Amplience Extension for SAP® Hybris® Commerce

-

Version 2.0.0

-

System Integration Guide

Author	Pablo Plaza Lastras, Nicko Cadell, Rohan Gadkari, Oliver Secluna
Date	20/10/2017

Contents

1. INTRODUCTION 4
1.1 PRODUCT MEDIA OVERVIEW 5
1.2 MEDIA SETS AND IMAGES IN AMPLIENCE 5
1.3 TRANSFORMATION TEMPLATES 5
2. EXTENSIONS OVERVIEW 6
3. CONFIGURATION 6
3.1 AMPLIENCEDM EXTENSION 6
3.2 AMPLIENCEDMADDON EXTENSION 7
3.3 AMPLIENCEDMBACKOFFICE EXTENSION 7
3.4 AMPLIENCEDMCOCKPITS EXTENSION 7
4. CORE DATA MODEL AND STRATEGIES 7
4.1 EXTENDED ITEM TYPES 7
4.2 NEW ITEM TYPES 7
4.3 EXTENDED BEAN TYPES 8
4.4 NEW BEAN TYPES 8
4.5 AMPLIENCECONFIGSERVICE 8
4.6 PRODUCT IMAGE URL STRATEGIES 8
4.6.1. AmplienceProductImageUrlResolver 9
4.6.2. AmplienceProductResolver 9
4.6.3. ApparelAmplienceProductResolver 9
4.6.4. AmplienceIdentifierSanitizer 10
4.6.5. AmplienceSeoImageNameStrategy 11
4.6.6. AmplienceProductSwatchUrlResolver 11
4.7 PRODUCT DATA POPULATORS 11
4.7.1. AmplienceProductPrimaryImagePopulator 11
4.7.2. AmplienceSearchResultVariantProductPopulator 12
4.7.3. ExamplePromoAmplienceSearchResultVariantProductPopulator 12
4.7.4. AmplienceEmailContextCatalogVersionPopulator 12
4.7.5. AmplienceEmailContextProductPopulator 12
4.7.6. AmplienceProductDataPopulator 12
4.7.7. ProductAmplienceMediaSetPopulator 13
4.7.8. DefaultVariantOptionDataPopulator 13
4.7.9. AmplienceVariantOptionDataPopulator 13

4.8 AMPLIENCE JOBS 13
4.8.1. AmplienceProductImageStatusUpdateCronJob 13
4.8.2. AmplienceProductImageEmailReportCronJob 15
5. CMS COMPONENTS 16
5.1.1. AmplienceResponsiveImageComponent 17
5.1.2. AmplienceImageCarouselComponent 17
5.1.3. AmplienceUGCCarouselComponent 17
5.1.4. AmplienceUGCMediaWallComponent 17
5.2 AMPLIENCEADDONCMSCOMPONENTRENDERER 18
6. STOREFRONT CHANGES 18
6.1 AMPLIENCEBEFOREVIEWHANDLER 19
6.2 ECOMMBRIDGE 19
6.2.1. AmplienceBridgeController 19
6.2.2. InstagramCallbackController 20
6.2.3. AmplienceProductFacade 20
6.2.4. AmplienceResourceUrlFacade 20
6.2.5. AmplienceLocaleStringStrategy 20
6.2.6. CurrencyFormatSupplier 20
6.2.7. ecommBridge.js 20
6.3 AMPLIENCEPAGESCRIPT.TAG 21
6.4 BINDCOMPONENTS.JS 21
6.4.1. CMS Components JSPs 22
AmplienceResponsiveImageComponent.jsp 22
responsiveImage.tag 22
AmplienceImageCarouselComponent.jsp 22
AmplienceUGCCarouselComponent.jsp 22
AmplienceUGCMediaWallComponent.jsp 22
6.4.2. StoreFront Pages and JSP Tag files. 23

staticContentLayoutPage.jsp 23
image.tag 23
productLayout1Page.jsp 23
standardPdpViewer.tag 23
productLayout2Page.jsp 23
renderKitPdpViewer.tag 23
quickViewPopup.jsp 24
6.5 PRODUCT IMAGE IMPORT INTO AMPLIENCE 24
6.5.1. ExportImages 24
exportImages.groovy 25
6.5.2. DeleteImages 26
7. AMPLIENCEDMBACKOFFICE EXTENSION 26
8. AMPLIENCEDMCOCKPITS EXTENSION 26
8.1 AMPLIENCEEDITORAREACONTROLLERIMPL 26
8.2 AMPLIENCETHUMBNAILURLATTRIBUTEHANDLER 26
9. HOW TO ADAPT THE EXTENSION TO YOUR NEEDS 27
9.1 HOW TO ADAPT THE EXTENSION TO YOUR MODEL 27
9.2 HOW TO ADD A SWATCH IMAGE TO A PRODUCT 27
9.3 VIDEOS 27
9.4 SPIN SETS 27
9.5 CREATE A NEW ROUNDEL 28
9.6 CREATE A NEW IMAGE SIZE 28
9.7 CHANGE YOUR SEO STRATEGY 29

1. INTRODUCTION

Amplience Dynamic Media (DM) is a powerful automated image and video management service which includes dynamic imaging and video transcoding capabilities, metadata, viewer configuration, reporting, and analytics.

The service will dynamically render any image variant on demand from a single master asset based on URL parameters (e.g. size), reducing drastically the number of assets that need to be created/maintained.

The Amplience DM assets are served from a high performance CDN which improves the performance of the storefront.

This document is aimed at Developers and Solutions Architects, and should be read after the Amplience Extension for SAP® Hybris® Commerce Business User Guide, which gives an overview of the main functionalities although some of them will be touched upon again in this document.

This document will explain the main classes and files where the logic of the extension resides and will give the integrators understanding of where to look at/what to change to adapt the extension to their Hybris implementation. The extensions work with the Hybris Apparel Accelerator Demo Store. If the customer solution uses a different product data model then certain customisations may need to be made which depend on the implementation's specific requirements.

The Amplience Extension for SAP® Hybris® Commerce is made up of four different extensions that will be covered in following sections. At a high level the solution provides support for the following capabilities:

- Master all product media in Amplience DM.
- Load all product images in the Apparel Accelerator Storefront from the Amplience DM CDN.
- Product details page updated to use Amplience media viewer that supports images, videos, and 360 spin sets.
- CMS components that utilise content from Amplience – images, carousels, user generated content.
- Updates to the Hybris Product Cockpit and Backoffice to load product images from Amplience DM.
- Embeds the Amplience UGC tools directly into the Hybris Backoffice.
- Example one time migration script to export product media from Hybris using a naming convention supported by Amplience ready to be bulk uploaded to Amplience.

The Amplience Dynamic Media extensions are pre-configured to work with an Amplience DM account called "**hybrisext**", this account already has all the product media from the Hybris Apparel Accelerator Demo Store loaded. This means that you can take the vanilla Apparel Accelerator, install the Amplience Dynamic Media extensions and start hybris and use the Apparel storefront without having to setup an Amplience DM account.

Of course when you have your own Amplience DM account you can configure the extensions to use it via Hybris configuration properties.

You will need to configure your own Amplience DM account and upload any product media you might need. The Amplience DM account can bulk import media via SFTP. Further details are available from Amplience.

The Apparel Demo store data including all the media has been preloaded but the media has been exported to Amplience and media URL references changed.

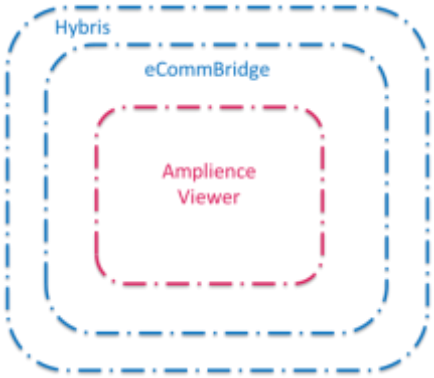
The product images are loaded from the Amplience CDN using various formatting parameters which control the size, quality, etc. Further details on the Amplience formatting options is available in the Amplience Playground website (http://playground.amplience.com/).

A key feature of the Amplience Dynamic Media extensions is that all product media is mastered in Amplience Dynamic Media and is not loaded into Hybris or managed in Hybris. When the Hybris Apparel Accelerator Demo Store initialises an empty system it will load all the sample product media into Hybris. When using the Amplience Dynamic Media extensions this product media in Hybris is not used. Included with the Amplience Dynamic Media extensions there is an example script to delete all the product media from the Hybris instance.

The Amplience Dynamic Media extensions use a naming convention to build the URL to a product's media in the Amplience DM CDN. This means that there is no item in Hybris to represent the product media. It is possible that a product in Hybris does not have product media in Amplience DM. In this case Amplience CDN will return a 'missing' image, which is configured in the Amplience DM account.

One of the capabilities of the Amplience Dynamic Media extensions is the integration of the Amplience media viewers into the product details page. The Amplience media viewers require the storefront to support a standard javascript API called the Amplience eCommBridge. The Amplience Dynamic Media extensions add the eCommBridge API to the Accelerator storefront to allow the Amplience viewers to be embedded in the storefront pages.

Figure 1. ECommBridge



The Amplience eCommBridge is an adapter/data layer implemented in the storefront to allow embedding of Amplience viewers.

This extension has been built using the Hybris Spring design patterns, where all the service and strategy beans are in spring configuration files and if they need changing/replacing, they can be done via Spring configuration changes. All the beans will have separate interface and implementation classes to help with dependency injection.

