styla extension for hybris commerce

Version: 1.0

Last modified: 02/12/2016

Contents

Styla integration for hybris

A full integration of styla with your online shop consists of:

- adding a content page to your online shop where the styla content is rendered
- adding styla's javascript and stylesheet definition to the content page that renders the styla content
- provide endpoints which respond to requests coming from styla in order to expose information about your category structure, productimages in a category and detailed product information for a given product code
- integrate with styla's SEO endpoint, which returns for requested styla content information that is relevant for search engine optimization
- provide a javascript function or endpoint for adding products to the user's cart when called from a magazine page

Overview

The styla integration for hybris consists of two extensions, which simplify the integration of styla with your online shop. The extensions use hybris' Service Layer API which lets you easily implement the interfaces on your own or extend the default implementations provided with this solution.

Extension "styla"

The styla extension is essential. It

- provides classes for convenient conversion of Java objects to JSON and vice versa
- defines the interfaces of service classes which are used by the implementation of the endpoints responding to requests coming from styla
- defines an interface and provides a default implementation for requesting SEO content from styla

Extension "stylaaddon"

The stylaaddon extension is optional and is a sample implementation adding the styla magazine to the hybris B2C Accelerator. It depends on the styla extension, as it implements some of its interfaces. Features of this extension:

- provides a controller implementing the endpoint for exposing data to styla
- provides default implementations of the service interfaces which are used for exposing data to styla
- provides a controller responsible for rendering the magazine page in your online shop
- html templates (*jsp, *tag) that render the html for the styla page and add the javascript/ stylesheet declarations
- ImpEx files for the creation of CMS items (pages, contentslots, etc.) for the magazine page

If your online shop doesn't use the standard hybris accelerator you won't use this extension in your system.

Get started

Disclaimer:

Before deploying this solution to your production environment, you should have installed and tested the proper functionality in your testing environment. Make sure you have a backup of your working system.

Adding the styla extension

1. Unzip hybris-styla-extension.zip to {HYBRIS_HOME}/hybris. It will add the styla extension to {HYBRIS_HOME}/hybris/bin/custom/styla

2. Add to {HYBRIS_HOME}/hybris/config/localextensions.xml this line within the extensions tag:
   …
   <extension name='styla' />
   …

3. Edit {HYBRIS_HOME}/hybris/config/local.properties and override the properties from {HYBRIS_HOME}/hybris/bin/custom/styla/project.properties with the custom values provided to you by styla.

4. Change to directory to {HYBRIS_HOME}/hybris/bin/platform/ and execute

   ant all

As this extension doesn’t modify the hybris type system, there is no need to perform a platform update.

Adding the stylaaddon extension

1. Unzip hybris-stylaaddon-extension.zip to {HYBRIS_HOME}/hybris. It will add the stylaaddon extension to {HYBRIS_HOME}/hybris/bin/custom/stylaaddon

2. Add to {HYBRIS_HOME}/hybris/config/localextensions.xml this line within the extensions tag:
   …
   <extension name='stylaaddon' />
   …

3. Change to directory to {HYBRIS_HOME}/hybris/bin/platform/ and execute

   ant addoninstall -Daddonnames="stylaaddon" -DaddonStorefront.yacceleratorstorefront="yacceleratorstorefront"

   If your storefrontname is different, change "yacceleratorstorefront" accordingly.

4. Copy {HYBRIS_HOME}/hybris/bin/custom/stylaaddon/acceleratoraddon/web/webroot/WEB-INF/views/desktop/pages/layout/* to

   {HYBRIS_HOME}/hybris/bin/ext-template/yacceleratorstorefront/web/webroot/WEB-INF/views/desktop/pages/layout/ and copy

   {HYBRIS_HOME}/hybris/bin/acceleratoraddon/web/webroot/WEB-INF/tags/shared/styla to

   {HYBRIS_HOME}/hybris/bin/ext-template/yacceleratorstorefront/web/webroot/WEB-INF/tags/shared/

   This will copy the html template files.

