eCommerce Integration with hybris

NOTE

This page contains links to the hybris website. For certain pages you will need an account to login.

Deploying eCommerce with hybris

NOTE

The following procedures use the following demonstration catalog to illustrate the deployment:
Geometrixx Outdoors Site English (US)

Deploying the necessary eCommerce packages will provide the full functionality of the eCommerce framework, together with a reference implementation of eCommerce functionality as provided with a hybris implementation (including a demonstration catalog)

This is available under the English (US) branch (/content/geometrixx-outdoors/en_US) of the Geometrixx Outdoors site:

- Product Information (with color variants when appropriate)
- Shopping Cart content overviews
- Customer Sign-Up and Customer Sign-In
- Access to the hybris management console

TECHNICAL REQUIREMENTS - HYBRIS SERVER

The hybris extension of the eCommerce Integration Framework has been updated to support Hybris 5 (as default), while maintaining backward compatibility with Hybris 4.

NOTE

hybris 5.1 is currently not supported.

NOTE

You will need Java 7 to run the hybris 5 server.

NOTE

The hybris add-on, the Telco Accelerator, is not supported by the AEM extension.

PACKAGES NEEDED FOR ECOMMERCE WITH HYBRIS

To install eCommerce functionality you need:

- Your hybris server, available from hybris.
- AEM eCommerce framework:
  - this is part of a standard AEM installation
- AEM hybris content packages, available through your Daycare account:
  - cq-commerce-hybris-update-content-5.6.200

- hybris-specific API implementation
- cq-geometrixx-hybris-content-5.6.200
- a reference implementation to illustrate use of hybris (geometrixx-outdoors/en_US)

NOTE

To download the package, please request access via Daycare.

INSTALLATION OF ECOMMERCE WITH HYBRIS

To install a fully-fledged configuration (using the demonstration catalog, Geometrixx Outdoors) the basic steps are:

1. Install AEM.
2. Install the demonstration content packages using the Package Manager:
   1. cq-commerce-hybris-update-content-5.6.200
   2. cq-geometrixx-hybris-content-5.6.200
3. Download and build your hybris Server.
4. Construct your catalog in your eCommerce engine:
   1. Setup the Geometrixx Outdoor Store.
5. Author any supplementary pages that you need in AEM.

NOTE

To download the package, please request access via Daycare.

CAUTION

Use of the hybris server requires a separate hybris license.

NOTE

For developers API documentation is also available for download.

DOWNLOAD AND BUILD YOUR HYBRIS SERVER

The steps in this procedure will download and build the hybris server. It will also make the initial configurations required for the connections between hybris and cq. The extension will then be usable with the default settings.

NOTE

To complete this, you will need Groovy installed on your system.

1. Download the **hybris Commerce Suite 5.0.4** distribution from:
   https://download.hybris.com/resources/releases/5.0.4/hybris-commerce-suite-5.0.4.0.zip

   CAUTION

   The **5.0.4** version **must** be used. No guarantee can be made for later versions.

CAUTION

You will need an account (from hybris) to access this.

2. Unzip the distribution file to the required location (referred as <hybris-root-directory>).
3. From the command line, execute the following:

```
cd <hybris-root-directory>/bin/platform
. ./setantenv.sh
ant clean all
cd ../..
```

NOTE

When executing:
ant clean all
Press Return when required.

4. Download the following files to the root folder of your extracted hybris distribution, <*hybris-root-directory*>:

5. From the command line, execute the following to:
   - update the configuration of the hybris server (as required by the extension)
   - rebuild the hybris server with the modified configuration
   - start the server

```
groovy setup.groovy
cd bin/platform
ant clean all
sh hybrisserver.sh
```

NOTE

Dependent on your system, several of these steps might take several minutes to complete.

6. In your browser, navigate to the **hybris administration console** at:
   http://localhost:9001

7. Click **Initialize** and then confirm the initialization action (as it will delete existing data).
   The progress will be shown on the console, with FINISHED indicating completion.

NOTE

Dependent on your system, this might take several minutes to complete.

8. Shutdown your hybris instance.
9. From the command line, execute the following in the root folder of your hybris distribution:

```
cd <hybris-root-directory>
groovy postinit.groovy
```

SETUP THE GEOMETRIXX OUTDOORS STORE

This procedure will upload and configure the demonstration store - Geometrixx Online.

1. Start your hybris instance. From the command line, execute the following:

```
cd <hybris-root-directory>/bin/platform
```

```
sh hybrisserver.sh
```

2. In your browser, navigate to the **hybris management console** at: http://localhost:9001/hmc/hybris
3. From the sidebar navigation, expland **System** and **Tools**. Then select **Import** to open the **Wizard: CSV Import** window.
4. In the **Configuration** tab, **Upload** the following **Import file**:
5. Set the **Locale Setting** to: en_US - English (United States)
6. Open the **Resources** tab.
7. **Upload** the following **Media-Zip**:
8. Click **Start** to import the specified files. The **Result** tab will show any log entries.
9. Click **Done** to close the import window.
10. From the sidebar, select **System**, then **Tools**, then **Import**.
11. **Upload** the following **Import file**:
12. Set the **Locale Setting** to: en_US - English (United States)
13. Click **Start** to import the specified files. The **Result** tab will show any log entries.
14. Click **Done** to close the import window.
15. You can now use the product cockpit to view the imported catalogs and products: http://localhost:9001/productcockpit

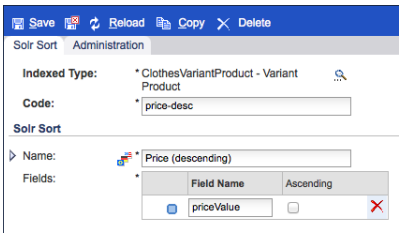
Administering hybris and AEM Commerce

After installation you can configure your instance:

1. Configure the Facetted Search for Geometrixx Outdoors.
2. Configure the Catalog Version.
3. Configure the Import Structure.
4. Configure the Product Attributes to Load.
5. Importing the Product Data.
6. Configure the Catalog Importer.
7. Use the importer to import the catalog into a specific location in AEM.

CONFIGURE THE FACETTED SEARCH FOR GEOMETRIXX OUTDOORS

1. In your browser, navigate to the **hybris management console** at: http://localhost:9001/hmc/hybris
2. From the sidebar, select **System**, then **Facet search**, then **Facet Search Config**.
3. **Open Editor** for the **Sample Solr Configuration for clothescatalog**.
4. Under **Catalog versions** use **Add Catalog version** to add outdoors-Staged and outdoors-Online to the list.
5. **Save** the configuration.
6. Open **SOLR Item types** to add **SOLR Sorts** to ClothesVariantProduct:
   - relevance ("Relevance", score)
   - name-asc ("Name (ascending)", name)
   - name-desc ("Name (descending)", name)
   - price-asc ("Price (ascending)", priceValue)
   - price-desc ("Price (descending)", priceValue)



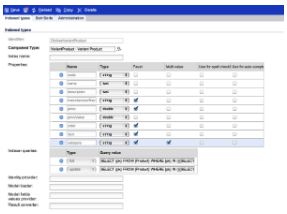
7. In the **Indexed Types** tab set the **Composed Type** to:
   Product - Product
8. In the **Indexed Types** tab adjust the **Indexer queries** for full:

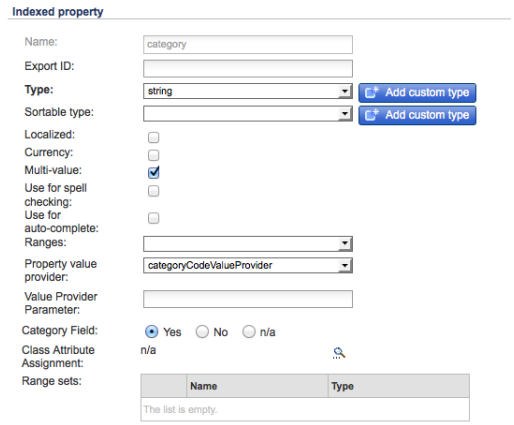
```
SELECT {pk} FROM {Product} WHERE {pk} IN ({{SELECT {baseProduct} FROM {variantproduct}}})
```

9. In the **Indexed Types** tab adjust the **Indexer queries** for incremental:

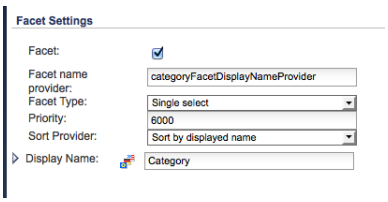
```
SELECT {pk} FROM {Product} WHERE {pk} IN ({{SELECT {baseProduct} FROM {variantproduct}}}) AND
 {modifiedtime} <= ?lastIndexTime
```