1.1 Product Media Overview

1.2 Media Sets and images in Amplience

Amplience DM supports storing 3 types of asset, Images, Videos, or Media Sets. A Media Set is simply an ordered list of Amplience assets and can contain any combination or images, videos, or even nested media sets. Each asset has a unique code – which must be unique across all types of assets stored in your account.

For examples of assets hosted within Amplience and to experiment with transformation parameters then look in the Amplience Playground for images http://playground.amplience.com/di/app/#/images and for media sets http://playground.amplience.com/di/app/#/sets.

The approach taken in the Amplience Dynamic Media extensions is to package all the product's images into a single media set. The product images also exist separately however they are typically requested from the media set rather than directly. The media set has a couple of nice features to support this. If the media set URL is requested with an image encoding, for example from within an HTML <img> tag, then Amplience will return the first image from the media set. It is possible to access the assets within the media set by using the media set's URL with a slash and the index position of the item in the media set.

These features of Amplience media sets allow the first product in the media set to be used as the primary image shown on the storefront (PLP, Cart, Order etc…) and also to drive the product gallery shown on the PDP pages.

The naming convention for the product's media set is "<product code>**-ms**".

The only exception to this is that it is possible to setup a swatch image override which is shown instead of the first image from the product's media set. This swatch image is loaded directly and not from the product's media set – it is likely that the swatch image will not be a member of the media set.

The naming convention for the alternative swatch image is "<product code>**-swatch**".

1.3 Transformation Templates

Amplience DM supports applying transformations to image via URL parameters. The most common transformation would be the size of the image, although there are many others defined in the Amplience Documentation.

Amplience DM also supports pre-defined named groups of parameters that are setup within your Amplience account, these are also specified on the URL but the actual transformation parameters that are applied are fully managed within the Amplience account.

The Amplience Dynamic Media extensions use transformation templates rather than actually applying specific transformation parameters, this allows the actual transformation parameters to be changed from within the Amplience account without having to make changes in Hybris or the Amplience Dynamic Media extensions.

Hybris already supports the concept of different sizes or formats for each product image. It does this using the ImageFormat type. Each ImageFormat has unique name. The ImageFormats that Hybris provides are named “product”, “zoom”, “styleSwatch”, “thumbnail”, and “cartIcon”.

The Amplience Dynamic Media extensions use the name of the Hybris ImageFormat as the Transformation Template used in the URL. For example the thumbnail image is just the main product image with the “thumbnail” Transformation Template applied as a URL parameter. The “thumbnail” Transformation Template in the Amplience account then controls the transformations that are applied to the image – i.e. resizing the image.

2. EXTENSIONS OVERVIEW

There are four extensions included in the Amplience Extension for SAP® Hybris® Commerce solution. These can be found in the hybris/bin/ext-amplience directory:

ampliencedm: This extension provides the core data model and integration with Amplience DM. All the functionality delivered in this extension is customisable via hybris spring configuration. This extension provides a number of strategies that can be replaced to customise the behaviour, reasonable defaults have been supplied. This extension has dependencies on the hybris commercefacades and acceleratorservices extensions, but it does not depend on any of the sample accelerator storefronts.

ampliencedmaddon: This extension is a hybris accelerator AddOn that integrates Amplience DM functionality directly into the Accelerator Apparel Demo Store (yacceleratorstorefront). This extension is designed to work with the Accelerator Apparel Demo Store and can be used as a starting point for developing a storefront that utilises Amplience DM functionality.

This extension is the only place where the Apparel Demo Store is specifically referred to, be that the data model for the Apparel product types, or the product and content catalogs, or the CMS content pages setup for the apparel storefronts.

ampliencedmbackoffice: This extension provides integration with the hybris backoffice. The backoffice admin area configuration is defined here. The extension also provides a new tab within the backoofice which loads the Amplience UGC tool.

ampliencedmcockpits: This extension provides the CMS Cockpit and Product Cockpit configuration for the new and modified item types created in the ampliencedm extension. There are further customisations to the Product Cockpit to display product images loaded from Amplience.

3. CONFIGURATION

Hybris has a hierarchical system of properties files. Base configuration can be defined in an extension’s project.properties file and still be overridden in the global local.properties file.

3.1 ampliencedm extension

The configuration in the ampliencedm extension project.properties file includes sensible defaults values that support the out of the box configuration. These values can be overridden in the global local.properties file.

Configuration can be overridden on the BaseSite

amplience.dm.config.accountIdentifier=hybrisext

amplience.dm.config.analyticsCollectorId=

Default Amplience hostnames, change if using your own DNS CNAMEs

amplience.dm.config.imageHostname=i1.adis.ws

amplience.dm.config.contentHostname=c1.adis.ws

amplience.dm.config.analyticsHostname=a1.adis.ws

amplience.dm.config.scriptHostname=s1.adis.ws

The Amplience account identifier set here is to the sample Amplience account that has been preconfigured by Amplience. You will need to override this with your account identifier.

These configuration values can be overridden per site within hybris by being set on the BaseSite item in the HMC or in the Backoffice Admin Area.

3.2 ampliencedmaddon extension

Just like any hybris AddOn the ampliencedmaddon extension's project.properties file has configuration for additional CSS and JavaScript files that need to be added to the storefront pages. The only reason to override these would be if newer versions of the Amplience CSS and JavaScript files are released.

3.3 ampliencedmbackoffice extension

The ampliencedmbackoffice extension's project.properties defines the URL to the Amplience UGC functionality that is embedded into the hybris Backoffice.

ampliencedmbackoffice.ugcmoderationwidget.URL=https://apps.dev-artifacts.adis.ws/dam-app-social/hotfix-ugcui-611-multiple-bulk-action-bug/694/index.html?appSource=external/#/feed

This will need to be overridden if Amplience release an updated UGC tool with a modified URL.

3.4 ampliencedmcockpits extension

The ampliencedmcockpits extension doesn't have any configuration that needs to be changed.

4. CORE DATA MODEL AND STRATEGIES

The core data model, services and strategies are implemented in the ampliencedm extension.

4.1 Extended Item Types

Type	Description
Product	Add amplienceImageStatus and amplienceAltSwatch.
BaseSite	Add site specific overrides for Amplience configuration parameters.

4.2 New Item Types

Type	Description
AmplienceProductImageStatusUpdateCronJob	Cron Job that queries Amplience image status for each matching product.

Type	Description
AmplienceProductImageEmailReportCronJob	Cron Job that sends an email report of products which are missing Amplience images.
AmplienceProductsMissingMediaView	HMC view that provides a way of reporting on the products that are missing media in Amplience.

4.3 Extended Bean Types

Type	Description
ProductData	Extend the hybris Product data to add the name of the Amplience media set holding the product's media.

4.4 New Bean Types

Type	Description
AmplienceConfigData	Data object to hold Amplience configuration
AmplienceProductData	Data object to hold Amplience product data to return to the Amplience ecommBridge
AmplienceResourceData	Data object to hold Amplience resource data to return to the Amplience ecommBridge
AmplienceProductImageEmailReportData	Data object to hold the missing images report data passed into the report email templates.

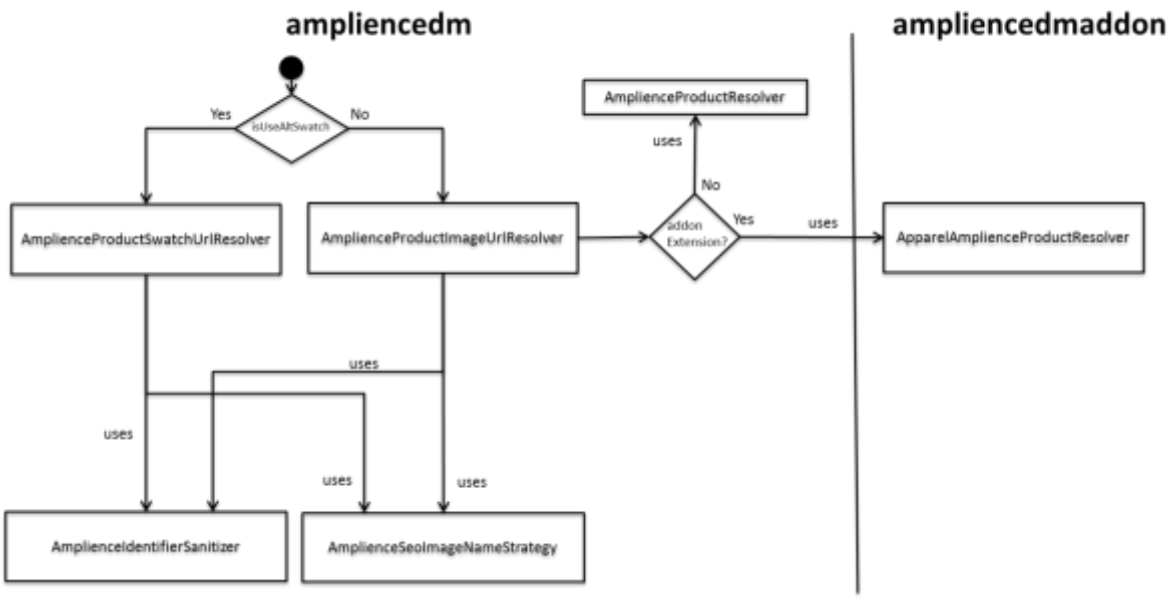
4.5 AmplienceConfigService

The AmplienceConfigService is responsible for loading the Amplience configuration either for a specified site or for the current site.

4.6 Product Image URL strategies

Image URL strategies are the classes that build the URLs to Amplience Media for Hybris Products. This is the summary of the Amplience Product Image Strategies, which are covered in the rest of this section.