5. Edit {yacceleratorstorefront}/web/webroot/WEB-INF/tags/desktop/template/master.tag adding

   <%@ taglib prefix="styla" tagdir="/WEB-INF/tags/shared/styla" %> to the taglib list and

   <styla:styla/> within the <head> element

6. Change directory to {HYBRIS_HOME}/hybris/bin/platform/ and execute

   ant all

7. Start hybris and perform an impex of the cms-content.impex file located under

   {HYBRIS_HOME}/hybris/bin/custom/stylaaddon/resources/stylaaddon/import/contentCatalogs/<storename>

   This will create the CMS items for the styla content page.

8. Synchronize your content catalog in order to add the items imported with cms-content.impex for the staged catalog version to the online catalog version.

As this extension doesn’t modify the hybris type system, there is no need to perform a platform update.

Migration-Guide to version 1.0

If you migrate from a previous version of this plugin to version 1.0 you might consider the following change. While the previous version supported only the paths /, /story, /user and /tag (the latter three with a pathvariable parameter, this version now supports any paths and also request parameters.

StylaSeoService

The StyleSeoService defines the new method getSeoForPath() which is a generic method to retrieve a SEO object from styla’s SEO Service. If you use a customized implementation of this interface you have to provide an implementation for this method.

The methods getStory(), getUser(), getTag() and getRoot() are deprecated and getSeoForPath() should be used instead. If you use any of these deprecated methods in your project, you should revise your code for replacing these methods with getSeoForPath().

StylaMagazineController

The StylaMagazineController now supports any magazine paths and optional request parameters. If you use a custom implementation of the StylaMagazineController, you should revise your code to add this support as well.

The styla extension in detail

This extension provides elementary functionalities for the integration with styla, which are:

- JSON: classes for convenient conversion of Java objects to JSON and vice versa
- Dataproviders: interfaces of service classes which are responsible for retrieving the right information from hybris commerce suite in order to expose to styla
- Styla Seo Service: interface and default implementation for requesting SEO content from styla
- StylaWebService: a client that communicates with styla's webservice

JSON

For the communication with the styla service JSON is used. The package **com.styla.json** provides classes for marshalling and unmarshalling objects to JSON string and vice versa. For instance, if you request a story from styla, styla will provide information for SEO as JSON, which you can unmarshall by using the Seo class for the creation of the corresponding Seo object. Another example: when you export product search results to styla, you will generate a list of ProductImage objects where a ProductImage contains information such as caption, shop, pageUrl etc. and then provide it as JSON to styla. The ProductImage provides the necessary fields and will be exported to valid JSON.

Dataproviders

Your shop implementation must provide endpoints that respond to requests from styla providing information about your shop. For the current styla API these endpoints are:

- /categories (export category/navigation menu of your shop)
- /products (export products for a given search)
- /product (product detail information)
- /version (provide the version number of the extension)

In order to support customized hybris implementations, this extension defines various interfaces which are used by the classes implementing the endpoints. These interfaces serve as dataproviders whose implementations are responsible for retrieving the required data from the hybris commerce platform.

In the next sections the business logic for information retrieval and the data mapping for each endpoint is briefly described:

The stylaaddon extension implements the endpoints and provides default implementations for these dataprovider interfaces and makes use of the hybris accelerator. In case that a customer doesn't use the hybris accelerator, uses a customized data model or uses a search engine other than hybris' default Apache's SOLR, it might be necessary to implement a custom implementation for these dataprovider interfaces.

StylaCategoryProvider

The StylaCategoryProvider interface is intended to be used for the implementation of the /categories-endpoint. It defines the method getCategories that returns a list of com.styla.json.Category objects. The returned categories must reflect the navigation menu of your shop.

For example, the DefaultStylaCategoryProvider (stylaaddon extension) which implements this interface for the hybris accelerator store, iterates over the hybris CMS components that build up the navigation menu in the shop frontend. For each link that has a (hybris) Category attached, a (styla) Category object is created and filled up with the necessary information. The approach of iterating over the CMS components of the navigation bar is used as the exported categories should represent the stores' navigation menu structure which would not be the case if exporting directly the categories from hybris.

