

Responsive Storefronts

The hybris Accelerator provides a starting point for setting up a responsive storefront that is configurable and ready for development.

Overview: The hybris Accelerator provides a base implementation of a responsive storefront that includes a template for developing a complete responsive site. This functionality does not come enabled as part of the default configuration of the hybris Accelerator: certain configurations need to be made to proceed with responsive development (as described in the following sections). The responsive Accelerator does not include the full functionality of the desktop Accelerator, but is rather a starting off point for developing a complete responsive site. The Accelerator provides two different combinations of user experience: the desktop/mobile user experience, or the responsive/mobile user experience.

Note: The responsive storefront does not explicitly support AddOns at this time.

Enabling the Responsive Experience: To enable the responsive Accelerator, you need to configure the **commerceservices.default.desktop.ui.experience** property. This property can be set to either **desktop** or **responsive**. The **commerceservices.default.desktop.ui.experience** property decides which paths the view resolvers use and which impex files the import services load during initialization. Previously, the desktop and the mobile UI experiences could be used at the same time. Now, the server can also use the responsive and the mobile UI experiences at the same time. To switch between these two combinations, desktop/mobile or responsive/mobile, set the property **commerceservices.default.desktop.ui.experience** in the appropriate properties file (defined in the commerce services **project.properties** file, and in the **local.properties** file that is located in the **config** folder) to **desktop** or **responsive**, respectively. If the property is not defined, the hybris Accelerator will default to desktop.

Impex Import Convention: Enabling the responsive storefront requires a re-initialization of the storefronts to update the different components and configurations required for the responsive UI experience. The data import works by applying a patch-like import over the classic desktop data. On initialization, the desktop data is imported, and if the property is set to **responsive**, the CMS responsive content is then loaded next. The convention loads the core data **site-responsive.impex**, **store-responsive.impex** and **cms-responsive-content.impex** files, as well as the sampled data **cms-responsive-content.impex** file of the store extension.

Note: The **CoreDataImportService** and **SampleDataImportService** have been modified to apply these responsive patch impex files.

Unimplemented Static Pages: The hybris Accelerator provides a minimal set of pages to get you started with a responsive site. However, not all of the storefront pages are fully implemented as responsive. Some pages provide a static markup implementation only.

For the static markup pages, the following global warning message is

shown: **Information:PageUnder Construction - Not Completely Functional".**

The following is a list of the static markup pages:

- All account pages
- All search pages
- `productGridPage.jsp`
- `productListPage.jsp`
- `categoryPage.jsp`
- `checkoutLoginPage.jsp`
- `serverError.jsp`
- `storeFinderSearchPage.jsp`

Updates for Responsive Pages: To release a functional subset of pages for the responsive storefront, some features have been modified or left out for later implementation. The most important features that are affected are those in the checkout flow, as described in the following sections.

Cart Page Modifications: On the **Cart** page, the option to choose **Pick Up** or **Ship** for a line item has been removed. Instead, each item lists the delivery option that was selected when the product was added to the cart. To update these fields, a customer must add the product to the cart with the appropriate delivery option (**Pick Up** or **Ship**) from the **Product Details** page.

Also note, updating the quantity of any item now takes effect after the Return key is pressed, or the input box loses focus.

Checkout Step Modifications: In the classic checkout flow, the first checkout step involves selecting a shipping address (that is, if there are items to be shipped in the cart), and the second checkout step involves selecting and consolidating the pick-up-in-store options (that is, if there are pick-up-in-store items in the cart). In the responsive checkout flow, the first checkout step combines both the selection of a shipping address as well as any pick-up-in-store options into a single page. Note that in this step, the consolidation of pick-up-in-store is not available. The customer is able to view the stores where their pickup items are available, but they cannot consolidate or modify these stores in this checkout step.

Checkout Flow Group For Responsive:The checkout flow in the responsive checkout is handled by a checkout flow group called the **responsiveCheckoutGroup**. In the responsive checkout flow, the step for consolidating store locations for pick-up-in-store items has been removed. The hybris Accelerator provides the appropriate impex files to update the **checkoutFlowGroup** for the electronics and apparel base stores.

The following sample impex updates a base store's **checkoutFlowGroup** to use the responsive checkout flow:

```
$storeId=electronics
```

```
$checkoutFlowGroup=responsiveCheckoutGroup
```

Updating the checkout flow group of Electronics Base Store to responsive

```
INSERT_UPDATE BaseStore;uid[unique=true];checkoutFlowGroup;  
;$storeId;$checkoutFlowGroup;
```

Note:This affects the checkout flow for both the mobile and the responsive storefronts.














Google Maps Embed API Key For PDP Pick-Up-In-Store:The Google Maps Embed API is used to enable the Pick-Up-In-Store pop-up window to display a map that shows the location of a selected store. This implementation requires an associated Google account, as well as a generated Google Maps Embed API key for that account.