Figure 2. Product Image URL strategies



4.6.1. AmplienceProductImageUrlResolver

This builds the Amplience media URL for a specific Product. Not every product has media. In hybris products can have multiple levels of variants. Typically only one level or type of variant would have media.

The AmplienceProductImageUrlResolver uses an AmplienceProductResolver to decide, given a specific starting product, which is the actual product that should be displayed, and therefore should have product media in Amplience.

If no product can be found by the AmplienceProductResolver then a URL to the unknown image media is created.

If a product can be found then a URL to the product's media set is created. The media set identifier in Amplience is created by taking the Hybris product code and appending "-ms" to it. If this is not the desired behaviour then the getImageIdentifierForProduct() method will need to be overridden with the appropriate behaviour.

The media set URL is further constructed using SEO content from the product, which is obtained using the AmplienceSeoImageNameStrategy.

Finally as part of the URLs parameters the hybris session locals are appended. The hybris session language is converted into the format expected by Amplience using the AmplienceLocaleStringStrategy.

An example URL is given below:

http://i1.adis.ws/s/hybrisext/102277_lime-ms/Categories/Helmets/Helmets-Snow/Trace-Helm-lime.jpg?locale=en-GB,en-*,*

4.6.2. AmplienceProductResolver

This strategy is responsible for deciding which hybris product should have media in Amplience. This does not mean that the product actually has to have media in Amplience at that time, but that it ought to.

The default implementation provided in the amplienceedm extension does not know anything about the product model used and therefore assumes that every product has media in Amplience.

If you have variant products it is very likely that this default implementation will not be suitable, however intimate knowledge of your custom product model would be required. This is exactly the scenario we needed to solve when trying to support the Apparel product model.

4.6.3. ApparelAmplienceProductResolver

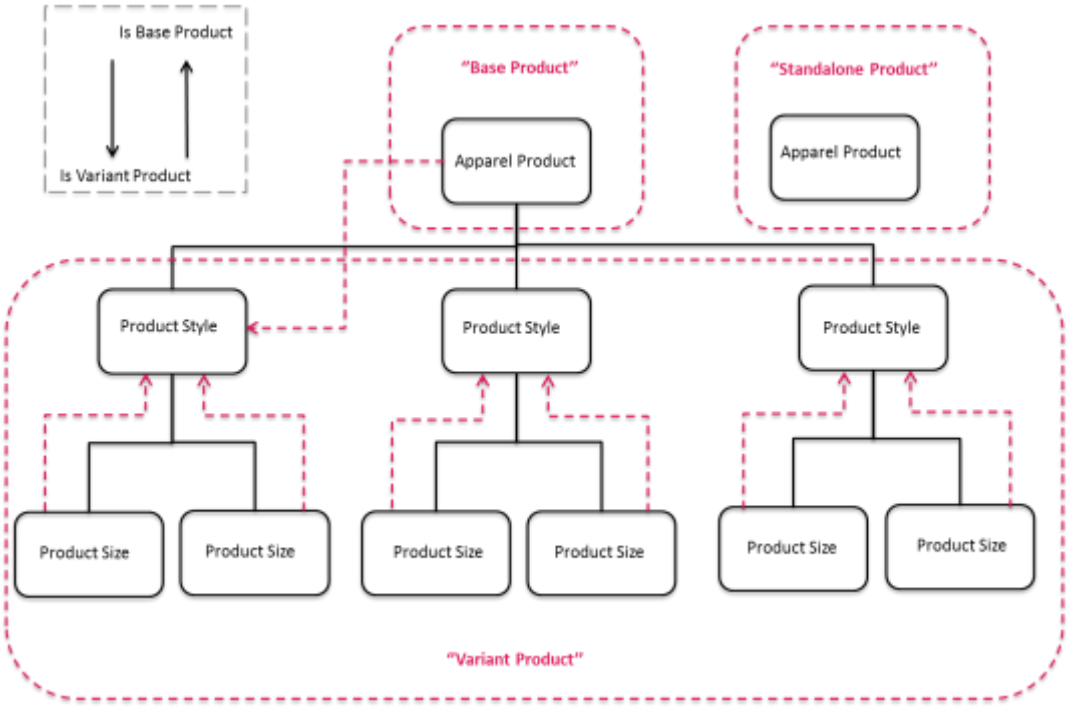
The solution is to replace the default AmplienceProductResolver. The ApparelAmplienceProductResolver is defined in the amplienceedmaddon extension and it provides support for the Apparel product data model.

The AmplienceProductImageUrlResolver does actually use the ApparelAmplienceProductResolver because the spring configuration in the ampliencedmaddon extension replaces the spring alias so that ApparelAmplienceProductResolver is loaded whenever the amplienceProductResolver alias name is used.

This Resolver will take in account the Apparel Store Demo Storefront product hierarchy and will return the appropriate product that should have media, which should be either ApparelStyleVariantProduct or an ApparelProduct with no variants.

The logic that this class follows is depicted in the following figure:

Figure 3. ApparelAmplienceProductResolver logic



On the figure above, the Demo Apparel Store product level hierarchy is depicted, with the top level Apparel Product, Product Styles and Product Size products.

In Hybris terminology a Base Product is the product above a Variant Product, and a Variant Product always has always a Base Product, so in general, an Apparel Product is a Base Product, a Product Style product is both a Base Product and a Variant product, and the Product Size product will only be a Variant Product (this logic depicted on the top left grey box).

However, the code defines the following (in pink text in figure above).

Variant Product is any product that is Variant of another product (either Product Style or Product Size product).

Base Product is the Apparel Product when the product has variants (e.g. a helmet with different colours).

Standalone Product is the Apparel Product when the product has no variants (e.g. a hoover).

The class returns always the image as per pink arrows in figure above, i.e.:

1. If the product is a Standalone Product it will return itself.
2. If the product is an Apparel Product, it will return the FIRST Product Style variant.
3. If the product is a Product Size product, it will return its Product Style product.
4. If the product is a Product Style Product, it will return itself.

It's unlikely that you will be using the exact same Apparel product model in your implementation and therefore it is very likely that you will need to provide your own implementation of AmplienceProductResolver.

4.6.4. AmplienceIdentifierSanitizer

This strategy is used to sanitise the Hybris product code used to build the URL to the Amplience media. The Hybris product code could contain any Unicode character sequence which may not be supported by Amplience.

The default implementation of this strategy takes the product code and performs the following:

1. The Unicode string is normalised into decomposed form. This splits characters with accents or other adornments into a sequence of multiple (decomposed) characters.
2. Replace specific common Unicode characters with ASCII equivalents.
3. Remove all non ASCII characters from the string.
4. Replace any characters that are not upper or lower case letters, or numbers, or dash (-) with an underscore (_) character.

If this approach to sanitising the product code is not suitable then provide your own implementation of this strategy.

4.6.5. AmplienceSeoImageNameStrategy

This strategy use used to get SEO text to include in the Amplience media URL. An example URL is shown below with the SEO text highlighted in bold.

http://i1.adis.ws/s/hybrisext/102277_lime-ms/**Categories/Helmets/Helmets-Snow/Trace-Helm-lime**.jpg?locale=en-GB,en-*,*

The SEO text is ignored by Amplience DM when processing the media URL, therefore the SEO text can be arbitrarily constructed so long as it remains a legal URL.

The default implementation of this strategy is to take the primary category path to the product and the product's name and compose them together into a URL path like structure. A product can be in many categories and the concept of a primary category is not something that is well defined within Hybris. The implementation takes the first category path from any root to the product – ignoring classification classes. This behaviour can be changed by overriding the getPrimaryCategoryForProduct() method or by replacing the whole strategy.

4.6.6. AmplienceProductSwatchUrlResolver

This media URL resolver is very similar to the AmplienceProductUrlResolver however this is used only in the contexts where a swatch image is desirable, i.e. to differentiate between variants of the same base product.

The implementation of this resolver checks to see if a product has an alternative swatch image, if it does it generates an Amplience DM URL to the swatch image, otherwise it delegates to the normal AmplienceProductUrlResolver implementation which will generate the normal media URL for the product.

The implementation detects that the product has an alternative swatch image by checking the product's *amplienceAltSwatch* attribute. If this has been set by the AmplienceProductImageStatusUpdateCronJob then there is an alternative swatch. The implementation assumes that the alternative swatch uses a naming convention of "<product code>-swatch".

4.7 Product Data Populators

4.7.1. AmplienceProductPrimaryImagePopulator

Populate the primary product image with the URLs to the Amplience product image. This populator is called as part of the conversion from a ProductModel to a ProductData.

Uses the AmplienceProductImageUrlResolver to create the URL to the image hosted in Amplience.

This populator is used for every product media except on PLP and the product variants (Swatch images) section on the PLP. The populator is configured with a list of hybris image formats and it generates a URL for each image format. The ProductData holds a list of images, and this implementation populates that list.

The AmplienceProductImageUrlResolver is called once to create the base URL. Each image format is added to the URL query string as an Amplience Transformation Template of the same name.

The image formats configured are:

- zoom
- product
- thumbnail
- cartIcon

An example URL for the zoom image format is below:

http://i1.adis.ws/s/hybrisext/102277_lime-ms/Categories/Helmets/Helmets-Snow/Trace-Helm-lime.jpg?locale=en-GB,en-*,*&$zoom$

4.7.2. AmplienceSearchResultVariantProductPopulator

This class is used in PLP and search pages to add product media URLs to the products displayed. It adds per product, ImageData to ProductData objects with their URLs populated.

Conceptually this is very similar to the AmplienceProductPrimaryImagePopulator above however this is called as part of the conversion of Solr search results into a list of ProductData objects.

