ACTIVE COMMERCE®
TAKE BACK YOUR CUSTOMER.

SKINNING SYSTEM GUIDE

SITECORE E-COMMERCE DONE RIGHT. ➡ ACTIVECOMMERCE.COM

Active Commerce Skinning System Guide

Overview

The Active Commerce Skinning System is designed to allow developers to dynamically replace the files used for Layouts, Sublayouts, LESS/CSS, JS, and theoretically any asset on a per-website basis in a Sitecore install. This is meant to allow quick implementations of Active Commerce, but also allows ultimate flexibility to replace styles or even full markup, without having to change Layout/Sublayout paths in Sitecore or change the Sublayouts that are used for a particular website. For styling, the system takes advantage of LESS (precompiler for CSS), to further speed skin customization and creation. Last, but not least, web optimization is built in, using the excellent Cassette plug-in, giving us all the benefits of combination, minification, and cache busting to reduce page load time.

The Basics

- Active Commerce base skin assets are stored under the web application root at /skins/. *(NOTE: this can be changed if necessary. Under App_Config\Include\ActiveCommerce.Skinning.config, modify the value stored in "ActiveCommerce.Skinning.Path" setting)*
- The base folder contains skin assets, subfolders for more skin assets (e.g., styles, scripts, templates) and subfolders for available skins.
- *IMPORTANT: Modifications to the base skin should be avoided; instead, a new skin should be created. This will make upgrading Active Commerce much easier.*
- Your skin should contain subfolders for base assets that will be overridden (e.g., styles, scripts, templates) and subfolders for available sub-skins.
- Skins can be further nested, without any theoretical limit.
- The skin used by a Site is configured on its Sitecore <site /> node, using the attribute s*kin*. The value should be the folder name of the skin. Sub-skin paths are separated by a forward-slash. e.g., *skin="omg/zomg"*
- When skin assets are referenced, the skinning token ~SKIN~ is replaced by the path to the asset within the appropriate skin path.
- Whether skinning a layout, sublayout, or other asset, the skinning system will first check for the existence of the file within the configured sub-skin, then climb up the parent skins until the desired asset is found. This means that creating a skin doesn't mean copying or overriding every asset. (Please don't.)
- So, going forward, ~SKIN~ (aka skin token) = resolved skin path. You'll see this referenced throughout this document and also throughout the code.
- **When deploying to production, be sure that you have *debug="false"* on the system.web/compilation element of Web.config.** This enables CSS and Javascript minification within Cassette. Setting *debug="true"* makes debugging easier.
    - For more info on Cassette: http://getcassette.net/documentation/v2/

Creating a New Skin

1. Pull down the scaffolding provided here: https://github.com/ActiveCommerce/skin-scaffolding
2. Rename “new-skin” to your skin name and copy the folder into your web project under “skins.”
3. In Visual Studio, make sure all .less files are set to Build Action of “Content” (under Properties).
4. Set the “skin” attribute of your site definition to your skin name. For example:

```
<site name="sherpa_winter_outfitters"
        hostName="sherpa.dev.activecommerce.com"
        physicalFolder="/"
        rootPath="/sitecore/content/Sherpa Winter Outfitters"
        ...
        skin="sherpa"
    />
```

Layouts & Sublayouts

- To skin a Layout or Sublayout, create an aspx or ascx file under your skin folder with the same name as the base file you wish to override, e.g., ~/skins/omg/Compare.ascx
- If you wish to use the same code-behind, delete the code-behind file(s), and modify the control to inherit from the corresponding base control (*NOTE: you’ll need to add a reference to ActiveCommerce.Web*).
- If you wish to extend the code-behind, inherit your code-behind from the corresponding base control.

