|  |
| --- |
|  |
| **hybris Multichannel Accelerator Technical Design Guide** |
|  |

Content

[Technical Introduction to hybris 2](#_Toc299098582)

[Features 2](#_Toc299098583)

[Multichannel Accelerator Extensions 5](#_Toc299098584)

[Key Design Principles 6](#_Toc299098585)

[System Architecture of the hybris Multichannel Accelerator 7](#_Toc299098586)

[Using this Document 8](#_Toc299098587)

[Approaching the Documentation 8](#_Toc299098588)

[Understanding Data Model Diagrams 9](#_Toc299098589)

[Multitier Architecture 10](#_Toc299098590)

[Web Server 11](#_Toc299098591)

[hybris Application Server Nodes 12](#_Toc299098592)

[Solr Standalone 18](#_Toc299098593)

[Relational Database 18](#_Toc299098594)

[Other Components Not Covered 18](#_Toc299098595)

[hybris Multichannel Accelerator Extensions 18](#_Toc299098596)

[Extension Structure 20](#_Toc299098597)

[The hybris Multichannel Suite Extension Dependencies 22](#_Toc299098598)

[acceleratorcore Extension - Technical Guide 23](#_Toc299098599)

[acceleratorfacades Extension - Technical Guide 29](#_Toc299098600)

[acceleratorstorefront Extension - Technical Guide 39](#_Toc299098601)

[acceleratorcockpits Extension - Technical Guide 60](#_Toc299098602)

[acceleratorsampledata Extension - Technical Guide 66](#_Toc299098603)

[commerceservices Extension - Technical Guide 70](#_Toc299098604)

[commercefacades Extension - Technical Guide 85](#_Toc299098605)

[Advanced Personalization in the hybris Multichannel Accelerator 103](#_Toc299098606)

[Advanced Personalization Integration 104](#_Toc299098607)

[Business Processes and Eventing in the hybris Multichannel Accelerator 108](#_Toc299098608)

[Business Processes and Eventing 109](#_Toc299098609)

[Customer Reviews in the hybris Multichannel Accelerator 112](#_Toc299098610)

[Language 113](#_Toc299098611)

[Anonymous Reviews 113](#_Toc299098612)

[Rating System 114](#_Toc299098613)

[Commerce Facades 115](#_Toc299098614)

[New Approval Process 116](#_Toc299098615)

[Extended Data Model 117](#_Toc299098616)

[Data Importing Capability in the Multichannel Accelerator 117](#_Toc299098617)

[Hot Folder Data Importing 119](#_Toc299098618)

[Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator 130](#_Toc299098619)

[WCMS Integration 132](#_Toc299098620)

[Email Processes 138](#_Toc299098621)

[Data Model 142](#_Toc299098622)

[Essential and Project Data in the hybris Multichannel Accelerator 143](#_Toc299098623)

[General concept of core vs sample data 144](#_Toc299098624)

[Features 145](#_Toc299098625)

[Acceleratorcore 145](#_Toc299098626)

[Acceleratorsampledata 147](#_Toc299098627)

[Search and Navigation in the hybris Multichannel Accelerator 149](#_Toc299098628)

[Solr Configuration 151](#_Toc299098629)

[Commerce Services 153](#_Toc299098630)

[Commerce Facades 156](#_Toc299098631)

[Storefront 159](#_Toc299098632)

[SEO URLs in the hybris Multichannel Accelerator 161](#_Toc299098633)

[SEO URL Implementation in Accelerator 162](#_Toc299098634)

[Example SEO URLs from Accelerator 163](#_Toc299098635)

[Setting Up the Store Locator in the hybris Multichannel Accelerator 163](#_Toc299098636)

[Adding Points of Sale to a Base Store 165](#_Toc299098637)

[Storefront API Key Configuration 167](#_Toc299098638)

[Setting Site Distance Unit 168](#_Toc299098639)

[CMS Storelocator Pages 168](#_Toc299098640)

[Geolocation Cron Job 168](#_Toc299098641)

[Spring Usage in the hybris Multichannel Accelerator 169](#_Toc299098642)

[Using Spring as IoC Container 169](#_Toc299098643)

[Spring MVC 170](#_Toc299098644)

[Spring Integration 171](#_Toc299098645)

[Spring Security 171](#_Toc299098646)

[Spring Security in the hybris Multichannel Accelerator 171](#_Toc299098647)

[Spring Security 172](#_Toc299098648)

[Configuration 174](#_Toc299098649)

[Stock Management in the hybris Multichannel Accelerator 175](#_Toc299098650)

[Stock Service 176](#_Toc299098651)

[Warehouses 177](#_Toc299098652)

[Stock Levels 177](#_Toc299098653)

[Top Navigation and Link Management in the hybris Multichannel Accelerator 178](#_Toc299098654)

[Top Navigation 179](#_Toc299098655)

[Link Management 181](#_Toc299098656)

[WCMS Integration in the hybris Multichannel Accelerator 181](#_Toc299098657)

[WCMS Integration Overview 183](#_Toc299098658)

[View 188](#_Toc299098659)

[CMS Page Configuration 195](#_Toc299098660)

[Getting Started with the hybris Multichannel Accelerator 199](#_Toc299098661)

[1. Sample Store Set-Up 200](#_Toc299098662)

[2. Source Code Adaptation 201](#_Toc299098663)

[3. Product Catalog Data Modeling 201](#_Toc299098664)

[4. Sample Data Scripts Creation 202](#_Toc299098665)

[Multichannel Accelerator Quick Installation Guide 203](#_Toc299098666)

[Basic Installation 204](#_Toc299098667)

[Accessing the Sample Storefronts 205](#_Toc299098668)

[Customizing the hybris Multichannel Accelerator with Extgen and Modulegen 208](#_Toc299098669)

[Modulegen Configuration and Usage 208](#_Toc299098670)

[Creating customised accelerator system using modulegen 210](#_Toc299098671)

[Storefront and Catalog Modelling in the hybris Multichannel Accelerator 212](#_Toc299098672)

[Sample Storefront Diagram 214](#_Toc299098673)

[Storefront Setup 215](#_Toc299098674)

[CMS Site 215](#_Toc299098675)

[Base Store 216](#_Toc299098676)

[Catalog Data 217](#_Toc299098677)

[Content Catalog 217](#_Toc299098678)

[Product Catalog 218](#_Toc299098679)

[Classification System 220](#_Toc299098680)

[Solr Facet Search 221](#_Toc299098681)

[Customer Data 221](#_Toc299098682)

[Order Data 221](#_Toc299098683)

[Stock 221](#_Toc299098684)

[Developer Trails 222](#_Toc299098685)

[Add New CMS Component Trail 223](#_Toc299098686)

[Motivation 223](#_Toc299098687)

[Tasks 224](#_Toc299098688)

[Summary 231](#_Toc299098689)

[Add New CMS Page Template Trail 231](#_Toc299098690)

[Motivation 231](#_Toc299098691)

[Tasks 231](#_Toc299098692)

[Summary 235](#_Toc299098693)

[Customization Possibilities in the hybris Multichannel Accelerator 235](#_Toc299098694)

[Just One Storefront 237](#_Toc299098695)

[Removing Apparel and the Apparel Stores 239](#_Toc299098696)

[acceleratorcore 240](#_Toc299098697)

[acceleratorfacades 245](#_Toc299098698)

[acceleratorstorefront 247](#_Toc299098699)

[acceleratorsampledata 248](#_Toc299098700)

[Re-Initialize 249](#_Toc299098701)

[Removing BTG 249](#_Toc299098702)

[User Interface and Creatives in the hybris Multichannel Accelerator 254](#_Toc299098703)

[The Front End 255](#_Toc299098704)

[Usability, Layout and Design Good Practice 256](#_Toc299098705)

[Colors 256](#_Toc299098706)

[Layout 257](#_Toc299098707)

[Accessibility 262](#_Toc299098708)

[Technical Structure 262](#_Toc299098709)

[HTML 263](#_Toc299098710)

[Theme Directory 264](#_Toc299098711)

[CSS Framework 267](#_Toc299098712)

[Javascript 267](#_Toc299098713)

[Browser Compatibility 268](#_Toc299098714)

The purpose of this document is to introduce you to the hybris Multichannel Accelerator and to help you gain an understanding into what comes pre-packaged, why it has been implemented, how it works and how you can use it to build out your projects requirements.

Essentially, the hybris Multichannel Accelerator is a collection of template extensions to hybris that are a starting point for your new e-commerce implementation project. The project development team takes the hybris Multichannel Accelerator source code, edit it, change it, add to it and make it your project's own. By the end of your project the hybris Multichannel Accelerator source may have taken quite a different shape. The point of supplying this source code for your development team is so they have something to start with on day one, that is implementing much of the functionality you will already need to fulfill your project requirements.

The hybris Multichannel Accelerator also makes technical architecture choices for you, such as which MVC framework to use, how to render HTML, how to handle emails, how to organize your software components, which CSS frameworks to use.

The hybris Multichannel Accelerator also provides the development team with source code that may already fulfill a reasonable percentage of your customers requirements. The hybris Multichannel Accelerator ships with lots of standard and typical B2C functionality implemented with best-practice, heavily adopted software frameworks. If your project requirements differ in some way to what the hybris Multichannel Accelerator offers, your developers can of course also adapt the code since it's all delivered as source. Your developers can of course also remove aspects of the source code that is not required for your project.

The hybris Multichannel Accelerator also ships with sample stores that provide good reference points as to how to correctly setup data in a hybris system and how to write scripts to load data in bulk in an efficient way.

* [Technical Introduction to hybris](#1)
* [Features](#2)
* [Multichannel Accelerator Extensions](#3)
* [Key Design Principles](#4)

# Technical Introduction to hybris

The hybris Multichannel Accelerator is based on the hybris Multichannel Suite. It provides a modular development framework with a rich type system for persistent data. It also provides a set of extensible tools to manage data and processes. To best leverage the Accelerator you should get familiar with the technical concepts of the hybris Multichannel Suite. There is comprehensive documentation available on hybris Wiki. We recommend to start your reading here:

* [Get a Quick Start](#5): A set of curriculum with different scopes and topics.
* [Platform Basics and Concepts](#6): A collection of documents about general technical concepts.
* [hybris Development Trail](#7): A tutorial to teach you developing with the hybris Multichannel Suite.

# Features

Key features delivered by the hybris Multichannel Accelerator source code include:

* Spring MVC 3 Based Storefront
  + Annotation Based Page Controllers
  + Annotation Based CMS Component Controllers
  + Annotation Based URL Mappings
* hybris WCMS Integration
  + Extensive set of new components
  + Extensive set of page templates
  + Live Edit
  + Preview
  + CMS Cockpit
  + Content Pages
  + CMS managed content on Functional Pages like cart, checkout, my account
* Website Themes
  + Grid Based CSS Framework
  + Quick Skinning of a site
  + Independent Themes Per Storefront
* JSP based View Layer
  + Page Templating tags
  + Pages
  + CMS Components
  + Tags
  + Locale, Theme and Site based Message Resources
* Web 2.0
  + Ajax Mini Cart
  + JQUERY
  + Ajax Modals
  + JSON Binding
* Facade Layer
  + Front-end agnostic Storefront API
  + Everything Customisable and Extensible in project using Spring Framework
* SOLR Integration
  + Category Landing Pages Integrated with the WCMS
  + Multi-select and refinement facets
  + Configurable facet and facet value ordering
  + Configurable Sorts
* Spring Security
  + Authentication
  + Role based Access Restrictions
  + HTTPS
  + Secure GUID cookie
* Configurable Hot Folder Batch Importer
  + Spring Integration
  + Simple CSV files with **impex** translator layer
  + Sequencing
  + Spring XML Based Configuration
  + Decoupled Design
* Emails
  + Process Engine Integration
  + CMS Integration
  + Velocity templates
  + Audit Trail
* Apparel Data Model
  + Multi level variants by style and size
* Multiple Storefronts residing on one server
  + Split By Country or Brand
  + Share Product Catalogs or separate
  + Share Content Catalogs for multiple language or separate
  + Regional delivery rules
  + Net/Gross Prices
* Richly functional Sample Stores to demonstrate available Accelerator functionality
  + Landing Pages
  + Product Pages
  + Facetted Search and Navigation
  + Cart
  + Full Checkout Process
  + My Account
  + Order History
  + Content Pages
* Extensive Sample Data
  + All complete examples of using **impex**
  + CMS Content
  + 2 Product Catalogs with classification, product references, categorisation, media, prices
  + Promotions
  + BTG Segments and Rules
  + Storelocator Points of Sale

# Multichannel Accelerator Extensions

The hybris Multichannel Accelerator adds multiple extensions on top of those you get with the hybris Multichannel Suite. All extensions are shipped as source, they are intended as a starting point for your project however the way you would work with these extensions on a project differs.

All **accelerator** extensions are templates in which a project would start by directly modifying the source code of the extension or using **extgen** and **modulegen** to make a copy with more suitable package and source naming for the project. The **commerce** extensions are designed to be extended in counterpart project specific extensions and not to have source code directly modified.

* The **acceleratorstorefront**, **acceleratorfacades** and **acceleratorcore** extensions are templates that you should use as a starting point on a project. You would adapt the supplied template code and configuration in these to extend the functionality provided out of the box by the accelerator to fulfil the requirements of your project.
* The **acceleratorcockpits** extension is also a template extension that adds functionality to the cockpits. Projects can add further code customizations here.
* The **acceleratorsampledata** extension is provided to showcase what the accelerator provides out of the box by presenting three sample fully functional storefronts that showcase all the accelerators functionality.
* The **commercefacades** extension provides a single storefront API that captures all typical B2C storefront functions while the **commerceservices** aggregates most B2C common extensions to provide more B2C oriented services out of the box.

|  |
| --- |
| **See Also**   * [hybris Multichannel Accelerator Extensions](#8) |

# Key Design Principles

* Use proven and well established technologies and frameworks
  + Well adopted in the development community
  + No bleeding edge
  + Adopt features of the Spring Framework where applicable and appropriate
* Performant
* Clear layers of separation
* Adopt hybris Best Practices
* Clear, appropriate and well documented examples of using hybris technology
* Real-world samples
* Less HTML more CSS
* Provide Template **accelerator** extensions to **extgen**
* Provide Front-end agnostic Storefront API **commerce** extensions to extend by adaptable plugin points using Spring IoC
* Provide support for Selenium tests
* Easy to understand and adapt

|  |
| --- |
| **See Also**   * [Customizing the hybris Multichannel Accelerator with Extgen and Modulegen](#9) * [Customization Possibilities in the hybris Multichannel Accelerator](#10)   + [Removing Apparel and the Apparel Stores](#11)   + [Removing BTG](#12) |

# System Architecture of the hybris Multichannel Accelerator

The following document will introduce you to the key architecture aspects of the hybris Multichannel Accelerator. It is intended as a reference for developers, software and system architects and consultants.

The hybris Multichannel Accelerator is essentially a feature rich web application and a collection of software components that fulfill many typical B2C use-cases that are common requirements on a vast number of hybris implementation projects. This document helps you understand the software as well as understand in more detail the features implemented by the software, how everything fits together and how you can adapt the code or configuration to fulfill your projects requirements.

While reading this document, you start with a brief guide on how to use this document, followed by some content around the architecture of an Multichannel Accelerator deployment so you get an broad understanding of the components involved. From here on you can then dive into whatever documentation you like, be it guides to extensions or guides to specific functionality topics.

* [Using this Document](#14)
  + [Approaching the Documentation](#15)
  + [Understanding Data Model Diagrams](#16)
* [Multitier Architecture](#17)
  + [Web Server](#18)
  + [hybris Application Server Nodes](#19)
    - [Web Request Application Server Node](#20)
      * [Presentation Layer](#21)
      * [ServiceLayer](#22)
      * [Persistence Layer](#23)
      * [Simple Request Interaction Diagram](#24)
    - [Back Office Application Server Node](#25)
      * [Administration Applications](#26)
      * [ServiceLayer](#22)
      * [Persistence Layer](#23)
    - [Diagram of Logical Organization of Software Components](#27)
  + [Solr Standalone](#28)
  + [Relational Database](#29)
  + [Other Components Not Covered](#30)

|  |
| --- |
| **About this Document**  This document describes the system architecture of the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, software architects, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Child Pages** |

|  |
| --- |
| **See Also**   * [Third-Party Components](#31) * [Third-Party Compatibility - Release 4](#32) |

## Using this Document

This section guides you on how to use this document.

### Approaching the Documentation

There are two ways you can approach this documentation.

1. You can deep-dive into one of the hybris Multichannel Accelerator extensions to find documentation related to the functionality and software components provided by that extension.
2. You can read through specific topic area documentation, where functionality may cross the boundaries of multiple extensions.

The first use-case tends to apply to developers who has already have the code in front of them and are looking to understand a part of it. The second use-case would apply to an architect or developer who wishes to understand the big picture of a more specific topic area such as how BTG is handled in the hybris Multichannel Accelerator or how Search and Navigation work.

All extension specific documentation has references to the topic specific guides in the **See Also** panels at the right of each page. In many cases there are specific links to areas in the topic documentation relating to the extension so there is little risk of missing anything.

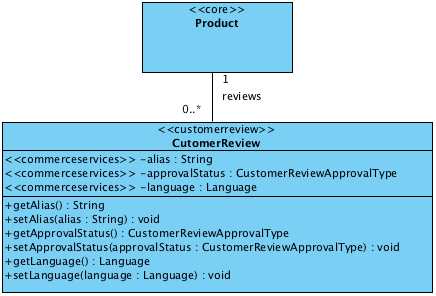
### Understanding Data Model Diagrams

The hybris Multichannel Accelerator uses a flexible type system that allows extensions to add additional attributes to existing types. During build time, model objects are generated and compiled from the type system definition, located in the items.xml file, and the result is the complete data model.

The hybris Multichannel Accelerator requires a specific set of extensions out of the box, but like the rest of the hybris Multichannel Suite, it does not impose any limitations on further extensions being added that include data model changes.

Unless it is required to improve the clarity of the diagram, all the hybris Multichannel Accelerator data model diagrams only show the attributes and relationships of types added in the hybris Multichannel Accelerator extensions. You would need to refer to other counterpart documentation to see a fuller picture of the data model. Where a type was extended, the UML stereotypes were used to identify at class level in which extension the type was added, or at attribute level which extension added the attribute.

The following diagram examples this:



The **Product** class is declared in the **core** extension, the **CustomerReview** class is declared in the **customerreview** extension. The **alias**, **approvalStatus** and **languages** attributes are all declared in the **commerceservices** extension.

## Multitier Architecture

There are many types of system and software components that can make up a hybris system deployment, indeed there is probably not one **typical** hybris system deployment. The numbers of servers and components completely differ depending on implementation requirements. To give you an understanding of the kind of system and software components you will typically work with on a project starting with the hybris Multichannel Accelerator as a base, the following section describes how various types of component physically and logically fit together. You can find more details about each component in counterpart sections of the hybris Multichannel Accelerator documentation.

|  |
| --- |
| **See Also**   * [Storefront Web Application Deconstructed](#33) * [Spring Usage in the hybris Multichannel Accelerator](#34) * [Business Processes and Eventing in the hybris Multichannel Accelerator](#35) * [Data Importing Capability in the Multichannel Accelerator](#36) * [Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator](#37) * [Search and Navigation in the hybris Multichannel Accelerator](#38) * [commercefacades extension - Technical Guide](https://wiki.hybris.com/display/acc/commercefacades+extension+-+Technical+Guide#commercefacadesextension-TechnicalGuide-BreakdownofaFacade) section **Breakdown of a Facade** |

The diagram below shows an example deployment setup of the hybris Multichannel Accelerator based implementation. The rest of this section provides more details on the deployment presented in the diagram.

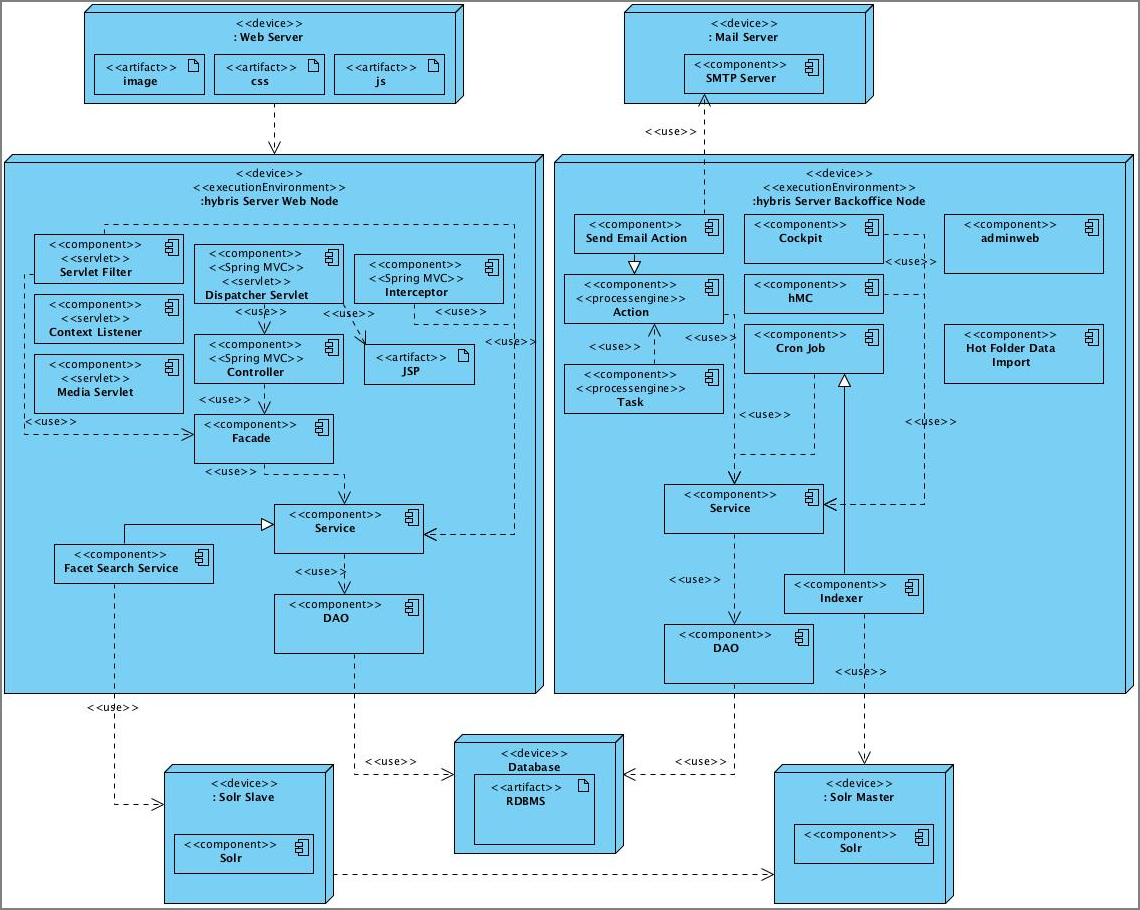


Figure: Example deployment of the hybris Multichannel Accelerator based system.