10. In the **Indexed Types** tab adjust the category facet according to the following screenshots. Double-click on the last entry in the category list to open the **Indexed property** tab:

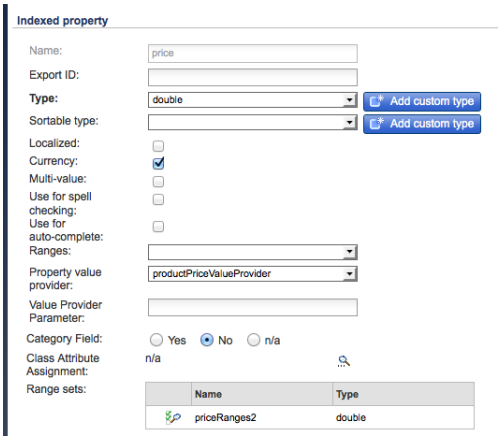




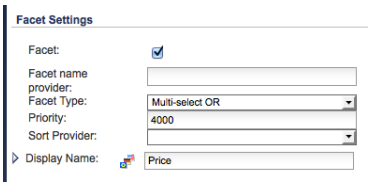
11. Open the **Facet Settings** tab and adjust the field values:



12. **Save** the changes.
13. Again from **SOLR Item types**, adjust the price facet according to the following screenshots. As with category, double-click on price to open the **Indexed property** tab:



14. Open the **Facet Settings** tab and adjust the field values:



15. **Save** the changes.
16. Open **System**, **Facet search**, then **Indexer operation wizard**. Start a cronjob:
    - **Indexer operation**: full
    - **Solr configuration**: Sample Solr Config for Clothes

CONFIGURE THE CATALOG VERSION

The **Catalog version** (hybris.catalog.version) that is imported can be configured for:
Day CQ Commerce Hybris Configuration
(com.adobe.cq.commerce.hybris.common.DefaultHybrisConfigurationService)
Catalog version is usually set to either Online or Staged (the default).
When working with AEM there are several methods of managing the configuration settings for such services; see Configuring OSGi for full details. Also see the console for a full list of configurable parameters and their defaults.

The log output provides feedback on the created pages and components and reports potential errors.

CONFIGURE THE IMPORT STRUCTURE

The following listing shows a sample structure (of assets, pages and components) that is created by default:

```
+ /content/dam/path/to/images
  + 12345.jpg (dam:Asset)
    + ...
  + ...
+ /content/site/en
  - cq:commerceProvider = "hybris"
  - cq:hybrisBaseStore = "basestore"
  - cq:hybrisCatalogId = "catalog"
  + category1 (cq:Page)
    + jcr:content (cq:PageContent)
      - jcr:title = "Category 1"
    + category11 (cq:Page)
      + jcr:content (cq:PageContent)
        - jcr:title = "Category 1.1"
      + 12345 (cq:Page)
        + jcr:content (cq:PageContent)
          + par
            + product (nt:unstructured)
              - cq:hybrisProductId = "12345"
              - sling:resourceType = "commerce/components/product"
              + image (nt:unstructured)
```

```
    - sling:resourceType = "commerce/components/product/image"
    - fileReference = "/content/dam/path/to/images/12345.jpg"
  + 12345.1-S (nt:unstructured)
    - cq:hybrisProductId = "12345.1-S"
    - sling:resourceType = "commerce/components/product"
    + image (nt:unstructured)
      - sling:resourceType = "commerce/components/product/image"
      - fileReference = "/content/dam/path/to/images/12345.1-S.jpg"
  + ...
```

Such a structure is created by the OSGi service DefaultImportHandler that implements the ImportHandler interface. An import handler is called by the actual importer to create products, product variations, categories, asset, etc.

NOTE

You can customize this process by implementing your own import handler.

The structure to be generated when importing can be configured for:
Day CQ Commerce Hybris Default Import Handler
(com.adobe.cq.commerce.hybris.importer.DefaultImportHandler)
When working with AEM there are several methods of managing the configuration settings for such services; see Configuring OSGi for full details. Also see the console for a full list of configurable parameters and their defaults.

CONFIGURE THE PRODUCT ATTRIBUTES TO LOAD

The response parser can be configured to define the properties and attributes to be loaded for (variant) products:

1. Configure the OSGi bundle:
   Day CQ Commerce Hybris Default Response Parser
   (com.adobe.cq.commerce.hybris.impl.importer.DefaultResponseParser)
   Here you can define various options and attributes needed for loading and mapping.

   NOTE

   When working with AEM there are several methods of managing the configuration settings for such services; see Configuring OSGi for full details. Also see the console for a full list of configurable parameters and their defaults.

IMPORTING THE PRODUCT DATA

There are a variety of ways to import the product data. The product data can be imported when initially setting up the environment, or after changes have been made in the hybris data:

- Full Import
- Incremental Import
- Express Update

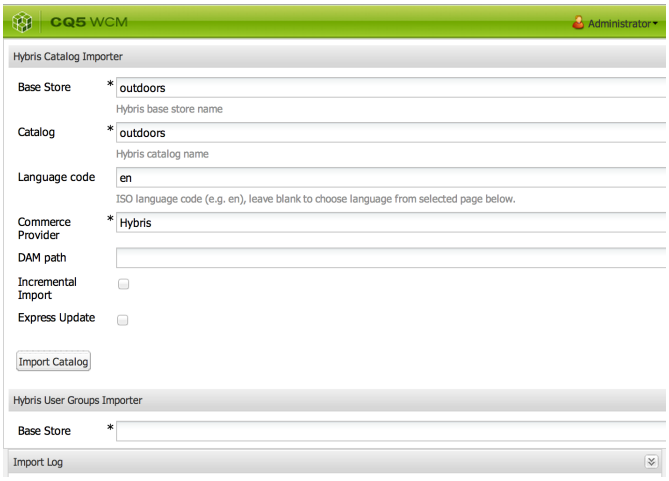
Full Import

1. If required, delete all existing product data using CRXDE Lite.
   1. Navigate to the sub-tree holding the product data:
      /etc/commerce/products
      For example:
      http://localhost:4502/crx/de/index.jsp#/etc/commerce/products
   2. Delete the node that holds your product data; for example, outdoors.
   3. **Save All** to persist the change.

2. Open the hybris importer in AEM:
   /etc/importers/hybris.html

For example:
http://localhost:4502/etc/importers/hybris.html

3. Configure the required parameters; for example:



4. Click **Import Catalog** to start the import.
When complete, you can verify the data imported at:
/etc/commerce/products/outdoors
You can open this in CRXDE Lite; for example:
http://localhost:4502/crx/de/index.jsp#/etc/commerce/products

Incremental Import

1. Check the information held in AEM for the relevant product(s), in the appropriate sub-tree under:
/etc/commerce/products
You can open this in CRXDE Lite; for example:
http://localhost:4502/crx/de/index.jsp#/etc/commerce/products
2. In hybris, update the information held on the revelant product(s).

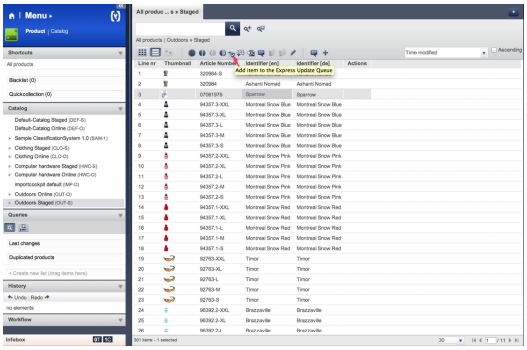
3. Open the hybris importer in AEM:
/etc/importers/hybris.html
For example:
http://localhost:4502/etc/importers/hybris.html
4. Select the clickbox **Incremental Import**.
5. Click **Import Catalog** to start the import.
When complete, you can verify the data updated in AEM under:
/etc/commerce/products

Express Update

The import process can take a long time, so as an extension to the Product Synchronization you can select specific areas of the catalog for an express update that is triggered manually. This uses the export feed together with the standard attributes configuration.

1. Check the information held in AEM for the relevant product(s), in the appropriate sub-tree under:
/etc/commerce/products
You can open this in CRXDE Lite; for example:
http://localhost:4502/crx/de/index.jsp#/etc/commerce/products
2. In hybris, update the information held on the revelant product(s).