Digging Deeper

If inserting a skinned Sublayout statically, i.e., in your Webforms markup, use the *ac:SkinnedSublayout* control instead of the *sc:Sublayout* control and be sure to include the skin token in the path, e.g., <ac:SkinnedSublayout ID="SkinnedSublayout1" runat="server" Path="/~SKIN~/Sublayout.ascx" />

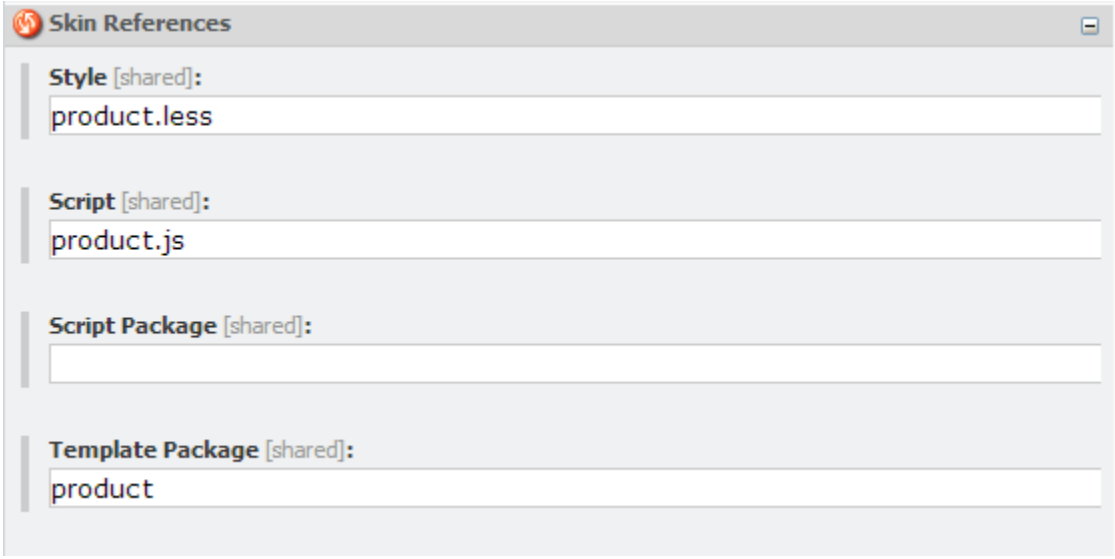
If adding a new Sublayout (not overriding base), and you’d like to take advantage of the skinning system’s ability to dynamically include assets and also allow sub-skins to override, follow these steps to make it “skinnable”:

- In Sitecore, on the Content tab of the Sublayout item, change its path to include the skinning token, e.g., /~SKIN~/Sublayout.ascx. (*NOTE: the Grid Designer won't work at this time, due to our dynamic path replacement.*)

- Change the template of the sublayout to “Skinned Sublayout” (/sitecore/templates/ActiveCommerce/Skinning/Skinned Sublayout). (*NOTE: be aware that due to a current bug in Sitecore, the Page Editor will no longer work for this sublayout*)

Adding Asset References

- Style, Script, and HTML Template are defined on Active Commerce sublayouts (“Skinned Sublayout”).
- To modify default references, or add new ones, navigate to the appropriate Active Commerce sublayout in Sitecore (those under /sitecore/layout/Sublayouts/ActiveCommerce/)
- Look for the “Skin References” section in the editor.



(*NOTE: Ignore the “Library References” section. These are core libraries, essential to Active Commerce.*)

- Add/remove assets as necessary. To include more than one, use a pipe-delimited format, e.g., cart.less|order.less
- See specific asset sections below for more information on each type.

Digging Deeper

The references are actually added through a custom processor in the renderLayout Sitecore pipeline. (Configured in xActiveCommerce.Skinning.config).

Styles

Active Commerce stylesheets are built with LESS. If you’re unfamiliar, checkout the official site here to learn more. The biggest pieces we’ve taken advantage of are variables and mixins.

While it's not required to use LESS in your skin (plain css works just fine), it will definitely help make things easier.

You'll want to make sure .less files are supported in your IDE. The latest version of Visual Studio VS2012 supports LESS natively. If you're using an older version there are Extensions to support LESS.