### Web Server

It is common to use a web server such as Apache to store static assets such as images, CSS and JavaScript files. This is typically done to ensure the application server resources are not troubled with serving assets that can be served from the web server with greater performance.

|  |
| --- |
| **See Also**   * [hybris Server](#39), section **Installation** * [hybris Server](#40), section **Configuring the hybris Server** * <http://httpd.apache.org/docs/2.2/install.html>: Compiling and Installing Apache Server * <http://learn.iis.net/page.aspx/26/installing-and-configuring-iis-7/>: Installing and Configuring IIS7 |

### hybris Application Server Nodes

Although it is possible to install all the software components in hybris on a single run time, it is more desirable to setup a cluster of servers to separate the functions that are dedicated to serving real-time customer web requests and those that perform asynchronous processes or administration interfaces. This preference is driven by performance, fault-tolerance and networking security reasons.

|  |
| --- |
| **See Also**   * [Cluster - Technical Guide](#41) * [Setup of a Production System](#42) * <http://tomcat.apache.org/tomcat-7.0-doc/setup.html>: Apache Tomcat Setup Guide |

The software components that run inside the hybris run time can be divided into separate logical layers. In each layer we will briefly describe and reference further information.

#### Web Request Application Server Node

One or more application servers can be set up to service requests from the web. These could be just requests on the web sites or requests from web services. The state is maintained using HTTP sessions. The requests are often proxied via a web server and often a session **id** aware load balancer is installed to ensure load is distributed evenly to all available application servers.

##### Presentation Layer

The presentation layer consist of following components:

* **Servlet Components**
  + Filters
  + Listeners
  + Dispatcher Servlet
  + Media Servlet
* **Spring MVC**
  + Dispatcher Servlet
  + Controllers
  + Interceptors
* **View**
  + JSP/JSTL

##### ServiceLayer

The hybris ServiceLayer consist of following components:

* Facades
* Services
* Data Access Objects

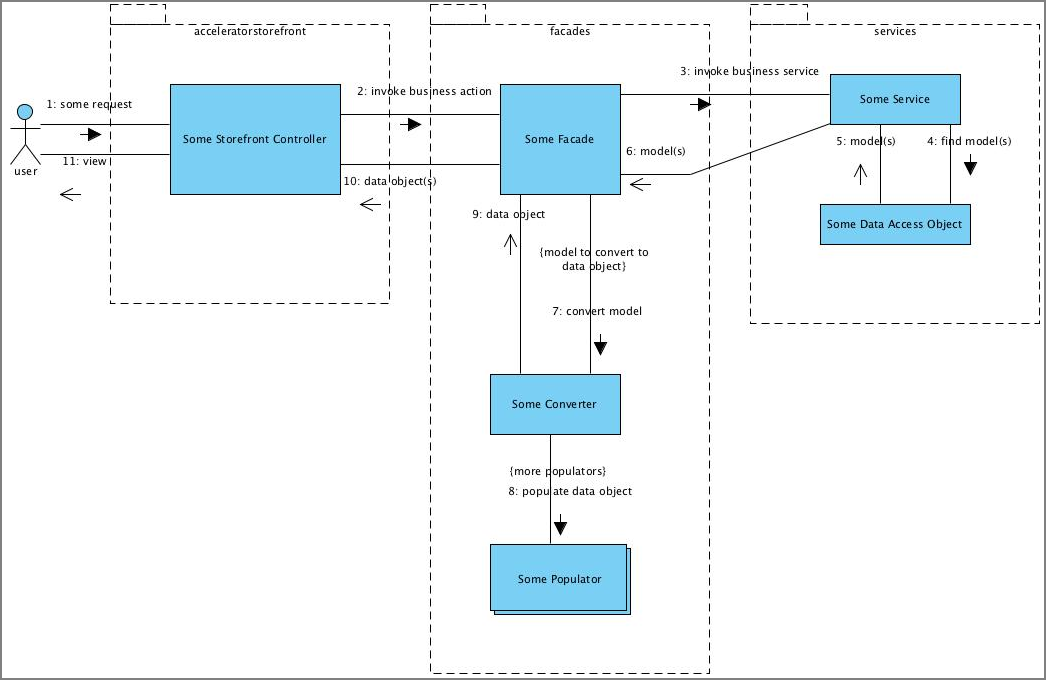
##### Persistence Layer

The persistence layer is all out of the box and would require very little work on the project side except describing/modifying types in the items.xml .

|  |
| --- |
| **See Also**   * [Storefront Web Application Deconstructed](#33) * [mediaweb Extension](#43) * [ServiceLayer](#44) * [items.xml](#45) * [ServiceLayer Architecture](#46), section **DAOs** |

##### Simple Request Interaction Diagram

The following interaction diagram shows how the application server components inter-operate to service a client request:



#### Back Office Application Server Node

Back office functions including the various admin interfaces can be separated onto different physical application servers. This can simplify security as admin applications that shouldn't be available across the Internet can simply be installed on a back office machine without Internet access. Back office processes tend to require different types of data to be cached, thus helping to improve cache performance. It can also be desirable to allow for a little downtime possibility on back office nodes, separating back office process allows you to do this without having downtime on the storefront.

##### Administration Applications

There are following applications which you can use for administration of your system:

* Management Console
* Administration Web
* Business and Administration Cockpits, for example:
  + Product Cockpit
  + WCMS Cockpit
  + Report Cockpit
  + Print Cockpit
* Customer Service Cockpit

You can find more details on how to use the front and back ends in documentation in the list of documents in the **See Also** panel on the right.

|  |
| --- |
| **See Also**   * End user documentation:   + [hMC - End User Guide](#47)   + [hybris Administration Console - End User Guide](#48)   + [Administration Cockpit Tutorial](#49)   + [Product Cockpit - End User Guide](#50)   + [WCMS Module - End User Guide](#51)   + [Reporting Module - End User Guide](#52)   + [Customer Service Module - End User Guide](#53) * Technical documentation:   + [hmc Extension](#54)   + [adminweb Extension](#55)   + [mcc Extension - Technical Guide](#56)   + [admincockpit Extension - Technical Guide](#57)   + [productcockpit Extension - Technical Guide](#58)   + [cmscockpit Extension - Technical Guide](#59)   + [cscockpit Extension - Technical Guide](#60) |

##### ServiceLayer

The hybris Multichannel Accelerator and hybris system are based on the hybris ServiceLayer API. The following technologies are used within the hybris Multichannel Accelerator:

* Cron Jobs
* Tasks, Processes and Actions
* Hot Folder Data Importer
* Services
* Data Access Objects
* SOLR Indexer

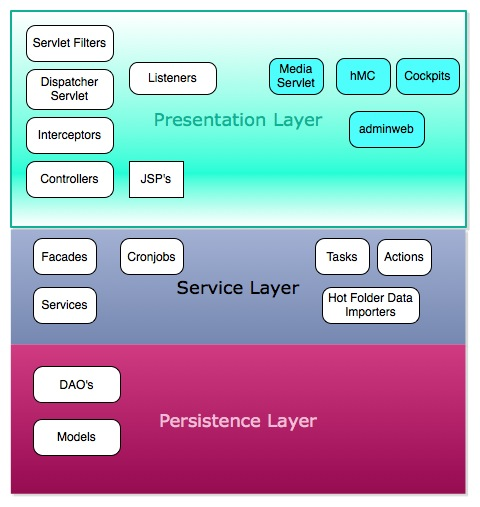
More information on these topics is available in the documentation provided in the hybris Wiki.

|  |
| --- |
| **See Also**   * [cronjob Extension - Technical Guide](#61) * [task Extension - Technical Guide](#62) * [processengine Extension - Technical Guide](#63) * [processengine Extension - Technical+Guide](#64), section **Actions** * [Data Importing Capability in the Multichannel Accelerator](#36) * [commerceservices Extension - Technical Guide](#65) * [ServiceLayer Architecture](#46) * [Search and Navigation in the hybris Multichannel Accelerator#Indexing](#66) |

##### Persistence Layer

The persistence layer is all out of the box and would require very little work on the project side except describing/modifying types in the items.xml .

#### Diagram of Logical Organization of Software Components

The following diagram shows how the components are arranged into their logical tiers:  


### Solr Standalone

Though Solr can be installed embedded in with the hybris Application Server, It's recommended to install Solr standalone especially when a cluster of web request nodes are installed. More information is available in the [Search And Navigation](#67) document, in the section section **Solr Configuration**. Typically you have one **master** Solr server that you send all index updates too and then these changes are replication to one or more **slave** servers.

|  |
| --- |
| **See Also**   * [solrfacetsearch Extension - Technical Guide](#68) * [Search and Navigation in the hybris Multichannel Accelerator](#38) |

### Relational Database

The hybris system supports a wide range of RDBMS. You can find more information in the [Third-Party Databases](#69) documentation.

### Other Components Not Covered

* Load Balancers
* VIP's
* Firewalls
* SFTP/FTP Servers
* Media Asset Management Systems
* ERP Systems
* Payment Service Providers
* CRM Systems
* Content Delivery Networks

## hybris Multichannel Accelerator Extensions

The hybris Multichannel Accelerator adds multiple extensions on top of those you get with the hybris Multichannel Suite. All extensions are shipped as source, they are intended as a starting point for your project however the way you would work with these extensions on a project differs.

All **accelerator** extensions are templates in which a project would start by directly modifying the source code of the extension or using **extgen** and **modulegen** to make a copy with more suitable package and source naming for the project. The **commerce** extensions are designed to be extended in counterpart project specific extensions and not to have source code directly modified.

* The **acceleratorstorefront**, **acceleratorfacades** and **acceleratorcore** extensions are templates that you should use as a starting point on a project. You would adapt the supplied template code and configuration in these to extend the functionality provided out of the box by the accelerator to fulfil the requirements of your project.
* The **acceleratorcockpits** extension is also a template extension that adds functionality to the cockpits. Projects can add further code customizations here.
* The **acceleratorsampledata** extension is provided to showcase what the accelerator provides out of the box by presenting three sample fully functional storefronts that showcase all the accelerators functionality.
* The **commercefacades** extension provides a single storefront API that captures all typical B2C storefront functions while the **commerceservices** aggregates most B2C common extensions to provide more B2C oriented services out of the box.
* [Extension Structure](#70)
* [The hybris Multichannel Suite Extension Dependencies](#71)

|  |
| --- |
| **About this Document**  This document introduces the extensions behind the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Child Pages** |

### Extension Structure

You can divide the hybris Multichannel Accelerator extensions into four conceptual groups:

* Accelerator project extensions,
* Accelerator cockpit extensions,
* Accelerator sample and test data,
* Commerce storefront API extensions.

All **accelerator** extensions are project template extensions where you can either directly modify code or a project named copy of the extension be made using **extgen**. For more information on how to use **extgen**, refer to [Creating a New Extension](#72) document. In addition we also supply our sandbox **commercewebservices** extension that uses the **commerce** B2C storefront API extensions also used by accelerator storefront to expose product search web services out of the box.

|  |
| --- |
| **See Also**   * [Customizing the hybris Multichannel Accelerator with Extgen and Modulegen](#9) * [Creating a New Extension](#72) |

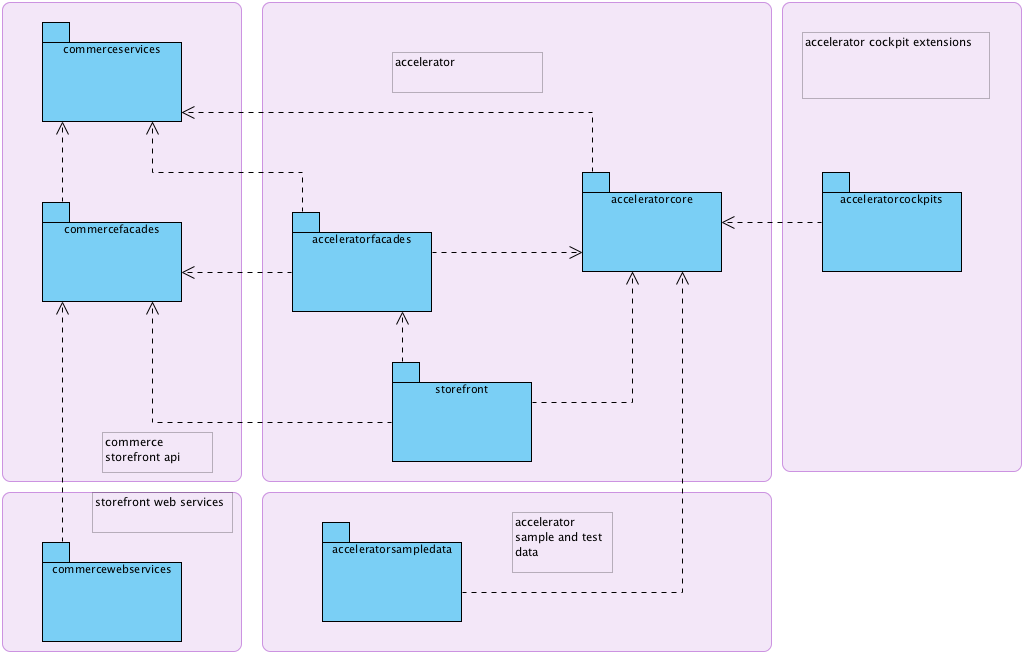


Figure: Dependencies behind the hybris Multichannel Accelerator extensions.

The above package diagram outlines the dependencies between each of the new extensions.

* The **acceleratorstorefront** and **commercewebservices** extensions contain the code for the front-end tiers that expose storefront functionality using web site or web service respectively. They do not have a business layer so do not come with a core or the hybris Management Console (hMC) module.
* The **acceleratorfacades** and **commercefacades**extensions organize and aggregate multiple business services to provide a storefront API that exposes actions and a data model that is optimized for a B2C commerce front-end use. The **acceleratorfacades** extension extends the functionality offered by the **commercefacades** extension and would be where a project would add all additional facades and customize the **commercefacades** extension functionality. Both extensions come with a core module but no hMC and web module.

|  |  |
| --- | --- |
|  | **Info**  In principal Accelerator facade classes could actually have resided in the **src** tree of the **acceleratorstorefront** extension, however this would require other front-end extensions needing a dependency on the **acceleratorstorefront** extension. The **acceleratorfacades** extension has been created to maintain this cleaner layer separation. |

* The **acceleratorcore** and **commerceservices** extensions are the business service layer extensions where the data model is extended, services from other hybris extensions are combined, added to and enriched and further services are added for project use. Both extensions come with a core and hMC module. All core data vital for the provisioning of an accelerator storefront is loaded in the **acceleratorcore** extension as a part of essential and project data. Projects would be expected to alter this scripts and add further data scripts here.
* The **acceleratorcockpit** extension is where you programmatically extend hybris cockpit business tools to customize existing features of a cockpit or add additional features. It is important to note that you can add any XML file based configuration to any extension, the Accelerator adds this to the **acceleratorcore** extension.
* The **acceleratorsampledata** extension is provided to create a number of reference stores. This extension adds all the products and content necessary for each of the three sample storefronts as well as test promotions, users, Advanced Personalization rules and so on. A project would not go live with this extension, it is provided purely as a sample. The data is separated into this extension to make it very easy to not include the sample data as a part of a project. In such a case, you simply do not add this extension to your localextensions.xml . This extension does not require the hybris **sampledata** extension to exist but can also run with this extension present.

### The hybris Multichannel Suite Extension Dependencies

In order to expose a wide amount of typical B2C functionality, the commerce- and accelerator- extensions add additional dependencies on extensions that exist outside of the core platform module. Find below an overview of dependencies on external extensions:

* The **commerceservices** extension depends on:
  + [basecommerce](#73) extension - provides a set of multi-purpose services especially related to order management but also fraud check, store locating, and so on.
  + ﻿[promotions](#74) extension - used to implement customer sales promotion functionality such as "Buy 2 get 1 free", "Free delivery when order value exceeds X".
  + ﻿[payment](#75) extension - a framework to provide payment related functionality like authorization of a payment amount, reserve amount, capture and storing payment data with the payment service provider.
  + ﻿[customerreview](#76) extension - provides functionality enabling front end users of a web shop to give a rating and comment on offered products.
  + [solrfacetsearch](#77) extension - provides full text and faceted search based on [Apache Solr](http://lucene.apache.org/solr/).
  + ﻿[cms2](#78) extension - provides content management functionality. Using the **cms2** extension you can create a website, that can be later managed by the WCMS Cockpit.
* The **acceleratorcore** extension depends on:
  + [fulfilmentprocess](#79) extension - implements an example order fulfillment process.
* The **acceleratorstorefront** extension depends on:
  + ﻿[btg](#80) extension - adds behavioral targeting functionality to the hybris Multichannel Suite allowing dynamic user group assignments based on user actions performed during web-browsing.
* The **acceleratorcockpits** extension depends on:
  + ﻿[cmscockpit](#59) extension - the back end application for the website pages content management, compared to the WCMS Cockpit that is the front end application.
  + ﻿﻿[productcockpit](#58) extension - provides a simple way for cockpit end users to create and update all information regarding products and catalogs.
  + ﻿[cscockpit](#60) extension - the cockpit of the Customer Service module.

|  |  |
| --- | --- |
|  | **Licensing**  A hybris extension may provide functionality that is licensed through different hybris modules. Make sure to limit your implementation to features as defined in your license contract. In case of doubt, please contact your hybris Sales representative. |

### acceleratorcore Extension - Technical Guide

The hybris **acceleratorcore** extension is the template business layer extension for your project. It is here where a project will extend or add further services and extend the hybris data model to fit a project requirements.

A project will start by either using **extgen** and **modulegen** to create a copy of this extension with project specific naming and packaging.

**acceleratorcore** also includes a sandbox Hot folder Batch Importer framework as well as working samples of how to write you own service and dao.

|  |  |
| --- | --- |
|  | **Note**  **Before Implementing** A hybris extension may provide functionality that is licensed through different hybris modules. Make sure to limit your implementation to features as defined in your license contract. In case of doubt, please contact your hybris Sales representative. |

* [WCMS Components](#82)
* [Hot Folder Batch Data Import Capability](#83)
* [Apparel Products](#84)

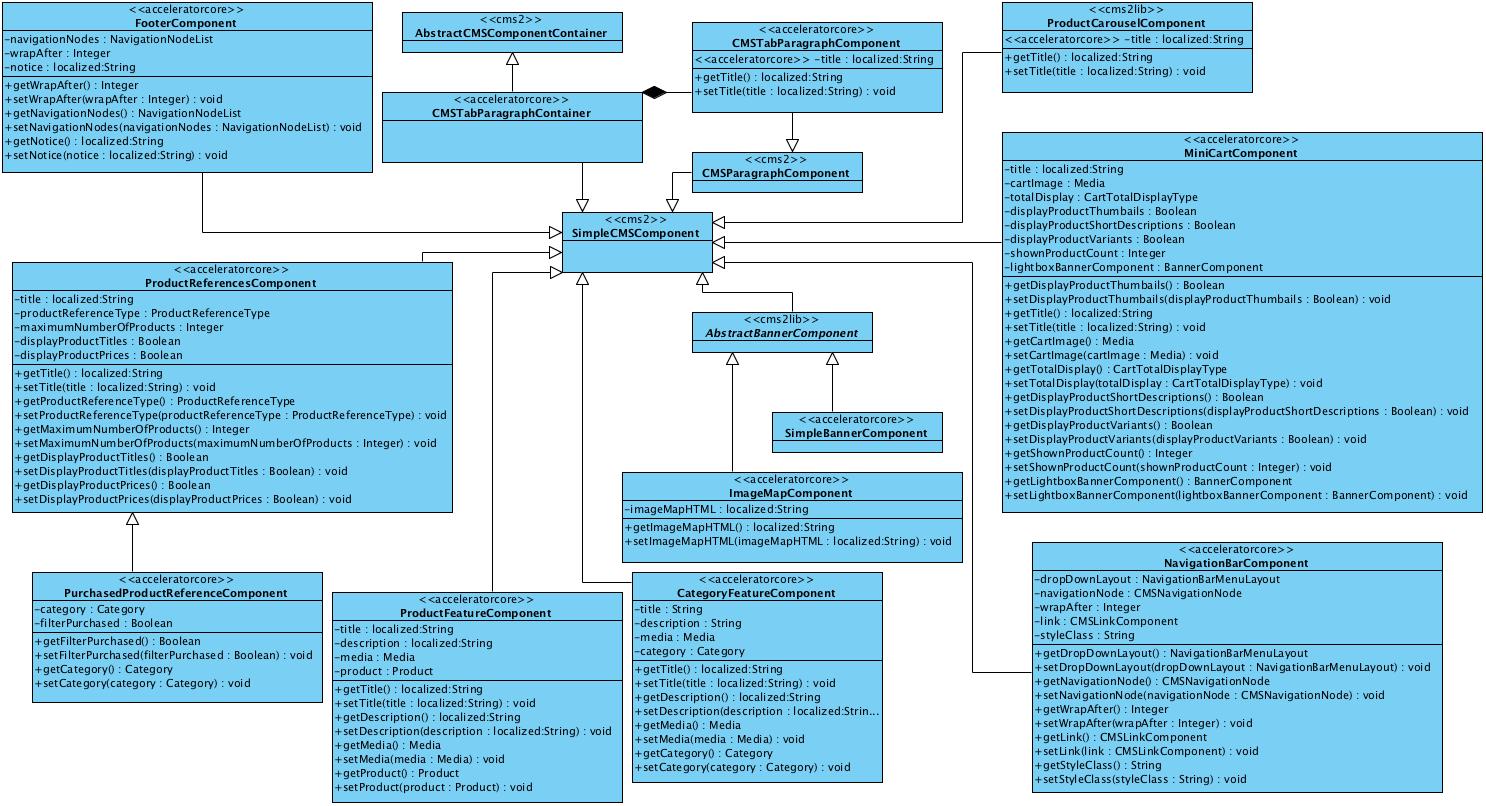
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About this Document**   | **Extension** | **Release 4.0** | **Release 4.1** | **Release 4.2** | **Release 4.3** | **Release 4.4** | | --- | --- | --- | --- | --- | --- | | **namename** | 4.0 | 4.1 | 4.2 | 4.3 | 4.4.0 |  | **Directory** | [**Download**](https://wiki.hybris.com/createpage.action?spaceKey=acc&title=Download&linkCreation=true&fromPageId=103809609) **With** | **Related To** | | --- | --- | --- | | bin/accelerator | hybris Multichannel Accelerator | * [hybris Multichannel Accelerator](#85) * [acceleratorfacades Extension - Technical Guide](#86) * [cms2 Extension - Technical Guide](#78) * [cms2lib Extension - Technical Guide](#87) |   This document introduces the hybris **acceleratorcore** extension.  **Audience**: Consultants, developers  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [Customizing the hybris Multichannel Accelerator with Extgen and Modulegen](#9) * [Data Importing Capability in the Multichannel Accelerator](#36) * [User Interface and Creatives in the hybris Multichannel Accelerator](#88), section **Standardized Components** * [WCMS Components for the hybris Multichannel Accelerator](#89) |

#### WCMS Components

The **acceleratorcore** adds a number of new WCMS components to those provided out of the box in the **cms2** and **cms2lib** extensions. Some existing components have also been extended with additional attributes.

