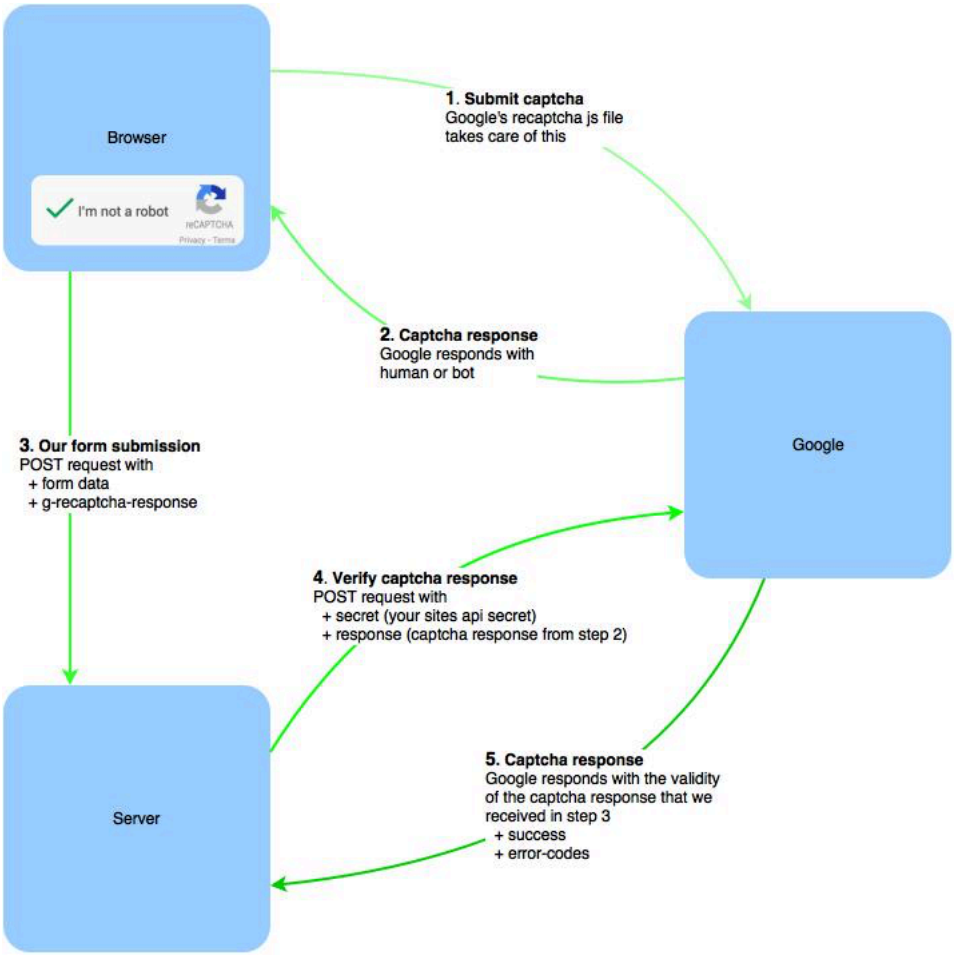
```
/*
 * [y] hybris Platform
 *
 * Copyright (c) 2000-2016 SAP SE or an SAP affiliate company.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of SAP
 * ("Confidential Information"). You shall not disclose such Confidential
 * Information and shall use it only in accordance with the terms of the
 * license agreement you entered into with SAP.
 */
ACC.captcha = {
   bindAll: function ()
   {
      this.renderWidget();
   },

   renderWidget: function ()
   {
      $.ajax({
         url: ACC.config.encodedContextPath + "/register/captcha/widget/recaptcha",
         type: 'GET',
         cache: false,
         success: function (html)
         {
            if ($(html) != [])
            {
               $(html).appendTo('.js-recaptcha-captchaaddon');
               $.getScript('https://www.google.com/recaptcha/api.js?hl=' + document.documentElement.lang, function ()
               {
                  if ($('#recaptchaChallangeAnswered').val() == 'false')
                  {
                     $('#g-recaptcha_incorrect').show();
                  }
               });
            }
         }
      });
   }
};

$(document).ready(function ()
{
   if ($('#registerForm').html() != null || $('#updateEmailForm').html() != null)
   {
      ACC.captcha.bindAll();
   }
});
```