As the PLP rendering doesn't query the hybris database but SOLR, this strategy populates ProductData image data from SearchResultValueData returned from SOLR using super.populate call to SearchResultProductPopulator.

4.7.3. ExamplePromoAmplienceSearchResultVariantProductPopulator

Search Result Populator that adds Amplience product image URLs which also adds contextual promotion roundels.

It extends AmplienceSearchResultVariantProductPopulator.getProductImageURLContextualQuery method which is set to null in the parent class.

This example displays 3 optional roundels on product image URLs. The 3 triggers are 'new', 'stock' and 'sale'.

The 'new' trigger is contrived as in this example all sample products are imported at the same time therefore we are just taking the product's PK mod 3 and using that. This means that about a third of the products will have this roundel.

The 'stock' trigger uses the lowStock condition in the product's StockData.

The 'sale' trigger looks at the categories a product is in to see if any of them are called 'sale'.

These 3 examples are contrived but show the potential functionality. For performance reasons the data for the trigger conditions should be indexed into SOLR and not require separate database lookups.

Returns a string with the extra parameters to be added to the Amplience URL so that three different roundels are displayed given a contrived made up example logic.

Any combination of the 3 conditions above is allowed. When the condition is true a query parameter is added to the URL, for example:

If 'new' is true then '&new=1' is appended to the image URL.

Otherwise if 'stock' is true then '&stock=1' is appended to the image URL.

Otherwise if 'sale' is true then '&stock=1' is appended to the image URL.

4.7.4. AmplienceEmailContextCatalogVersionPopulator

Populates the catalogue version data with the catalogue name and version.

Used as part of the product missing images report email generation.

4.7.5. AmplienceEmailContextProductPopulator

Populates the Amplience product data with the product SKU and name.

Used as part of the product missing images report email generation.

4.7.6. AmplienceProductDataPopulator

Populates the AmplienceProductData from a ProductData instance.

Used as part of the response to the ecommBridge.site.getProduct, (the AmplienceBridgeController.getProducts calls the DefaultAmplienceProductFacade which uses the amplienceProductDataConverter bean which in turn uses the AmplienceProductDataPopulator).

It sets the following fields in AmplienceProductData from ProductData

- **ProductID**: ProductData.code.
- **Sku**: ProductData.code.
- **Name**: ProductData.name.
- **Description**: ProductData.summary.
- **DescriptionLong**: ProductData.description.
- **URL**:ProductData.URL.
- **Image and thumbnails URLs:** ImageData.URL for 'product' and 'thumbnail' formats.
- **RatingValue** and **Ratings Image** set to null.
- **Price**:ProductData.PriceData.Value.
- **PriceFormatted**: ProductData.PriceData.formattedValue.
- **PriceCurrency**: ProductData.PriceData.currencyIso.
- PriceWas: null.
- **Availability**: ProductData.StockData.stockLevelStatus.
- **Custom (HashMap<String, String>)**:"rating", ProductData.averageRating.

4.7.7. ProductAmplienceMediaSetPopulator

Populates the product data with the Amplience media set name (adding '-ms' suffix).

It is used to construct the name of the media set, rather than the URL to the product image (which is done in the AmplienceProductImageUrlResolver). It populates the ProductData.amplienceMediaSet property with this value which is then used to configure the Render Kit PDP viewer and the standard PDP viewer – which need the name of the media set rather than a URL.

4.7.8. DefaultVariantOptionDataPopulator

Extends the VariantOptionDataPopulator to limit the variant options which are inherited from the base product.

It extends it via overriding the populate method that calls getVariantAttributeDescriptorModels new method rather than getVariantsService().getVariantAttributesForVariantType().

Main functionality setting VariantOptionData attributes:

- variantOptionQualifiers
- code
- URL
- stock
- priceData

4.7.9. AmplienceVariantOptionDataPopulator

This is the Amplience specific Variant Option Data Converter that populates ImageData parameters into VariantOptionData and populates swatch image URL in the PDP.

This populator is configured in spring to map specific variant attributes to specific image formats. The image formats in turn are mapped to a UrlResolver.

For example the ApparelStyleVariantProduct.style variant attribute is mapped to the 'styleSwatch' image formant, and in turn that image format is mapped to the amplienceProductSwatchUrlResolver.

There is a default fallback configured to use the generic amplienceProductImageUrlResolver.

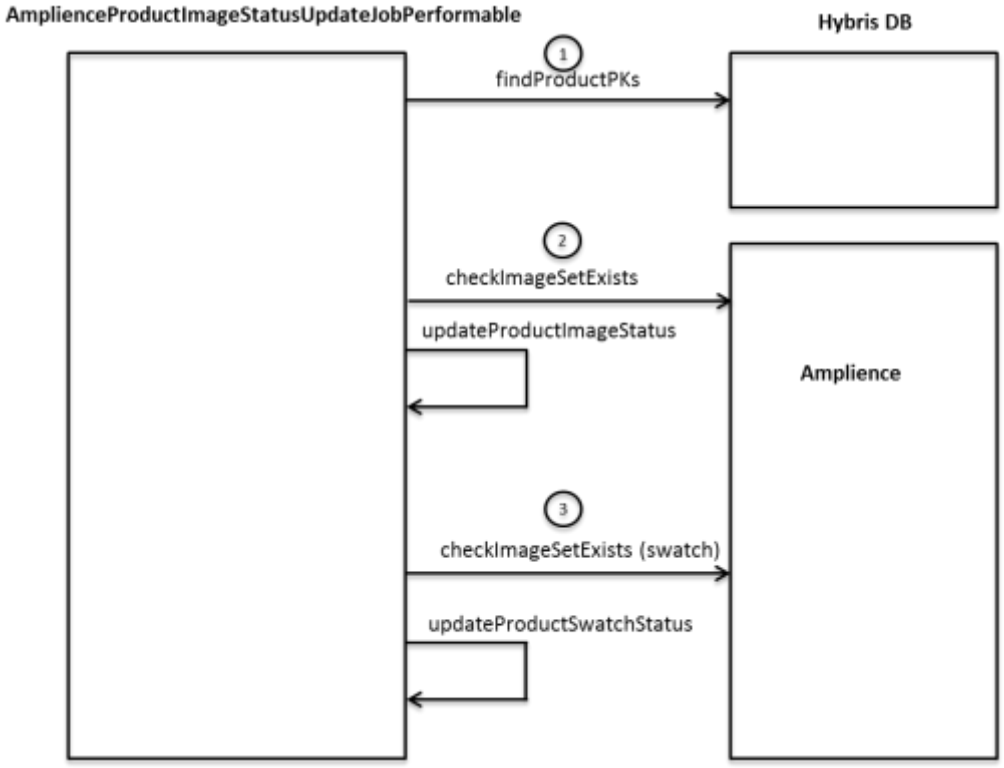
4.8 Amplience Jobs

4.8.1. AmplienceProductImageStatusUpdateCronJob

The AmplienceProductImageStatusUpdateCronJob executes the AmplienceProductImageStatusUpdateJobPerformable code when it runs.

The purpose of this job is to check if a product has a media set in Amplience, i.e. will Amplience display a product image or the fallback missing product image.

Figure 4. AmplienceProductImageStatusUpdateJobPerformable tasks



The job performs the following tasks:

- Runs a flexible search query to find the hybris products that need to be checked.
- For each product:
    - Builds the URL to retrieve the metadata from the Amplience media set for that product. This uses the same naming convention strategies as used by the storefront but uses a different resolver: amplienceProductImageMetadataUrlResolver.
    - Requests the metadata URL.
    - Based on the response status code updates the Product's amplienceImageStatus field as follows:

Response Status	Amplience Image Status
200	FOUND
404	MISSING

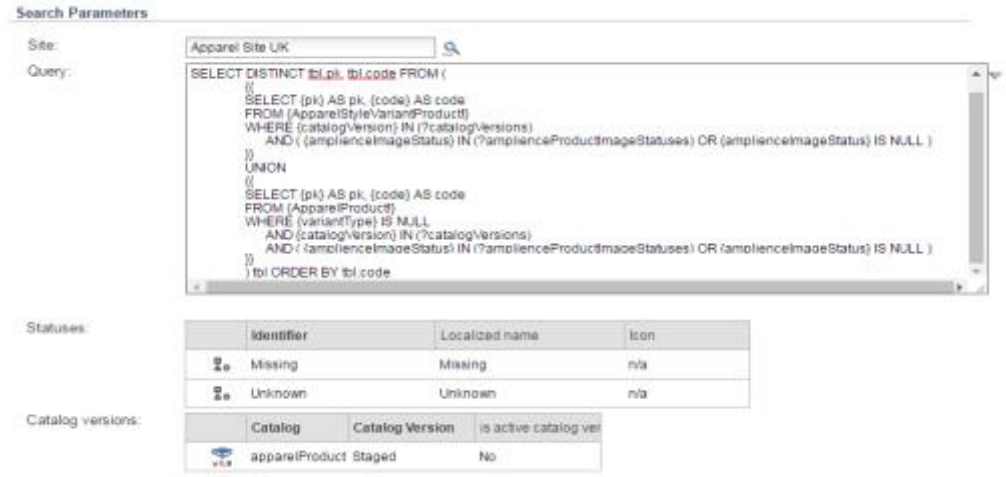
- For each product that is marked with status FOUND
    - Build the URL to reteive the metadata from the Amplience image that is the alternative swatch image for the product. This uses the amplienceProductSwatchMetadataUrlResolver.
    - Request the alternative swatch image metadata URL.
    - If the metadata is found update the Product's amplienceAltSwatch field to TRUE.

The metadata for the media set is retrieved because, unlike the image data, Amplience DM does not apply a fallback to the request. If the media set does not exist then the caller will receive a 404 error response when requesting the metadata.