StylaSearchService

The StylaSearchService is intended to be used for the implementation of the /products-endpoint. This interface returns a list of relevant products for given search parameters. The returned search result list contains objects of type com.styla.json.ProductImage.

The search result must correspond to the result for the same search directly executed in the shop, for instance, a text search made in the shop frontend. For this reason, it may be advisable to use the same search implementation as used for the frontend. For example, the DefaultStylaSearchService (stylaaddon extension) uses the ProductSearchFacade of the hybris accelerator.

StylaProductProvider

The StylaProductProvider is intended to be used for the implementation of the /product endpoint. This interface returns detailed product information for a given product code. The returned object is of type com.styla.json.Product.

Styla SEO Service

In order to optimize your styla content pages in your shop for search engines, styla provides an endpoint which returns seo content that can be added to your content page. The interface StylaSeoService defines the method getSeoForPath(String path, String reqParams) which expects the path (e.g. /story/the-story) and optional request parameters (e.g. offset=1234) which returns an object of type com.styla.json.Seo which can be used to add the SEO related content to your page.

The class DefaultStylaSeoService provides an implementation for this interface and essentially forwards requests for a SEO object to the webservice client implementation with its required parameters. The DefaultStylaSeoService uses a cache that stores SEO objects in order to reduce the number of requests to the styla webservice.

With version 1.0 of this extension, the methods getRoot(), getStory(), getUser() and getTag() are deprecated and the method getSeoForPath() which supports any paths and request parameters should be used.

StylaWsClient

StylaWsClient represents the remote interface of the style service. DefaultStylaWsClient implements this interface, does the communication with styla's webservice and processes the webservice's responses.

Configuration (project.properties)

In order to configure the styla extension the following properties are relevant and should be overridden in your local.properties file:

Property	**Values/Example**	**Description**
styla.settings.productionmode	{true, false}	Production environment?
styla.seo.baseurl	http://seo.styla.com/clients/ example	Baseurl of styla's webservice
styla.production.js.url	//cdn.styla.com/scripts/clients/ example.js?v=19c5d2e1de-23	Default url for styla javascript in production environment
styla.production.css.url	//cdn.styla.com/styles/clients/ example.css?v=19c5d2e1de-23	Default url for styla css in production environment
styla.staging.js.url	//cdn.styla.com/scripts/clients/ example.js?v=19c5d2e1de-23	Default url for styla javascript in non-production environment
styla.staging.css.url	//cdn.styla.com/styles/clients/ example.css?v=19c5d2e1de-23	Default url for styla css in non-production environment
styla.production.js.<site-uid>.<language-iso>.url Example: styla.production.js.apparel-uk.en.url	//cdn.styla.com/scripts/clients/ example1.js?v=19c5d2e1de-1	Url for styla javascript in multisite/-language production environment

styla.staging.js.<site-uid>.<language-iso>.url Example: styla.staging.js.apparel-uk.en.url	//cdn.styla.com/scripts/clients/example1.js?v=19c5d2e1de-1	Url for styla javascript in multisite/-language non-production environment
Styla.endpoint.key.<site-uid> Example: styla.endpoint.key.apparel-de	KRgmBvgyQOCiC3wd	Authentication key for protection your endpoints

The stylaaddon extension in detail

If your online shop doesn't use the standard hybris accelerator you won't use this extension. Nevertheless, this documentation might serve as a starting point for your custom implementation.

The stylaaddon extension is optional and is a sample implementation adding the styla magazine to the hybris B2C Accelerator. It depends on the styla extension, as it implements its interfaces and uses the StylaWsClient implementation for requesting SEO content.