The `captchaaddon.js` file renders the widget by loading Google's JavaScript API with the necessary parameters. For more information on displaying the widget, see the following Google documentation: https://developers.google.com/recaptcha/docs/display

7. Update the backend to validate the captcha response.

When the user completes the captcha and submits the form, a captcha response is returned. You can validate this response, using Google's API, by submitting a POST request to `https://www.google.com/recaptcha/api/siteverify` with the recaptcha response and secret key as parameters. This returns a JSON object with a boolean under **success** that indicates whether the captcha is valid or not.

The following diagram illustrates the process:



8. Update `captchaaddon/acceleratoraddon/web/src/de/hybris/platform/security/captcha/ReCaptchaAspect.java`. The following is an example:

```
/*
 * [y] hybris Platform
 *
 * Copyright (c) 2000-2016 SAP SE or an SAP affiliate company.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information of SAP
 * ("Confidential Information"). You shall not disclose such Confidential
 * Information and shall use it only in accordance with the terms of the
 * license agreement you entered into with SAP.
 */
package de.hybris.platform.security.captcha;

import atg.taglib.json.util.JSONException;
import atg.taglib.json.util.JSONObject;
import de.hybris.platform.acceleratorservices.config.SiteConfigService;
import de.hybris.platform.store.BaseStoreModel;
import de.hybris.platform.store.services.BaseStoreService;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;

import javax.servlet.http.HttpServletRequest;

import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.collections.PredicateUtils;
import org.apache.commons.httpclient.DefaultHttpMethodRetryHandler;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpStatus;
import org.apache.commons.httpclient.methods.PostMethod;
import org.apache.commons.httpclient.params.HttpMethodParams;
import org.apache.commons.lang.StringUtils;
import org.apache.log4j.Logger;
import org.aspectj.lang.ProceedingJoinPoint;
import org.springframework.beans.factory.annotation.Required;
import org.springframework.validation.BindingResult;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;


/**
 * An aspect which uses google ReCaptcha api to validate captcha answer on the storefront Registration form.
 */

public class ReCaptchaAspect
{
   private static final Logger LOG = Logger.getLogger(ReCaptchaAspect.class);

   private static final String RECAPTCHA_SITE_KEY_PROPERTY = "recaptcha.publickey";
   private static final String RECAPTCHA_SECRET_KEY_PROPERTY = "recaptcha.privatekey";
   private static final String RECAPTCHA_RESPONSE_PARAM = "g-recaptcha-response";
   private static final String RECAPTCHA_VERIFY_URL = "https://www.google.com/recaptcha/api/siteverify";

   private SiteConfigService siteConfigService;
   private BaseStoreService baseStoreService;

   public Object prepare(final ProceedingJoinPoint joinPoint) throws Throwable
```

```
{
   final List<Object> args = Arrays.asList(joinPoint.getArgs());
   final HttpServletRequest request = (HttpServletRequest) CollectionUtils.find(args,
         PredicateUtils.instanceofPredicate(HttpServletRequest.class));

   if (request != null)
   {
      final boolean captchaEnabledForCurrentStore = isCaptchaEnabledForCurrentStore();
      request.setAttribute("captchaEnabledForCurrentStore", Boolean.valueOf(captchaEnabledForCurrentStore));
      if (captchaEnabledForCurrentStore)
      {
         request.setAttribute("recaptchaPublicKey", getSiteConfigService().getProperty(RECAPTCHA_SITE_KEY_PROPERTY));
      }
   }
   return joinPoint.proceed();
}

public Object advise(final ProceedingJoinPoint joinPoint) throws Throwable