It is expected that only a subset of the products should have media in Amplience, therefore it is necessary to configure the cron job with the query to run to find the products to check.

For the Apparel product model the job is configured as follows:

Figure 5. AmplienceProductImageStatusUpdateCronJob config in HMC



The flexible search query is parameterised with the Statuses and Catalog Versions set on the cron job.

The query finds all products of type ApparelStyleVariantProduct or ApparelProduct (with a Null variantType) which have an amplienceImageStatus or Null, Missing or Unknown.

In general, where products do have their media uploaded to Amplience, the job should only check new products that have been added since the last execution or products which are currently missing their Amplience media.

Setting the Product's amplienceImageStatus field makes it easy to report on.

4.8.2. AmplienceProductImageEmailReportCronJob

The AmplienceProductImageEmailReportCronJob executes the AmplienceProductImageEmailReportJobPerformable code when it runs.

The purpose of this job is to send an email report of the hybris products which do not have media in Amplience. The list of products is determined by a flexible search query – typically this would use the Product's amplienceImageStatus field.

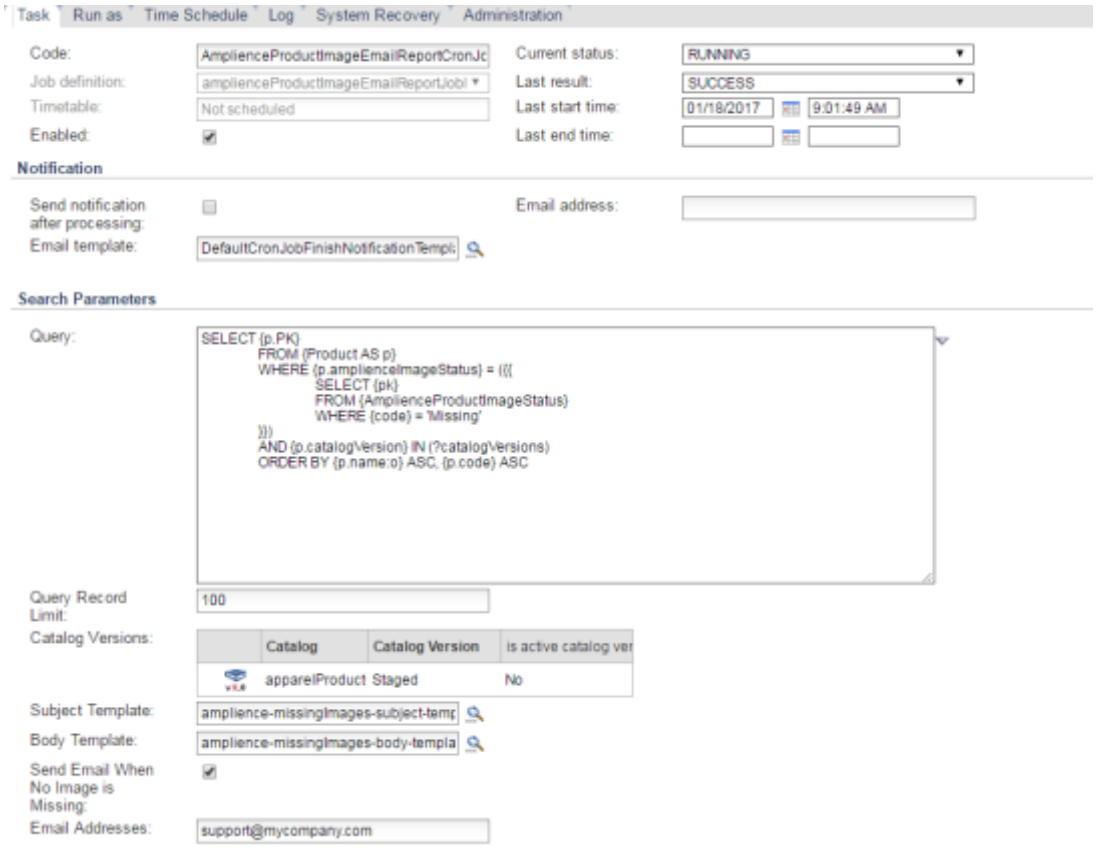
The result of the query is checked and there is the option to skip sending the report if no products have been found which are missing their Amplience media.

If the email report is sent then the subject line and body of the email are generated using reporting velocity templates.

This job would send an email to a configurable email address (or comma separated email addresses) when there are images missing in Amplience.

For the Apparel product model the job is configured as follows:

Figure 6. AmplienceProductImageEmailReportCronJob in HMC



The job queries Hybris DB for products that have their amplienceImageStatus set to MISSING, therefore this job needs to be scheduled to run after the AmplienceProductImageStatusUpdateCronJob.

The cron job is setup with render templates configured with the velocity templates (both of them of type RendererTemplate) that are in: amplicencedm/resources/amplicencedm/emails/email-missingProductImages-body.vm. The render template has a string attribute called TemplateScript that holds the velocity template script to execute to generate the subject or bodyThe templates are executed with a context object that provides the 'data' to be rendered into the template.

For AmplienceProductImageEmailReportCronJob the context object is AmplienceProductImageEmailReportData.which has the following properties: catalogVersions, missingImageCount, products. From these the velocity template can render the content required for the emails.

5. CMS COMPONENTS

The following CMS components are also defined in the amplicencedm extension. There are more details on how these can be configured and used in the Business Guide.

- Amplience Page Script Component
- Amplience Responsive Image Component.
- Amplience Image Carousel Component.
- Amplience UGC Carousel Component.
- Amplience UGC Media Wall Component.

All these CMS Components use the default Accelerator rendering method where the component's attributes are copied directly into the request attributes and then rendered via a JSP include. This means that there is a separate JSP for each component and that during execution of the JSP the component's attributes are in scope as named attributes of the request.

The JSP used to render the components are found in the amplicencedmaddon extension in the acceleratoraddon/web/webroot/WEB-INF/views/desktop/cms folder. The JSPs have the same name as the CMS Component type but with the name lowercased.

Note: CMS component java classes are model classes used to store all the CMS attributes and are in the amplicencedm extension. JSP files resulting from the CMS Component creation will be hosted on the amplienceaddone extension and binded by the AmplienceAddonCMSComponentRenderer. JSPs will be covered on in this document inside the amplicencedmaddon section.

5.1.1. AmplienceResponsiveImageComponent

Component used to provide an Amplience responsive Point of Interest image.

The component has the following attributes:

Attribute	Description
imageName	The Amplience image identifier. This is unique within your Amplience account.
params	Extra transformation parameters to add to the request URL.
title	The title of the image. Used as a hover tip.
alt	The alternate text to used when the image is not available.

5.1.2. AmplienceImageCarouselComponent

Component used to display an image carousel backed by an Amplience media set.

The component has the following attributes:

Attribute	Description
setName	The Amplience media set identifier. This is unique within your Amplience account.

5.1.3. AmplienceUGCCarouselComponent

Component used to display an Amplience User Generated Content stream as a carousel of images.

The component has the following attributes:

Attribute	Description
streamID	The Amplience UGC stream identifier.
hashtag	
tags	
title	The title to display.
callToAction	The call to action to display on the carousel images
autoPlay	Automatically start rotating the carousel
numberOfSlides	The number of images to show in the carousel. Defaulted to 6.
enableModal	Enable the popup showing image details on click. Defaulted to true.
showModalText	Defaulted to true.

5.1.4. AmplienceUGCMediaWallComponent

Component used to display an Amplience User Generated Content stream as a wall of tiled images.

The component has the following attributes:

Attribute	Description
streamID	The Amplience UGC stream identifier.
hashtag	
tags	
wallLayout	Layout is either “Standard” or “Hero”. Defaulted to “Hero”.
title	The title to display.
callToAction	The call to action to display on the carousel images
enableModal	Enable the popup showing image details on click. Defaulted to true.
showModalText	Defaulted to true.

5.2 AmplienceAddOnCMSComponentRenderer

The DefaultAddOnCMSComponentRenderer assumes that a cms component will be rendered by the same extension that defined the component type. In this case the cms component types are defined in the amplifiencedm extension but the renderer is the amplifiencedmaddon.