3. In hybris, add the product(s) to the Express Queue; for example:



4. Open the hybris importer in AEM:
   /etc/importers/hybris.html
   For example:
   http://localhost:4502/etc/importers/hybris.html
5. Select the clickbox **Express Update**.
6. Click **Import Catalog** to start the import.
   When complete, you can verify the data updated in AEM under:
   /etc/commerce/products

CONFIGURE THE CATALOG IMPORTER

The hybris catalog can be imported into CQ, using the batch importer for hybris catalogs, categories and products.

The parameters used by the importer can be configured for:
Day CQ Commerce Hybris Catalog Importer
(com.adobe.cq.commerce.hybris.impl.importer.DefaultHybrisImporter)
When working with AEM there are several methods of managing the configuration settings for such services; see Configuring OSGi for full details. Also see the console for a full list of configurable parameters and their defaults.

Concepts of eCommerce with hybris

The integration framework provides the mechanisms and components for:

- connection to an eCommerce system
- pulling data into AEM
- displaying that data and collecting the shopper's responses
- returning transaction details
- search over the data from both systems

This means that:

- shoppers can register and shop without waiting
- price changes will be seen by shoppers without delay
- products can be added as required

NOTE

The eCommerce framework can be used with any eCommerce solution. Certain specifics and examples dealt with here will refer to the hybris solution.

To optimize operation each application concentrates on its own area of expertise with information being transferred between the two in real time; for example:

- AEM:
  - Requests:
    - Product Information from hybris.

    - Provides:
        - User views for Product Information, Shopping Cart and Checkout.
        - Shopping cart and checkout information to hybris.
        - Search Engine Optimization (SEO).
        - Community functionality.
        - Unstructured marketing interactions.
- hybris:
    - Provides:
        - Product Information from the database.
        - Product variant management.
        - Search within the product information.
    - Processes:
        - The Shopping Cart.
        - The Checkout.
        - Order fulfillment.

A number of out-of-the-box AEM components are provided to use the integration layer. Currently these are:
- Product display component
- Shopping cart
- Check-out
- my Account

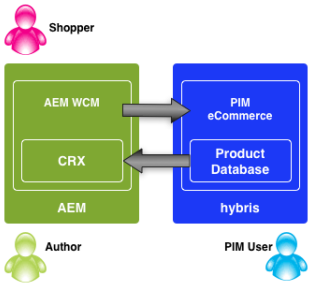
Various search options are also available.

PAGE AND COMPONENTS

The following components and pages can use eCommerce data:
- **Landing page**
  Rendered by AEM (**Commerce Category**).
  This provides primarily static information, for example, an introduction and overview with links to the underlying product pages.
- **Product**
  Rendered by AEM (**Commerce Product**).
  Dynamic information about individual products; for example, price changes, special offers. Updates from live data are reflected.
  The product component can be added to any page where the parent page delivers the required metadata (i.e. the paths to cartPage and cartObject). In Geometrixx Outdoors this is supplied by UserInfo.jsp.
- **Search results / product lists** (cached)
  See search below. This component can be added to any page.
- **myAccount**
  Transaction data in hybris is combined with personal information about the shopper. AEM uses some of this data as profile data. A forms action in AEM is used to write information back to hybris.
- **Checkout**
  Checkout is implemented with standard AEM forms. This allows the marketing manager to customize the experience with marketing content.
  The checkout process is a very complex process that is hybris owned.
- **Shopping cart**
  hybris provides all relevant data (xml). AEM renders this information using the appropriate components.

ROLES

The integrated system caters for the following roles to maintain the data:



- Product Information Management (PIM) User who maintains:
  - Product information.
  - Taxonomy, categorization, approval.
  - Interacts with digital asset management.
  - Pricing - often this comes from an ERP system and is not explicitly maintained in the commerce system.
- Author / Marketing Manager who maintains:
  - Marketing content for all channels.
  - Promotions.
  - Vouchers.
  - Campaigns.
- Surfer / Shopper who:
  - Views your product information.
  - Places items into the shopping cart.
  - Checks out their orders.
  - Expect order fulfillment.

SINGLE SIGN-ON

Single-sign-on (SSO) is provided, so that authors are known in both systems without having to login twice.

PRODUCTS AND PRODUCT VARIANTS

Products

Product data is maintained in hybris and can be made available in AEM.
Actual product information imported from hybris is held in the CRX repository under:
/etc/commerce/products
The following properties indicate the link with hybris:

- commerceProvider
- cq:hybrisCatalogId
- cq:hybrisProductID

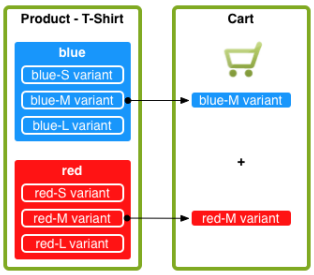
This data is synchronized as necessary.

NOTE

The hybris implementation (i.e. geometrixx-outdoors/en_US) only stores product IDs and other basic information under /etc/commerce.
The hybris server is referenced every time information about a product is requested.

Product Variants

For appropriate products information about variants can also be held. For example, for items of clothing the different colors available are held as variants:



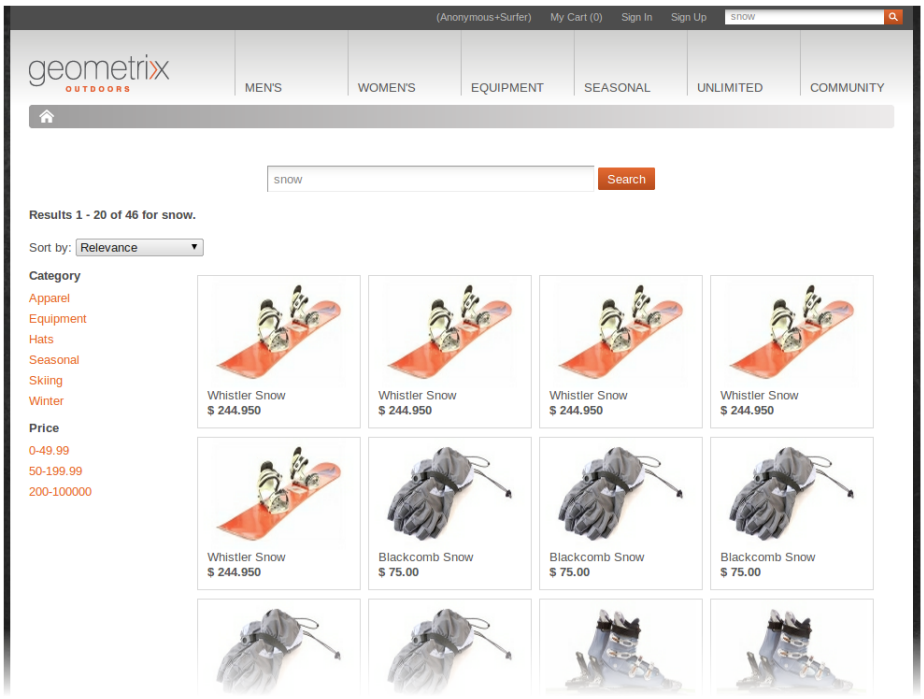
DATA SYNCHRONIZATION

Some of the product data that is maintained in hybris needs to be available in AEM.
Depending on the type of data either:

- A periodic synchronization is used together with a data feed of changes. In addition to this, you can select specific updates for an express update.
- Highly volatile data, such as price information, is retrieved from the commerce engine for each page request.

SEARCH

The eCommerce search API is fully implemented in the hybris solution, so you can use the eCommerce search component that is provided out-of-the-box. The faceted search allows you to search either JCR and/or hybris:



PROMOTIONS AND VOUCHERS

Vouchers are a tried and tested method of offering discounts to either attract customers into making a purchase and/or rewarding customer's loyalty.
Promotions, together with vouchers, allow you to realize scenarios such as:

- A company provides custom prices for employees, which is a handcrafted list of users.
- Long-term customers receive discounts on all orders.
- A sale price offered over a well-defined time period.
- A customer receives a voucher when their previous order exceeded a specific amount.

- A customer who buys *product-X* is offered a discount on *product-Y* (pair products).

Promotions are not usually maintained by product information managers, but by marketing managers. The promotions are also integrated into the Campaign Management.