The following class diagram shows the extensions made to the WCMS data model in **acceleratorcore**. For your reference the declaring extension is stereotyped on the class. If an existing type has been extended, the attribute will be stereotyped.



* **Mini Cart Component**  
  Configure how the mini cart works on the storefront. This includes title, how totals are calculated, how many recently added products to display and a merchandising banner.
* **Product References Component**  
  Choose what types of cross-sell to show on a product page. The system will fill the slot with cross-sells of the specified type if they exist for the currently viewed product.
* **Simple Banner Component**  
  Simple banner with only image and link attributes.
* **Image Map Component**  
  Offers all the attributes of the **SimpleBannerComponent** but also adds the ability to embed image map HTML.

|  |
| --- |
| **See Also**   * [Wikipedia on Image Maps](http://en.wikipedia.org/wiki/Image_map) |

* **Category Feature Component**  
  Spotlight a particular category of products. Add some marketing text, an image and an emotive title. Link will be auto-generated.
* **Product Feature Component**  
  Spotlight a particular product. Add some marketing text, an image and an emotive title. Link will be auto-generated.
* **Navigation Bar Component**  
  Placeholder component for building the ever-present top navigation on a website.

|  |
| --- |
| **See Also**   * [Top Navigation and Link Management in the hybris Multichannel Accelerator](#90) |

* **Footer Component**  
  A counterpart to the **NavigationBarComponent**, this allows you to build a typical footer menu present like those present on many e-Commerce sites.
* **Product Carousel Component**  
  This **cms2lib** component has been extended to add a title.
* **Purchased Product References Component**  
  This is provided for sample purposes primarily and extends the **ProductRefrencesComponent** by incorporating a sample suggestion service as part of a functionality demonstration of advanced personalization and WCMS restrictions.
* **CMS Tab Paragraph Component**  
  Support the ability to add additional tabs of content on the product details page.

#### Hot Folder Batch Data Import Capability

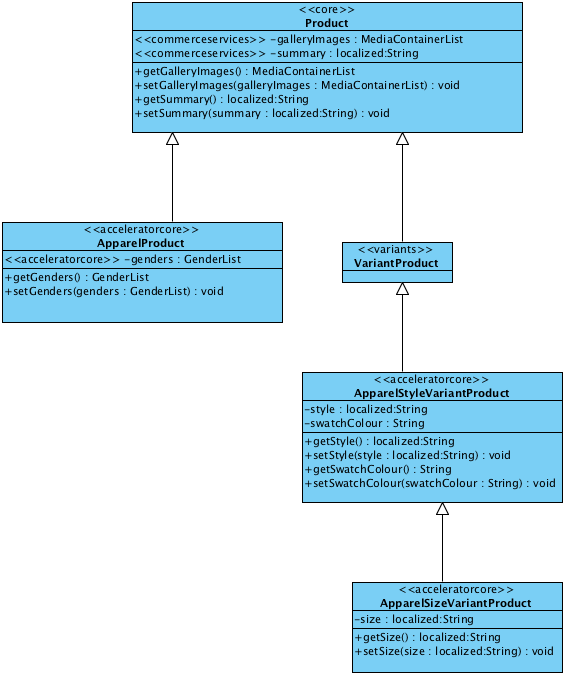
A sandbox hot folder data import capability, which uses Spring Integration is provided with the **acceleratorcore** extension. This is built using ImpEx and is intended as an integration plug-in point for automated data updates from external systems. For more information refer to the dedicated [Data Importing Capability in the Multichannel Accelerator](#36) documentation.

#### Apparel Products

**acceleratorcore** extends the Product data model to support Apparel Products. The concept of variants of Style and Size has been added.

The recommended modeling approach is a 'variant of a variant' model which is organized as follows:

* The **ApparelProduct** extends the **Product** and adds a genders attribute that enables you to modify if the product is suited only to a certain sex.
* The **ApparelStyleVariantProduct** extends the **VariantProduct** and allows you to vary products by different style, for example Red or Green T-Shirt.
* The **ApparelSizeVariantProduct** extends the **ApparelStyleVariantProduct** and allows you to vary a style by a different Size, for example XXL Red T-Shirt.



This approach allows you to set promotions and cross-sells at style level rather than having to set them up for every size variation. It is of course also possible to setup size level promotions and cross-sells still. If a product just has one size it is possible to create a direct link to the size variant, since it extends the style variant all the style attributes are available.

### acceleratorfacades Extension - Technical Guide

The hybris **acceleratorfacades** extension is a starting point template for a project to extend the functionality provided by the **commercefacades** extension. It enables to add additional project-specific facades, as well as extending or adding further data objects, customizing or adding new converters and adding additional populators. Out of the box, the **acceleratorfacades** extension is quite light. It just provides some examples of how to extend **commercefacades** with some extended Data Objects, Converters, Populators and a new Facade. Lastly, it provides an extension point for the email generation processes to inject facade data object context variables into the velocity context used when generating emails.

As with all template extensions of the, a project would first create a copy of the extension using **extgen** and **modulegen**.

|  |  |
| --- | --- |
|  | **Note**  **Before Implementing** A hybris extension may provide functionality that is licensed through different hybris modules. Make sure to limit your implementation to features as defined in your license contract. In case of doubt, please contact your hybris Sales representative. |

* [Apparel](#91)
* [SimpleSuggestionFacade](#92)
* [Email](#93)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About this Document**   | **Extension** | **Release 4.0** | **Release 4.1** | **Release 4.2** | **Release 4.3** | **Release 4.4** | | --- | --- | --- | --- | --- | --- | | **acceleratorfacades** | 4.0 | 4.1 | 4.2 | 4.3 | 4.4.0 |  | **Directory** | [**Download**](#94) **With** | **Related To** | | --- | --- | --- | | bin/accelerator | hybris Multichannel Accelerator | * [hybris Multichannel Accelerator](#85) * [commercefacades Extension - Technical Guide](#95) * [acceleratorcore Extension - Technical Guide](#81) |   This document introduces the hybris **acceleratorfacades** extension.  **Audience**: Consultants, developers  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **See Also**   * [Customizing the hybris Multichannel Accelerator with Extgen and Modulegen](#9) |

#### Apparel

**ProductFacade** from the **commercefacades** extension is extended to include additional support for the **ApparelProduct** provided by **acceleratorcore** extension. This includes:

* Extending and replacing the default **ProductConverter**
* Supporting the handling of Apparel Size and Style variants with an extended **VariantOptionDataConverter**
* Adding a new Data Object type **GenderData** with counterpart converter and Populator.

These serves as providing an example as to how to extend **commercefacades**, as well as providing the adaptations necessary to expose the data required for the **acceleratorstorefront** Apparel front end storefronts.

|  |
| --- |
| **Related Resources**   * [acceleratorcore extension Technical Guide](https://wiki.hybris.com/display/acc/acceleratorcore+extension+-+Technical+Guide#acceleratorcoreextension-TechnicalGuide-ApparelProducts) section **ApparelProducts** * [How To Extend the ProductFacade - Tutorial](#96) |

#### SimpleSuggestionFacade

The **SimpleSuggestionFacade** is used by the **acceleratorstorefront** to render the content required for the **PurchasedProductReferences** CMS Component. It is an example of how to add an additional standalone facade to complement those provided out of the box with the **commercefacades** extension.

#### Email

Emails are generated asynchronously using the business process email functionality provided in **commerceservices**. Like a storefront, much of the content of the email will be generated using the facade layer. Prototype beans are setup in the application context to configure which facade data objects are injected into the velocity context for rendering of email content.

For each email the hybris Multichannel Accelerator generates, the AcceleratorFacades layer provides a context type extending **AbstractEmailContext** type. During email generation process, the **DefaultEmailContextFactory** in **commerceServices** retrives the name of the email context type from the renderer template (which holds the velocity script referencing the email context object) associated with the email template. Then it tries to get the bean instance of the email context type from the application context.

The **CustomerEmailContext** type populates customer data into email context. The **ForgottenPasswordEmailContext** adds details like token in addition to customer data, and the **OrderNotificationEmailContext** populates order data into the email context. The data objects that get generated and injected into the email context is exactly the same data objects generated for storefront. This makes the scripting of velocity template to generate the email text is simpler, and relatively similar to that of a web page that uses the same data objects.

The following code sample is the **AbstractEmailContext** prototype bean defined in **commercesservices** extension:

|  |
| --- |
| <bean id="abstractEmailContext" class="de.hybris.platform.commerceservices.process.email.context.AbstractEmailContext" abstract="true" scope="prototype">  <property name="customerEmailResolutionService" ref="customerEmailResolutionService"/>  <property name="siteBaseUrlResolutionService" ref="siteBaseUrlResolutionService"/>  <property name="configurationService" ref="configurationService"/> </bean> |

Table 1 commerceservices-spring.xml

And the three email context prototype beans defined in **acceleratorfacades** extension are show below:

|  |
| --- |
| <bean id="customerEmailContext" class="de.hybris.platform.acceleratorfacades.process.email.context.CustomerEmailContext" parent="abstractEmailContext" scope="prototype" >  <property name="customerConverter" ref="customerConverter"/> </bean>  <bean id="forgottenPasswordEmailContext" class="de.hybris.platform.acceleratorfacades.process.email.context.ForgottenPasswordEmailContext" parent="customerEmailContext" scope="prototype" >  <property name="customerConverter" ref="customerConverter"/> </bean>  <bean id="orderNotificationEmailContext" class="de.hybris.platform.acceleratorfacades.process.email.context.OrderNotificationEmailContext" parent="abstractEmailContext" scope="prototype" >  <property name="orderConverter" ref="orderConverter"/> </bean> |

Table 2 acceleratorfacades-spring.xml

#### How To Extend the ProductFacade - Tutorial

To deal with additional **gender** attribute that is defined for type **ApparelProduct**, you need to extend **ProductFacade**. To do it, you should perform the steps described below.

|  |  |  |
| --- | --- | --- |
| **About this Document**  This document instructs you on how to extend the **ProductFacade**.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Consultants, developers  **Related extension**: [acceleratorfacades Extension - Technical Guide](#86)  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

1. Create data container that holds representation of gender from **ApparelProduct**:

|  |
| --- |
| public class GenderData {  private String code;  private String name;   // public getters and setters } |

Table 3 GenderData.java

1. Create a new class that extends the existing **ProductData** object so it can hold your gender data:

|  |
| --- |
| public class ApparelProductData extends ProductData {  List<GenderData> genders;   // public getter and setter } |

Table 4 ApparelProductData.java

1. Create a converter for the gender, that takes the gender as enumeration and returns **GenderData**:

|  |
| --- |
| public class GenderConverter extends AbstractPopulatingConverter<Gender, GenderData> {  private TypeService typeService;   protected TypeService getTypeService()  {  return typeService;  }   @Required  public void setTypeService(final TypeService typeService)  {  this.typeService = typeService;  }    @Override  protected GenderData createTarget(final Gender source)  {  return new GenderData();  }    @Override  public void populate(final Gender source, final GenderData target)  {  target.setCode(source.getCode());  target.setName(getTypeService().getEnumerationValue(source).getName());   super.populate(source, target);  } } |

Table 5 GenderConverter.java

1. Implement a populator that populates **ApparelProductData** with the **gender** attribute. Basing on a type of passed arguments, you need to decide where to get gender from:

|  |
| --- |
| public class ApparelProductPopulator implements Populator<ProductModel, ProductData> {  private GenderConverter genderConverter;   protected GenderConverter getGenderConverter()  {  return genderConverter;  }   @Required  public void setGenderConverter(final GenderConverter genderConverter)  {  this.genderConverter = genderConverter;  }    @Override  public void populate(final ProductModel source, final ProductData target) throws ConversionException  {  ApparelProductModel apparelProductModel = null;  if (source instanceof ApparelProductModel)  {  apparelProductModel = (ApparelProductModel) source;  }  else if (source instanceof VariantProductModel)  {  if (((VariantProductModel) source).getBaseProduct() instanceof ApparelProductModel)  {  apparelProductModel = (ApparelProductModel) ((VariantProductModel) source).getBaseProduct();  }  else if (((VariantProductModel) source).getBaseProduct() instanceof ApparelStyleVariantProductModel)  {  apparelProductModel = (ApparelProductModel) ((ApparelStyleVariantProductModel) ((VariantProductModel) source)  .getBaseProduct()).getBaseProduct();  }  }   if (target instanceof ApparelProductData && apparelProductModel != null)  {  if (CollectionUtils.isNotEmpty(apparelProductModel.getGenders()))  {  final List<GenderData> genders = new ArrayList<GenderData>();  for (final Gender gender : apparelProductModel.getGenders())  {  genders.add(getGenderConverter().convert(gender));  }  ((ApparelProductData) target).setGenders(genders);  }  }  }  } |

Table 6 ApparelProductPopulator.java

1. Having all required classes, you can create **AcceleratorProductConverter**. It should take care of creating proper data holder object of required type and then populate it with all required data:

|  |
| --- |
| public class AcceleratorProductConverter extends ProductConverter {  private Populator<ProductModel, ProductData> apparelProductPopulator;    protected Populator<ProductModel, ProductData> getApparelProductPopulator()  {  return apparelProductPopulator;  }   @Required  public void setApparelProductPopulator(final Populator<ProductModel, ProductData> apparelProductPopulator)  {  this.apparelProductPopulator = apparelProductPopulator;  }    @Override  protected ProductData createTarget(final ProductModel source)  {  if (source instanceof ApparelProductModel)  {  return new ApparelProductData();  }  else if (source instanceof VariantProductModel)  {  if (((VariantProductModel) source).getBaseProduct() instanceof ApparelProductModel)  {  return new ApparelProductData();  }  else if (((VariantProductModel) source).getBaseProduct() instanceof ApparelStyleVariantProductModel)  {  return new ApparelProductData();  }  }  return super.createTarget(source);  }    @Override  public void populate(final ProductModel source, final ProductData target)  {  getApparelProductPopulator().populate(source, target);   super.populate(source, target);  } } |

Table 7 AcceleratorProductConverter.java

1. To connect all pieces together, reference all classes as Spring beans and use the Spring configuration to make them work. A sample of Spring configuration file:

|  |
| --- |
| <alias name="acceleratorGenderConverter" alias="genderConverter"/>   <bean id="acceleratorGenderConverter" class="de.hybris.platform.acceleratorfacades.converters.GenderConverter" scope="tenant">  <property name="typeService" ref="typeService"/>  </bean>   <bean id="apparelProductPopulator" class="de.hybris.platform.acceleratorfacades.converters.populator.ApparelProductPopulator" scope="tenant">  <property name="genderConverter" ref="genderConverter"/>  </bean>   <alias name="acceleratorProductConverter" alias="productConverter"/>  <bean id="acceleratorProductConverter" class="de.hybris.platform.acceleratorfacades.converters.AcceleratorProductConverter" scope="tenant" parent="defaultProductConverter">  <property name="apparelProductPopulator" ref="apparelProductPopulator"/>  </bean> |

Table 8 acceleratorfacades-spring.xml

The **acceleratorProductConverter** is aliased with **productConverter**, so it overrides the existing product converter in the system, and takes all of its responsibilities. From now on all product conversions are performed by this new converter that checks if a product is related to apparel variant. It also takes care of filling proper attributes.

### acceleratorstorefront Extension - Technical Guide

The hybris **acceleratorstorefront** extension is a template for a ready to be adapted web front end. It uses Spring MVC.

The **acceleratorstorefront** extension unites all the functions typically found in an online shop front end and interacts with the hybris Management Console (hMC), Product and CMS Cockpits using the essential data from **acceleratorcore** and optionally sample store content from the **acceleratorsampledata** extension. It supports multiple store fronts served from the same web application, as well as splits by country, language and currency. Each storefront has independently managed cms content and product catalogs, both can be separated or shared between multiple storefronts.

The **acceleratorstorefront** extension is intended as a starting point to accelerate project development or as a cheat sheet for your own website. If getting live quick is your aim, then using the site theme feature, advanced CMS site configuration and flexibility to attach your site to a different search index, it is possible to rebrand the current web store, add your own products and cms content and go live with few code modifications.

The **acceleratorstorefront** extension uses the **commercefacades** extension to provide much of the storefront functionality exposed on the front-end. This makes it much easier to share business logic and storefront functionality across multiple channels.

|  |  |
| --- | --- |
|  | **Note**  **Before Implementing** A hybris extension may provide functionality that is licensed through different hybris modules. Make sure to limit your implementation to features as defined in your license contract. In case of doubt, please contact your hybris Sales representative. |

* [Best Practice Storefront](#98)
* [Getting Started](#99)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About this Document**   | **Extension** | **Release 4.0** | **Release 4.1** | **Release 4.2** | **Release 4.3** | **Release 4.4** | | --- | --- | --- | --- | --- | --- | | **acceleratorstorefront** | 4.0 | 4.1 | 4.2 | 4.3 | 4.4.0 |  | **Directory** | [**Download**](#100) **With** | **Related To** | | --- | --- | --- | | bin/accelerator | hybris Multichannel Accelerator | * [hybris Multichannel Accelerator](#85) * [acceleratorcore Extension - Technical Guide](#81) * [acceleratorsampledata Extension - Technical Guide](#101) * [commercefacades Extension - Technical Guide](#95) |   This document introduces the hybris **acceleratorstorefront** extension.  **Audience**: Consultants, developers  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Child Pages**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **Related Resources**  **Architectural Overviews**   * [Storefront Web Application Deconstructed](#33) * [User Interface and Creatives in the hybris Multichannel Accelerator](#102) * [WCMS Integration in the hybris Multichannel Accelerator](#103) * [Search and Navigation in the hybris Multichannel Accelerator](#104) section **Storefront**   **Configuration and Customization Guides**   * [Setting Up the Store Locator in the hybris Multichannel Accelerator](#105) * [Top Navigation and Link Management in the hybris Multichannel Accelerator](#90) * [SEO URLs in the hybris Multichannel Accelerator](#106) * [Stock Management in the hybris Multichannel Accelerator](#107) * [Spring Security in the hybris Multichannel Accelerator](#108) * [Advanced Personalization in the hybris Multichannel Accelerator](#109)   **Configuration and Customization Trails**   * [Add New CMS Component Trail](#110) * [Add New CMS Page Template Trail](#111) |

#### Best Practice Storefront

The **acceleratorstorefront** adopts good practice web application design principles. Like our **storetemplate**, **mobiletemplate** and **b2bstore** it is based on one of the industries most heavily adopted MVC frameworks, **Spring MVC**. No unproven bleeding edge technologies have been used to ensure project developers have the greatest chance of immediate familiarity with the frameworks and the architecture when they initially take their first glance at the source code. Controllers are annotation based for fastest development turnaround and form objects use the **javax validation framework**. Controllers are free of business logic and lightweight due to the inclusion of a facade layer provided in separate extensions that exposes a single Storefront API that cleanly separates the storefront from the business layer. This ensures projects have the greatest chance of re-using business logic with different front-ends if they so desire.

**JSP** and **JSTL** are used for the view with a less HTML, more CSS policy for all UI component mark up to ensure storefront's can be easily re-skinned with the **acceleratorstorefronts** theming functionality. All pages have been designed to fit a standard 950 pixel wide Grid-Based canvas using the **Blueprint CSS** framework. Good practice examples are shown of how to use the hybris WCMS to dynamically include content all used throughout the site. All UI components and assets are cleanly organized into a logical folder structure. Tags have been used to UI functionality to components. Additionally, UI components are also broken up into Pages, Fragments and WCMS components using simple templating tags and the tags provided by the **cms2lib** extension. Message Resource Bundles have also been used to internationalize the storefront with an extra dimension of adapting content by theme and site. All content is internationalized and UTF-8 encoded that has been tested end-to-end. The front-end has also been accessibility tested.

The Storefront is made secure using **Spring Security** and also has had a level of XSS testing applied. A Secure GUID cookie function as well as limited time link support has been added for extra security.

The Storefront also provides Automated Smoke Test support with tools such as Selenium by introducing a tag that enables component id's to be output when the Smoke Test feature is enabled.

The Storefront also demonstrates best practice integration with other hybris B2C commerce features not directly available through a business facade layer such as BTG, WCMS (pages, components, preview and live edit).

#### Getting Started

The following [Storefront Web Application Deconstructed](#33) guide will give you a deeper understanding of how the storefront fits together.

Prior to starting development, you should first start your project by making a copy of this **acceleratorstorefront** extension together with counterpart extensions from the **accelerator** module by using our extgen and modulegen functionality. For more information about extgen and modulegen tools, go to the [Customizing the hybris Multichannel Accelerator with Extgen and Modulegen](#9) document.

#### Storefront Web Application Deconstructed

This section provides a general overview of the Multichannel Accelerator storefront construction and technology used.