Recommended Approach

1. **Start with the scaffolding** (see *Creating a New Skin* section).
2. **Override base variables** in /styles/common/variables.less. Available base variables are at ~/skins/styles/common/variables.less. Alternatively, copy the base as a start and tweak away. Configure as many things as makes sense for your skin's design here (fonts, headers, site width, colors, buttons, etc). The goal is to get the site looking as close to your design as possible so you won't have to manually override as many styles later on. Let Active Commerce and .less do the work for you! You can also declare any new global skin variables here.
3. (optional) **Add additional mixins** in /styles/common/mixins.less. There are a bunch already included in the base (~/skins/styles/common/mixins.less), so no need to re-add those.
4. (optional) **Override page-specific variables** throughout. Look for "// DEFINE OVERRIDES OF xxxxxx VARIABLES HERE" comments.
5. **Add necessary styles** throughout. Look for "// DEFINE ADDITIONAL xxxxxx STYLES HERE" comments. Don't forget to take advantage of variables and included mixins (rounded-corners, gradients, and all that good stuff).

Including a New Stylesheet (less or css) File

1. Create a file under your skin "styles" folder, e.g, ~/skins/omg/styles/foo.less
2. Add "foo.less" as a "Style" reference (*see Adding Asset References*) to all sublayouts that require it.

Scripts

All base scripts use modules defined as object literals. This makes it easy for you to tweak as necessary, without having to make modifications directly. Each module extends a base "ActiveCommerce" object, and also provides a "config" object where applicable.

Scripts come in two flavors — files and packages. A file is what you'd expect — a single .js file. A package is represented by a folder (folder name = the package name). All files within that folder will be combined into a single file (i.e., request) at runtime.

Extending a Base Skin Javascript File or Package

1. This will allow you to tweak the configuration of base scripts, add functionality, or override properties and methods entirely.
2. Create a file under your skin "scripts" folder with the same name as the base file you wish to extend, e.g., ~/skins/omg/scripts/product.js
3. For packages, same rules apply, but also place the file under the same folder name, e.g., ~/skins/omg/scripts/main/app.js
4. This file will automatically be included on all pages that the base file (or package) is included on, *after* the base skin file, of course.

Example – Overriding default page size of browse products page

1. Add a new file under your skin "scripts" folder, named "browse-product.js".
2. Enter the following text: `ActiveCommerce.BrowseProduct.config.productsPerPage = 20;`
3. The browse product page will now show 20 products.

Including a New Javascript File

1. Create a file under your skin "scripts" folder, e.g., ~/skins/omg/scripts/foo.js
2. Add "foo.js" as a "Script" reference (see *Adding Asset References* section) to all sublayouts that require it.
3. Another way to include a new file is to place that file within a known package folder, e.g., ~/skins/omg/scripts/main/foo.js

Including a New Javascript Package

1. Create a folder under your skin "scripts" folder, e.g., ~/skins/omg/scripts/bar
2. Add whichever files (.js) you want to this folder.
3. Add "foo" as a "Script Package" reference (see *Adding Asset References* section) to all sublayouts that require it.

HTML Templates

Active Commerce uses jQuery-tmpl for html templating. If you'll be extending or adding your own, make sure to check out the syntax to ensure compatibility.

Templates are always delivered as packages. A package is represented by a folder (folder name = the package name). All files within that folder will be combined into a single file (i.e., request) at runtime.

Overriding a Base Skin Template

1. Create a folder and file under your skin "templates" folder with the same name as the base file you wish to override, e.g., ~/skins/omg/templates/browse-product/product.htm
2. This file will now override the base file wherever the template is used.

Including a New Template Package

1. Create a folder under your skin “templates” folder, e.g., ~/skins/omg/templates/foo
2. Add whichever files (.htm) you want to this folder.
3. Add “foo” as a “Template Package” reference (see *Adding Asset References* section) to all sublayouts that require it.
4. Templates are automatically precompiled, thanks to the handy Cassette jQuery-tmpl add-in. To reference the pre-compiled jQuery-tmpl template in your javascript, use the following syntax: $.tmpl(<template_filename>, <data>). So, in our example, if we’ve added a file “bar.htm”, it would be $.tmpl(“bar”, <data>).