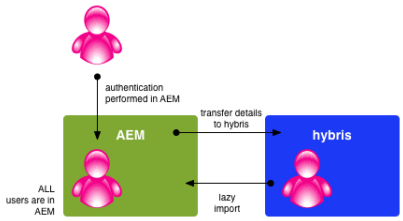
hybris promotions and hybris vouchers cover everything that influences the shopping cart and is related to pricing. Promotion specific marketing content (such as banners, etc) is not part of the hybris promotion.

NOTE

AEM uses the term **Voucher**, this is synonymous with the term **Coupon**.

CUSTOMER REGISTRATION AND ACCOUNTS

When a shopper registers, the account details need to be synchronized between AEM and hybris. Sensitive data is held independently, but profiles are shared:



NOTE

When using hybris, you need to ensure that accounts created for users who log into an AEM instance are replicated (e.g. via workflows) to any other AEM instances that communicate with hybris.
Otherwise, these other AEM instances will also try to create accounts for these users in hybris. These actions will fail with a DuplicateUidException coming from hybris.

CUSTOMER-SPECIFIC PRICING

hybris uses the context (essentially the shopper information) to determine the price on the hybris side then provide the correct information back to AEM.

PAYMENT SECURITY

Payment details, including credit card information, is managed by hybis. AEM forwards such transactional information to hybris (from where it is then forwarded to a payment processing service).

Payment Card Industry (PCI) complicance can be achieved.

ADDRESS BOOK

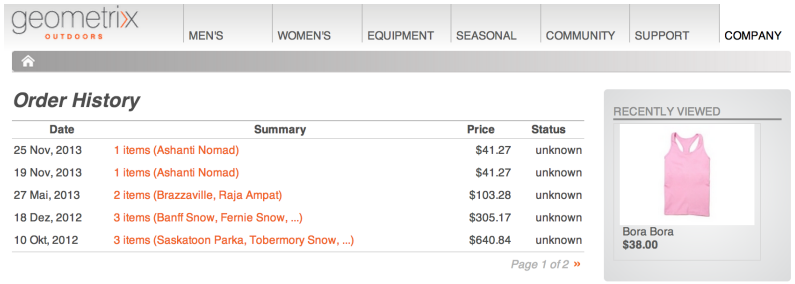
Your site will need to store delivery, billing and alternative addresses. This can be implemented using forms based on your default address format.

ORDER FULFILLMENT AND TRACKING

After placing an order, shoppers will often return to:

- Check the status of their order
- Remove products from the order
- Add products to the list

After receiving the order delivery, shoppers may also want to view the history of orders made over a period of time.
Order fulfillment and tracking is managed by hybris. Information can be displayed by AEM using the Order History component, which shows all relevant details, including the vouchers and promotions applied. For example:



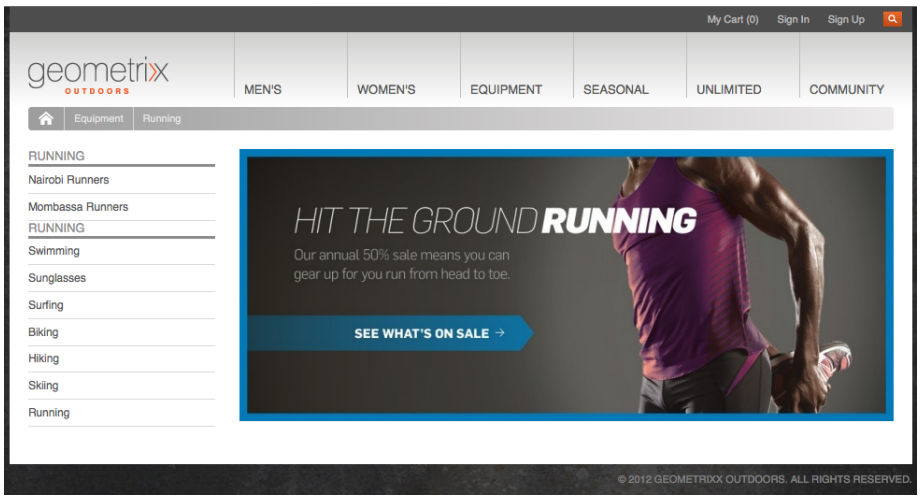
CATALOGS IN MULTIPLE LANGUAGES

hybris supports product content in multiple languages, this can be used by AEM.
When requesting data from hybris, the integration framework retrieves the language from the current tree (for example, en_US for pages under /content/geometrixx-outdoors/en_US).
For a multi-lingual store, you can import your catalog in each language tree individually (or copy it by means of MSM).

PRODUCT CATALOG SECTIONS

The product catalog sections provide you with, for example:

- an introduction (image and/or text) to the category; this can also be used for banners and teasers to promote special offers
- links to the individual products in that category
- links to the other categories



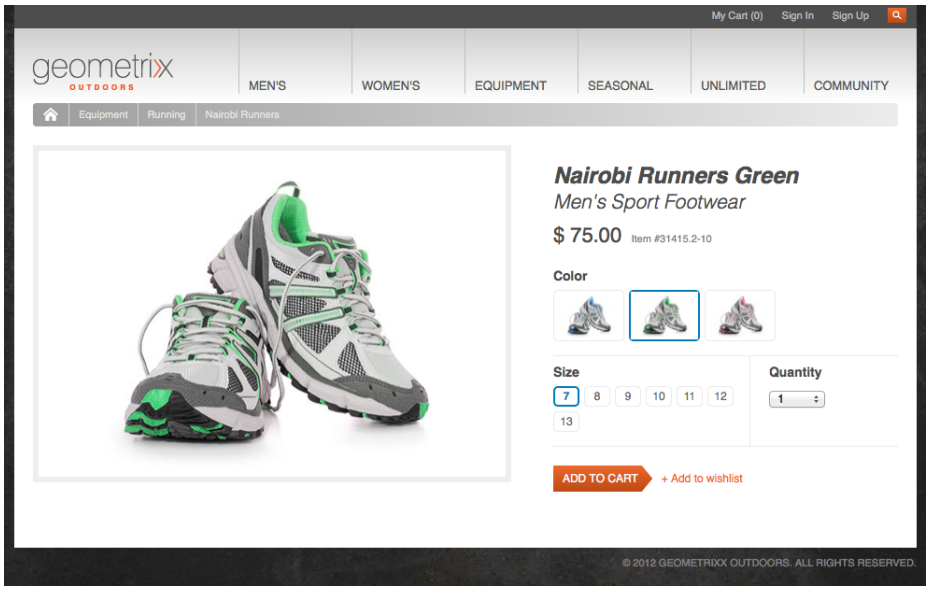
PRODUCT INFORMATION WITH COLOR VARIANTS

A AEM page (for example, using the **Commerce Product** template) can provide:

- general product information; text and images
- pricing

And allow the shopper to select:

- color and size variants
- quantity
- add to basket



SHOPPING CART CONTENT OVERVIEW

The shopping cart provides:
- an overview of items selected
- links to the individual product pages
- updates to quantity
- removal of the item

hybris stores the cart in a session so items stay in the cart (and can be restored) across log-in/log-out. For example:
- browse as anonymous and add products to the cart
- sign in as Allison Parker - her cart is empty
- add products to her cart
- sign out - the cart will show the products for anonymous
- sign in again as Allison Parker - her products are restored

NOTE

An anonymous cart can only be restored on the same machine.

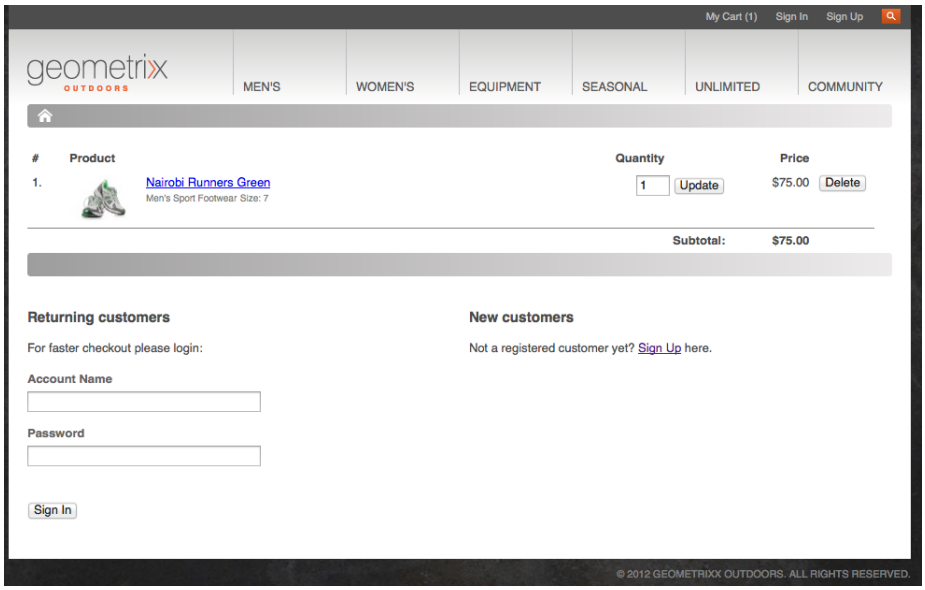
NOTE

It is not recommended to test restoring the cart contents with the admin account, as this can conflict with the hybris admin account.