{

   final boolean captchaEnabledForCurrentStore = isCaptchaEnabledForCurrentStore();
   if (captchaEnabledForCurrentStore)
   {
      final List<Object> args = Arrays.asList(joinPoint.getArgs());
      HttpServletRequest request = (HttpServletRequest) CollectionUtils.find(args,
            PredicateUtils.instanceofPredicate(HttpServletRequest.class));

      if (request == null && RequestContextHolder.getRequestAttributes() instanceof ServletRequestAttributes)
      {
         final ServletRequestAttributes requestAttributes = (ServletRequestAttributes) RequestContextHolder
               .getRequestAttributes();
         request = requestAttributes.getRequest();
      }

      if (request != null)
      {
         request.setAttribute("captchaEnabledForCurrentStore", Boolean.valueOf(captchaEnabledForCurrentStore));
         request.setAttribute("recaptchaPublicKey", getSiteConfigService().getProperty(RECAPTCHA_SITE_KEY_PROPERTY));
         final String recaptchaResponse = request.getParameter(RECAPTCHA_RESPONSE_PARAM);
         if (StringUtils.isBlank(recaptchaResponse) || !checkAnswer(recaptchaResponse))
         {
            // if there is an error add a message to binding result.
            final BindingResult bindingResult = (BindingResult) CollectionUtils.find(args,
                  PredicateUtils.instanceofPredicate(BindingResult.class));
            if (bindingResult != null)
            {
               bindingResult.reject("recaptcha.challenge.field.invalid", "Challenge Answer is invalid.");
            }
            request.setAttribute("recaptchaChallangeAnswered", Boolean.FALSE);
         }
      }
   }
   return joinPoint.proceed();
}

protected boolean checkAnswer(final String recaptchaResponse)
{
   final HttpClient client = new HttpClient();
   final PostMethod method = new PostMethod(RECAPTCHA_VERIFY_URL);

   method.getParams().setParameter(HttpMethodParams.RETRY_HANDLER, new DefaultHttpMethodRetryHandler(3, true));
   method.addParameter("secret", getSiteConfigService().getProperty(RECAPTCHA_SECRET_KEY_PROPERTY));
   method.addParameter("response", recaptchaResponse);

   try
   {
      final int statusCode = client.executeMethod(method);

      if (statusCode != HttpStatus.SC_OK)
      {
         return false;
      }

      final JSONObject response = new JSONObject(method.getResponseBodyAsString());
      return response.getBoolean("success");
   }
   catch (IOException | JSONException e)
   {
      LOG.error("Exception occurred while checking captcha answer", e);
      return false;
   }
   finally
   {
      method.releaseConnection();
   }
}

protected boolean isCaptchaEnabledForCurrentStore()
{
   final BaseStoreModel currentBaseStore = getBaseStoreService().getCurrentBaseStore();
   return currentBaseStore != null && Boolean.TRUE.equals(currentBaseStore.getCaptchaCheckEnabled());
}

protected SiteConfigService getSiteConfigService()
{
   return siteConfigService;
}

@Required
public void setSiteConfigService(final SiteConfigService siteConfigService)
{
   this.siteConfigService = siteConfigService;
}

protected BaseStoreService getBaseStoreService()
{
   return baseStoreService;
}

@Required
public void setBaseStoreService(final BaseStoreService baseStoreService)
{
```

```
            this.baseStoreService = baseStoreService;
        }
    }
```

The `json-taglib` has also been added to the `captchaaddon` AddOn. The `prepare()` method passes along the sitekey required to render the widget. The `advise()` method is called when the user submits the registration form and uses the `checkAnswer()` method to validate the user's response. The remaining methods are unchanged.

In the above example, a new library has also been added to parse the JSON object response.