Images

- Any static file system asset can be "skinned" by using the skin token in the URL.
- The skinning system will intercept the request, determine the best match for the asset within the Website's configured skin, and transfer the request to that asset.
- This can be used for asset references in CSS files, static images on pages, etc.
- Example path: /~SKIN~/image.jpg

Sitecore Media

- This includes assets served up from the Sitecore Media Library, and also Active Commerce Media items (e.g., YouTube/Vimeo videos)
- All images and videos used throughout the base skin use setting values found in xActiveCommerce.Skinning.config.

Overriding Specific Settings in Base Configuration

1. Create a file “xActiveCommerce.Skinning.z<skin_name>.config” under App_Config/Include folder. (*NOTE: The “z” is important here, as we’re taking advantage of Sitecore’s configuration factory and file name loading order.*)
2. Add the following:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
        <sitecore>
        <skins>
        <skin name="base">
                <settings type="ActiveCommerce.Skinning.Settings.SkinSettings,
ActiveCommerce.Kernel">
                        <images hint="raw:AddImage">
                                <image name="Browse.Category">
                                        <Width>300</Width>
                                        <Height>100</Height>
                                </image>
```

```
                    </images>
                    <videos hint="raw:AddVideo">
                        <video name="Generic.Video">
                            <Width>700</Width>
                            <Height>500</Height>
                        </video>
                    </videos>
                </settings>
            </skin>
            </skins>
            </sitecore>
    </configuration>
```

3. Add entries for whichever base settings you'd like to override. In this case, we've got an override for the image used for category browsing, and also for generic videos.

Overriding All Settings for a Specific Skin

1. Only really applies when you've got multiple skins in use on a single Active Commerce instance.
2. Similar to previous, except you'll need to copy the *entire* list set of settings. This is because, currently, a skin config setting will not fallback to base.
3. Change the name of the <skin> node "name" attribute to your skin name.
4. File should look something like this:

```
<configuration xmlns:patch="http://www.sitecore.net/xmlconfig/">
        <sitecore>
        <skins>
        <skin name="omg">
            <settings type="ActiveCommerce.Skinning.Settings.SkinSettings,
ActiveCommerce.Kernel">
                ALL SKIN SETTINGS FOR OMG SKIN
            </settings>
        </skin>
        <skin name="foo">
            <settings type="ActiveCommerce.Skinning.Settings.SkinSettings,
ActiveCommerce.Kernel">
                ALL SKIN SETTINGS FOR FOO SKIN
            </settings>
        </skin>
        </skins>
        </sitecore>
</configuration>
```

Email templates

- Must be done manually (no css here). Includes the following files:
    - ~SKIN~\Mail-OrderReceipt.ascx (Order Receipt)
    - ~SKIN~\Mail-Generic.ascx (all others)
- Follow instructions for Sublayouts.
- Corresponds to templates located here: /sitecore/content/<site name>/Email Templates

Mobile

Active Commerce uses RESS (Responsive design plus Server Side components) to deliver a mobile optimized view on a single code base.

Device Detection

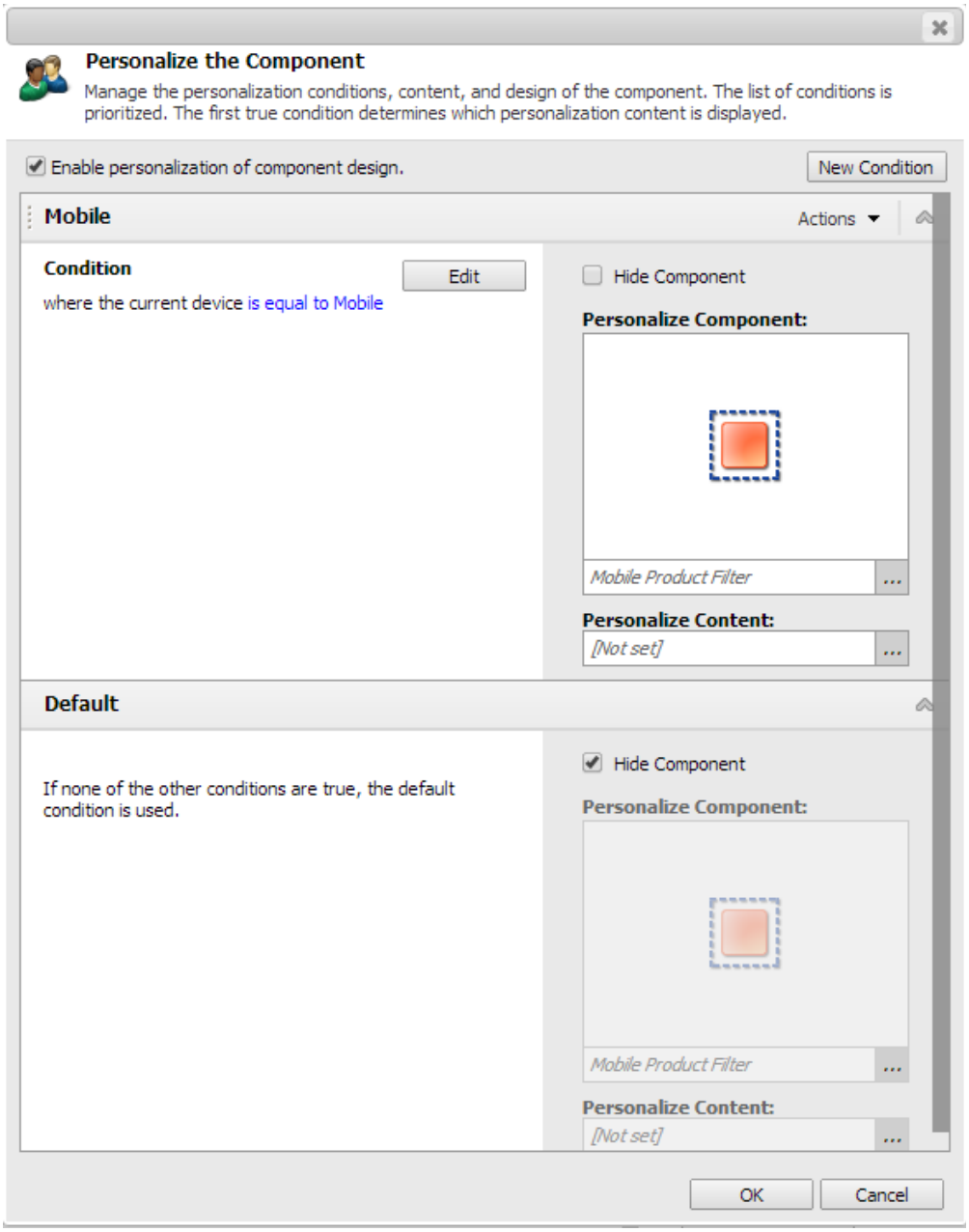
Device detection is necessary to allow Sitecore to change the context device based on the end user's current device. A device resolver such as one of the following is recommended.

- Sitecore Mobile Device Detector: http://marketplace.sitecore.net/en/Modules/Mobile_Device_Detector.aspx
- Sitecore Responsive Device Resolver: http://marketplace.sitecore.net/en/Modules/Responsive_Device_Resolver.aspx

Component Personalization

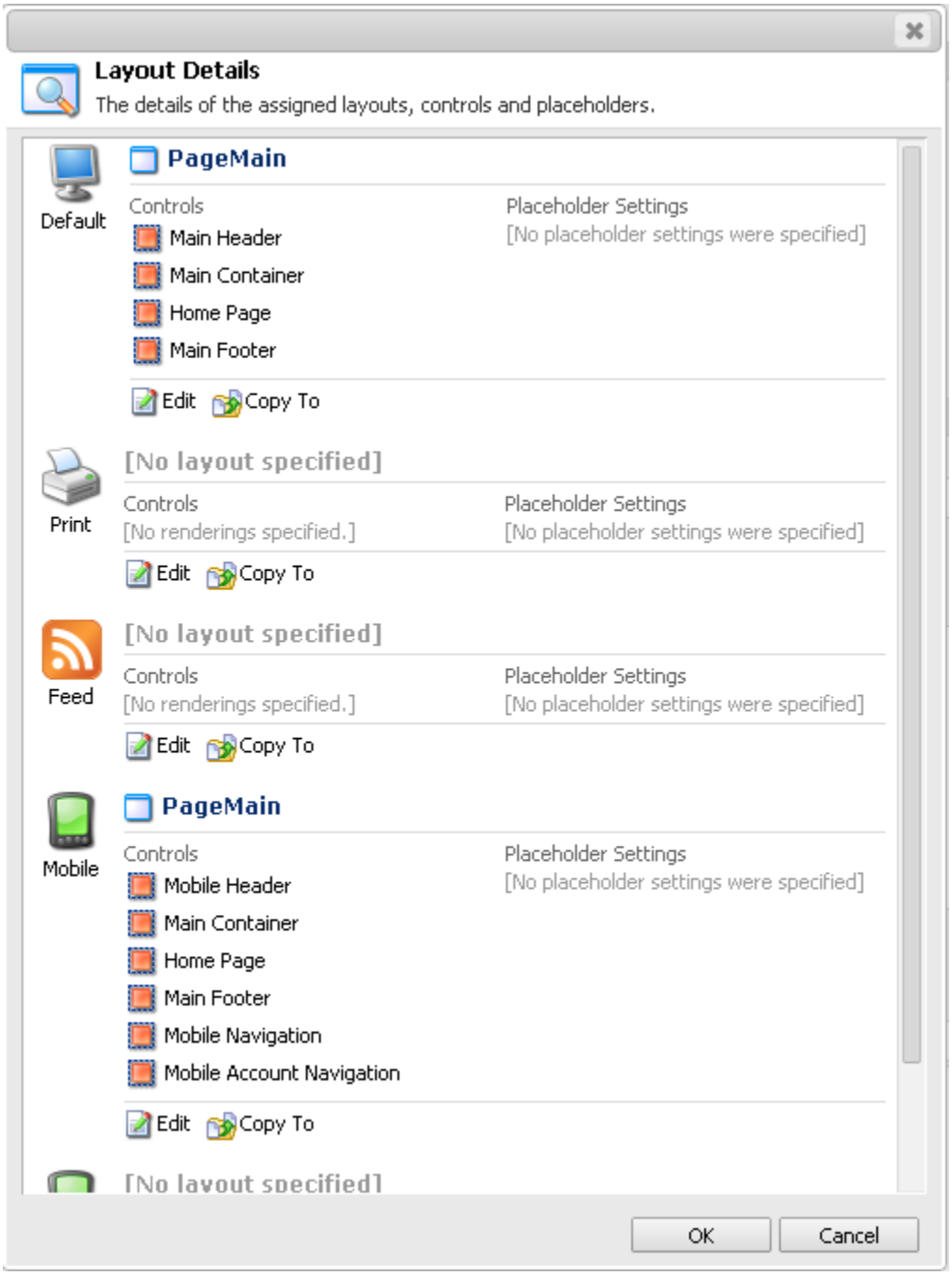