NOTE

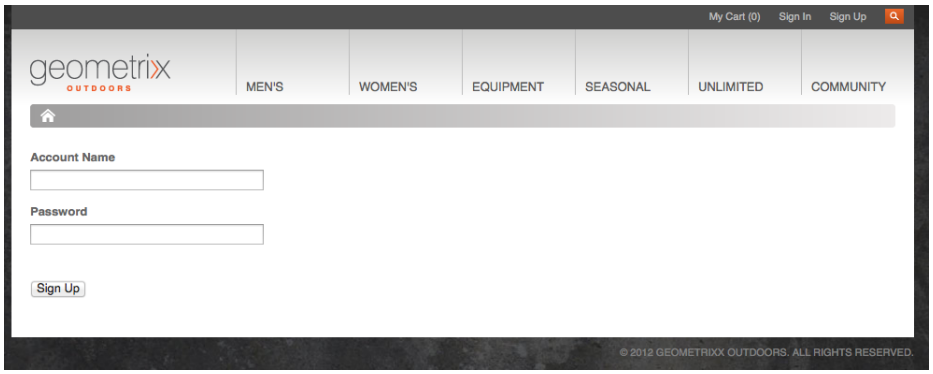
hybris can be configured to remove pending carts after a defined period of time.

Price changes are reflected in both systems as they occur.



CUSTOMER SIGN-UP

Often sign-up is required for the shopper to have access to the shopping cart. This requires registration so that a customer-specific account can be created.

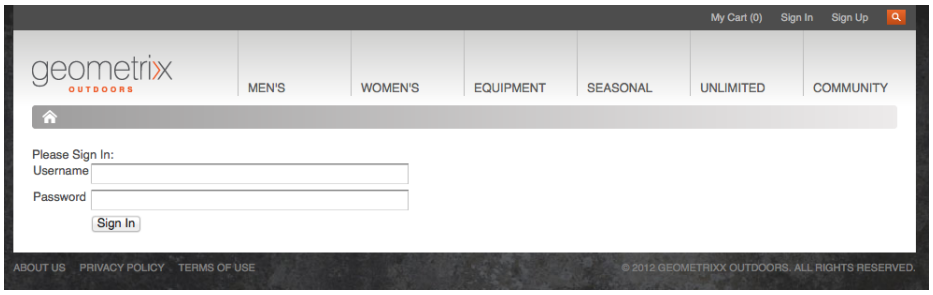


NOTE

An anonymous shopping cart and checkout is also supported.

CUSTOMER SIGN-IN

After sign-up the shopper must login with their account so that their actions can be tracked and their orders fulfilled.

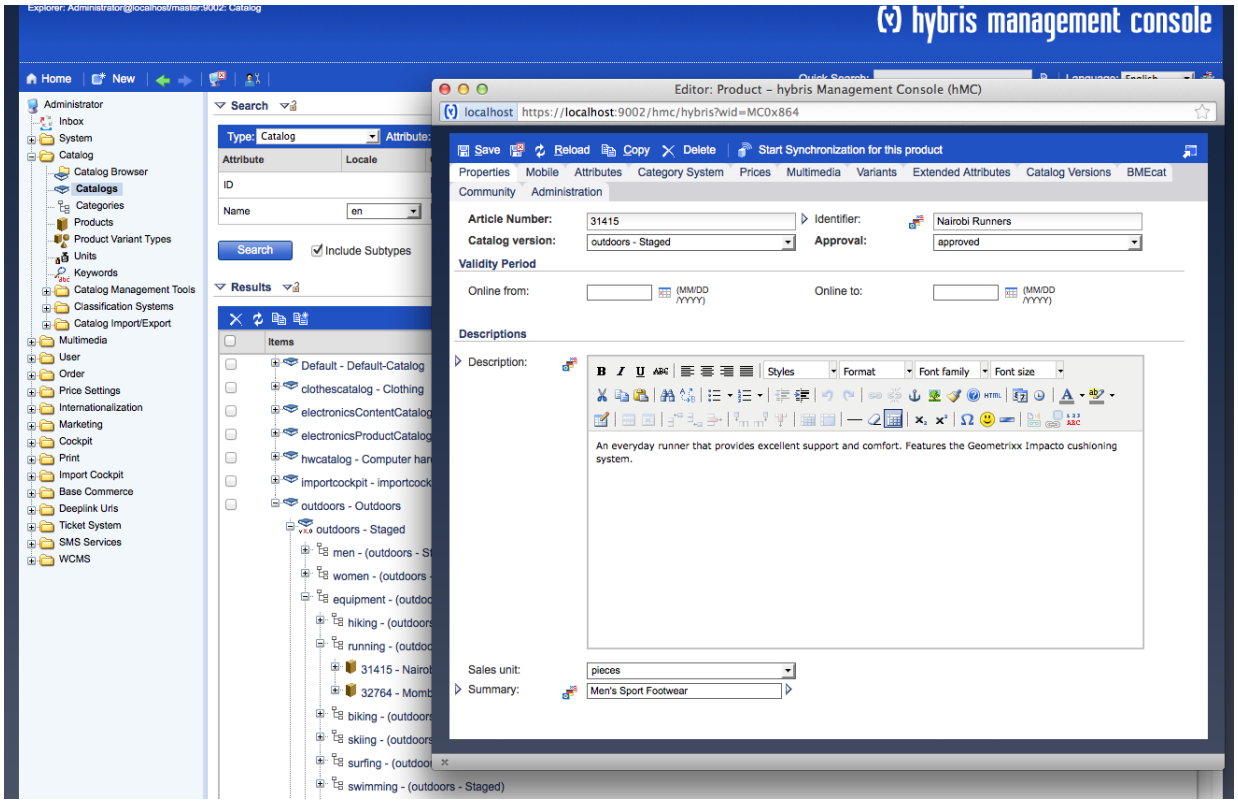


ACCESS TO THE HYBRIS MANAGEMENT CONSOLE

The hybris console is used for managing your product information; see the hybris documentation for more information.
The local instance included in the demonstration packages can be accessed using:
https://localhost:9001/hmc/hybris

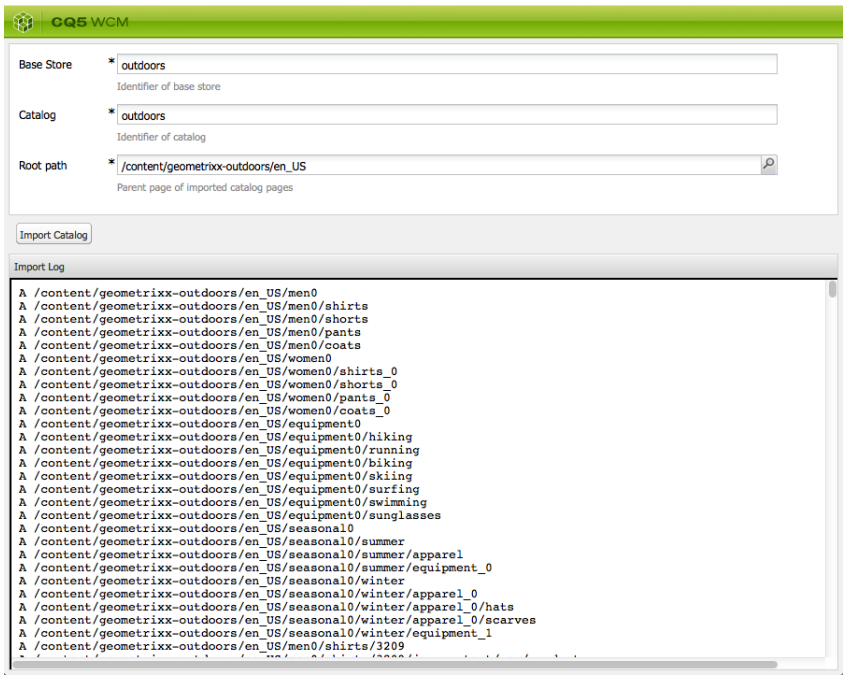
Login is required, the default details are:

- Account name:
  admin
- Password:
  nimda



CATALOG IMPORTER

The hybris package comes with a catalog importer for setting up the initial page structure.
This is available from:
http://localhost:4502/etc/importers/hybris.html



The following information has to be provided:

- **Base store**
  The identifier of the base store configured in hybris.
- **Catalog**
  The identifier of the catalog to import.
- **Root path**
  The path where the catalog should be imported into.