* [Web Application Configuration](#112)
  + [Structure of a Web Application](#113)
  + [WEB.XML](#114)
    - [Filter Chain](#115)
    - [Listeners](#116)
    - [Servlet Mapping](#117)
* [MVC](#118)
  + [Annotation Based Page Controllers](#119)
  + [Annotation Based CMS Component Controllers](#120)
* [Live Edit and Preview](#121)
* [Validation](#122)
* [JSON Binding for Servicing Ajax Requests](#123)
* [Messages](#124)
* [bean.xml files](#125)
* [Interceptors](#126)
* [View](#127)
  + [JSPs](#128)
  + [Page JSPs](#129)
  + [Fragment JSPs](#130)
  + [CMS JSPs](#131)
  + [TAGs](#132)
  + [Template Tags](#133)
  + [Component Tags](#134)
* [Theme Support](#135)
* [Resources](#136)
* [Blueprint CSS](#137)
* [JavaScript](#138)
* [Model](#139)
* [Breadcrumb Support](#140)

|  |  |  |
| --- | --- | --- |
| **About this Document**  This document instructs you on the storefront structure in the hybris Multichannel Accelerator.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Consultants, developers, software architects  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

##### Web Application Configuration

* web/src  
  Contains the Java source for the store front. Thisd includes page and cms controllers, interceptors, various custom implementations of classes required for spring security and custom store front components.
* **web/webroot**
  + **\_ui**
    - **black**  
      The Black theme
    - **blue**  
      The Blue theme
    - **blueprint**  
      bluprint library
    - **common**  
      commonly used stylesheets, JavaScript libraries and images
    - **purple**  
      The Purple theme
  + **js**  
    contains JavaScript required for the CMS
  + **stylesheets**  
    contains stylesheets required for the CMS
  + **WEB-INF**
    - **config**  
      Spring application context files
    - **tags**  
      Tags that are used within views
    - **views**  
      JSP pages, fragments and cms components
    - **lib**  
      Libraries required by the store front.
    - **messages**  
      Localization files.
    - **tld**  
      Tag library descriptor files for cms and ycommerce tags.

[About Extensions](#141)

###### Structure of a Web Application

###### WEB.XML

web.xml sets the default values for the various web applications deployed in an application server.

Filter Chain

The filters are in a chain and are called in the order provided below. Finally the request is delivered to the servlet.

| **Filter** | **Description** | **See Also** |
| --- | --- | --- |
| **ResourceFilter** | The filter is used to server file resources by bypassing the other filters. It is only mapped to paths that contain files on disk and should be served unaltered. This is used to serve the JavaScript, CSS, images and theme files. |  |
| **RequestLoggerFilter** | It logs each HTTP request received by this web application. This is a useful filter during development, but is not recommended for use in production. It logs each request along with time taken to process the request and generate the response. |  |
| **HybrisInitFilter** | Filter that does initialization work for the hybris web application, including:   * Getting the hybris session from the **HttpSession** or, if not available, create a new hybris session * Activate this hybris session so that calls to **jaloSession.getCurrentSession()** can be performed * Notify all extension managers or each request | * [hybrisInitFilter](#142) |
| **CMSSiteFilter** | The filter sets up the CMS integration for the application. It uses the requested URL to select the current site and sets up the session catalog versions. It also handles the preview data and redirecting to specific pages based on deep preview links. | * [storetemplate Extension Tutorial](#143), section **The CMSSiteServletFilter** * [How To Set Up the Preview in the WCMS Cockpit - Tutorial](#144), section **StoreTemplateAlgorithm** |
| **StoreFrontFilter** | This is the store front application specific filter that initializes new sessions. It is responsible for calling the **StoreSessionFacade** to initialize each new session. The **StoreSessionFacade** sets up the initial language and currency for the session. This filter is also responsible for recording each request in the **BreadcrumbBuilder**, which is used when building a historical breadcrumb trail. |  |
| **FlashScopeFilter** | It supports extracting the flash scoped variables from the session and adding them to the current request. Flash scope allows the use of the post/redirect/get design pattern. It lightens many of the problems associated with handling multiple submits or resubmission of data in browser requests to the server. Flash scope is an additional scope to those provided in a standard Java web application (page, request, session and application). Any attributes held in flash scope are available for the duration of the current request, and the subsequent request too. |  |
| **Spring Security Filter** | This filter is used to enable Spring Security support in the application. This filter is of type **DelegatingFilterProxy**, but the filter name is **springSecurityFilterChain**. The filter name has to match the bean id defined by Spring Security. | * [Spring Security](#145) * [http://static.springsource.org/.../security-filter-chain.html](http://static.springsource.org/spring-security/site/docs/3.0.x/reference/security-filter-chain.html): The Security Filter Chain * [How to enable SSO?](#146) |
| **BTG Filters:**   * **BTGSegmentFilter** * **AbstractBtgFilter** * **AbstractPkResolvingBtgFilter** * **BTGSegmentFilter** * **CategoryVisitedBtgFilter** * **ProductVisitedBtgFilter** * **RefererHeaderBtgFilter** * **RequestParamsBtgFilter** | To enable Advanced Personalization, you need to find a place in your application where you can hook in with evaluation of customer segment rules. Each rule type provided by the **btg** extension requires a different data for evaluation. It is very important that evaluation method is called whenever a request is made to the server. This is why the **acceleratorstoretemplate** extension is equipped with the Filter. It is responsible for rule computation that is triggered on each request. The following BTG filter beans are executed for each request:   * **refererHeaderBtgFilter** * **requestParamsBtgFilter** * **productVisitedBtgFilter** * **categoryVisitedBtgFilter** | * [How To Integrate Advanced Personalization with Front-end Application - Tutorial](#147) * [Advanced Personalization in the hybris Multichannel Accelerator](#109) * [Advanced Personalization Module](#148) |

Listeners

These are servlet context listeners.

| **Listener** | **Description** | **See Also** |
| --- | --- | --- |
| **HybrisContextLoaderListener** | The hybris Multichannel Suite provides a **HybrisContextLoaderListener** for setting the global **ApplicationContext** automatically as parent of your **WebApplicationContext**. With that you can use the global bean definitions, for example to inject them into your **WebApplicationContext** beans. If you make a **getBean** call to your **WebApplicationContext**, it is checked whether there is a definition available. If not, the parent **ApplicationContext** is used. | * [Spring in the hybris Multichannel Suite](#149) |
| **RequestContextListener** | If you want to enable the usage of the web application-specific scopes: **session** and **request**, you additionally have to add the Spring **RequestContextListener**. | * [Spring in the hybris Multichannel Suite](#149) |

Servlet Mapping

Servlet mapping specifies the web container of which java servlet should be invoked for a url given by client.

| **Servlet** | **Description** | **See ALso** |
| --- | --- | --- |
| **DispatcherServlet** | The storefront web application uses the Spring MVC framework to handle requests. The core of the Spring MVC framework is the **DispatcherServlet**. It has to be configured in your web.xml file for activating the dispatching of incoming requests to the special MVC controller. | [Spring in the hybris Multichannel Suite](#150) section **UsingSpringMVC** |

##### MVC

The Accelerator store front is based on the Spring MVC framework. The [MVC](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) pattern provides a clean seperation of objects into 3 categories, models for maintaing data, controllers for handling events and views for rendering data.

###### Annotation Based Page Controllers

Page Controllers have the responsibility of providing data that will be rendered on the views. These are Spring MVC 3 controllers that use annotations for URL mapping both at the controller and method level. Further reading for how to provide [SEO URLs in the hybris Multichannel Accelerator](https://wiki.hybris.com/display/acc/SEO+URLs+in+the+hybris+Multichannel+Accelerator).

Constants defined in ControllerConstants provide the common view mappings for various pages, cms components and fragments.

|  |
| --- |
| ControllerConstants.Actions.Cms ControlelrConstants.Views.Cms ControllerConstants.Views.Pages.Account ControllerConstants.Views.Pages.Password ControllerConstants.Views.Pages.Checkout ControllerConstants.Views.Pages.Error ControllerConstants.Views.Pages.Cart ControllerConstants.Views.Pages.StoreFinder ControllerConstants.Views.Pages.Misc ControllerConstants.Views.Fragments.Cart ControllerConstants.Views.Fragments.Checkout ControllerConstants.Views.Fragments.Product |

Table 9 List of interfaces containing Controller Constants

AbstractPageController is used as the super class of all page controllers, it inserts the request, languages, currencies, currentLanguage, currentCurrency, and user variables into all page models. These are the most common objects used by various pages, fragments and cms components. For example the header makes use of all of them for the language and currency selector as well as to display the wecome message for the user.

###### Annotation Based CMS Component Controllers

All functionality is provided by the DefaultCMSComponentController unless a specific functionality is desired. The job a CMSComponentController is to populate the page model with component properties which is accomplished in the fillModel method. Much like the Page Controllers a CMS controller will return a view. The getView method from the AbstractCMSComponentController can be overriden to return the view specific to the new component. The CMS component controllers are always invoked by the [CMS component tag](https://wiki.hybris.com/display/acc/WCMS+Integration+in+the+hybris+Multichannel+Accelerator#WCMSIntegrationinthehybrisMultichannelAccelerator-TagFilesExample).

|  |  |
| --- | --- |
|  | brief overview of how this works but refer to separate cms documentation for finer detail of each component |

##### Live Edit and Preview

The Accelerator provides support for Live Edit and Preview by having the CMSSiteFilter preconfigured and through the use of CMS tags throughout the application.

For more information view the [WCMS Integration in the hybris Multichannel Accelerator Documentation](https://wiki.hybris.com/display/acc/WCMS+Integration+in+the+hybris+Multichannel+Accelerator)

##### Validation

Form validation in the accelerator store front is handled by Spring Forms. Two custom validators have been introduced the PaymentDetailsValidator and the EqualAttributesValidator. Additional tags have been added to simplify error display for form elements in web/webroot/WEB-INF/tags/form. If the provided tags aren't sufficient Spring form tags can also be wrapped in errorSpanField elements as in paymentDetailsForm.tag

|  |
| --- |
| <template:errorSpanField path="expiryMonth">  <form:select id="ExpiryMonth" path="expiryMonth" cssClass="card\_date">  <option value="" label="<spring:theme code='payment.month'/>"/>  <form:options items="${months}" itemValue="code" itemLabel="name"/>  </form:select> </template:errorSpanField> |

Table 10 paymentDetailsForm.tag

##### JSON Binding for Servicing Ajax Requests

Some controllers for example the CheckoutController return json data for ajax requests. This is handled by the jackson message converter org.springframework.http.converter.json.MappingJacksonHttpMessageConverter, defined in spring-mvc-config.xml. The actual string is eval()'ed into a json object by jquery, by specifying the dataType parameter as json.

|  |
| --- |
| $.ajax({  url: "/accelerator/url/",  dataType: "json",  success: function(data){   } }); |

In the @RequestMapping for these controller methods .json is appeneded to the value just as a clue to the developer.

##### Messages

Error, status and information messages for the acceleratorstorefront are located in base.properties files, with translations in base\_de.properties and base\_ja.properties. The Spring message tags can be used to display the messages.

|  |
| --- |
| <spring:message code="message.code"/> |

##### bean.xml files

Spring context files in the Accelerator store front define the beans used within the Spring environment.

|  |  |
| --- | --- |
| **web-application-config.xml** | Main entry point that imports all other config files. |
| **acceleratorstorefront-spring-btg.xml** | Contains beans used in by the BTG extension. |
| **spring-mvc-config.xml** | Contains beans for spring MVC functionality. |
| **spring-security-config.xml** | Spring security related beans, url interceptors. |

##### Interceptors

There are several configured in the spring-mvc-config.xml which are used for:

* setting the current locale (de.hybris.platform.acceleratorstorefront.interceptors.LocaleInterceptor)
* putting the Google API Key into the request (de.hybris.platform.acceleratorstorefront.interceptors.CmsPageInterceptor)
* loading the appropriate CMS page slots into the model (de.hybris.platform.acceleratorstorefront.interceptors.GoogleAPIKeyInterceptor)
* additional security for https requests (de.hybris.platform.acceleratorstorefront.security.GUIDInterceptor)
* creating BTG events for visited content pages (de.hybris.platform.acceleratorstorefront.servlets.btg.ContentPageVisitedBtgInterceptor)

Additional interceptors can be added if additional low level functionality is required.

##### View

The storefront uses JSP 2.1 as its view rendering technology. The Spring MVC controllers specify a view name to use and spring is configured to look in the **/WEB-INF/views/** folder for a JSP file that matches the view name.  
The JSP files are all located under the web applications **/WEB-INF** folder which mean that they are not directly accessible from the browser. The only way that the JSPs are used is as the view renderer.  
TAG files are used to provide common functionality for the JSPs. Also TAG files are used to build a very simple common master template that dispenses with the need for a 3rd party tool like sitemesh.  
The JSP and TAG files predominantly use JSP Expression Language (EL) and the JSP Standard Tag Library (JSTL). The use of standard technologies in the view layer should simplify adoption of the accelerator storefront.

###### JSPs

The JSP files are organised into 3 groups under the **/WEB-INF/views/** folder.

* **pages**  
  The pages folder contains JSPs that are used to render a complete HTML web page that can be returned to the client browser.
* **fragments**  
  The fragments folder contains JSPs that are used to render parts of HTML web pages that are requested or updated asynchronously. This typically happens via an AJAX request from the client browser.
* **cms**  
  The cms folder contains JSPs that are used to render the view for specific CMS components.

###### Page JSPs

The Page JSPs are used to render the view for a complete HTML web page. These JSPs are organised into subfolders based on their main area of functionality.  
An example simple page JSP that is used to render CMS content pages:

|  |
| --- |
| <%@ page trimDirectiveWhitespaces="true" %> <%@ taglib prefix="template" tagdir="/WEB-INF/tags/template" %> <%@ taglib prefix="cms" uri="/cms2lib/cmstags/cmstags.tld" %> <%@ taglib prefix="breadcrumb" tagdir="/WEB-INF/tags/nav/breadcrumb" %>  <template:page pageTitle="${pageTitle}">  <div id="breadcrumb" class="breadcrumb">  <breadcrumb:simpleBreadcrumbBar title="${title}" />  </div>  <cms:slot var="feature" contentSlot="${slots.Section1}">  <div class="span-24 section1 advert">  <cms:component component="${feature}"/>  </div>  </cms:slot>  <div class="span-24 section2">  <div class="span-4 zone\_a advert">  <cms:slot var="feature" contentSlot="${slots.Section2A}">  <cms:component component="${feature}"/>  </cms:slot>  </div>  <div class="span-20 zone\_b last">  <cms:slot var="feature" contentSlot="${slots.Section2B}">  <cms:component component="${feature}"/>  </cms:slot>  </div>  </div>  <div class="span-24 section3 advert">  <cms:slot var="feature" contentSlot="${slots.Section3}">  <cms:component component="${feature}"/>  </cms:slot>  </div> </template:page> |

This page uses the <template:page> tag to build the HTML page. The <template:page> ensures that the correct HTML directives are includes, includes all the required javascript and CSS resources, adds in the page header, navigation bar, and footer. This allows the page to concentrate on the main body of the page.  
The page uses the <breadcrumb:simpleBreadcrumbBar> to populate the breadcrumb bar then uses the <cms:slot> and <cms:component> tags to output the CMS configured components in to the appropriate positions on the page.  
More detail on the JSP tags used will be provided below in the section called Template Tags.

###### Fragment JSPs

The Fragment JSPs are used to render a view which is only part of the HTML web page and will typically be integrated into the web page on the client side. This is used to handle functionality like the product quick view popup. The product quick view popup is requested by the client browser via an AJAX request. The view is rendered for a specific product and returned to the client browser. The client browser then inserts this HTML in to the web page.  
This means that the fragment JSPs are not responsible for rendering the whole HTML page and therefore do not need to use the <template:page> tag to generate a whole page.

###### CMS JSPs

The CMS is built up using different types of components. Each of these components needs to be rendered to HTML and included in the response. As you have seen above a custom Spring MVC controller may be used to handle any logic associated with rendering the CMS component, but ultimately the component is rendered by a JSP view found in the cms folder. The JSP file name is looked up using the lowercase type code of the component.  
An example CMS component JSP is:

|  |
| --- |
| <%@ page trimDirectiveWhitespaces="true" %>  <div class="cmsimage">  <img title="${media.altText}" src="${media.url}" alt="${media.altText}"/> </div> |

This is the **cmsimagecomponent.jsp** which is used to render the view for the **CMSImageComponent**.  
As you can see this is a relatively simple component to render but this same pattern can be applied to more complex components.

###### TAGs

The JSPs use TAGs to provide common functionality or just to simplify the JSP and improve readability. TAGs cannot be rendered directly bust must be included in a JSP file or in another TAG file. The TAG files are organised into several different folders under the **/WEB-INF/tags/** folder. TAG files were introduced in the JSP 2.0 standard.  
The TAGs are split roughly into 2 separate groups.

* template tags  
  The template tags are used as a master template to include all the required layout and structural elements on each page.
* component tags  
  Component tags are used to encapsulate simple functionality into a reusable component.

###### Template Tags

We have already seen a brief example of the <template:page> tag above. This tag is responsible to putting together the common layout required for each page.  
This is the content of the page.tag:

|  |
| --- |
| <%@ tag body-content="scriptless" trimDirectiveWhitespaces="true" %> <%@ attribute name="pageTitle" required="false" rtexprvalue="true" %> <%@ attribute name="pageCss" required="false" fragment="true" %> <%@ attribute name="pageScripts" required="false" fragment="true" %> <%@ taglib prefix="template" tagdir="/WEB-INF/tags/template" %> <%@ taglib prefix="header" tagdir="/WEB-INF/tags/common/header" %> <%@ taglib prefix="footer" tagdir="/WEB-INF/tags/common/footer" %> <%@ taglib prefix="cart" tagdir="/WEB-INF/tags/cart" %> <%@ taglib prefix="nav" tagdir="/WEB-INF/tags/nav" %>  <template:master pageTitle="${pageTitle}">  <jsp:attribute name="pageCss">  <jsp:invoke fragment="pageCss"/>  </jsp:attribute>  <jsp:attribute name="pageScripts">  <jsp:invoke fragment="pageScripts"/>  </jsp:attribute>  <jsp:body>  <div id="wrapper">  <div id="page">  <a href="#skip-to-content" class="skiptocontent">skip to content</a>  <a href="#skiptonavigation" class="skiptonavigation">skip to main navigation</a>  <header:header/>  <a name="skiptonavigation"></a>  <nav:topNavigation/>  <cart:addToCart/>  <div id="content">  <a name="skip-to-content"></a>  <jsp:doBody/>  </div>  <footer:footer/>  </div>  </div>  </jsp:body> </template:master> |

This tag uses the <template:master> tag to generate the actual HTML directives, html element, head element, body element, include all scripts and CSS resources. The main function of the page.tag is to include the header and top navigation sections followed by the main body of the page, and finally the standard footer.  
The page.tag can be customised if the main layout elements of the page need to be changed.  
The master.tag is used to generate the required HTML to actually make the compliant HTML web page. The advantage of putting this all in one tag is that it only needs to be done and maintained in one place and is then shared by all the pages that need it.

###### Component Tags

The rest of the TAG files are reusable components that are included on JSP pages. For example the **<format:price>** tag is used to render a price. Here is an example of the tag being used:

|  |
| --- |
| <format:price priceData="${entry.basePrice}" displayFreeForZero="true"/> |

In the above example the tag is being used to render the base price for a cart entry product. In this case if the price is zero then we want to display the "Free" promotional text instead of the numerical value.  
Here is the implementation of the tag:

|  |
| --- |
| <%@ tag body-content="empty" trimDirectiveWhitespaces="false" %> <%@ attribute name="priceData" required="true" type="de.hybris.platform.commercefacades.product.data.PriceData" %> <%@ attribute name="displayFreeForZero" required="false" type="java.lang.Boolean" %> <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> <%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>  <c:choose>  <c:when test="${priceData.value > 0}">  ${priceData.formattedValue}  </c:when>  <c:otherwise>  <c:if test="${displayFreeForZero}">  <spring:theme code="text.free" text="FREE"/>  </c:if>  <c:if test="${not displayFreeForZero}">  ${priceData.formattedValue}  </c:if>  </c:otherwise> </c:choose> |

##### Theme Support

The accelerator storefront extends the theme support provided by Spring MVC to load resources based on both the current CMS Site as well as the Theme set on the site. The resource bundles are located in the **/WEB-INF/messages/** folder. Resources are loaded in the following order (highest priority first):

1. Site  
   For example: site-electronics.properties
2. Theme  
   For example: theme-blue.properties
3. Base  
   For example: base.properties

As these are standard Java resource bundles each properties file can be specialised for each language.  
Each theme has a separate directory of resources under the **/\_ui/<theme name>/** path (for example **/\_ui/blue/**). This resource directory can be used to hold CSS files and images included from the CSS.  
Theme resource lookup is done via the **<spring:theme>** tag, for example:

|  |
| --- |
| <spring:theme code="basket.page.total"/> |

This will look up the resource with the key "basket.page.total" in the resource bundle for the current site, if it cannot be found there it will fall back to the resource bundle for the current theme, and if not found there will look in the base resource bundle.  
In order to customise the behaviour of the Spring MVC theme support the acceleratorstorefront redefines the **themeSource** and **themeResolver** beans. For more information on Spring theme support see <http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html#mvc-themeresolver>.

##### Resources

The accelerator storefront exposes simple resource files under the **/\_ui/** web application path. These resources include CSS files, JavaScript files, images (related to CSS), and theme specific versions of the same. These files are not interpreted by the web application and are served straight from disk. The **/\_ui/** path prefix is used as this is unlikely to interfear with any of the content or SEO URLs required by the site itself. In a production environment it is recommended that these static file resources are served directly by the front end web server.

##### Blueprint CSS

The accelerator storefront uses the Blueprint CSS as the basis for its CSS layout. Blueprint CSS provides a solid grid that can support the most complex of layouts. More details on Blueprint CSS can be found here <http://www.blueprintcss.org/>.

The /\_ui web site path is used as the prefix for all

The following CSS files are loaded (lowest priority first):

* **/\_ui/blueprint/screen.css**  
  The main blueprint CSS file.
* **/\_ui/common/css/common.css**  
  Basic storefront CSS that is not theme related.
* **/\_ui/<theme name>/css/changes.css**  
  Theme specific CSS.

##### JavaScript

The accelerator storefront uses a number of different JavaScript libraries to provide dynamic functionality. The JavaScript files are located in the **/\_ui/common/js/** folder.  
The main library used is jQuery. jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for web development that has become very popular. More information on jQuery can be found here <http://jquery.com/>.  
The jQuery UI library is used to provide basic user interface components and behaviours. More information on jQuery UI can be found here <http://jqueryui.com/>.  
Several other jQuery plugins are used to provide additional behaviours, including: image carousel, client side HTML templating, blocking user interactions during AJAX updates, popup windows, review stars, form elements, and many more. Review the files in the **/\_ui/common/js/** folder for more details.