Active Commerce uses Sitecore personalization to conditionally render components based on the current context device as set by the device detection solution you implement. For example, the Mobile Navigation sublayout is rendered when the context device is Mobile. Otherwise, it is hidden.

Note the use of the Sitecore context device condition not device detection conditions.
For example, use "where the current device is equal to Mobile" and not "where the device is mobile". This is important to take advantage of Active Commerce device extensions such as viewport settings and skinning device key.



Mobile Renderings

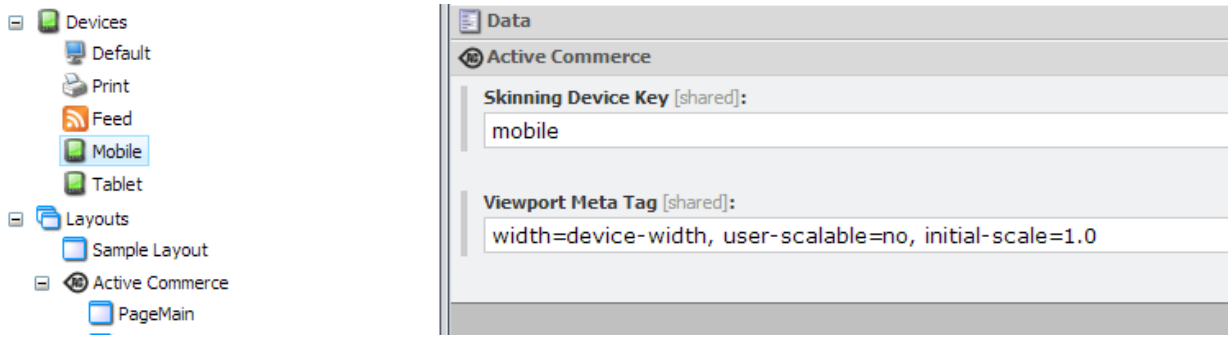
While most items use personalization on the Default device as described above, the shopping cart and homepage take advantage of separate Mobile renderings to be more friendly in Page Editor mode. Shown below are the Layout Details for the homepage.



Notice Personalization is not used in this scenario. Instead, a Mobile rendering is used.

Device Extensions

Active Commerce adds additional fields to the Sitecore Device template for use by the skinning system.



Skinning Device Key

The Skinning Device Key allows you to load stylesheets, images and videos targeted toward end user devices such as mobile phones.

Stylesheets

For stylesheets, the Skinning Device Key is used to load additional stylesheets. For every base stylesheet included, Active Commerce will also look for another file with the following format:

<base name>.<skinning device key>.<less | css>

For example, when viewing a product detail page, if the current Sitecore context device is Mobile, product.less and product.mobile.less will be loaded.

Mobile Optimized Sitecore Media