PROXY PAGES

Proxy pages are used to simplify the structure of the repository.
Creating a catalog will use 10 nodes per product as it provides individual components for each product that you can update and customise within AEM.
This large number of nodes can become an issue if your catalog contains hundreds or even thousands of products. To avert any issues you can create your catalog using proxy pages.
These proxy pages use a two-node structure (cq:Page and jcr:content), which proxies for a complete product page by referencing the product data and the template page. The proxy page will then be used to recreate the catalog page.
However, there is a trade-off. You will not be able to customize your product information within AEM, a standard template (defined for your site) will be used.

NOTE

You will not experience any problems if you import a large catalog without proxy pages.
You can convert from one methodology to the other at any time. You can also convert a sub-section of your catalog.

BUCKETING

As a JCR node can only have 1000 direct child nodes, buckets (phantom folders) are required to ensure that this limit is not exceeded. These are generated according to an algorithm when importing.
These buckets take the form of phantom folders that are introduced to your catalog structure, but can be configured so they are not apparent in public URLs.

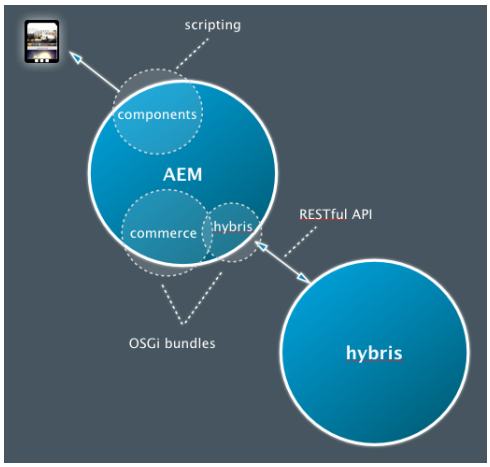
Developing eCommerce with hybris

NOTE

The eCommerce framework can be used with any eCommerce solution. Certain specifics and examples dealt with here will refer to the hybris solution.

The integration framework includes an integration layer with an API. This allows you to:

- plug in an eCommerce system and pull product data into AEM
- build AEM components for commerce capabilities independent of the specific eCommerce engine



NOTE

API documentation is also available for download.

A number of out-of-the-box AEM components are provided to use the integration layer. Currently these are:

- a product display component
- a shopping cart
- check-out

For search an integration hook is provided that allows you to use the AEM search, the search of the eCommerce system, a third party search (like Search&Promote) or a combination thereof.

ECOMMERCE ENGINE SELECTION

The eCommerce framework can be used with any eCommerce solution, the engine being used needs to be identifiable by AEM:

- eCommerce Engines are OSGi services supporting the CommerceService interface
  - Engines can be distinguished by a commerceProvider service property
- AEM supports Resource.adaptTo() for CommerceService and Product
  - The adaptTo implementation looks for a cq:commerceProvider property in the resource's hierarchy:
    - If found, the value is used to filter the commerce service lookup.
    - If not found, the highest-ranked commerce service is used.
  - A cq:Commerce mixin is used so the cq:commerceProvider can be added to strongly-typed resources.
- The cq:commerceProvider property is also used to reference the appropriate commerce factory definition.
  - For example, a cq:commerceProvider property with the value hybris will correlate to the OSGi configuration for **Day CQ Commerce Factory for Hybris** (com.adobe.cq.commerce.hybris.impl.HybrisServiceFactory) - where the parameter commerceProvider also has the value hybris.
  - Here further properties, such as Catalog version can be configured (when appropriate and available).

See the following examples below:

`cq:commerceProvider = geometrixx`	in a standard AEM installation a specific implementation is required; for example, the geometrixx example, which includes minimal extensions to the generic API
`cq:commerceProvider = hybris`	hybris implementation

```
/content/store
+ cq:commerceProvider = hybris
  + mens
    + polo-shirt-1
    + polo-shirt-2
    + employee
+ cq:commerceProvider = jcr
  + adobe-logo-shirt
    + cq:commerceType = product
    + price = 12.50
  + adobe-logo-shirt_S
    + cq:commerceType = variant
    + size = S
  + adobe-logo-shirt_XL
    + cq:commerceType = variant
    + size = XL
    + price = 14.50
```

NOTE

Using CRXDE Lite you can see how this is handled in the product component for the hybris implementation:
/apps/geometrixx-outdoors/components/hybris/product/product.jsp

DEVELOPING FOR HYBRIS 4

The hybris extension has been updated to support Hybris 5, while maintaining backward compatibility with Hybris 4.

The default settings in the code are tuned for Hybris 5.
To develop for Hybris 4 the following is required:

- When invoking maven add the following command line argument to the command
  -P hybris4
  It downloads the pre-configured Hybris 4 distribution and embeds it in the bundle:
  cq-commerce-hybris-server
- In the OSGi configuration manager:
  - Disable Hybris 5 support for the Default Response Parser service.
  - Ensure that Hybris Basic Authentication Handler service has a lower service ranking than Hybris OAuth Handler service.

SESSION HANDLING

hybris uses a user session to store information such as the customer's shopping cart. The session id is returned from hybris in a JSESSIONID cookie that needs to be sent on subsequent requests to hybris. To avoid storing the session id in the repository it is encoded in another cookie that is stored in the shopper's browser. The following steps are performed:

- On the first request no cookie is set on the shopper's request; so a request is sent to the hybris instance to create a session.
- The session cookies are extracted from the response, encoded in a new cookie (for example, hybris-session-rest) and set on the response to the shopper. The encoding in a new cookie is required, because the original cookie is only valid for a certain path and would otherwise not be sent back from the browser in subsequent requests. The path information must also be added to the cookie's value.
- On subsequent requests, the cookies are decoded from the hybris-session-<*xxx*> cookies and set on the HTTP client that is used to request data from hybris.

NOTE

A new, anonymous session is created when the original session is no longer valid.

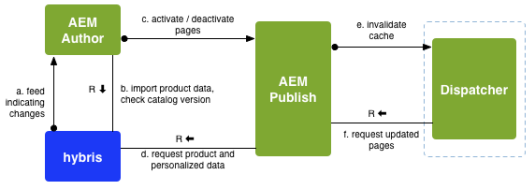
CommerceSession

- This session "owns" the **shopping cart**
  - performs add/remove/etc
  - performs the various calculations on the cart; commerceSession.getProductPrice(Product product)
- Owns the *storage location* for the **order** data CommerceSession.getUserContext()
- Also owns the **payment** processing connection
- Also owns the **fulfillment** connection

PRODUCT SYNCHRONIZATION AND PUBLISHING

Product data that is maintained in hybris needs to be available in AEM. The following mechanism has been implemented:

- An initial load of IDs is provided by hybris as a feed. There can be updates to this feed.
- hybris will supply update information via a feed (which AEM polls).
- When AEM is using product data it will send requests back to hybris for the current data (conditional get request using last modified date).
- On hybris it is possible to specify feed contents in a declarative way.
- Mapping the feed structure to the AEM content model happens in the feed adapter on the AEM side.



- The importer (b) is used for the initial setup of the page tree structure in AEM for catalogs.
- Catalog changes in hybris are indicated to AEM via a feed, these then propagate to AEM (b)
  - Product added/deleted/changed with respect to catalog version.
  - Product approved.
- The hybris extension provides a polling importer ("hybris" scheme"), which can be configured to import changes into AEM at a specified interval (for example, every 24 hours where the interval is specified in seconds):
  - http://localhost:4502/content/geometrixx-outdoors/en_US/jcr:content.json
    {
    * "jcr:mixinTypes": ["cq:PollConfig"],
    * "enabled": true,
    * "source": "hybris:outdoors",
    * "jcr:primaryType": "cq:PageContent",
    * "interval": 86400
    }
- The catalog configuration in AEM recognizes **Staged** and **Online** catalog versions.
- Syncing products between catalog versions will require a (de-)activation of the corresponding AEM page (a, c)
  - Adding a product to an **Online** catalog version requires activation of the product's page.
  - Removing a product requires deactivation.
- Activating a page in AEM (c) requires a check (b) and is only possible if
  - The product is in an **Online** catalog version for product pages.
  - The referenced products are available in an **Online** catalog version for other pages (e.g. campaign pages).