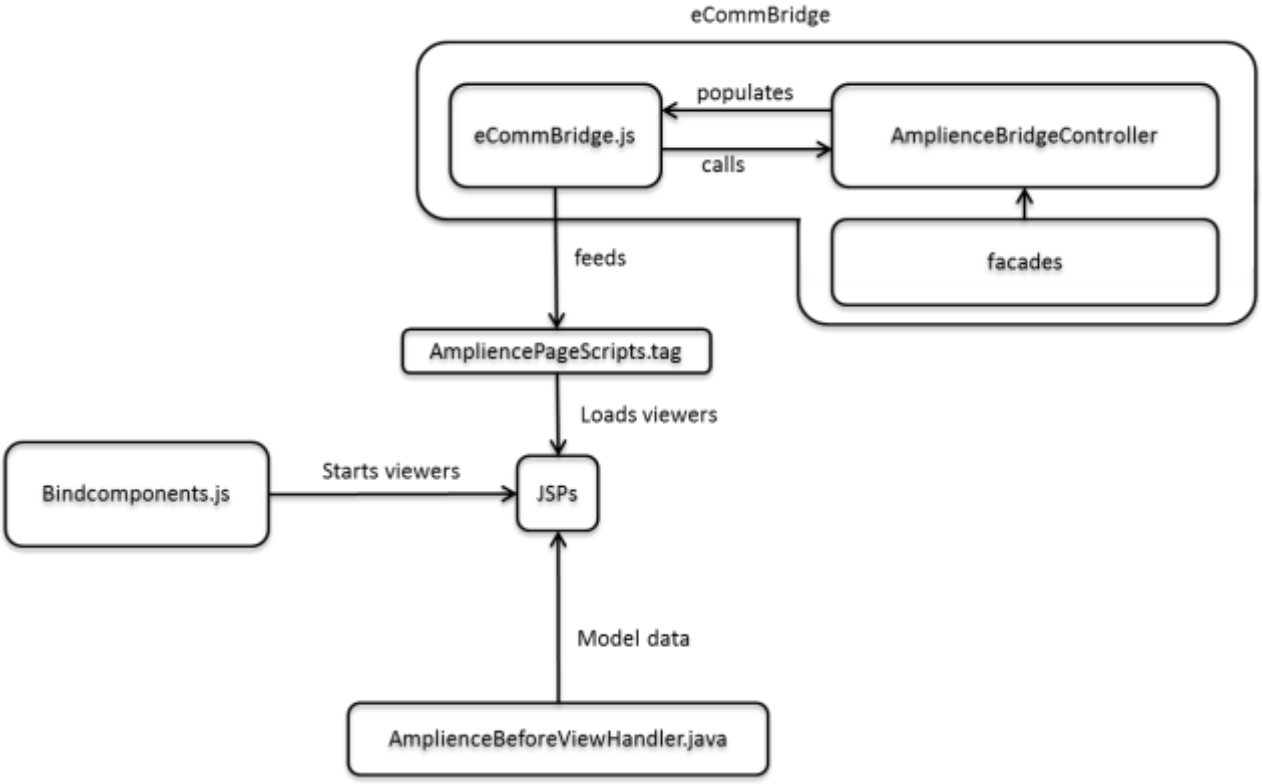
The DefaultAddOnCMSComponentRenderer exposes all the component's attributes to the renderer template, however it does not expose the cms component itself to the template. The component is added to the list of variables to expose to the template here.

6. STOREFRONT CHANGES

The amplifiencedmaddon has four main areas of functionality to support Amplience viewers

1. -A BeforeViewHandler that adds to the model Amplience specific data.
2. -The eCommBridge, which sets some javascript objects and exposes some operations which can be used by the Amplience Viewers. The eCommBridge has a javascript frontend (ecommBridge.js) and some backend code (AmplienceBridgecController and facades) that populate the javascript variables from the Hybris Backend and allow operations to be called by the eCommBridge.
3. -AmpliencePageScript.tag, which will set all the necessary javascript files for the Amplience Viewers.
4. -Bindcomponents.js, which starts all the viewer’s javascript files.
5. -JSPs and tags that support Amplience Viewers (for storefront and CMS pages).

Figure 7. Amplifiencedmaddon main components



6.1 AmplienceBeforeViewHandler

Loads model with different context values. It is called before any page (view) is rendered.

Adds the following the model:

- External URL resources to the lists of CSS and JS resources.
- Amplience account configuration.
- The session currency in format expected by the Amplience ecomm bridge
- The session local (including fallbacks) in Amplience format.
- The category code and SEO path if a category page is being viewed.
- The Instagram authentication callback URL.
- The customer ID from the current order or current session.

It also changes the view used to render the quickViewPopup to one provided by the Amplience addon on (ampliencedmaddon/fragments/product/quickViewPopup.jsp).

6.2 eCommBridge

6.2.1. AmplienceBridgeController

This is the controller that handles requests from the Amplience ecommBridge. It uses the AmplienceProductFacade, amplienceResourceUrlFacade and the hybris Platform CartFacade.

The methods in the controller map directly to the lookup methods in the Amplience's ecommBridge javascript:

- **getProducts** - Returns a list of AmplienceProductData in json or a list of product codes at a "/misc/amplience/product" ajax request from the ecommBridge, using the AmplienceProductFacade.getProductsByCode method.

- **getResourceUrl** – Gets Amplience resource data for specified resource given using the amplienceResourceUrlFacade.getResourceUrl method.
- **addToBasket** – Supports adding multiple products to the basket in a single request.

6.2.2. InstagramCallbackController

This is the controller handles requests made to the Instagram authentication callback URL. The only method looks forwards on to the "addon:/ampliencedmaddon/pages/misc/instagramAuth" view.

6.2.3. AmplienceProductFacade

Gets list of AmplienceProductData for specified list of products.

Gets the ProductData for each product code and then converts the ProductData into AmplienceProductData using the injected amplienceProductDataConverter.

6.2.4. AmplienceResourceUrlFacade

Gets the Amplience resource data for a specific type of resource.

The getResourceUrl method gets a page URL based on type and param.

Type is one of the following: ‘’category’’,’’ product’’, ‘’standalone-page’’ and ‘’search’’ param will be the categoryCode, productCode, pageUid, queryText for each of the above.

When there is no param, type is one of “home”, "basket", "checkout", "stores", "my-account", “checkout-confirmation” or “wish-list”.

An exception will be triggered if type is not any of the above.

6.2.5. AmplienceLocaleStringStrategy

Gets the current Amplience locale string for the current hybris site and language settings.

The Amplience locale string is a list of locales which is built from the Hybris main language and fallback languages.

The locale string is cached per site & language so that it does not need to be regenerated for each request.

Some mapping is required between hybris locales and what Amplience expects.

- Replaces any ‘_’ with ‘-‘
- Adds to any two character string a ‘-*
- Adds a suffix on the string list of ‘,*’

For example if the hybris locale is ‘en_GB’ the result would be en-GB,en-*,*.

6.2.6. CurrencyFormatSupplier

It creates a currency DecimalFormat for the current currency and locale.

6.2.7. ecommBridge.js

The ecommBridge javascript defines the following objects:

accountConfig. Object that would hold the Amplience configuration.

capability. This object will indicate to Viewers the capabilities supported. It has been set as:

- quickView: true
- getProduct: true
- URL: true

- wishList: false
- transactional: true

user: where all the Hybris user data will be held.

site: site specific information and operations. It holds three variables and three methods:

- Variables: page, locale, currency.
- Methods:
    - getURL: gets an Amplience URL given a type and a parameter. It triggers the AmplienceBridgeController.getResourceUrl method
    - getProduct. Gets Amplience product data. It triggers the AmplienceBridgeController. getProducts method
    - bind.: Not implemented

interaction: This object holds methods that, when called, make AJAX calls to the AmplienceBridegeController.

- quickView: Launches a QuickView for the specified product
- addToBasket: Add products to the Basket. Not used in the Apparel Demo storefront.
- addtoWishlist. Add products to the WishList.. Not supported functionality by default as there is no wishlist functionality on the Apparel Demo Storefront. To enable implementing this method the ecommBridge.capability.wishList (above) variable needs to be set to true and a new method implemented in AmplienceBridgeController.

6.3 AmpliencePageScript.tag

It performs three tasks:

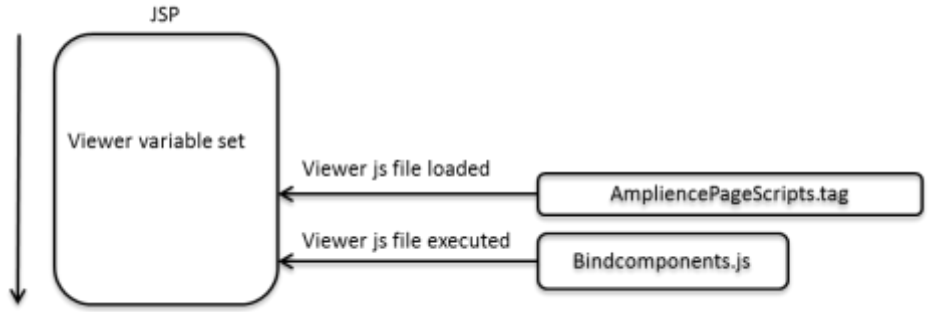
1. It populates the dynamic elements of the ecommBridge javascript on every page:
    - The account config
    - The locale
    - The currency
    - The current page type & page specific data like product media set or order number.
2. It loads Amplience Javascripts resources that require the ecommBridge values above to be setup before they can be loaded:
    - Carousel Scripts.
    - Standard PDP Viewer Scripts.
    - UGC Carousel & MediaWall Viewer Scripts.
    - Render Kit Viewer Scripts.
3. It populates the Amplience analytics tracking on the order confirmation page with the following values for each order entry:
    - orderEntry.totalPrice.Value
    - orderEntry.product.code
    - orderEntry.quantity
    - orderEntry.totalPrice.currencyIso
    - orderData.code

6.4 bindComponents.js

This Javascript file performs late binding between HTML elements on the page and the Amplience Javascript which will render them. This script is loaded at end of the page and it finds which have specific classes set which indicates which Javascript functionality they should be bound to.

This approach is necessary because the Accelerator tries to defer all Javscript loading to the end of the page. This means that at the point where a Viewer is being used in the HTML the Javascript behaviours are not yet available.

Figure 8. Viewer load sequence



6.4.1. CMS Components JSPs

7. THE LIST AVAILABLE COMPONENTS AND HOW DO THEY PRESENT THE AMPLIENCEDM EXTENSION (SEE CMS COMPONENTS CMS COMPONENTS

The following CMS components are also defined in the amplicencedm extension. There are more details on how these can be configured and used in the Business Guide.

).

AmplienceResponsiveImageComponent.jsp

Retrieves the Model fields (imageName-as ‘code’, params, title, alt) to feed responsiveImage.tag.

responsiveImage.tag

This tag constructs the URL to an Amplience media and builds a responsive image given some different media queries on min-width: 1400px, 1024px, 800px, 640px, 480px, 320px).