For images and video that are defined in xActiveCommerce.Skinning.config (or your override config), the Skinning Device Key is used to load an optimized version. When resolving a media item, Active Commerce will look for an override that corresponds to the current key, using the following format:

<base name>.<skinning device key>

For example, a product gallery image has a default entry, Product.GalleryThumbnail. It also has an override, Product.GalleryThumbnail.Mobile. When the current Skinning Device Key equals “mobile”, the image will be rendered as 40 x 40px.

```
<image name="Product.GalleryThumbnail">
        <Width>70</Width>
        <Height>70</Height>
</image>
<image name="Product.GalleryThumbnail.Mobile">
        <Width>40</Width>
        <Height>40</Height>
</image>
```

Viewport Meta Tag

The Viewport Meta Tag content will be written to the viewport meta tag on the PageMain layout. The Mobile device that comes with Active Commerce has a recommended default value included.

Testing Mobile Views

Developers and designers needing to test mobile views can use Chrome emulation on the desktop.

1. In Chrome launch a new Incognito window
2. Activate the developer tools
3. Click the settings icon
4. Select Overrides
5. Check “Show 'Emulation' view in console drawer” and close the settings dialog
6. Show the console if it is hidden
7. In the console, click the Emulation tab
8. Select the device to emulate and click Emulate

(as of Chrome Version 33.0.1750.117 m)

BrowserStack (http://www.browserstack.com/) is a great service for testing your site and skin in more realistic approximations of various mobile devices. Individual pages can also be viewed using the Simulator within Sitecore Preview as shown below.