The extension contains:

- StylaController: a controller that implements the endpoints for exporting data to styla (/categories, /products, /product, /version)
- StylaMagazineController: a controller that is responsible for rendering the magazine page in your online shop
- Dataproviders: default implementations of the dataprovider interfaces
- html templates (*jsp, *tag) that render the html for the styla page and add the javascript/stylesheet declarations
- ImpEx files for the creation of CMS items (pages, contentslots, etc.) for the magazine page

StylaController (/styla)

This class implements the endpoints for

- /categories
- /products
- /product
- /version

and responds to requests coming from styla. This class makes use of the dataprovider implementations, which are responsible to retrieve the requested information from the hybris commerce platform.

Some endpoints require that styla sends an authentication key as a request parameter ("key"). If this parameter is empty or doesn't match, the response returns the HTTP status code 401 - (unauthorized).

StylaMagazineController (/magazine)

This class is responsible for responding to the /magazine path and returns the content page for the styla magazine. The styla magazine can be requested with several parameters which are passed as path variables and optional request parameters. The basic paths are:

- /story/{storytitle} - return a specific story
- /tag/{tag} - filter stories for a specific tag
- /user/{username} - filter stories for a specific user

In version 1.0 of this plugin, the StylaMagazineController supports any path under the magazine root which is passed as request parameter ("url") to styla's Seo-Service. In this context, the StylaMagazineController now also considers request parameters.

The returned HTML page must contain the script tag for the styla-javascript and styletag for the styla-css which are necessary to load the styla magazine on your page and to render the content properly.

Additionally, styla recommends to add additional tags for search engine optimization to the HTML of the magazine page. The content for these tags is requested from styla’s webservice. For your convinience you can use the implementation of the StylaSeoService from the styla extension which returns an object of type com.styla.json.Seo after requesting the information from the webservice.

Customizations

If you don’t use the hybris accelerator or a web application framework other than Spring MVC, you have to implement the endpoints that render the styla magazine page (/magazine/*) and expose your shop data to styla (/styla/*) on your own.

FAQ

I’ve installed the addon extension. How can I test if it is working properly?

First of all make sure that hybris starts up without error, i.e. pay attention on the logs.

In your browser, check the endpoints (adapt domain and path according to your configuration):

/magazine (example)
https://stylademo-uk.example.com/yacceleratorstorefront/en/magazine/

https://stylademo-uk.example.com/yacceleratorstorefront/magazine/story/the-urban-nomad_981034

https://stylademo-uk.example.com/yacceleratorstorefront/magazine/user/bob

https://stylademo-uk.example.com/yacceleratorstorefront/magazine/tag/fashion

/styla/categories (example)
https://stylademo-uk.example.com/yacceleratorstorefront/styla/categories?key=1234

/styla/products (example)
https://stylademo-uk.example.com/yacceleratorstorefront/styla/products?key=1234

https://stylademo-uk.example.com/yacceleratorstorefront/styla/products?key=1234&search=Jacke

https://stylademo-uk.example.com/yacceleratorstorefront/styla/products?key=1234&category=200000

/styla/product (example)
https://stylademo-uk.example.com/yacceleratorstorefront/styla/product?id=300044624

/styla/version (example)
https://stylademo-uk.example.com/yacceleratorstorefront/styla/version

We don’t use the hybris Accelerator/Spring MVC for our shop frontend. What do I have to do to get styla running for my shop?

You basically have to add the styla extension to your hybris configuration and implement the dataprovider interfaces, the endpoint for the stylamagazine and the RESTful webservices on your own. Please check chapter The styla extension in detail and The stylaaddon extension in detail for more details.

Relevant interfaces: StylaProductProvider, StylaSearchService, StylaCategoryProvider

We use a customized data model. What do I have to do to get styla running for my shop?

You basically have to write your custom implementation of the dataprovider interfaces.

Relevant interfaces: StylaProductProvider, StylaSearchService

We don’t use the default Apache SOLR installation of the hybris Accelerator for the product search in our shop frontend. What do I have to do?

You basically have to write your custom implementation of the dataprovider interfaces.

Relevant interfaces: StylaProductProvider, StylaSearchService