##### Model

##### Breadcrumb Support

The **acceleratorstorefront** module uses simple design to display breadcrumb info. Breadcrumb is represented in the system with a list of object of type **Breadcrumb**. Each breadcrumb element contains three pieces of information:

1. **url** that is relative URL addres where breadcrumb piece will point to
2. **name** that is displayed name of breadcrumb piece
3. **linkClass** represents CSS class that the single piece of breadcrumb will be decorated with.

Page that is required to display breadcrumb uses **breadcrumb** tag that requires single attribute that is list of **Breadcrumb** elements. Default breadcrumb tag displays at the beginning one link pointing to homepage, and after this link all elements of provided list are displayed.  
There are number of sample implementations of **BreadcrumbBuilder** interface in the **acceleratorstorefront** module that can be used to build breadcrumbs for some common purposes like for product, category, search result page or simple breadcrumb with one single link at the end. These breadcrumb builders are defined as spring beans and used in the MVC controllers. Each breadcrumb builder returns ready to use list of **Breadcrumb** objects that can be diectly passed to the breadcrumb tag at the displayed page.  
Here is sample example of breadcrumb related pieces of code for product details page:

|  |
| --- |
| // Somewhere in product page controller (...) model.addAttribute("breadcrumbs", productBreadcrumbBuilder.getBreadcrumbs(productData)); (...) |

|  |
| --- |
| // Somewhere in product page (...) <div id="breadcrumb" class="breadcrumb">  <breadcrumb:breadcrumb breadcrumbs="${breadcrumbs}"/> </div> (...) |

﻿

|  |  |
| --- | --- |
|  | Brief discussion of facades and data objects before referencing commercefacades and acceleratorfacades extensions and counterpart documentation |

### acceleratorcockpits Extension - Technical Guide

The hybris **acceleratorcockpits** extension is a template extension that demonstrates how a project can apply software customisations to one or more existing cockpits without needing to create a project specific versions of a cockpit using the **ycockpit** template.

|  |  |
| --- | --- |
|  | **Note**  **Before Implementing** A hybris extension may provide functionality that is licensed through different hybris modules. Make sure to limit your implementation to features as defined in your license contract. In case of doubt, please contact your hybris Sales representative. |

* [Spring beans redefinition](#152)
  + [Global spring context](#153)
  + [Web spring context](#154)
* [Cockpit UI configuration redefinition](#155)
* [Overriding or adding other files to cockpits](#156)
* [Acceleratorcockpits cscocpit customisations](#157)
  + [Cockpit UI configuration](#158)
  + [Spring beans customisations](#159)
* [Acceleratorcockpits cmscocpit customisations](#160)
  + [Spring bean customisations](#161)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About this Document**   | **Extension** | **Release 4.0** | **Release 4.1** | **Release 4.2** | **Release 4.3** | **Release 4.4** | | --- | --- | --- | --- | --- | --- | | **namename** | 4.0 | 4.1 | 4.2 | 4.3 | 4.4.0 |  | **Directory** | [**Download**](https://wiki.hybris.com/createpage.action?spaceKey=acc&title=Download&linkCreation=true&fromPageId=109152035) **With** | **Related To** | | --- | --- | --- | | bin/accelerator | hybris Multichannel Accelerator | * [hybris Multichannel Accelerator](#85) |   This document introduces the hybris **acceleratorcockpits** extension.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Consultants, developers  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

#### Spring beans redefinition

One of the purposes of the *acceleratorcockpits* extension is to put in one place spring bean redefinitions that will be used in the cockpits. There are two kinds of spring bean redeclarations in acceleratorcocktpis.

##### Global spring context

The first one is in the file acceleratorcockpits-spring.xml . Here we can redefine beans that will be then used in all cockpit extensions. Simple example that overrides *calculationService* can be:

|  |
| --- |
| <alias name="defaultImpersonatingCalculationServiceWrapper" alias="calculationService"/> <bean id="defaultImpersonatingCalculationServiceWrapper" scope="tenant" class="de.hybris.platform.acceleratorcockpits.cscockpit.services.order.impl.ImpersonatingCalculationServiceWrapper">  <property name="calculationService" ref="defaultCalculationService"/>  <property name="acceleratorImpersonationService" ref="acceleratorImpersonationService"/> </bean> |

Table 11 acceleratorcockpits-spring.xml

The name of this file is defined in project.properties file:

|  |
| --- |
| # Specifies the location of the spring context file putted automatically to the global platform application context. acceleratorcockpits.application-context=acceleratorcockpits-spring.xml |

Table 12 project.properties

##### Web spring context

Second kind of spring beans redeclaration is related to specific cockpit extension. For example we can redefine product perspective used in the *productcockpit* extension. Here is simple example:

|  |
| --- |
| <bean id="ProductPerspective" class="de.hybris.platform.productcockpit.session.impl.ProductPerspective" scope="session"  parent="BasePerspective">  <property name="uid" value="productcockpit.perspective.product" />  <property name="label" value="perspective.product" />  <property name="customCsaURI" value="/productcockpit/productCSA.zul" />  <property name="infoBoxTimeout" value="2000" />  (...) </bean> |

Table 13 productocockpit-web.xml

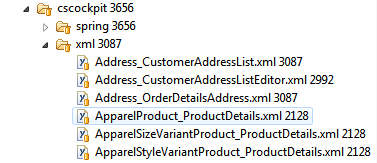
Second step will be to put this file name to project.properties so it will be included to the specified extension spring configuration:

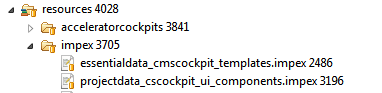
|  |
| --- |
| # Inject additional config into the productcockpit productcockpit.additionalWebSpringConfigs=classpath:/acceleratorcockpits/productcockpit/spring/productocockpit-web.xml |

Table 14 project.properties

Now our redeclared **ProductPerspective** bean will be used in the *productcockpit*.

#### Cockpit UI configuration redefinition

In *acceleratorcockpits* extension we can also redefine cockpit configurations that will be imported to the database and used in cockpit extensions. It will be imported in old fashioned way, to keep the clear folder structure in *acceleratorcockpits* extension. You can read more on this topic here: [How to import Cockpit Configuration - Tutorial](#162). Here is presented sample structure of cockpit configuration that we want to import:  


And now these files need to be imported. We use import by name convention using correctly named and placed impex files:  


In projectdata\_cscockpit\_ui\_components.impex we need to import all required configuration files (see [impex syntax](#163)). Below is excerpt from the sample impex file:

|  |
| --- |
| $defaultCatalog=Default $defaultCatalogVersion=Staged $catalogVersion=catalogVersion(catalog(id[default='$defaultCatalog']),version[default='$defaultCatalogVersion'])[unique=true,default='$defaultCatalog:$defaultCatalogVersion'] $media=media(code, $catalogVersion);  $jarResource=jar:de.hybris.platform.acceleratorcockpits.constants.AcceleratorCockpitsConstants&/acceleratorcockpits/cscockpit  # # Load medias # INSERT\_UPDATE Media;code[unique=true];$catalogVersion;mime;realfilename;@media[translator=de.hybris.platform.impex.jalo.media.MediaDataTranslator][forceWrite=true]  ;Acc\_Customer\_NewCustomerDetails\_ui\_config;;text/xml;Customer\_NewCustomerDetails.xml;$jarResource/xml/Customer\_NewCustomerDetails.xml;Acc\_Customer\_EditCustomerDetails\_ui\_config;;text/xml;Customer\_EditCustomerDetails.xml;$jarResource/xml/Customer\_EditCustomerDetails.xml  # # Setup cockpit configuration # INSERT\_UPDATE CockpitUIComponentConfiguration;code[unique=true];factoryBean;objectTemplateCode[unique=true];principal(uid)[unique=true];$media;  ;newCustomerDetails;editorConfigurationFactory;Customer;cockpitgroup;Acc\_Customer\_NewCustomerDetails\_ui\_config; ;editCustomerDetails;editorConfigurationFactory;Customer;cockpitgroup;Acc\_Customer\_EditCustomerDetails\_ui\_config; |

Table 15 projectdata\_cscockpit\_ui\_components.impex

#### Overriding or adding other files to cockpits

Another thing that can be done using *acceleratorcockpit* extension is to override or add frontend files that will be copied during build process to the destination extensions. If we want to do this we need to modify buildcallbacks.xml file of *acceleratorcockptis* extension. In this file we need to find macro called **acceleratorcockpits\_after\_build** and put there things we want to be copied. Below is simple example that will copy all files from */resources/acceleratorcockpits/cmscockpit/zul* to *cmscockpit* webroot folder:

|  |
| --- |
| <echo level="info" message="Start copying custom cmscockpit files to cmscockpit..."/> <copy todir="${ext.cmscockpit.path}/web/webroot">  <fileset dir="${ext.acceleratorcockpits.path}/resources/acceleratorcockpits/cmscockpit/zul">  <include name="\*\*/\*\*.\*" />  </fileset> </copy> |

Table 16 buildcallbacks.xml

After building the system with ant the files will be copied to the destination.  
The **acceleratorcockpits\_after\_build** macro is also used to copy/override *cmscockpit* images and icons. The following existing code snippet is responsible for this action:

|  |
| --- |
| <copy todir="${ext.cmscockpit.path}/web/webroot/cmscockpit/images">  <fileset dir="${ext.acceleratorcockpits.path}/resources/acceleratorcockpits/cmscockpit/images">  <include name="\*\*/\*\*.\*" />  </fileset> </copy> |

Table 17 buildcallbacks.xml

#### Acceleratorcockpits cscocpit customisations

In *acceleratorcockpits* extension much customisations has been made to *cscockpit* extension. Both spring configuration and cockpit UI configurations were extended and new widgets and controllers as well as java classes have been added.

##### Cockpit UI configuration

Cockpit component configurations have been added to *cockpitgroup* user group for several item types. New configuration files are placed in **/resources/acceleratorcockpits/cscockpit/xml/** folder and are imported in projectdata\_cscockpit\_ui\_components.impex file. This impex file is imported automatically by file name convention.

##### Spring beans customisations

There are several new spring configration files introduced in *acceleratocockpits* for *cscockpit*. These files are placed in /resources/acceleratorcockpits/cscockpit/spring/ folder and imported by inclusion in import.xml file that is referenced in project.properties file. Here is short description of new spring configurations introduced for *cscockpit*:

* cscockpit-web.xml - here we have new *messageSource* bean defined that overrides exsiting one and joins message sources from original *cscockpit* extension and in *acceleratorcockpits*
* cscockpit-services.xml - in this file only *addressModelLabelProvider* bean is redefined
* cscockpit-controllers.xml - here we have several bean redefinitions including redefinition of *csCallContextController* that is extended to use *baseStoreSelectorStrategy*
* cscockpit-widgets.xml - several renderers bean implementations were redefined in this file as well as *csProductUrlStrategy*

#### Acceleratorcockpits cmscocpit customisations

##### Spring bean customisations

/resources/acceleratorcockpits/cmscockpit/spring/cmscockpit-wizards.xml redefines *cmsSiteWizard* spring bean. Other beans defined in this file is related to the new defined wizard. This wizard extends default CMS site wizard with new attributes defined for CMS site type in *commerceservices* extension, that are *theme*, *defaultLanguage* and *locale*.

### acceleratorsampledata Extension - Technical Guide

The hybris **acceleratorsampledata** extension adds the necessary dataset for the 3 accelerator reference Storefronts (Electronics JP, Apparel UK and Apparel DE) across 2 product catalogs.

One Storefront has multi-lingual options, as well as offering multiple currency choices. 2 offer shipping to multiple countries, all have separate tax rules set at the Store Point Of Sale Country. Each Storefront has a css 'Theme' that changes the look and feel in the form of a re-skinning of a base html/collection of stylesheets. The Apparel Storefronts share the same Theme (black), and 2 are available for Electronics (blue and purple). All storefronts are served from just one web-application using the same codebase. All Storefronts have separate content catalogs.

|  |  |
| --- | --- |
|  | **Note**  **Before Implementing** A hybris extension may provide functionality that is licensed through different hybris modules. Make sure to limit your implementation to features as defined in your license contract. In case of doubt, please contact your hybris Sales representative. |

* [Japanese Electronics Store](#164)
* [UK Apparel Store](#165)
* [DE Apparel Store](#166)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About this Document**   | **Extension** | **Release 4.0** | **Release 4.1** | **Release 4.2** | **Release 4.3** | **Release 4.4** | | --- | --- | --- | --- | --- | --- | | **acceleratorsampledata** | 4.0 | 4.1 | 4.2 | 4.3 | 4.4.0 |  | **Directory** | [**Download**](https://wiki.hybris.com/createpage.action?spaceKey=acc&title=Download&linkCreation=true&fromPageId=106532714) **With** | **Related To** | | --- | --- | --- | | bin/accelerator | hybris Multichannel Accelerator | * [hybris Multichannel Accelerator](#85) |   This document introduces the hybris **acceleratorsampledata** extension.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Consultants, developers  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [Essential and Project Data in the hybris Multichannel Accelerator](#167) |

The Sample Data Extension Loads the Product and WCMS content for the three sample stores.

#### Japanese Electronics Store

* Exclusive Product Catalog with content in 3 languages:
  + Electronics
* Product Data Model :
  + Vanilla with Classification System
* Exclusive WCMS Content Catalog with content in 3 languages
* Languages:
  + Japanese
  + English (default)
  + German
* Currencies:
  + JPY
  + USD (default)
* Gross Prices
* Shipping:
  + Japan
  + USA
  + Europe
* Site Theme:
  + Blue (default) and Purple
* Enhanced SOLR Config
* Advanced Personalisation Samples
* Promotions
* Customer Review Samples
* Cross-Sells

#### UK Apparel Store

* Shared Product Catalog (with DE Apparel) with content in 2 languages :
  + Snowboard and Surf
* Product Data Model:
  + Variants
* Exclusive Content Catalog with content in 1 language
* Languages:
  + English GB
* Currencies:
  + GBP
* Gross Prices
* Shipping:
  + UK territory only
* Site Theme:
  + Black
* Enhanced SOLR Config (shared with DE Apparel)
* Some Promotions

#### DE Apparel Store

* Shared Product Catalog (with UK Apparel) with content in 2 languages :
  + Snowboard and Surf
* Product Data Model:
  + Variants
* Exclusive Content Catalog with content in 1 language
* Languages:
  + German (default)
* Currencies:
  + EUR
* Gross Prices
* Shipping:
  + Continental Europe
* Site Theme:
  + Apparel
* Enhanced SOLR Config (shared with UK Apparel)
* Some Promotions

### commerceservices Extension - Technical Guide

The hybris **commerceservices** extension organizes functionality from one or more hybris Platform services. These services fulfill certain tasks on their own but often need to be combined to provide a complete B2C commerce use case. This often involves combining functionality from separate extensions that are licensed separately. The **commerceservices** extension also extends more generic functionality from certain hybris extensions to add a more B2C commerce features.

|  |  |
| --- | --- |
|  | **Note**  **Before Implementing** A hybris extension may provide functionality that is licensed through different hybris modules. Make sure to limit your implementation to features as defined in your license contract. In case of doubt, please contact your hybris Sales representative. |

* [Overview of Services](#168)
  + [Catalog Services](#169)
    - [Commerce Category Service](#170)
    - [Commerce Price Service](#171)
      * [Net Gross Strategy](#172)
      * [Sales Tax](#173)
    - [Commerce Product Service](#174)
  + [Facet Search Enhancements](#175)
  + [Order Services](#176)
    - [Commerce Cart Service](#177)
    - [Commerce Checkout Service](#178)
    - [Delivery Service](#179)
  + [Customer Account Services](#180)
    - [Customer Account Service](#181)
    - [Customer Email Resolution Service](#182)
    - [Secure Token Service](#183)
      * [DefaultSecureTokenService](#184)
  + [Email Services](#185)
    - [Email Service](#186)
    - [Email Business Process](#187)
      * [CMS Email Page Service](#188)
    - [Email Generation Service](#189)
    - [Site Base URL Resolution Service](#190)
  + [Store Locator Services](#191)
    - [Store Lookup Service](#192)
  + [I18n Services](#193)
    - [Commerce Common I18n Service](#194)
* [Data Model](#195)
  + [Product](#196)
    - [Using Media Containers for a Gallery](#197)
    - [Customer Reviews](#198)
  + [Order](#199)
    - [Tracking the Site in which the order was placed](#200)
    - [Payment](#201)
  + [Customer](#202)
  + [Storefront Setup](#203)
    - [BaseStore Financial Settings](#204)
    - [CMSSite Localization Settings](#205)
    - [CMSSite Presentation Settings](#206)
  + [Solr Facet Search](#207)
  + [Email](#208)
  + [Store Locator](#209)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About this Document**   | **Extension** | **Release 4.0** | **Release 4.1** | **Release 4.2** | **Release 4.3** | **Release 4.4** | | --- | --- | --- | --- | --- | --- | | **namename** | 4.0 | 4.1 | 4.2 | 4.3 | 4.4.0 |  | **Directory** | [**Download**](#100) **With** | **Related To** | | --- | --- | --- | | bin/accelerator | hybris Multichannel Accelerator | * [hybris Multichannel Accelerator](#85) |   This document introduces the hybris **commerceservices** extension.  **Audience**: Consultants, developers, technical deciders, software architects  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

#### Overview of Services

The following is a guide to all the new services and existing service enhancements provided with the **commerceservices** extension. Dependencies on extensions outside of the core **Platform** are highlighted for your reference.

##### Catalog Services

###### Commerce Category Service

The **CommerceCategoryService** adds category lookup functionality within the scope of only the current session product catalogs. Other types of session catalog, i.e. content catalogs and classification catalogs are excluded.

###### Commerce Price Service

|  |  |
| --- | --- |
|  | **Note**  Note the default implementation adds a dependency to the **basecommerce** extension |

The **CommercePriceService** is intended to replace the use of the existing **PriceService**. This provides a reduced interface to only expose typical B2C product price requests such as getting the current web price or a from price, in event of variants. All the additional **PriceService** complexities such as price banding and deciding which price is appropriate is either hidden away or simply not supported by this service.

|  |
| --- |
| **See Also**   * [basecommerce Extension - Technical Guide](#73) |

Net Gross Strategy

The **commerceservices** extension extends **basecommerce** **BaseStore** to include a net flag attribute that identifies whether prices on a storefront should be net or gross of sales tax. A **NetGrossStrategy** is used by all services wanting to determine if net or gross prices should be returned. The **NetPriceService** which is the **commerceservices** extension extended version of **PriceService** uses this strategy. There are two implementations out of the box; **DefaultNetGrossStrategy** which uses the default **PriceFactory** rules and **CommerceNetGrossStrategy** which uses the value from the **BaseStore**.

Sales Tax

The **commerceservices** extension extends the **basecommerce** extension **BaseStore** to allow a **UserTaxGroup** to be specified. This can be used to set the tax group to ensure correct sales tax is picked up by the **PriceService** when sites are deployed across multiple countries.

###### Commerce Product Service

|  |  |
| --- | --- |
|  | **Note**  Note the default implementation adds an implicit dependency to the **basecommerce** extension. |

The commerce product service adds extra B2C relevant functions not provided in default **ProductService**. This includes classification class filter functionality and switchable stock level support.

|  |
| --- |
| **See Also**   * [basecommerce Extension - Technical Guide](#73) |

##### Facet Search Enhancements

|  |  |
| --- | --- |
|  | **Note**  The **commerceservices** extension has a dependency on the **solrfacetsearch** extension |

A number of B2C commerce functions have been added to extend the functionality provided in the **solrfacetsearch** extension in the **commerceservices** extension. For more information please refer to the [Search and Navigation in the hybris Multichannel Accelerator](#38) document.

|  |
| --- |
| **See Also**   * [solrfacetsearch Extension - Technical Guide](#68) * [Search and Navigation in the hybris Multichannel Accelerator](#38) |

##### Order Services

###### Commerce Cart Service

|  |  |
| --- | --- |
|  | **Note**  Note that the default implementation adds an implicit dependency to the **basecommerce** and **promotions** extensions |

The **CommerceCartService** replaces the **CartService** by adding promotions calculation and stock checks to the Cart services.

|  |
| --- |
| **See Also**   * [basecommerce Extension - Technical Guide](#73) |

###### Commerce Checkout Service

|  |  |
| --- | --- |
|  | **Note**  Note that the default implementation adds an implicit dependency to the **basecommerce**, **promotions** and **cms2** extensions. The interface also has an explicit dependency on the **payment** extension. |

The **CommerceCheckoutService** adds services to orchestrate typical operations during checkout, such as registering a new credit card, adding and setting a delivery address, choosing a delivery method and placing the order. It is intended to replace direct use of the **CartService** and **OrderService**.

|  |
| --- |
| **See Also**   * [basecommerce Extension - Technical Guide](#73) * [cms2 Extension - Technical Guide](#78)   + [cms2 Extension Tutorial](#210) * [payment Extension - Technical Guide](#75) |

###### Delivery Service

The **DeliveryService** provides functionality around available and supported delivery rules that are scoped to a single store front rather than an entire platform deployment which may include multiple store fronts. This includes identifying supported delivery countries and delivery options.

##### Customer Account Services

###### Customer Account Service

|  |  |
| --- | --- |
|  | **Note**  Note that the default implementation adds an implicit dependency to the **cms2** extension. Furthermore the interface adds an explicit dependency to the **basecommerce** and **payment** extensions. |

Customer account service handles typical customer account management capabilities. Provides methods for registering, verifying user credentials, updating profile settings, forgotten password services, managing address books, viewing users orders for the current store front and managing payment information using a PCI compliance friendly subscription interface.

|  |
| --- |
| **See Also**   * [basecommerce Extension - Technical Guide](#73) * [cms2 Extension - Technical Guide](#78)   + [cms2 Extension Tutorial](#210) * [payment Extension - Technical Guide](#75) * [Authentication with Credit Card by Saved Token](#211) |

###### Customer Email Resolution Service

It simply adds a layer of abstraction on top of **CustomerModel** to resolve their email address.

###### Secure Token Service

It is a service to encrypt **SecureToken** data into a short encrypted token and to decrypt a token.

The **SecureToken** data consists of a data string and a timestamp. And the resultant encrypted string, the token is in Base64 format.