Add the API key to the **project.properties** file, as follows:

```
googleEmbedApiKey=
```

Note: You are responsible for getting your own API Key. For more information, see <https://developers.google.com/maps/documentation/embed/guide>.

Available Responsive WCMS Components: The responsive implementation of the hybris Accelerator does not define all of the WCMS components that are included in the desktop implementation of the Accelerator. However, there is no restriction to prevent the addition of these components to a responsive page. The fully functional responsive pages use the WCMS components that are defined. You can see which components are defined for responsive by referring to `/yacceleratorstorefront/web/webroot/WEB-INF/views/responsive/cms/`. If a non-implemented component is required, you can provide a JSP definition in this directory. For a complete list of components and their availability in the responsive storefront, expand the following table:

| CMS Component | Available for Responsive Storefront |
|--|---|
| accountnavigationcollectioncomponent.jsp |  |
| accountnavigationcomponent.jsp |  |
| addtocartaction.jsp |  |
| bannercomponent.jsp |  |
| breadcrumbcomponent.jsp |  |
| categoryfeaturecomponent.jsp |  |
| cmsimagecomponent.jsp |  |
| cmsproductlistcomponent.jsp |  |
| cmstabparagraphcomponent.jsp |  |
| dynamicbannercomponent.jsp |  |
| footercomponent.jsp |  |
| imagemapcomponent.jsp |  |
| jspincludecomponent.jsp |  |
| languagecurrencycomponent.jsp |  |
| listaddtocartaction.jsp |  |

| CMS Component | Available for Responsive Storefront |
|--------------------------------------|---|
| listpickupinstoreaction.jsp |  |
| minicartcomponent.jsp |  |
| navigationbarcollectioncomponent.jsp |  |
| navigationbarcomponent.jsp |  |
| pickupinstoreaction.jsp |  |
| productaddtocartcomponent.jsp |  |
| productcarouselcomponent.jsp |  |
| productfeaturecomponent.jsp |  |
| productgridcomponent.jsp |  |
| productreferencescomponent.jsp |  |
| productrefinementcomponent.jsp |  |
| productvariantselectorcomponent.jsp |  |
| rotatingimagescomponent.jsp |  |
| searchboxcomponent.jsp |  |
| searchresultsgridcomponent.jsp |  |
| searchresultslistcomponent.jsp |  |
| shareonsocialnetworkaction.jsp |  |
| simplebannercomponent.jsp |  |
| simpleresponsivebannercomponent.jsp |  |
| simplesuggestioncomponent.jsp |  |
| vieworderaction.jsp |  |
| viewstoreaction.jsp |  |

Responsive Images:

In the hybris Accelerator, the **simpleResponsiveImageComponent** handles images in a responsive way. This component is currently implemented within the storefront **Homepage**.

Overview: The hybris Accelerator has a WCMS **ImageComponent** that is used for containing media, and can be configured for different image components corresponding to different UI experience levels. However, unlike the desktop or mobile UI experience, the image solution in a responsive UI experience requires multiple images to be uploaded for the same image component, where the best image is then selected based on the screen width. In this case, the CMS component is mapped to a media container that holds different images, as well as to a JavaScript library that can detect the browser width and pixel density to select the appropriate image.

Abstract Responsive Banner Component: The hybris Accelerator provides the **AbstractResponsiveBannerComponent**, which is mapped to a media container that holds multiple images. The **SimpleResponsiveBannerComponent** is also provided by default as a concrete implementation of this abstract class. The **Imager.min.js** JavaScript library handles the images and their respective sizes as key-value pairs. Media files are therefore mapped to the following media formats, which are specific to a responsive UI experience: **widescreen**, **desktop**, **tablet**, and **mobile**. It is important to note that there is not necessarily a one-to-one mapping between the formats and the breakpoints. If the Imager library determines that an image with a mobile format is more suitable to a tablet element, then this will be selected.

Responsive Image Populators: The hybris Accelerator provides two populators to handle responsive images: the **ResponsiveMediaContainerPopulator** and the **ResponsiveImagePopulator**. These populators extract a key-value pair, where the key is the image width, and the value is the image URL that is provided to the JavaScript implementation. The **ResponsiveMediaContainerPopulator** is responsible for sending the appropriate information to the **ResponsiveImagePopulator**, as well as for sorting the media widths after the images have been populated. This is done to conform with the Imager implementation, which is explained in greater detail in the [Imager Implementation](#) section below.

Default Filename Parsing and Convention: Each media that is uploaded as part of the responsive image component is required to have a certain naming convention to identify the image size. The default implementation looks for `-XXXx or [_][0-9]+[xX][0-9]+[_]`, where XXX is the width of the image. If the image name is **sample_landing_home_page_150x70.png**, the regex filters the filename as **-150x70**, and from this, **150** is selected as the width. Since each responsive image component is mapped to a media container, the **ResponsiveMediaContainerPopulator** and the **ResponsiveImagePopulator** are used to fetch the required information for the image width and

the URL. The **ResponsiveImagePopulator** contains the parsing logic to fetch the width information from the media. It uses a regex pattern that can be customized to a particular implementation.