This is different to the rest of the CMS components JSPs in the fact that it doesn’t need any js file for any viewer loaded.

The example page built in CMS that uses this component is https://localhost:9002/yacceleratorstorefront/en/amp-poi

AmplienceImageCarouselComponent.jsp

This JSP configures the image carousel Javascript viewer to display a media set as a carousel.

The example page built in CMS that uses this component is https://localhost:9002/yacceleratorstorefront/en/amp-carousel.

AmplienceUGCCarouselComponent.jsp

This JSP configures the Amplience UGC carousel component.

The page built in CMS that uses this component is https://localhost:9002/yacceleratorstorefront/en/amp-ugc-carousel

AmplienceUGCMediaWallComponent.jsp

This JSP configures the Amplience UGC media wall of image components.

The page built in CMS that uses this component is https://localhost:9002/yacceleratorstorefront/en/amp-ugc-mediawall

7.1.1. StoreFront Pages and JSP Tag files.

There are some pages created inside this extension for demonstration purposes and some tags that are inserted in existing pages. These are:

staticContentLayoutPage.jsp

This a demo Static Content Page to show usage of amplience:image and amplience:responsiveImage tags. Its URL is https://localhost:9002/yacceleratorstorefront/en/amp-static.

It uses amplience:image (image.tag) and amplience:responsiveImage (responsiveImage.tag) with fixed parameters and displays a normal image and a responsive (to be exact, 'art direction' responsive) to show the difference between the two (please resize the browser to see the difference in behaviour between the two images).

Bear in mind that the AmplienceResponsiveImageComponent also uses responsiveImage.tag,but it uses it with parameters that are set at Component creation on the CMS.

image.tag

This is a very simple tag that is used for staticContentLayoutPage.jsp that builds an Amplience image URL based on an image ('womanwall' in this specific static case) using several resolution images inside a srcset tag for responsiveness.

productLayout1Page.jsp

This jsp is used to replace the storefront view for *ProductDetailsPageTemplate* only in the UK site.

It is being created to show how to add the Amplience Standard PDP viewer in PDP pages. It includes productDetailsPanel.tag which in turn includes standardPdpViewer.tag.

standardPdpViewer.tag

This tag sets the viewer configuration values onto the element which is used as the placeholder for the viewer in the DOM. The bindComponents.js file will find the placholder element and load the viewer script passing it the correct configuration.

Please note the configured values are used by the PDP viewer and can be changed to modify how PDP is displayed. Contact Amplience for more information.

It sets enableAmplienceStandardPdpViewerJavascript variable so that in amplienceScript.tag the standard PDP javascript is loaded.

productLayout2Page.jsp

This jsp is used to replace the storefront view for *ProductDetailsPageTemplate* only in the German site

It is being created to show how to add the Amplience Render Kit PDP viewer in PDP pages. It includes productDetailsPanelRenderKit.tag which in turn includes renderKitPdpViewer.tag.

renderKitPdpViewer.tag

This tag is similar to the standardPdpViewer.tag but uses Amplienc's more modern Render Kit framework. It sets the viewer configuration values onto the element which is used as the placeholder for the viewer in the DOM. The bindComponents.js file will find the placholder element and load the viewer script passing it the correct configuration.viewer:

It sets enableAmplienceRenderKitJavascript variable so that in amplienceePageScript.tag the Render Kit PDP javascript is loaded.

quickViewPopup.jsp

This is used to render the product quick view popup. This view is substituted in the AmplienceBeforeViewHandler.

This is done so that the product quick view popup uses also the Render Kit viewer (renderKitPdpViewer.tag), which is included in the jsp.

7.2 Product image import into Amplience

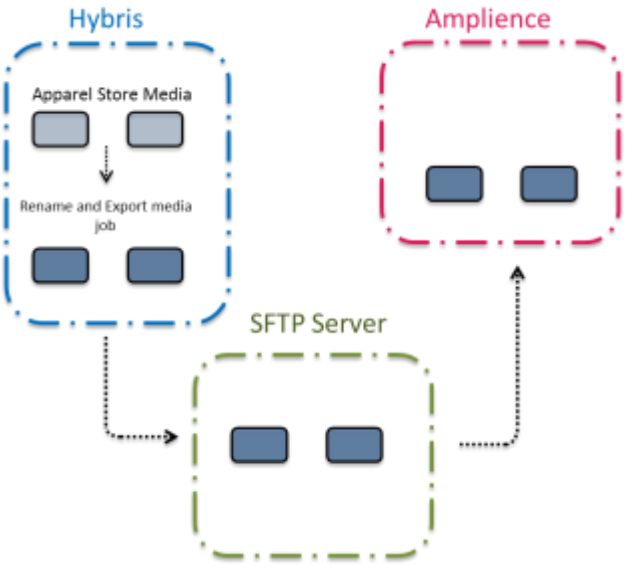
To be able to build URLs in Hybris pointing to a media asset in Amplience, a naming strategy needs to be followed. Amplience has a URL structure that requires the identifier for the media asset to be in a specific place in the URL.eg http://i1.adis.ws/i/playground/**womanonstep**.jpg?w=250&h=250&sm=TL for images or http://i1.adis.ws/s/playground/**bag_spin**?w=150&h=150 for media sets.

The naming convention that we want to use is to name the Amplience assets with the product code. In the Apparel Demo Store product media is not named according to the product it belongs to. This is why the media needs to be exported from hybris using the correct naming convention.

Typically a hybris system will have several copies of an image at different resolutions. For export to Amplience only the highest quality source images are required there is no advantage in exporting lower resolution versions of the images.

They typical way to load products into Amplience is to publish them on a SFTP site where Amplience will collect them and process them with an import and ingest script. If required Amplience may run the SFTP server for you or if you already have an SFTP service Amplience can connect to it.

Figure 9. Apparel Demo Store media export to Amplience



7.2.1. ExportImages

This is the main class that handles the export of the Hybris product media.

Finds products that should have their product images exported (see below for logic) and takes the media containers from the Picture and Gallery Images (videos and swatch images are not present in the Demo Apparel Store data).

The media containers are exported in order – replicating the Gallery order of the Accelerator StoreFront. The export is configured with a prioritised list of ImageFormats. Each media container is search for a media in each specified ImageFormat in turn. When found it is exported and the process moves on to the next container. This allows the existing ImageFormats to be ranked and the export will take the highest resolution format that it can find for each container.

The exported images are written to the local disk of the hybris server running the export. It is expected that this process will be used as part of initial data setup or as part of a migration exercise rather than as a frequent operational process.

The ExportImages class is triggered from the exportImages.groovy script found in ampliencedmaddon/resources/ampliencedmaddon/script.

exportImages.groovy

This script is run to trigger the export of images from hybris.

The script can be found in ampliencedmaddon/resources/ampliencedmaddon/script.

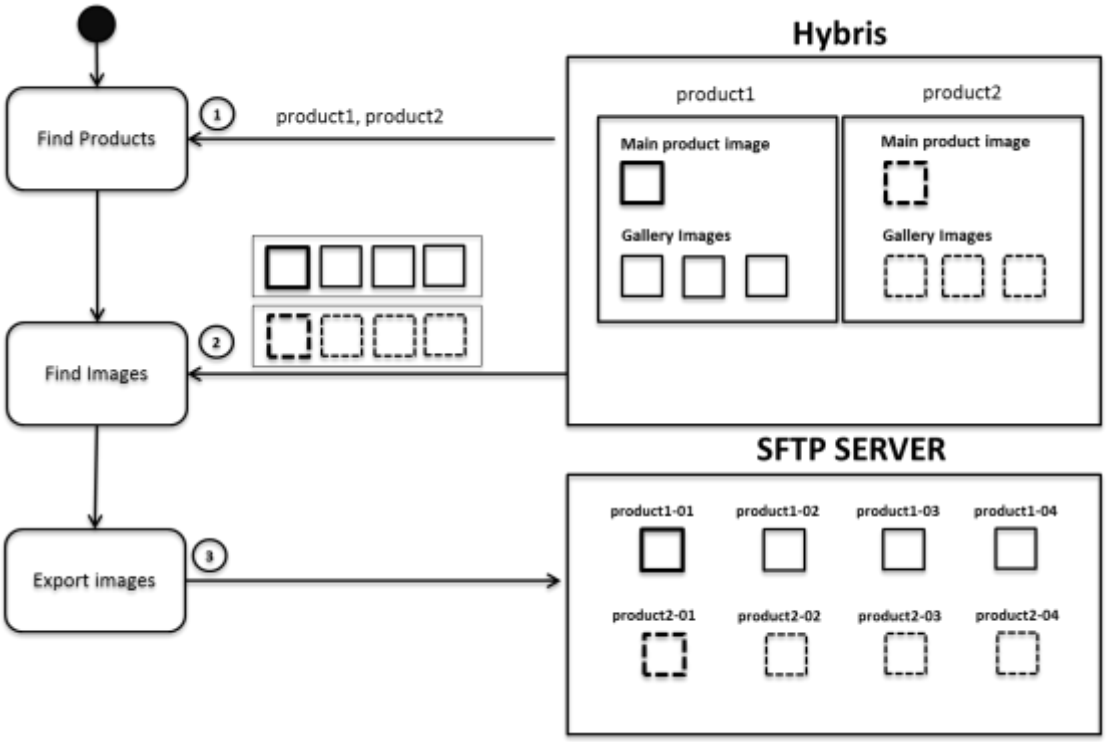
It sets the following configuration properties:

- **catalogCode** – the product catalog to export, e.g.'apparelProductCatalog'
- **catalogVersion** the product catalog version to export, e.g. 'Online'
- **imageFormats** – the prioritised list of ImageFormats, e.g. ["superZoom", "zoom", "product"]. The first image format matched will be exported.
- **query** – the query to use to find the products to export. This query is specific to the product data model. For the Apparel product model the query is:

```
SELECT DISTINCT tbl.pk, tbl.code
FROM (
        {{ SELECT {pk} AS pk, {code} AS code
        FROM {ApparelStyleVariantProduct!}
        WHERE {catalogVersion} = ?catalogVersion }}
UNION
        {{ SELECT {pk} AS pk, {code} AS code
        FROM {ApparelProduct!}
        WHERE {catalogVersion} = ?catalogVersion AND {variantType} IS NULL }}
) tbl
ORDER BY tbl.code
```

And triggers the ExportImages.doExport method.