- Activated product pages need to access the product data's **Online** version (d).
- The AEM publish instance requires access to hybris for the retrieval of product and personalized data (d).

ARCHITECTURE

Architecture of Product and Variants

A single product can have multiple variations; for instance, it might vary by color and/or size. A product must define which properties drive variation; we term these *variant axes*.
However, not all properties are variant axes. Variations can also affect other properties; for example, the price might be dependant on size. These properties cannot be selected by the shopper and therefore are not considered variant axes.
Each product and/or variant is represented by a resource, and therefore maps 1:1 to a repository node. It is a corollary that a specific product and/or variant can be uniquely identified by its path.
The product/variant resource does not always hold the actual product dataIt might be a representation of data actually held on another system (such as hybris). For example, product descriptions, pricing, etc, are not stored in AEM, but retrieved in real-time from the eCommerce engine.

Any product resource can be represented by a Product API. Most calls in the product API are variation specific (although variations might inherit shared values from an ancestor), but there are also calls which list the set of variations (getVariantAxes(), getVariants(), etc.).

NOTE

In effect a variant axes is determined by whatever Product.getVariantAxes() returns:
- hybris defines it for the hybris implementation

While products (in general) can have many variant axes, the out-of-the-box product component only handles two:
1. size
2. plus one more
   This additional variant is selected via the variationAxis property of the product reference (usually color for Geometrixx Outdoors).

Product References and Product Data

In general:
- product data is located under /etc
- and product references under /content.

There must be a 1:1 map between product variations and product data nodes.
Product references must also have a node for each variation presented - but there is no requirement to present all variations. For instance, if a product has S, M, L variations, the product data might be:

```
etc
  commerce
    products
      shirt
        shirt-s
        shirt-m
        shirt-l
```

While a "Big and Tall" catalog might have only:

```
content
  big-and-tall
    shirt
      shirt-l
```

Finally, there is no requirement to use product data. You can place all product data under the references in the catalog; but then you cannot really have multiple catalogs without duplicating all the product data.

API

```
public interface Product extends Adaptable {

    public String getPath();            // path to specific variation
    public String getPagePath();        // path to presentation page for all variations
    public String getSKU();             // unique ID of specific variation

    public String getTitle();           // shortcut to getProperty(TITLE)
    public String getDescription();     // shortcut to getProperty(DESCRIPTION)
    public String getImageUrl();        // shortcut to getProperty(IMAGE_URL)
    public String getThumbnailUrl();    // shortcut to getProperty(THUMBNAIL_URL)

    public <T> T getProperty(String name, Class<T> type);

    public Iterator<String> getVariantAxes();
    public boolean axisIsVariant(String axis);
    public Iterator<Product> getVariants(VariantFilter filter) throws CommerceException;
}
```

```
/**
 * Interface for filtering variants and AxisFilter provided as common implementation
 *
 * The <code>VariantFilter</code> is used to filter variants,
 * e.g. when using {@link Product#getVariants(VariantFilter filter)}.
 */
public interface VariantFilter {
    public boolean includes(Product product);
}

/**
 * A {@link VariantFilter} for filtering variants by the given
 * axis and value. The following example returns a list of
 * variant products that have a value of <i>blue</i> on the
 * <i>color</i> axis.
 *
 * <p>
 * <code>product.getVariants(new AxisFilter("color", "blue"));</code>
 */
public class AxisFilter implements VariantFilter {

    private String axis;
    private String value;

    public AxisFilter(String axis, String value) {
        this.axis = axis;
        this.value = value;
    }

    /**
     * {@inheritDoc}
     */
    public boolean includes(Product product) {
        ValueMap values = product.adaptTo(ValueMap.class);

        if(values != null) {
            String v = values.get(axis, String.class);

            return v != null && v == value;
        }

        return false;
    }
}
```

- **General Storage Mechanism**
  - Product nodes are nt:unstructured.
  - A product node can be either:
    - A reference, with the product data stored elsewhere:
      - Product references contain a productData property, which points to the product data (typically under /etc/commerce/products).
      - The product data is hierarchical; product attributes are inherited from a product data node's ancestors.

        - Product references can also contain local properties, which override those specified in their product data.
    - A product itself:
        - Without a productData property.
        - A product node which holds all properties locally (and does not contain a productData property) inherits product attributes directly from its own ancestors.
- **AEM-native Product Structure**
    - Each variant must have its own leaf node.
    - The product interface represents both products and variants, but the related repository node is specific about which it is.
    - The product node describes the product attributes and variant axes.

```
+ banyan_shirt
    - cq:commerceType = product
    - cq:productAttributes = [jcr:title, jcr:description, size, price, color]
    - cq:productVariantAxes = [color, size]
    - jcr:title = Banyan Shirt
    - jcr:description = Flowery, all-cotton shirt.
    - price = 14.00
    + banyan_shirt_s
        - cq:commerceType = variant
        - size = S
        + banyan_shirt_s_red
            - cq:commerceType = variant
            - color = red
        + banyan_shirt_s_blue
            - cq:commerceType = variant
            - color = blue
    + banyan_shirt_m
        - cq:commerceType = variant
        - size = M
        + banyan_shirt_m_red
            - cq:commerceType = variant
            - color = red
        + banyan_shirt_m_blue
            - cq:commerceType = variant
            - color = blue
    + banyan_shirt_l
        - cq:commerceType = variant
        - size = L
        + banyan_shirt_l_red
            - cq:commerceType = variant
            - color = red
        + banyan_shirt_l_blue
            - cq:commerceType = variant
            - color = blue
    + banyan_shirt_xl
        - cq:commerceType = variant
        - size = XL
        - price = 18.00
```

Architecture of the Shopping Cart

Components

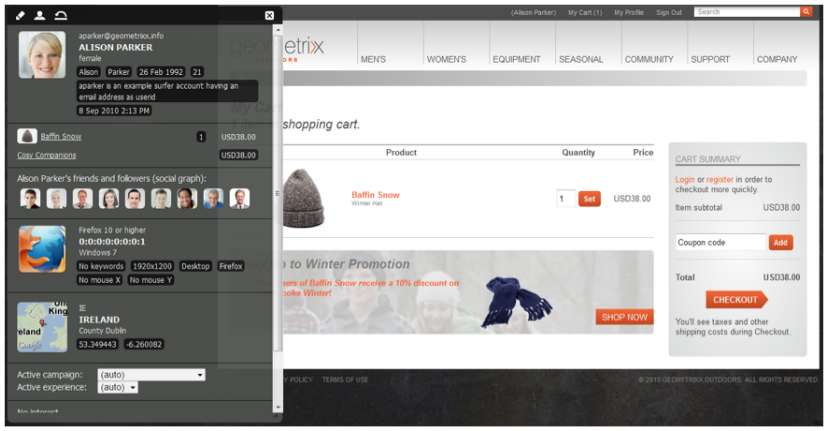
- The shopping cart is owned by the CommerceSession:
    - The CommerceSession performs add/remove/etc.
    - The CommerceSession also performs the various calculations on the cart.
- While not directly cart-related, the CommerceSession must also provide catalog pricing information (since it owns pricing)
    - Pricing might have several modifiers:
        - Quantity discounts.
        - Different currencies.
        - VAT-liable and VAT-free.
    - The modifiers are completely open-ended with the following interface:
        - int CommerceSession.getQuantityBreakpoints(Product product)
        - String CommerceSession.getProductPrice(Product product)

Storage

- Storage
  - In the hybris case, the hybris server owns the cart.
  - In the AEM-native case carts of are stored in the ClientContext

Personalization

- Personalization should always be driven through the ClientContext.
- A ClientContext /version/ of the cart is created in all cases:
  - Products should be added by using the CommerceSession.addCartEntry() method.
- The following illustrates an example of cart information in the ClientContext cart:



Architecture of Checkout

Cart and Order Data

The CommerceSession owns the three elements:

1. Cart contents
2. Pricing
3. The order details

1. **Cart contents**
   The cart contents schema is fixed by the API:

```
public void addCartEntry(Product product, int quantity);
public void modifyCartEntry(int entryNumber, int quantity);
public void deleteCartEntry(int entryNumber);
```

2. **Pricing**
   The pricing schema is also fixed by the API:

```
public String getCartPreTaxPrice();
public String getCartTax();
public String getCartTotalPrice();
public String getOrderShipping();
public String getOrderTotalTax();
public String getOrderTotalPrice();
```

3. **Order Details**
   However, order details are *not* fixed by the API:

```
public void updateOrderDetails(Map<String, String> orderDetails);
public Map<String, String> getOrderDetails();
public void submitOrder();
```

Shipping Calculations

- Order forms often need to present multiple shipping options (and prices).
- The prices might be based on items and details of the order, such as weight and/or delivery address.
- The CommerceSession has access to all the dependencies, so it can be treated in a similar manner as product pricing:
  - The CommerceSession owns shipping pricing.