Fallback Parsing: If the default parsing logic fails, or you opt to use a fallback strategy in your customization, then the mapping in the example below will be used. Since each media has **mediaFormat** qualifier information, we map these media qualifiers to a predefined width in the **spring.xml** file. However, this is not accurate for images that occupy only half the width of the browser, or for images that are enclosed in a **<div>** that occupies less than the full width of the browser, because Imager takes the parent width of the element inside which the image is enclosed. For example, when the **<div>** enclosing an image is just 50% of the screen width, the appropriate image will not be rendered for that particular breakpoint. The Imager will instead select a bigger image. As of now, this mapping is configured in the **spring.xml** file of the **acceleratorfacades** extension. This can be changed to breakpoint value, or screen width. Since the Imager JS looks for image width, we have used the breakpoint width.

```
<alias alias="responsiveImageFormats" name="responsiveImageFormats" />

<util:map id="responsiveImageFormats" map-class="java.util.LinkedHashMap"
  value-type="java.lang.Integer">
  <entry key="widescreen" value="1200"/>
  <entry key="desktop" value="992"/>
  <entry key="tablet" value="768"/>
  <entry key="mobile" value="480"/>
</util:map>
```

Sample Impex: Once all the configurations are completed, you must provide the appropriate resources to define a **SimpleResponsiveBannerComponent**. These resources include media files, media formats, media containers, as well as concrete **SimpleResponsiveBannerComponent**, as follows:

```
INSERT_UPDATE
Media;mediaFormat(qualifier);code[unique=true];@media[translator=de.hybris.platform
orm.impex.jalo.media.MediaDataTranslator]
[forceWrite=true];realfilename;altText;mime[default='image/jpeg'];
$contentCV[unique=true];folder(qualifier)[default=images];
;mobile;Elec_480x320_HomeSpeed_EN_01_480W.jpg;
$siteResource/images/banners/homepage/responsive/Elec_480x320_Home
Speed_EN_01_480W.jpg;Elec_480x320_HomeSpeed_EN_01_480W.jpg;"Fast
Precise";
;tablet;Elec_770x350_HomeSpeed_EN_01_770W.jpg;
$siteResource/images/banners/homepage/responsive/Elec_770x350_HomeSpeed_EN_01_77
0W.jpg;Elec_770x350_HomeSpeed_EN_01_770W.jpg;"Fast Precise";
```

```
;desktop;Elec_960x330_HomeSpeed_EN_01_960W.jpg;
$siteResource/images/banners/homepage/responsive/Elec_960x330_Home
Speed_EN_01_960W.jpg;Elec_960x330_HomeSpeed_EN_01_960W.jpg;"Fast
Precise";
;widescreen;Elec_1400x440_HomeSpeed_EN_01_1400W.jpg;
$siteResource/images/banners/homepage/responsive/Elec_1400x440_HomeSpeed_EN_01_1
400W.jpg;Elec_1400x440_HomeSpeed_EN_01_1400W.jpg;"Fast Precise";

INSERT_UPDATE MediaContainer;qualifier[unique=true];$medias;
$contentCV[unique=true]
;electronics-homepage-splash-
en;Elec_480x320_HomeSpeed_EN_01_480W.jpg,Elec_770x350_HomeSpeed_EN_01_770
W.jpg,Elec_960x330_HomeSpeed_EN_01_960W.jpg,Elec_1400x440_HomeSpeed_EN_01
_1400W.jpg
INSERT_UPDATE SimpleResponsiveBannerComponent;
$contentCV[unique=true];uid[unique=true];$mediaContainer
;;ElectronicsHomepageSplashBannerComponent;electronics-homepage-splash-en
```

Markup and Lazy Loading: In general, images are rendered by tags in HTML. This works when there is only one URL for the image. But in a responsive design, multiple images are required, and as a result, different URLs for the tag's src attribute are also required.

Imager Implementation: The hybris Accelerator provides an **Imager.min.js** that is based on the BBC Imager, which selects the image based on the screen width and the parent element size. The Imager javascript looks for an element with the class attribute **js-responsive-image** in the browser's DOM, and if it finds one, it gets the JSON data from the data-media attribute. The following is an example: **Sample markup for the Imager JavaScript to render images**

```
<img class="js-responsive-image" data-media=
'{"100":"Image of width 100x50",
"200":"Image of width 200x50",
"400":"Image of width 400x50",
"600":"Image of width 600x50",
"800":"Image of width 800x50",
"1000":"Image of width 1000x50"}'
alt="">
```

The **Imager.min.js** processes the JSON and loads the key value to available widths. It then determines the size of the parent element and creates the **src** attribute for the image tag based on the selected key.

Note: The Imager requires that the list of available widths be sorted in ascending order to be able to properly select the appropriate image width.