Figure 10. Export Images conceptual diagram



Note: There is no error reporting within the ExportImages code. It will throw an exception if there is a fatal error.

The simplest way to run the export is to find the exportImages.groovy script and use the hybris HAC script console to execute it. Simply paste the contents of the script into the script console, update the script contents to use appropriate configuration parameters and execute the script.

7.2.2. DeleteImages

After you have exported your product media to Amplience (see above) you no longer need to have the product media in hybris. This class contains the code to remove all the product media for a catalog.

Given a catalogue and version will find all products in that catalogue version, and then for each product will find the medias and media containers referenced in the following product attributes: Picture, Thumbnail, Normal, Thumbnails, Detail, Logo, GalleryImages. The medias will then be deleted.

There is a groovy script called deleteImages.groovy found in amplicencedmaddon/resources/amplicencedmaddon/script which will trigger the deletion of product images. The script sets the product catalog and version and triggers the deletion of product images. Again this script can easily be run through the hybris HAC scripting console, however remember to enable commit mode when running this script otherwise it won't delete anything.

8. AMPLIENCEDMBACKOFFICE EXTENSION

This extension enables embedding of Amplience business tooling into the hybris backoffice tool. It does this by adding support for an Iframe widget in the hybris backoffice.

One Iframe is setup to embed the Amplience UGC business tooling into the backoffice. It is performed by the IFrameController, which builds the iframe with the URL set up in the extension project.properties file (see configuration options section above).

This extension also provides configuration for the hybris Backoffice Admin Area. This provides the basic configuration for the enhancements to the hybris data model made in the amplicencedm extension.

9. AMPLIENCEDMCOCKPITS EXTENSION

This extension customises the Hybris product cockpit to load the product images from Amplience using Amplience DM media URLs.

If the hybris instance no longer contains the product images then the default hybris product cockpit would not be able to show any product images.

9.1 AmplienceEditorAreaControllerImpl

This is the Amplience specific version of the product cockpit's EditorAreaControllerImpl that uses the product's AmplienceThumbnailURL to get the preview image URL to display.

9.2 AmplienceThumbnailURLAttributeHandler

DynamicAttributeHandler for the Product.amplienceThumbnailURL attribute

This handler is read-only and generates a fully qualified URL to a preview thumbnail image for the product. This needs to be done as an item attribute because the product cockpit viewers that shown the product preview thumbnails consume the URL from an item attribute.

All the configuration files that use this attribute are in amplicencedmcockpits/resources/amplicencedmcockpits-config/cockpitgroup folder.

10. HOW TO ADAPT THE EXTENSION TO YOUR NEEDS

This section contains common scenarios that you are likely to face in an implementation. These sections cover topics that you should consider as well as pointers to which services or strategies you will need to customise.

10.1 How to adapt the extension to your model

The amplicencedm extension makes the assumption that every hybris Product will have product images. The amplicencedmaddon replaced this behaviour with support for the Apparel 3 tier product / variant hierarchy.

It's very likely that your product data model will not match exactly either of these 2 approaches and you will need to implement your own AmplienceProductResolver strategy.

When implementing the AmplienceProductResolver strategy you need clearly understand which hybris product type will logically have a product image, and how to get from any other type of product or variant to the nearest or most relevant product that does have an image.

You also need to look at the query used as part of the AmplienceProductImageStatusUpdateCronJob and the query passed into the ExportImages class as both of these queries should only select products which should have product images.

10.2 How to add a swatch image to a product

In the hybris Apparel Demo store swatch images are simply resized versions of the main product image. The amplicencedm extensions add support for an alternative image to be used as the swatch image.

In order use an alternative swatch image for a product you need to create the image and upload it to Amplience along with the other images for the product. If you are using the standard Amplience ingest script then the alternative swatch image should be named **<product code>-swatch.jpg**.

After having uploaded the product images to Amplience via an SFTP server and allowing Amplience time to import the product images they will appear in your Amplience account. The Amplience ingest script can publish the imported product images automatically or it can leave them to you to publish manually – contact Amplience for details. If you need to manually publish the product images then do so.

Once the product images are published run the AmplienceProductImageStatusUpdateCronJob so that hybris will notice that the product in question has an alternative swatch image. If the AmplienceProductImageStatusUpdateCronJob is run on a schedule you may choose to wait until its next run rather than triggering it manually.

The AmplienceProductImageStatusUpdateCronJob will update the Product.amplienceAltSwatch attribute on the Staged version of the hybris product so you will need to publish the product from Staged to Online.

Now when you go to the PDP page for the product the alternative swatch image will be shown instead of a small version of the main product image.

10.3 Videos

The Amplience media set used for the product gallery can contain videos. The Amplience viewers used on the product details page to display the product gallery support displaying and playing videos.

This means that if you have a video for a product then you need to upload the video to Amplience and put it into the media set for you product. Publish your changes in Amplience. Visit the PDP page for the product. Scroll through the product gallery and you will see the video at the relevant position.

Note you should not put a video as the first item in the product's media set.

10.4 Spin Sets

Just like Videos the same is true for 360 spin sets and 720 spin sets. Simply create the spin set in Amplience, add it to the product's media set, publish it to online, and visit the PDP page for the product. The Amplience viewer should pick up on the fact that the gallery now had a spin set.

10.5 Create a new roundel

The magic behind the product roundels is all in the Amplience transformation template called 'roundel'. If you need a different behaviour then you need to review the layer options that are document in the Amplience playground to come up with the effect that you want to achieve.

The behaviour of the Amplience extensions is to try to put all the logic into the transformation template and then use custom query string parameters to drive the actual behaviour. This provides a loose coupling between the 2 systems. Once the hybris system is creating the appropriate parameters then most changes can be made directly in Amplience.

The ExamplePromoAmplienceSearchResultVariantProductPopulator outputs up to 3 boolean parameters in the query string. These are then used in the roundel transformation template to trigger the actual behaviour.

Just by changing the transformation template you can change the actual images that are shown in the roundels, the size of the roundels, which roundel is shown in which corner and the exact positioning of the roundel relative to the edges of the main image.

You could change the parameters so that rather than passing simple booleans you pass more complex data or even pass in the name of an image to display as the roundel. There is huge scope to play with when working with image layers.

Remember that the AmplienceSearchResultVariantProductPopulator is run with the data returned from the Solr search and needs to be quick as it is run for each product shown during rendering of a lister page. To maintain acceptable performance the AmplienceSearchResultVariantProductPopulator should not cause the product to be loaded from the database.

10.6 Create a new image size

The sizes of specific images are controlled by the Amplience transformation template. The transformation template is named after the hybris ImageFormat that is used. The product lister pages use the ImageFormat named 'product' therefore to change the size of this image all you need to do is to modify the 'product' transformation template in Amplience and publish it.

If you want to create a new image size rather than changing one of the existing ones then there are quite a few steps to go through.

You would need to create new ImageFormat in your hybris solution and decide in which scenarios (i.e. pages) you would want to use this format. For example let's say that you wanted to create a new ImageFormat named 'hero'.

To create a new hybris ImageFormat you would need to modify the ImageFormat enum type in the items.xml file in one of your project's hybris extensions.

Then, in your Amplience account, create a transformation template also called 'hero' and set it up with the parameters you want to control the image size and how the image is displayed.

Then, depending on where you would want to use this new image format in the storefront you would need to modify either the configuration of the any of these spring beans amplienceProductPrimaryImagePopulator, amplienceSearchResultVariantProductPopulator, or amplienceVariantOptionDataPopulator:

If you want to use the new image format in pages that don't currently use one of the above populators then it's likely that you don't already have product data on that page and you will need to setup a specific converter / populator spring configuration for your particular use case.

In the JSP or TAG file where you want to display the new image format you would need to use the accelerator EL function 'productImage' to lookup the relevant ImageData from the ProductData, for example:

<c:set value="${ycommerce:productImage(product, 'hero')}" var="heroImage"/>

**

The URL generated for the image tag will use the $hero$ query parameter to activate the hero transformation template triggering the appropriate sizing and display parameters to be applied.

10.7 Change your SEO strategy

When the Amplience media URLs are built using the UrlResolvers in the amplience dm extension (e.g. AmplienceProductImageUrlResolver or AmplienceProductSwatchUrlResolver) there is a SEO element to the URL which is shown in an example below as the emboldened portion.

https://i1.adis.ws/s/Hybrisext/102277_lime-ms/**Categories/Helmets/Helmets-Snow/Trace-Helm-lime**.jpg?locale=en-GB,en-*,*&$product$

This SEO text is built by the AmplienceSeoImageNameStrategy given a specific product. The default implementation is to use the category path and the product name in the current language. You can provide your own implementation of this interface by overriding the spring bean definition. Your own implementation can return any value that can be derived from the product itself and can be customised to suite your own requirements.