The **SecureTokenService** is being used in forgotten password request process. When a customer requests for forgotten password, a token will be generated using the service by passing **SecureToken** data consists of customer uid and a timestamp. The generated token will be maintained in the system against the customer and an email will be sent to the customer with a password reset URL incorporating the token. When the customer tries to reset the password by visits the password reset page using the password reset URL, the token will be decrypted and validated if its still valid. It is also verified with the token maintained in the system against the customer before resetting the customer's password.

DefaultSecureTokenService

The **DefaultSecureToeknService** implements the **SecureTokenService** as described below:

The data to be encrypted is signed and encrypted using two separate keys. The keys are specified as hex character strings. Unlimited strength cryptography is required. The data assembled to encrypt as follows: the **Signature** key (not included in data block) and the data block consist data string in the SecureToken, generated checksum for data string in the **SecureToken** and the timestamp in the **SecureToken**. The data block also prefixed and postfixed with padding lengths. All the above, except signature key, is AES encrypted using encryption key. Result is Base64ed.

##### Email Services

The **commerceservices** extension adds extra functionality to asynchronously generate and send emails and track emails that have been sent with business processes using Process Engine. More detailed information can be found in the [Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator](#37) document.

|  |
| --- |
| **See Also**   * [processengine Extension - Technical Guide](#63) * [Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator](#37) |

###### Email Service

The **EmailService** handles the creation and sending of emails. Services are provided to enable a record of the email to be persisted for auditing purposes.

###### Email Business Process

The **commerceservices** extension introduces a new set of business process actions that support the ability to asynchronously generate email content with CMS managed sections and then send the emails using an SMTP server.

The following actions have been implemented:

* Generate Email Body
* Send Email
* Remove Email From Database.

CMS Email Page Service

|  |  |
| --- | --- |
|  | **Note**  Note that the interface has an explicit dependency on the **cms2** extension. |

The **CMSEmailPageService** adds the ability to manage parts of an emails content in the WCMS Cockpit. A specialized kind of page template called an **EmailPageTemplate** allows the template to be linked to a **RendererTemplate** from the **commons** extension whilst a special type of CMS page called an **EmailPage** allows you to add WCMS components to allocated slots defined by your **EmailPageTemplate** and placed in the email by the **RendererTemplate**.

|  |
| --- |
| **See Also**   * [cms2 Extension - Technical Guide](#78) * [commons Extension - Technical Guide](#212) |

###### Email Generation Service

The **EmailGenerationService** is responsible for generating the email content and attaching it to a new **EmailMessage**.

###### Site Base URL Resolution Service

|  |  |
| --- | --- |
|  | **Note**  Note that the interface has an explicit dependency on the **cms2** extension. |

Emails are typically sent asynchronously. The luxury of having the **HttpRequest** object to build up URL paths is not available. The **SiteBaseUrlResolutionService** returns the base site or media URL for the specified **CMSSite**. This enables you to generate emails with valid links to site content or imagery. This service can also be applied for other asynchronous exports, such as site map generation, that might require the base URL.

|  |
| --- |
| **See Also**   * [cms2 Extension - Technical Guide](#78) |

The default implementation simply uses the **ConfigurationService** and a naming convention using the site UID.

|  |
| --- |
| website.apparel-uk.http=http://apparel.uk.local:9001/acceleratorstorefront website.apparel-uk.https=https://apparel.uk.local:9002/acceleratorstorefront website.apparel-de.http=http://apparel.de.local:9001/acceleratorstorefront website.apparel-de.https=https://apparel.de.local:9002/acceleratorstorefront website.electronics.http=http://electronics.local:9001/acceleratorstorefront website.electronics.https=https://electronics.local:9002/acceleratorstorefront   media.apparel-uk.http=http://apparel.uk.local:9001 media.apparel-uk.https=https://apparel.uk.local:9002 media.apparel-de.http=http://apparel.de.local:9001 media.apparel-de.https=https://apparel.de.local:9002 media.electronics.http=http://electronics.local:9001 media.electronics.https=https://electronics.local:9002 |

Table 18 Sample config from accelerator sample storefronts

##### Store Locator Services

###### Store Lookup Service

|  |  |
| --- | --- |
|  | **Note**  Note the interface has an explicit dependency on the **basecommerce** extension. |

The **StoreLookupService** orchestrates the various services provided by the hybris Store Locator in the **basecommerce** extension to deliver typical store locator page functionality like nearest store, search by town or postal code, maps for locations in proximity to current location and content required for a store information page.

Additional **Bricks and Mortar** store attributes are added to **PointOfService** to capture content like a store image, features and some marketing content for a store details page. You can find more information in the [Setting Up the Store Locator in the hybris Accelerator](#213) document, section **Store Content**.

##### I18n Services

###### Commerce Common I18n Service

|  |  |
| --- | --- |
|  | **Note**  Note the interface has an explicit dependency on the **cms2** extension. |

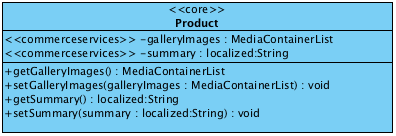
The **CommerceCommonI18nService** complements hybris [Internationalization and Localization](#214) capabilities by providing storefront scoped services for obtaining default or supported languages, currencies, supported territories and operating locale to the current executing storefront.

|  |
| --- |
| **See Also**   * [cms2 Extension - Technical Guide](#78) |

#### Data Model

**commerceservices** adds a number of data model modifications to add further B2C commerce support to existing more generic extensions or combine data model from multiple extensions to enable additional B2C commerce functions.

##### Product



* A marketing purpose **summary** attribute has been added to product to complement the existing name and large description attribute. This can be used when a more concise product description is required.
* The **galleryImages** attribute is used to store multiple images each resized to a number of standard formats expected by the storefront. A list of **MediaContainer** objects are used to model the gallery, more information can be found in the next section.

###### Using Media Containers for a Gallery

Media Containers are a useful way of grouping related images. **commerceservices** updates the **Product** data model to support a gallery which enables us to store multiple groups of images. Each image group is just the same image that is resized up-front to dimensions that are expected to fit on the storefront, such as thumbnail, detail or a zoom. This enables the storefront to serve a link to an image that is correctly size optimized thus not requiring any browser or client side resizing. This also ensures bandwidth is not unnecessarily wasted when downloading images to the client browser. For galleries to work well and look good, each image should be resized to a set of standard dimensions and formats.

hybris uses Medias to store image information such as its location on the web server, and a Media has an attribute of type Media Format that can be used for the purpose of logically defining the dimensions of an image.

For example a 'thumbnail' format could be 60x40px and a 'zoom' format could be 1000x700px.

A Media Container's primary function is to group related Media's with some extra context information about the related Media's.

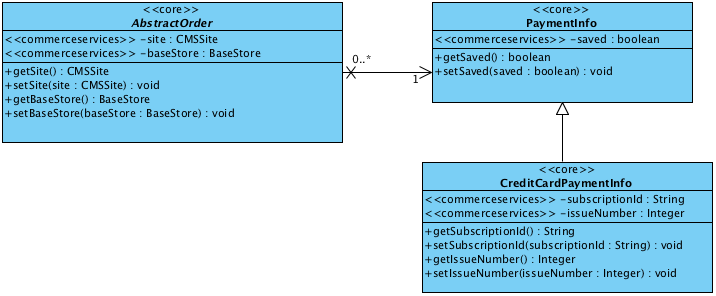
The Gallery Images attribute is modeled as a List of Media Containers. If we need to group all the related media that differ only by format, for example by different dimensions, together, to do this we can use ﻿Media Containers. Media Containers are catalog version aware as well which means changes can be uploaded to a Staged Product Catalog and then published Online.

|  |
| --- |
| **See Also**   * [Handle Medias Using the MediaContainer](#216) * [Managing Media in the hybris Product Cockpit](#217) |

###### Customer Reviews

The **customerreview** extension data model has been extended to support an approval process and additional functionality.  
See the [Customer Reviews in the hybris Multichannel Accelerator](#218) document for the data model and more information about the extended functionality.

##### Order



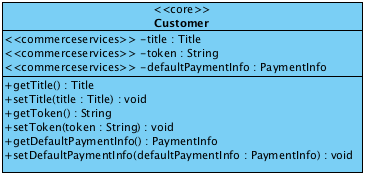
###### Tracking the Site in which the order was placed

* When an order is placed, the current active **CMSSite** and **BaseStore** is saved against the Order. When a cart is created, the current active **CMSSite** and **BaseStore** will be saved against the Cart. In a multiple storefront scenario, this can be used to determine which storefront the order was placed for reporting as well as filtering purposes.

###### Payment

* It is possible now to flag whether a **PaymentInfo** should be used as saved payment for a customer.
* The **CreditCardPaymentInfo** now has a place to save the subscription ID when sensitive credit card details such as the full card number and CVV number are saved in a third party payment system. The subscription ID is passed to our payment service during order placement.
* We have also added an issue number to support a subset of Switch and Maestro credit cards.

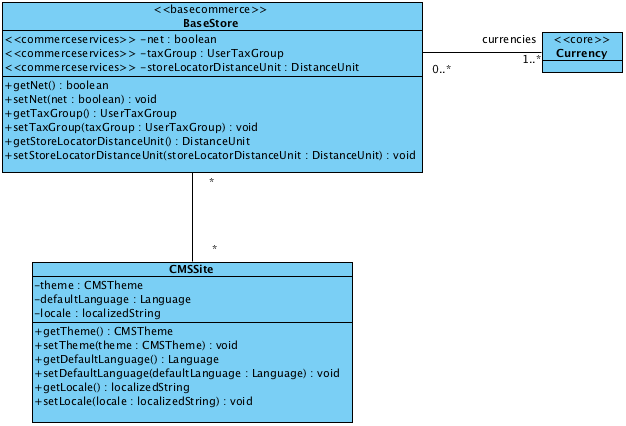
##### Customer



* A title attribute has been added to Customer.
* A customers default payment information can now be stored against the customer.
* A special token attribute has been added to support the ability to authenticate certain Customer Account requests that would need the same token to be passed back from the client as a request parameter. The forgotten password functionality, described in the [Customer Account Service](#181) section above, uses this store a secure token that can only be validated once.

##### Storefront Setup

A number of additional attributes have been added to **BaseStore** and **CMSSite** to capture configuration settings for each storefront.



###### BaseStore Financial Settings

* A net flag has been added to identify if the store prices are calculated net or gross of sales tax. This is used by the price calculation logic.
* The list of currencies supported for the store are now captured against the store.
* The user tax group from the **europe1** extension is stored to enable the price calculation logic to pick appropriate store specific sales tax rates in a multi storefront setup.

###### CMSSite Localization Settings

* The default language of the storefront is configurable. This is especially relevant if content on a specific storefront comes in multiple languages. The total list of languages available comes from the active content catalog version.
* The Java Locale to use for each language is now configures to ensure numbers and currency information is formatted appropriately on the storefront.

###### CMSSite Presentation Settings

* A theme attribute has been added to allow the look and feel of the Storefront to be switched.

|  |
| --- |
| **See Also**   * [Storefront and Catalog Modelling in the hybris Multichannel Accelerator](#219) * [Website Configuration in the hybris Multichannel Accelerator](#220) |

##### Solr Facet Search

A number of additions to the data model have been made to support a more enriched B2C SOLR configuration out of the box.  
See the [Search and Navigation in the hybris Multichannel Accelerator](#38) document for the data model and more information.

##### Email

The **commerceservices** extension adds Email Generation and Audit functionality combined with additional processes for use with the hybris process engine.  
See the [Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator](#37) document for the data model and more information.

##### Store Locator

The Store Locator data model has been extended to include more attributes specific for Bricks And Mortar stores.  
See the [Setting Up the Store Locator in the hybris Multichannel Accelerator](#105) document for the data model and more information.

### commercefacades Extension - Technical Guide

The hybris **commercefacades** extension provides a typical suite of storefront actions that make up a unified multichannel storefront API. Multiple front-ends can therefore use the same storefront facades. The facades responsibility is to take requests from the front-end and service the request by integrating business services from a range of the hybris extensions and exposing a Data Object Model adjusted to meet the storefront requirements.

|  |  |
| --- | --- |
|  | **Note**  **Before Implementing** A hybris extension may provide functionality that is licensed through different hybris modules. Make sure to limit your implementation to features as defined in your license contract. In case of doubt, please contact your hybris Sales representative. |

* [Breakdown of a Facade](#221)
  + [Facade Bean](#222)
  + [Data Objects](#223)
  + [Converters and Populators](#224)
    - [Converters](#225)
    - [Populators](#226)
    - [Configurable Populators and Data Options](#227)
    - [Base Classes](#228)
  + [Source Code Example](#229)
  + [Extending](#230)
  + [Conceptual Interaction Diagram](#231)
* [Packaged Facades](#232)
  + [Search](#233)
  + [Product](#234)
  + [Order](#235)
  + [User](#236)
  + [Store Session](#237)
  + [Store Locator](#238)
  + [Packaged Converters](#239)
  + [Packaged Populators](#240)
* [Extending Facades](#241)
* [URL Resolvers](#242)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **About this Document**   | **Extension** | **Release 4.0** | **Release 4.1** | **Release 4.2** | **Release 4.3** | **Release 4.4** | | --- | --- | --- | --- | --- | --- | | **namename** | 4.0 | 4.1 | 4.2 | 4.3 | 4.4.0 |  | **Directory** | [**Download**](#100) **With** | **Related To** | | --- | --- | --- | | bin/accelerator | hybris Multichannel Accelerator | * [hybris Multichannel Accelerator](#85) * [acceleratorfacades Extension - Technical Guide](#86) |   This document introduces the hybris **commercefacades** extension.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Consultants, developers, technical deciders  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

#### Breakdown of a Facade

##### Facade Bean

A facade in the **commercefacades** extension is a Spring managed bean that is an implementation of a software design pattern of abstracting the underlying implementation responsible for servicing a storefront **action** by exposing a simplified interface optimized for the storefront.

|  |
| --- |
| **See Also**   * [Using Facades and DTOs - Best Practice](#243) |

In general, facades typically integrate one or more hybris ServiceLayer method calls that, when combined, fulfill some form of **action** requested by a storefront user. The **action** is fulfilled by the front-end framework, which typically is an MVC controller, by building up the necessary parameters for the action and invoking a single method on the facades interface. The facade interface is however totally independent of the ServiceLayer model, so it is possible to completely substitute the underlying implementation of a facade **action**.

Examples of storefront actions include:

* Viewing product details.
* Adding a product to a cart.
* Adding a delivery address during checkout.
* Posting a review.
* Searching for products with a free text search.
* Refining to products under a certain price in the camera lenses category.
* Sorting products by price.

For more information on general facade concept, refer to [Using Facades and DTOs - Best Practice](#243) document.

##### Data Objects

Facades return a Data Object model for the front end that is the view of the information a storefront user would require exposure to. They typically are populated using a subset of data from the hybris ServiceLayer models. A Data Object may also be constructed from multiple models as well as from data that is derived after executing business logic on services. These Data Objects form part of the **Model** delivered to construct the View in the default MVC pattern. The hybris ServiceLayer Models form no part of a facades interface, maintaining a clean abstraction of the business layer and the presentation layer.

|  |
| --- |
| **See Also**   * [Models](#244) * [ServiceLayer](#44) |

##### Converters and Populators

###### Converters

Facade Data Objects are constructed from Models or other Service Layer objects using Converters. Converters create new instances of Data Objects using a source business object as a baseline. This instance is called the Data Object prototype.

|  |
| --- |
| **See Also**   * [Using Facades and DTOs - Best Practice](#245), section **Converters**. |

###### Populators

The **commercefacades** extension adds an additional type of component for use in the conversion process called a **Populator**. Populators allow for the conversion process of filling out a Data Object prototype from Business Objects and/or services to be broken down into a pipeline of Population tasks or steps. Each Populator carries out one or more related updates to the Data Object prototype. Each Population step can invoke services or copy data from the source business object to the prototype Facade Data Object. Facades always use a Converter to create a new instance of a Data Object prototype but can directly invoke Populators or Converters to fulfill the task of building up the Data Object.

###### Configurable Populators and Data Options

Configurable Populators extend the Populator interface by allowing a collection of Enum type data options to be provided. The Configurable Populator then invokes only Populators that add data for the given data options. This functionality may appear complex but is vital from a performance and bandwidth perspective. Constructing a whole object graph is often totally unnecessary for the client needs. This may be costly from a performance point of view server side, for example consider evaluating promotions for a product unnecessarily or attaching classification data when not required, as well as a data bandwidth point, for example(product description can be rather sizable, and therefore returning it in web service call is not necessary. Configurable Populators also allow for extra population logic to be injected into a population pipeline without the need to replace or override existing code.

###### Base Classes

* Since Converters can populate data objects and therefore can also be deemed Populators, ***AbstractConverter*** is a convenience base class that implements both the *Converter* and *Populator* interface and provides the basic implementation of the *Converter* interface delegating to a *createTarget* template method that allows a concrete sub-class to pick the appropriate target data object implementation.
* ***AbstractPopulatingConverter*** simply extends *AbstractConverter* and in addition provides an implementation of *populate* from the *Populator* interface. It is recommended that most Converters extend this base class and simply override the *createTarget* template method and inject the necessary pipeline of Populators to fill out the object graph of the target. This approach leads to the most flexible and adaptable code where much of the behaviour can be adapted externally through spring bean configuration.
* ***DefaultConfigurablePopulator*** is an implementation of a Populator pipeline where each population step is evaluated against a Set of *Enum* values passed by the caller. Each Populator in the pipeline will only run if it is mapped to an *Enum* that is contained in the Set passed by the caller.

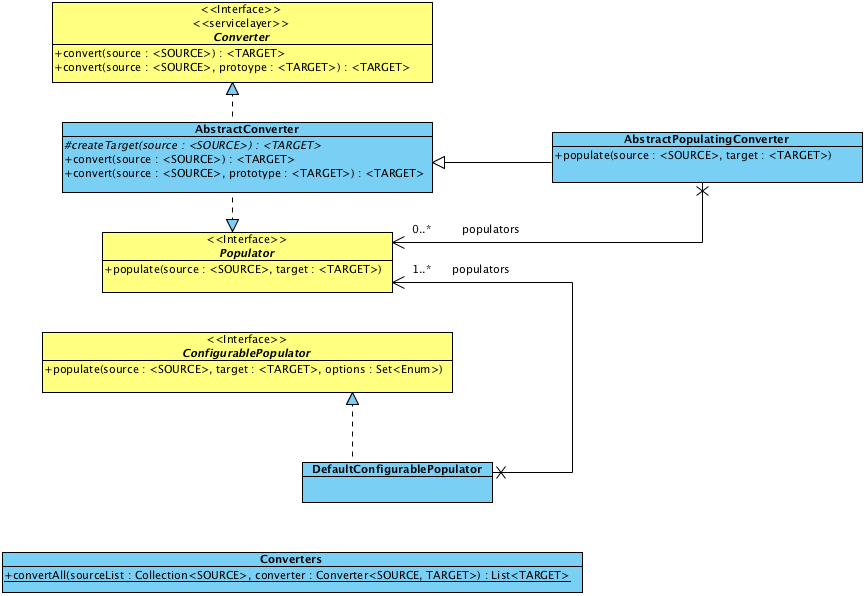


Figure: Diagram of converters and populators.