- Can retrieve/update delivery details by using updateOrder(Map<String, Object> delta)

NOTE

You could implement a shipping selector; for example: yourProject/commerce/components/shippingpicker:

- Essentially this could be a copy of foundation/components/form/radio, but with callbacks to the CommerceSession for:
  - Checking if the method is available
  - Adding pricing information
  - To enable shoppers to update the order page in AEM (including the superset of shipping methods and the text describing them), while still having the control to expose the relevant CommerceSession information.

Payment Processing

- The CommerceSession also owns the payment processing connection.
- Implementors need to add specific calls (to their chosen payment processing service) to the CommerceSession implementation.

Order Fulfillment

- The CommerceSession also owns the fulfillment connection.
- Implementors will need to add specific calls (to their chosen payment processing service) to the CommerceSession implementation.

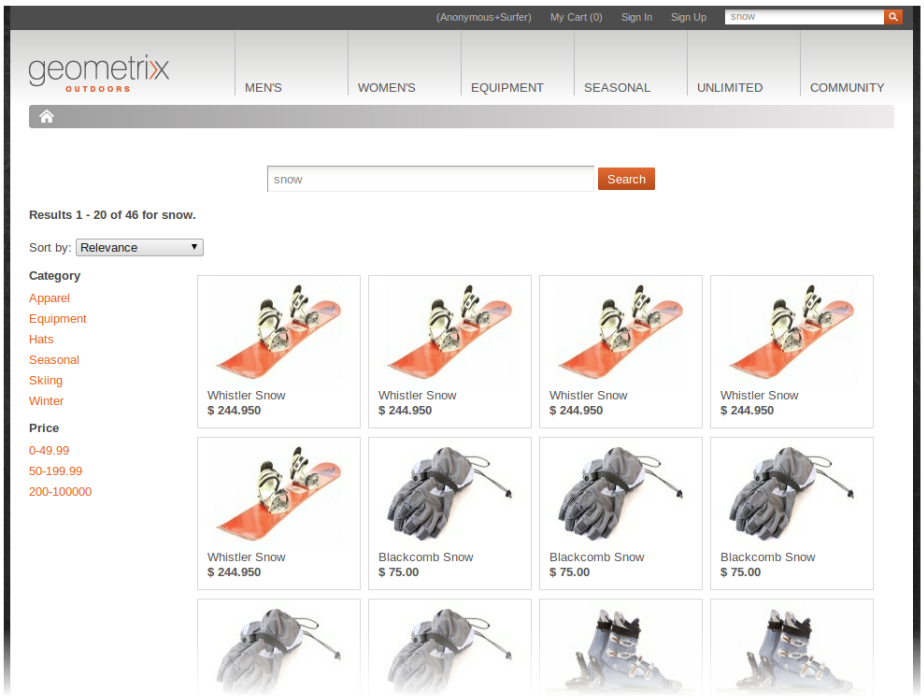
SEARCH DEFINITION

Following the standard service API model, the eCommerce project provides a set of search-related APIs that can be implemented by individual commerce engines.

NOTE

Currently, only the hybris engine implements the search API out-of-the-box. However, the search API is generic and can be implemented by each CommerceService individually.

The eCommerce project contains a default search component, located in:
/libs/commerce/components/search



This makes use of the search API to query the selected commerce engine (see eCommerce Engine Selection):

Search API

There are several generic / helper classes provided by the core project:

1. CommerceQuery
   Is used to describe a search query (contains information about the query text, current page, page size, sort and selected facets). All eCommerce services that implement the search API will receive instances of this class in order to perform their search. A CommerceQuery can be instantiated from a request object (HttpServletRequest).
2. FacetParamHelper
   Is a utility class that provides one static method - toParams - that is used for generating GET parameter strings from a list of facets and one toggled value. This is useful on the UI side, where you need to display a hyperlink for each value of each facet, such that when the user clicks on the hyperlink the respective value is toggled (i.e. if it was selected it is removed from the query, otherwise added). This takes care of all the logic of handling multiple/single-valued facets, overriding values, etc.

The entry point for the search API is the CommerceService#search method which returns a CommerceResult object. See the API Documentation for more information on this topic.

USER INTEGRATION

Integration is provided between AEM and various eCommerce systems. This requires a strategy for synchronizing shoppers between the various systems so that AEM-specific code only has to know about AEM and vice-versa:

- Authentication
  AEM is presumed to be the *only* web front-end and therefore performs *all* authentication.
- Slave Accounts
  AEM creates a slave account in hybris for each shopper. The username of the slave account is the same as the AEM username. A cryptographically-random password is auto-generated and stored (encrypted) in AEM.

Pre-existing Users

A AEM front-end can be positioned in front of an existing hybris implementation. Also a hybris engine can be added to an existing AEM installation. To do this, the systems must be able to gracefully handle existing users in either system:

- AEM -> hybris
  - When logging in to hybris, if the AEM user does not already exist:
    - create a new hybris user with a cryptographically random password
    - store the hybris username in the user directory of the AEM user
  - See: com.adobe.cq.commerce.hybris.impl.HybrisSessionImpl#login()
- hybris -> AEM
  - When logging in to AEM, if the system recognizes the user:
    - attempt to log in to hybris with supplied username/pwd
    - if successful, create the new user in AEM with the same password (AEM-specific salt will result in AEM-specific hash)
  - The above algorithm is implemented in a Sling AuthenticationInfoPostProcessor
    - See: com.adobe.cq.commerce.hybris.impl.user.LazyUserImporter.java

CUSTOMIZING THE IMPORT PROCESS

To build upon existing functionality your custom import handler:

- has to implement the ImportHandler interface
- can extend the DefaultImportHandler

```
/**
 * Services implementing the <code>ImportHandler</code> interface are
 * called by the {@link HybrisImporter} to create actual commerce entities
```

```
 * such as products.
 */
public interface ImportHandler {

    /**
     * Not used.
     */
    public void createTaxonomie(ImporterContext ctx);

    /**
     * Creates a catalog with the given name.
     * @param ctx   The importer context
     * @param name  The catalog's name
     * @return Path of created catalog
     */
    public String createCatalog(ImporterContext ctx, String name) throws Exception;

    /**
     * Creates a product from the given values.
     * @param ctx               The importer context
     * @param values            The product's properties
     * @param parentCategoryPath The containing category's path
     * @return Path of created product
     */
    public String createProduct(ImporterContext ctx, ValueMap values, String parentCategoryPath)
 throws Exception;

    /**
     * Creates a variant product from the given values.
     * @param ctx             The importer context
     * @param values          The product's properties
     * @param baseProductPath The base product's path
     * @return Path of created product
     */
    public String createVariantProduct(ImporterContext ctx, ValueMap values, String baseProductPath)
 throws Exception;

    /**
     * Creates an asset for a product. This is usually a product
     * image.
     * @param ctx             The importer context
     * @param values          The product's properties
     * @param baseProductPath The product's path
     * @return Path of created asset
     */
    public String createAsset(ImporterContext ctx, ValueMap values, String productPath) throws
 Exception;

    /**
     * Creates a category from the given values.
     * @param ctx           The importer context
     * @param values        The category's properties
     * @param parentPath    Path of parent category or base path of import in case of root category
     * @return Path of created category
     */
    public String createCategory(ImporterContext ctx, ValueMap values, String parentCategoryPath)
 throws Exception;
}
```

For your custom handler to be recognized by the importer, it must specify the service.ranking property with a value higher than 0; for example:

```
@Component
@Service
@Property(name = "service.ranking", value = 100)
public class MyImportHandler extends DefaultImportHandler {
    ...
}
```