9. Add the `json-taglib-0.4.1` library to the `captchaaddon` AddOn by adding the following lines to `captchaaddon/acceleratoraddon/web/webroot/WEB-INF/external-dependencies.xml`:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    ...
    <dependencies>
        <dependency>
            <groupId>de.hybris.external</groupId>
            <artifactId>json-taglib</artifactId>
            <version>0.4.1</version>
        </dependency>
    </dependencies>
</project>
```

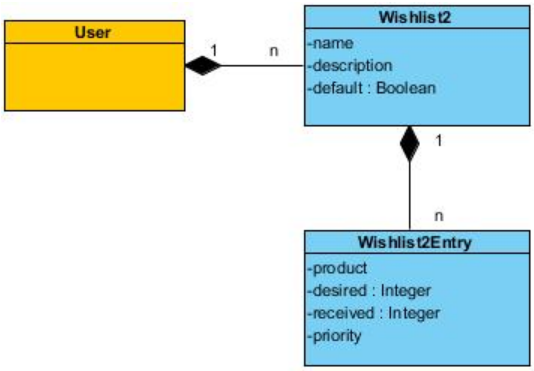
This adds `captchaaddon/acceleratoraddon/web/webroot/WEB-INF/lib/json-taglib-0.4.1.jar` when you build your project by running `ant clean all`. Add this jar as a dependency to the `captchaaddon` AddOn. You can do this in Eclipse by right-clicking on Project > Build Path > Configure Build Path, and under the Libraries tab, clicking Add Jars and selecting the jar.

Using Wishlist2Service API

The `wishlist` extension is based on the ServiceLayer and is service-oriented. Use the Wishlist2Service API to manage wishlist.

About Wishlists

The `wishlist` extension provides the possibility for the user to organize and keep track of the desired products. One user can have as many wishlists as needed, and the wishlist can contain as many entries as possible. However, in each wishlist the same product can only occur once. Duplicated products are not allowed in the same wishlist.



Using the Wishlist2Service API

The de.hybris.platform.wishlist2.Wishlist2Service provides the necessary methods to manage wishlists. You can refer to the service within your `spring.xml` file.

```
<bean id="yourService" class="yourServiceClass">
        <property name="wishlist2Service" ref="wishlist2Service"/>
</bean>
```

Alternatively, you can accomplish that by using Java code as in the code sample below.

```
Wishlist2Service wsService = (Wishlist2Service) Registry.getApplicationContext().getBean("wishlistService");
```

Creating a Wishlist

In order to create the default wishlist for a user, call the method as shown in the following code snippet.

```
public Wishlist2Model createDefaultWishlist(UserModel user, String name, String description);
```

To create a normal wishlist, call the method as shown in following code snippet.

```
public Wishlist2Model createWishlist(UserModel user, String name, String description);
```

Adding a Wishlist Entry to the Default Wishlist

A wishlist entry can be added to the default wishlist of the specific user.

```
public void addWishlistEntry(final UserModel user, final ProductModel product,
                             final Integer desired, final Wishlist2EntryPriority priority,
                             final String comment);
```

ℹ Note

Avoiding Duplicated Wishlist Entries

The same product should not be added twice in the same wishlist. To prevent duplication problems, proceed as follows: Before the wishlist entry is created and added to the active wishlist, iterate the wishlist and check whether there is already a wishlist entry with the specified product:

- If true, then give the user a message and return.
- If false, create the wishlist entry and add it to the current wishlist.

Retrieving Wishlists of the User

If you want to retrieve all wishlists of the specific user, call this method:

```
public List<Wishlist2Model> getWishlists(final UserModel user);
```

Similarly, you can retrieve the default wishlist.

```
public Wishlist2Model getDefaultWishlist(final UserModel user);
```

Removing a Wishlist Entry from the Wishlist

If one product is no longer needed, call the method from the following code sample to remove the corresponding wishlist entry from the wishlist.

```
public void removeWishlistEntry(final Wishlist2Model wishlist, final Wishlist2EntryModel entry);
```

The entry being removed by that method is an object of the public Wishlist2EntryModel class that extends the ItemModel. It has several attributes, for example:

- `received`: The number of products that the user has received. Initial value: 0
- `desired`: The number of products that the user wants. Initial value: 1

If the users buy a product, then the `received` property can be set as 1 or to the number of products that users have bought. The value of the `desired` property depends on users, because only the users know how many products they still want to have.