##### Source Code Example

We can refer to the **ProductFacade** **getProductByCode** method for a suitably complex example. The facade interface takes a product article number together with a set of data options as instruction from the front end as to the level of data that needs to be returned in the resulting **ProductData** object. The example uses a Converter, Populators, Configurable Populators and Services.

|  |
| --- |
| ProductData getProductForCode(String code, Set<ProductOption> options) throws UnknownIdentifierException,  IllegalArgumentException; |

The implementation in **DefaultProductFacade** looks like this:

|  |
| --- |
| @Override public ProductData getProductForCode(final String code, final Set<ProductOption> options) {  final ProductModel productModel = getProductService().getProductForCode(code);  final ProductData productData = getProductConverter().convert(productModel);   if (options != null)  {  getProductConfiguredPopulator().populate(productModel, productData, options);  }   return productData; } |

It first calls the **ProductService** to get the product model for the provided article number. The **ProductConverter** is then used to create the prototype **ProductData** object with just basic data filled out.

|  |
| --- |
| // From ProductUrlConverter @Override protected ProductData createTarget(final ProductModel source) {  return new ProductData(); }  // From ProductConverter @Override public void populate(final ProductModel source, final ProductData target) {  getProductBasicPopulator().populate(source, target);  getVariantSelectedPopulator().populate(source, target);  getProductPrimaryImagePopulator().populate(source, target);   super.populate(source, target); } |

The **ProductConverter** extends **ProductUrlConverter** of which extends are recommended base class **AbstractPopulatingConverter**. The base classes starts by creating a new instance of the prototype by calling the *createTarget* template method. In this instance a **ProductData** object is returned.

|  |  |
| --- | --- |
|  | **Extensibility Tip**  If you wish to add extra attributes to the baseline **ProductData** for your project, you can subtype the **ProductData** class and override the *createTarget* method to return your subtype instead. |

For this case, the **ProductConverter** defers to some hardwired Populators to set a few essential attributes that are always set when a user gets a skeleton **ProductData** back from the converter. This is a design decision to ensure these attributes are already set, it could also have been possible to just rely on the correct populators being configured on the converter as the base class infrastructure will run these.

|  |  |
| --- | --- |
|  | **Extensibility Tip**  This means that you may only need to replace or extend a Populator instance rather than a whole Converter to populate additional attributes on your customized data model. Alternatively you can configure additional populators on a Converter using spring configuration or extend the Converter class by overriding the convert method and invoking additional hard configured Populators |

The **ProductBasicPopulator** looks like this:

|  |
| --- |
| @Override public void populate(final ProductModel productModel, final ProductData productData) throws ConversionException {  productData.setName((String) getProductAttribute(productModel, ProductModel.NAME));  productData.setManufacturer((String) getProductAttribute(productModel, ProductModel.MANUFACTURERNAME));   productData.setAverageRating(productModel.getAverageRating());  if (productModel.getVariantType() != null)  {  productData.setVariantType(productModel.getVariantType().getCode());  }   productData.setPurchasable(Boolean.valueOf(productModel.getVariantType() == null && isApproved(productModel))); }  protected boolean isApproved(final ProductModel productModel) {  final ArticleApprovalStatus approvalStatus = productModel.getApprovalStatus();  return approvalStatus != null && ArticleApprovalStatus.APPROVED.equals(approvalStatus); } |

|  |  |
| --- | --- |
|  | **Implementation Detail**  Product Populators are a little unique in that they typically extend a **AbstractProductPopulator** class that is variant aware and supports the ability of falling back to a variants parent product for attribute values in the event of the source product value being **null**. |

The **ProductFacade** uses then its configured Populator to populate the prototype Data Object based on the requested data options. The configured Populator is simply a spring configured pipeline of populators that are invoked in a configured order and only invokes if they are assigned to an option that has been specified. Here's a sample spring configuration of the **ConfigurablePopulator** used by the **ProductFacade**.

|  |
| --- |
| <alias name="defaultProductConfiguredPopulator" alias="productConfiguredPopulator" />  <bean id="defaultProductConfiguredPopulator" scope="tenant" class="de.hybris.platform.commercefacades.convert.impl.DefaultConfigurablePopulator">  <property name="populators">  <map key-type="de.hybris.platform.commercefacades.product.data.ProductData$ProductOption">  <entry key="GALLERY" value-ref="productGalleryImagesPopulator" />  <entry key="SUMMARY" value-ref="productSummaryPopulator" />  <entry key="DESCRIPTION" value-ref="productDescriptionPopulator" />  <entry key="CATEGORIES" value-ref="productCategoriesPopulator" />  <entry key="PROMOTIONS" value-ref="productPromotionsPopulator" />  <entry key="STOCK" value-ref="productStockPopulator" />  <entry key="REVIEW" value-ref="productReviewsPopulator" />  <entry key="CLASSIFICATION" value-ref="productClassificationPopulator" />  <entry key="REFERENCES" value-ref="productReferencesPopulator" />  <entry key="VARIANT\_FULL" value-ref="variantFullPopulator" />  </map>  </property>  </bean> |

As it was mentioned above, some Populators may use services to fill out the prototype Data Object rather than properties on the source model. You can review the **ProductPricePopulator** as an example of a Populator that uses a service to fill out part of the data model rather than the source object directly.

|  |
| --- |
| @Override public void populate(final ProductModel productModel, final ProductData productData) throws ConversionException {  final PriceData.PriceType priceType;  final PriceInformation info;  if (CollectionUtils.isEmpty(productModel.getVariants()))  {  priceType = PriceData.PriceType.BUY;  info = getCommercePriceService().getWebPriceForProduct(productModel);  }  else  {  priceType = PriceData.PriceType.FROM;  info = getCommercePriceService().getFromPriceForProduct(productModel);  }   if (info != null)  {  final PriceData priceData = getPriceDataFactory().create(priceType, BigDecimal.valueOf(info.getPriceValue().getValue()),  info.getPriceValue().getCurrencyIso());  productData.setPrice(priceData);  }  else  {  productData.setPurchasable(Boolean.FALSE);  } } |

##### Extending

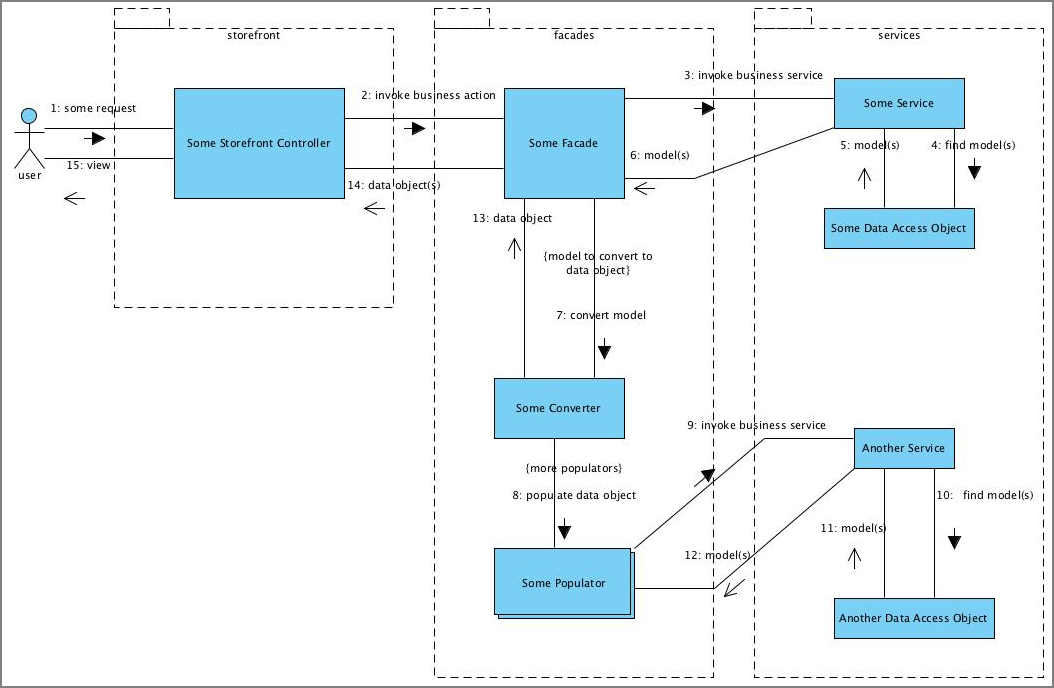
As you may have deemed from reading the previous content, there are many ways to extend the functionality of facades and converters. All of this can be achieved by re-wiring up beans in the application context. This includes but is not limited to:

* Adding new facades
* Extending existing facades interface and implementation to add additional methods
* Extending existing facade implementation to replace the original method implementation
* Creating a different or additional versions of a converter for a particular type
* Extending an existing converter implementation
* Adding a new Populator to a Populator pipeline
* Removing or replacing Populator implementations
* Adding new data objects and counterpart Populators and Converters
* Extending existing data objects and adding additional converter and/or Populator logic

|  |
| --- |
| **See Also**   * [How To Extend the ProductFacade - Tutorial](#96) |

##### Conceptual Interaction Diagram

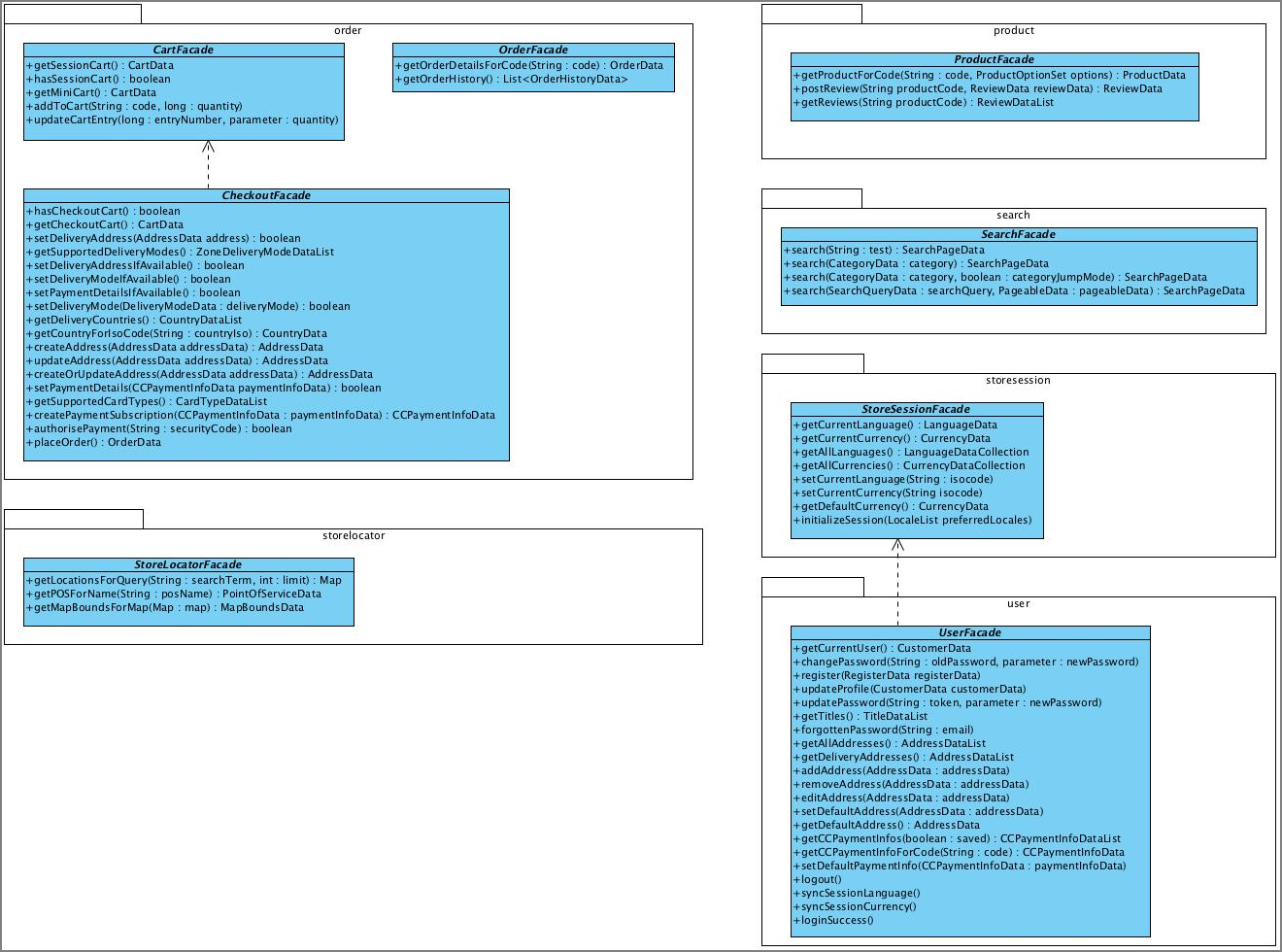
The following interaction diagram describes conceptually how the various components interact during a typical storefront request:



A typical storefront interaction can be described as follows:  
1. A user performs some action which is processed by a storefront component such as an MVC controller.  
2. The controller translates the request and invokes a business action on the appropriate facade.  
3. The facade invokes one or more business services to fulfill the action.  
4-6. The service most likely needs to return some model object and perhaps uses Data Access Objects to find the models to return to the calling facade.  
7. The facade uses converters to create front-end data objects from service layer models.  
8. Converters use a pipeline of Populators **fill** the skeleton front end data objects graph.  
9-12. Populators may also need to invoke additional business logic to fill the front end data object graph so may also invoke services in order to provide the necessary data required by the front-end data object.  
13-14. The filled data object is returned to the facade and the facade then returns this to the controller.  
15. The controller uses the data object as a model to construct  the view which will be returned to the user.

#### Packaged Facades

The following class diagram outlines the various facades offered by the **commercefacades** extension.



##### Search

The search facade exposes product free text search and facetted navigation capability.

|  |
| --- |
| **See Also**   * [Search And Navigation](#246), section **Commerce Facades** |

##### Product

The product facade delivers product information. It is possible to control the amount of information returned for the product by specifying data options. The product facade also enables reviews to be posted for products.

##### Order

The order facades provide the full shopping capability with all the typical cart, checkout and order history functions. The functionality is broken down into three separate facades, cart facade for the shopping phase, checkout facade for the order placement process and order facade for bringing up details of submitted orders and order history.

##### User

The User facade provides user account operations such as registration, payment and address book management, profile updates including password amendments.

##### Store Session

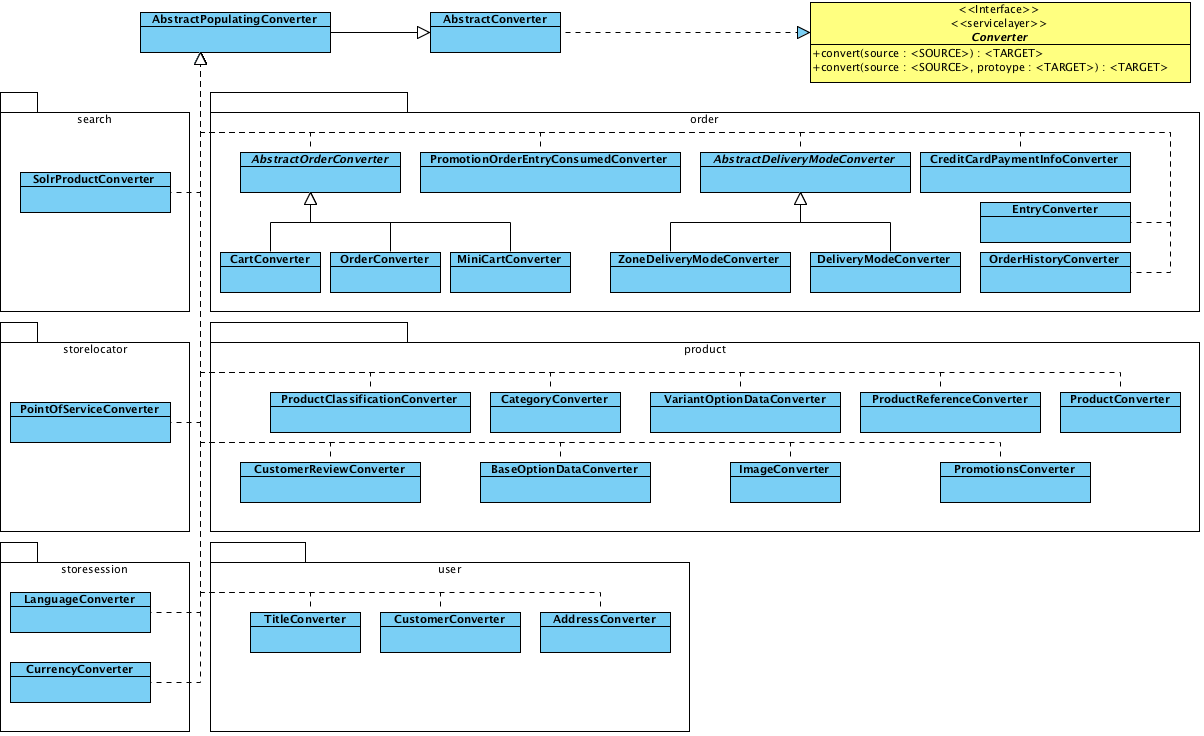
The Store Session facade provides access to various internationalization switches that the user can make when they visit a specific storefront.

##### Store Locator

The Store Locator facade exposes Point Of Sale search capability and provides the ability to expose store information and store specific content. Find more information on the Store Locator within the hybris Accelerator in the [Setting Up the Store Locator in the hybris Multichannel Accelerator](#105) document.

##### Packaged Converters

The following diagram shows the Converters shipped with the **commercefacades** extension:



##### Packaged Populators

The following diagram shows the Populators shipped with the **commercefacades** extension:



#### Extending Facades

Facades are tenant or singleton scoped Spring managed beans that have an interface and a default implementation class that can be customized by a number of means including replacing the entire bean, subclassing the bean class and overriding specific methods or customization by composition, typically by populating the data object model using different or additional instances of converters and populators.

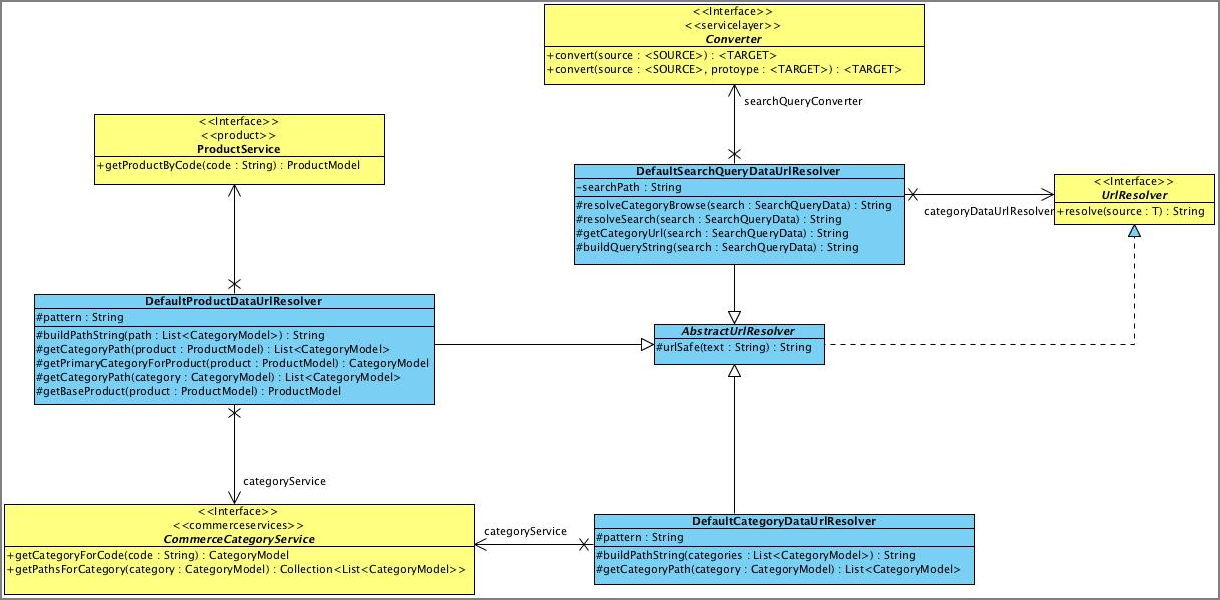
You should extend the **commercefacades** extension facades in a counterpart **facades** extension for your project. The **acceleratorfacades** extension is the template shipped with the hybris Accelerator that you can use as a starting point for your own extension. This extension must have a dependency on the **commercefacades** extension in its extensioninfo.xml file as well as any other necessary business layer extension required to complete the functionality offered by your customized or additional facades.

|  |  |
| --- | --- |
|  | **Implementation Detail**  Adding the dependency on the **commercefacades** extension in the extensioninfo.xml ensures you can replace bean definitions defined in the **commercefacades** extension application context. You can find more information on extending Spring beans can in [Spring Integration](#247), [ServiceLayer](#44) and more specifically [How To Extend a Service - Tutorial](#248) documents. |

#### URL Resolvers

A URL resolver creates a relative URL to a provided source item that can be used on the front end to generate a link back to the source item. This ensures the facade layer is able to supply URLs in the data object model without the MVC layer having to execute an additional pass over the data object model. This approach also allows multiple client applications to use different URL formats by simply using facades and converters configured with different resolver instances. Out of the box, SEO friendly URLs can be generated for product and category pages as well as facetted navigation pages.

|  |
| --- |
| **See Also**   * [Search And Navigation](#249), section **URL Generation** |



## Advanced Personalization in the hybris Multichannel Accelerator

The hybris Advanced Personalization lets you easily set up the rules, depending on several criteria, enabling the segmentation of your customers into different groups. In a second step you can use this segmentation, like personalizing the content which is shown to your customers.

Technically this functionality is provided by the **btg** and **btgcockpit** extensions. Find more information on the Advanced Personalization in [Advanced Personalization Module](#148) document.

* [Advanced Personalization Integration](#250)
  + [Event Producer](#251)
  + [Rules](#252)
  + [Configuration](#253)

|  |
| --- |
| **About this Document**  This document provides information on the integration of the Advanced Personalization Module with the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, software architects, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [Advanced Personalization Module](#148) * [btg Extension - Technical Guide](#80) * [btgcockpit Extension - Technical Guide](#254) * [acceleratorstorefront Extension - Technical Guide](#97) * [Storefront Web Application Deconstructed](#33) |

### Advanced Personalization Integration

The **btg** extension is integrated in the hybris Multichannel Accelerator via one of the Spring configurable **RequestFilters**, **DelegatingFilterProxy**). There are two main parts of this integration:

* Event publishing - adding scoped data to the knowledge base used to validate rules against segmentation.
* Segmentation - adding the user to segments based on fulfilled rules.  
  The first part is realized by event producing filters, the second by adding the **BTGSegmentFilter** to the active filter chain. For details regarding the inner working of the **BTGSegmentFilter** see [How To Integrate Advanced Personalization with Front-end Application - Tutorial](#147).

#### Event Producer

Out of the box there are two different types of events:

* Request Scoped: Referer Header, Request Params.
* Session Scoped: Category Visited, Product Visited, ContentPage Visited.



Figure: Organization of Advanced Personalization within the hybris Multichannel Accelerator.

The **AbstractBtgFilter** is responsible for producing and publishing Advanced Personalization events in a fail-safe manner while taking care of the proper scope. To realize this, an inverse event is produced in **getCleanupEvent** for all request scoped event producers (**RefererHeaderBtgFilter**, **RequestParamsBtgFilter**).

Events for a visited **Category** or **Product** are produced in two steps by **AbstractPkResolvingFilter**:

1. Determining the primary key via a **PkResolvingStrategy**.
2. Producing the corresponding event using the factory method **internalGetEvent**.

The **PkResolvingStrategy** is responsible for

* Parsing the URL via a configured **UrlParsingStrategy**, returning a product or category code.
* Looking up the product or category via the hybris ServiceLayer API service, returning the primary key.

Events identifying a visited content page are produced via an interceptor processed after the Spring MVC controller (**ContentPageVisitedBtgInterceptor**), since CMS aware controllers in the hybris Multichannel Accelerator are responsible for identifying the CMS page.

#### Rules

Rules and output actions are defined in the **btgcockpit** extension. Out of the box, the hybris Multichannel Accelerator provides two example segments:

* **Regular Customer Segment**: the user is added to the user group **regulargroup** if the number of orders in the last six months is greater or equals four. This triggers a promotion if the spent amount on an order exceeds a specific value.
* **Camera Cross Selling Segment**: the user is shown a cross-selling product carousel on the homepage if he bought, for example, a DSLR camera but no accessory.

|  |
| --- |
| **See Also**   * [WCMS Components for the hybris Multichannel Accelerator](#255), section **Product Reference Component** * [Rule Types Overview](#256) * [Customer Segments Overview](#257) |

#### Configuration

The integration of Advanced Personalization is configured in Spring and web.xml . The URL-pattern for determining the product or category code is maintained in acceleratorstorefront-spring-btg.xml file:

|  |
| --- |
| <bean id="productUrlParsingStrategy" scope="tenant">  <property name="regex" value="^/.+/p/(\w+)$" /> </bean>  <bean id="categoryUrlParsingStrategy" scope="tenant">  <property name="regex" value="^/.+/c/(\w+)$" /> </bean> |

Table 19 acceleratorstorefront-spring-btg.xml

The filters and their respective mappings are activated in web.xml file:

|  |
| --- |
| <filter>     <filter-name>BTGSegmentFilter</filter-name>     <filter-class>de.hybris.platform.acceleratorstorefront.servlets.btg.BTGSegmentFilter</filter-class> </filter> <filter>     <filter-name>refererHeaderBtgFilter</filter-name>     <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter> <filter>     <filter-name>requestParamsBtgFilter</filter-name>     <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter> <filter>     <filter-name>productVisitedBtgFilter</filter-name>     <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter> <filter>     <filter-name>categoryVisitedBtgFilter</filter-name>     <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter>  <filter-mapping>     <filter-name>BTGSegmentFilter</filter-name>     <url-pattern>/\*</url-pattern> </filter-mapping> <filter-mapping>     <filter-name>refererHeaderBtgFilter</filter-name>     <url-pattern>/\*</url-pattern> </filter-mapping> <filter-mapping>     <filter-name>requestParamsBtgFilter</filter-name>     <url-pattern>/\*</url-pattern> </filter-mapping> <filter-mapping>     <filter-name>productVisitedBtgFilter</filter-name>     <url-pattern>/\*</url-pattern> </filter-mapping> <filter-mapping>     <filter-name>categoryVisitedBtgFilter</filter-name>     <url-pattern>/\*</url-pattern> </filter-mapping> |

Table 20 web.xml

## Business Processes and Eventing in the hybris Multichannel Accelerator

The hybris Multichannel Accelerator provides integration with the hybris Process Engine out of the box. This comes implicitly with its integration with the **fulfilmentprocess** extension as well as a new processes to handle storefront user events and asynchronously constructing and sending Emails.

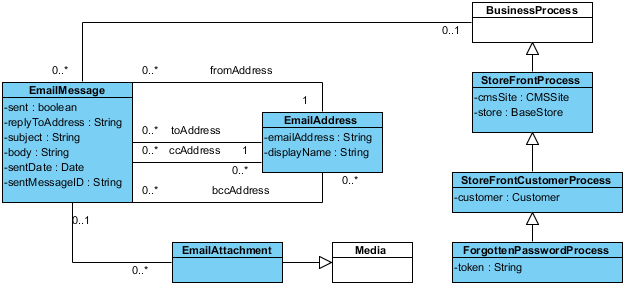
* [Business Processes and Eventing](#258)
  + [Customer Registration Email Process](#259)
  + [Example](#260)

|  |
| --- |
| **About this Document**  This document describes Business Processes and Eventing in the hybris Multichannel Accelerator  **Audience**: Consultants, developers, software architects, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [processengine Extension - Technical Guide](#63) * [fulfilmentprocess Extension - Technical Guide](#79) * [Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator](#37) |

### Business Processes and Eventing

Generating and sending emails is integrated into WCMS and the platform **processengine** extension. The process is triggered by sending an event which is then delivered by Spring to configured event listeners for this particular event.

The data model used for sending emails is shown in the next diagram:  
  
The main persistent items are

| **Class** | **Description** |
| --- | --- |
| **EmailMessage** | Stores an email message, mainly the subject, the body and status related information |
| **EmailAttachment** | Optionally, stores attachments as **Media** |
| **EmailAddress** | Stores an email address together with the display name for all different address types (to, from, cc, bcc) |
| **StoreFrontProcess** | Extends a business process with site and store related information required while rendering the email content |
| **StoreFrontCustomerProcess** | Extends **StoreFrontProcess** with an additional customer attribute |
| **ForgottenPasswordProcess** | Extends **StoreFrontCustomerProcess** with an additional token attribute used for the forgotten password process |

The extended business process models have been added to support capturing of email content as well as to trigger storefront processes such as handling registration of forgotten password.

#### Customer Registration Email Process

This process is used for creating and sending an email as shown in the next diagram:  
  
The steps triggered after receiving an **RegisterEvent** are:

| **Step** | **Description** |
| --- | --- |
| **generateCustomerRegistrationEmail** | create a new **EmailMessage** and initialise it using CMS integration classes and EmailGenerationService |
| **sendEmail** | send the **EmailMessage** to the intended recipients using **EmailService** |
| **removeSentEmail** | removes the **EmailMessage** from the database |

If an error occurs during processing, the process is stopped with an error status. For more details regarding the WCMS integration see the [Commerce Services Email Business Process](#261) document .

#### Example

How a business process is started in response to an previously published event using the platform **EventService** is shown in this section. The approach leverages the built-in support in Spring for publishing and receiving events using the Spring application context.

This example shows how to:

* Create and start a business process in response to a received event.
* Configure an event listener in Spring

1. First you provide an event listener parameterized with the event type:

|  |
| --- |
| public class RegistrationEventListener extends AbstractEventListener<RegisterEvent> {  private ModelService modelService;   public BusinessProcessService getBusinessProcessService()  {  return (BusinessProcessService) Registry.getApplicationContext().getBean("businessProcessService");  }   protected ModelService getModelService()  {  return modelService;  }   public void setModelService(final ModelService modelService)  {  this.modelService = modelService;  }   @Override  protected void onEvent(final RegisterEvent registerEvent)  {  final StoreFrontCustomerProcessModel storeFrontCustomerProcessModel = (StoreFrontCustomerProcessModel) getBusinessProcessService()  .createProcess("customerRegistrationEmailProcess" + System.currentTimeMillis(), "customerRegistrationEmailProcess");  storeFrontCustomerProcessModel.setCmsSite(registerEvent.getCmsSite());  storeFrontCustomerProcessModel.setCustomer(registerEvent.getCustomer());  getModelService().save(storeFrontCustomerProcessModel);  getBusinessProcessService().startProcess(storeFrontCustomerProcessModel);  } } |

Table 21 RegistrationEventListener

This code first creates a new process using **BusinessProcessService** and initializes the newly created process instance using received event data. In a second step, the process is started.

1. Next, this event listener has to be configured in Spring:

|  |
| --- |
| <bean id="customerRegistrationEventListener" class="de.hybris.platform.acceleratorcore.event.RegistrationEventListener" scope="tenant">  <property name="modelService" ref="modelService"/> </bean> |

Table 22 acceleratorcore-spring.xml

## Customer Reviews in the hybris Multichannel Accelerator

Customer reviews are built on top of the **customerreview** extension. The features offered include customer reviews and ratings that can be moderated.

* [Language](#262)
* [Anonymous Reviews](#263)
* [Rating System](#264)
* [Commerce Facades](#265)
* [New Approval Process](#266)
* [Extended Data Model](#267)

|  |  |  |
| --- | --- | --- |
| **About this Document**  This document describes the customer review integration within the hybris Multichannel Accelerator.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Consultants, developers, technical deciders  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [customerreview Extension - Technical Guide](#76)   + [Customer Review Tutorial](#268) |

### Language

If a website is available in 2 languages for example English and German and a review is posted on the German version of the website it will only be displayed on that version of the site. The language of a review can be changed in the hMC.

The current language for the storefront is managed by: de.hybris.platform.commercefacades.storesession.StoreSessionFacade.

### Anonymous Reviews

All reviews are anonymous the user can provide an optional alias if the user doesn't do so "by Anonymous" will be shown as the author of the review.

This behavior can be altered to show the logged in user by editing the reviewsTab.jsp.

For example if the review was posted by a logged in user and the desired behavior is to show that users ID if he/she doesn’t provide an alias the following code:

|  |
| --- |
| <c:choose>  <c:when test="${not empty review.alias}">  ${review.alias}  </c:when>  <c:otherwise>  <spring:theme code="review.submitted.anonymous"/>  </c:otherwise> </c:choose> |

Can be changed to:

|  |
| --- |
| <c:choose>  <c:when test="${not empty review.alias}">  ${review.alias}  </c:when>  <c:when test="${not empty review.principal.name && empty review.alias }">  ${review.principal.name}  </c:when>  <c:otherwise>  <spring:theme code="review.submitted.anonymous"/>  </c:otherwise> </c:choose> |

### Rating System

The Accelerator supports a 5 star rating system out of the box. Each review will have a rating associated with it. An average review is calculated for the product and will be displayed next to it.  
The Accelerator uses the jquery Star Rating Widget from: <http://orkans-tmp.22web.net/star_rating/>. Changing it to support a different rating system such as 1 – 4 or 1 – 10 stars is just a matter of changing the graphics, CSS and HTML.

For example to provide support for a 10 star rating system change the productPageReviewsTab.tag HTML markup from:

|  |
| --- |
| <div id="stars-wrapper">  <form:select path="rating" >  <form:option value='1'>1/5</form:option>  <form:option value='2'>2/5</form:option>  <form:option value='3'>3/5</form:option>  <form:option value='4'>4/5</form:option>  <form:option value='5'>5/5</form:option>  </form:select> </div> |

To:

|  |
| --- |
| <div id="stars-wrapper">  <form:select path="rating" >  <form:option value='1'>1/10</form:option>  <form:option value='2'>2/10</form:option>  <form:option value='3'>3/10</form:option>  <form:option value='4'>4/10</form:option>  <form:option value='5'>5/10</form:option>  <form:option value='6'>6/10</form:option>  <form:option value='7'>7/10</form:option>  <form:option value='8'>8/10</form:option>  <form:option value='9'>9/10</form:option>  <form:option value='10'>10/10</form:option>  </form:select> </div> |

The ratings widget is not used to display the ratings next to the product or the reviews so those graphics and/or CSS will need to be altered.

### Commerce Facades

There are 2 ways to get the reviews for a product the first method will is more efficient if only the reviews are desired. The second method will actually return a ProductData object which will contain the reviews for the product.

Method 1:

|  |
| --- |
| List<ReviewData> productReviews = productFacade.getReviews(productCode) |

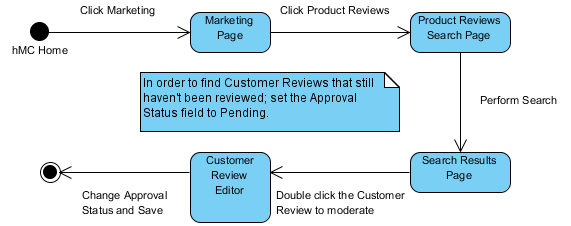
This will return the reviews that the current user can view based on the language and search restrictions.

Method 2:

|  |
| --- |
| final ProductData productData = productFacade.getProductForCode(productCode, Arrays.asList(ProductOption.BASIC, ProductOption.PRICE, ProductOption.SUMMARY, ProductOption.DESCRIPTION, ProductOption.GALLERY, ProductOption.CATEGORIES, ProductOption.REVIEW, ProductOption.PROMOTIONS, ProductOption.CLASSIFICATION)); List<ReviewData> productReviews = productData.getReviews(); |

For more information on commercefacades see: <https://wiki.hybris.com/display/acc/commercefacades+extension+-+Technical+Guide>

### New Approval Process



There has been a new approvalStatus attribute added to the CustomerReviewModel which is set to “Pending” by default.

To approve a review:

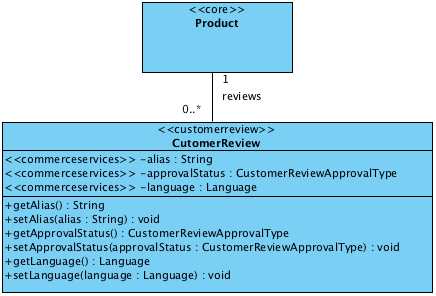
* Log into the hMC
* On the left hand side click on Marketing
* Click on Product Reviews
* Perform a search selecting reviews whose status is set to "Pending"
* Double click review in search results
* In the Approval Status combo box select "Approved"
* Save the review by clicking "Save" in the upper left corner

A search restriction on Reviews will ensure that only users in the customergroup will only view reviews that have been approved through the hMC.

The search restriction is created in de.hybris.platform.commerceservices.setup.CommerceServicesSystemSetup

For more information on moderating reviews see: <https://wiki.hybris.com/display/internal/Moderating+Reviews+in+the+hybris+Multichannel+Accelerator>

### Extended Data Model



## Data Importing Capability in the Multichannel Accelerator

The **acceleratorcore** extension comes with a batch package that enables projects to provide automated importing of data from **hot folders**. The infrastructure enables the import of simple CSV files that are internally translated into hybris fast, multi-threaded **impex** scripts that import content directly into your product catalogs. Files can also be grouped into batches to split the data load across multiple files. This feature is intended for automated updates such as stock, price and product catalog feeds that you may receive from external systems or from other areas of your business. The infrastructure uses the Spring Integration to provide a pluggable, highly configurable service based design to enable you to extend to fit your application specific requirements.

* [Hot Folder Data Importing](#269)
  + [Diagram Of Components](#270)
  + [Diagram of General Flow](#271)
  + [Configuration](#272)
    - [Task Configuration](#273)
    - [Converter Configuration](#274)
    - [Spring Integration Configuration](#275)
  + [Configuration Guide](#276)
    - [Adding a New Import](#277)
    - [Restricting to a Certain Catalog Version](#278)
    - [Applying Updates to Multiple Catalog Versions](#279)
    - [Sequencing Support](#280)
  + [Guide to Sample Imports](#281)
    - [New Product Feed](#282)
    - [Stock Update Feed](#283)
    - [Variant Feed](#284)
    - [Price Update Feed](#285)
    - [Merchandising Feed](#286)
    - [Media Feed](#287)
    - [Modifying File Mappings](#288)

|  |
| --- |
| **About this Document**  This document introduces the hybris Multichannel Accelerator data importing capabilities.  **Audience**: Consultants, developers, technical deciders, system administrators  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [ImpEx API](#289) * [ImpEx - User Guide](#290) * [Spring Integration](#247) * [acceleratorcore Extension - Technical Guide](#81) |

### Hot Folder Data Importing

The idea of **hot folder data importing** is that CSV files are imported automatically by moving them to a folder which is scanned periodically by the system. The system processes the found files.

#### Diagram Of Components

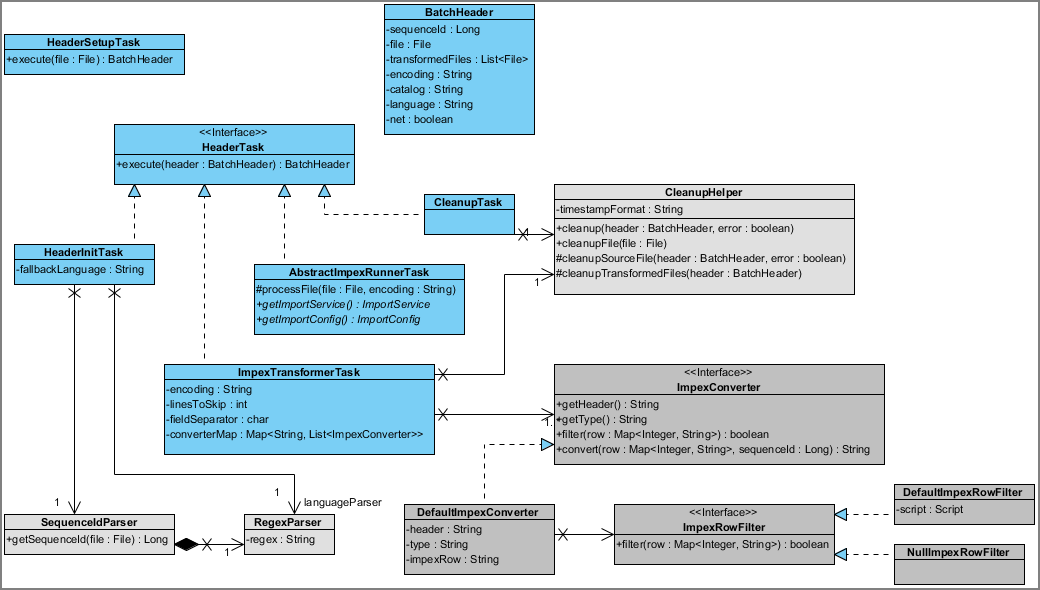


Figure: Components behind hot folder data importing.

The classes are structured into three major parts:

* Tasks executed by the Spring integration infrastructure.
* Converters providing the ImpEx header and converting CSV rows into ImpEx rows with optional filtering.
* Helper and utility classes: **SequenceIdParser**, **RegexParser**, **CleanupHelper**.

Tasks obey the general contract of a **HeaderTask** with the exception of the initial **HeaderSetupTask** creating the **BatchHeader**. The contract allows for

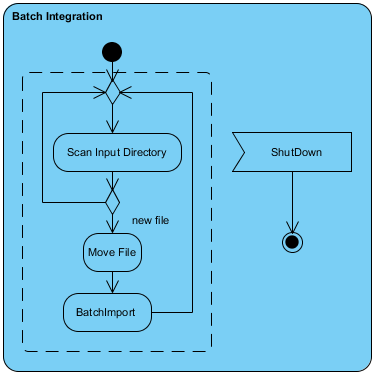
* Flexible configuration of processing pipelines.
* Providing a shared process context.
* Cleanup of all generated process files.
* Error handling by means of an **ErrorHandler** listening on an error channel.

CSV files are transformed into ImpEx files by **ImpexTransformerTask** processing following steps:

1. Retrieve all configured converters matching the file name prefix.
2. For every converter found, they convert the input file by following means:
   * Adding the ImpEx file header once with substitutions.
   * Converting all rows if they are not filtered.
   * If the line has missing input fields, adding the line together with the error message to a file in the error subdirectory.

Afterwards, all generated impex files are sent to the platform ImportService in **AbstractImpexRunnerTask** sequentially.

#### Diagram of General Flow

The general flow is shown in the diagram below:  


Spring integration periodically scans the configured input directory for new files. If new files are found, they are moved to the processing subdirectory and then sent to the Batch Import pipeline consisting of these tasks:

* **HeaderSetupTask**: creates a new **BatchHeader**
* **HeaderInitTask**: retrieves sequence ID and optionally language from the file name
* **ImpexTransformerTask**: creates one or many ImpEx files from the CSV input writing error lines to the error subdirectory.
* **ImpexRunnerTask**: processes all ImpEx files sequentially with multiple threads.
* **CleanupTask**: deletes all transformed files and moves the imported file with an optionally appended timestamp to the archive subdirectory.
* **ErrorHandler**: deletes all transformed files and moves the imported file with an optionally appended timestamp to the error subdirectory.

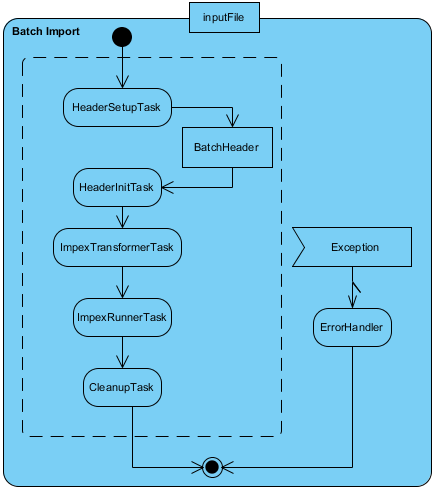


Figure: Batch import diagram.

#### Configuration

The **hot folder import** processes are configured in the following files:

| **File** | **Configuration Items** |
| --- | --- |
| acceleratorcore-spring-integration.xml | * Spring integration infrastructure including AOP setup. * ImpEx converters, filters and converter mappings. |
| project.properties | * **acceleratorcore.batch.impex.max-threads**: number of threads used for ImpEx. * **acceleratorcore.batch.impex.basefolder**: base folder of the import infrastructure containing catalog specific sub folders. |

The files are located in the $[HYBRIS\_BIN\_DIR|HYBRIS\_BIN\_DIR]/ext-hybris/acceleratorcore directory.

##### Task Configuration

Relevant configuration options for tasks include:

| **Configuration Option** | **Parameters** |
| --- | --- |
| **HeaderSetupTask** | * **catalog**: the catalog to use. This setting is applied to the default header substitution $CATALOG$ * **net**: the net setting to apply to prices. This setting is applied to the default header substitution $NET$ |
| **HeaderInitTask** | * **sequenceIdParser**: the regular expression used to extract the sequence ID from the file name. In the sample configuration, the sequence ID precedes the file extension separated by a hyphen, for example: base\_product-1.csv . * **languageParser**: the regular expression used to extract the language from the file name. In the sample configuration, the language optionally precedes the sequence ID, for example: base\_product-de-1.csv . * **fallbackLanguage**: the language to use if the language is not set in the file name. |
| **ImpexTransformerTask** | * **converterMap**: used to map file prefixes to one or multiple converters to produce ImpEx files. * **encoding**: the file encoding to use (default: UTF-8). * **linesToSkip**: the lines to skip in all CSV files (default: 0). * **fieldSeparator**: the separator to use to read CSV files (default: ,). |
| **CleanupTask** | * **CleanupHelper.timeStampFormat**: if set, append timestamp in the specified format to input files moved to the archive or error subdirectory |

##### Converter Configuration

Converters are the central configuration element used for importing. The following configuration options are available:

| **Configuration Option** | **Parameters** |
| --- | --- |
| **header**: the impex header to use including following header substitutions | * $NET$: the net setting * $CATALOG$ : the catalog prefix * $LANGUAGE$ : the language setting * $TYPE$ : an optional type attribute that can be applied if filtering is configured |
| **impexRow**: the template for an impex row adhering to this syntax | * **{('+')? (<columnId> | 'S')}**. The '+' character adds a mandatory check to this column causing lines with missing attributes being written to an error file in the error subdirectory. For example, writing the first CSV column to the ImpEx file can be accomplished by configuring **{+1}**. The 'S' can be used for writing the current sequence ID at the template position. Optionally, columns can be quoted by enclosing the column in the template with quotation marks. * **rowFilter**: an optional row filter. The supplied expression has to be a valid Groovy expression. The current row map consisting of column ID and value is referenced by **row**. The Groovy semantics for converting **String** to **Boolean** apply. For example, querying if the first column is not empty can be written as **!row[1]**. Configuring multiple converters with row filters gives the option to split a supplied CSV input file into different ImpEx files according to specified filter criteria. * **type**: an optional type that can be retrieved in the header using the header substitution $TYPE$ . |

##### Spring Integration Configuration

To allow an easier configuration, the Spring integration namespace is leveraged, namely the following elements:

* **int:channel**: setup of a channel
* **int:service-activator**: activates a referenced bean when receiving a message on a configured channel. The bean response is again wrapped in a message and sent to the configured output channel.
* **[file:inbound-channel-adapter:]** which is capable of scanning a directory in a configurable interval and sending files to a configured channel under following conditions:
  + Only files matching a specified regular expression are retrieved (filename-regex).
  + Files are processed in the order defined by the **FileOrderComparator**, under following priority:
    - If a priority is configured for the file prefix, use the priority.
    - For files with equal priority: process the older file first.
* **[file:outbound-gateway:]** which is used for moving a file to the processing subdirectory.

#### Configuration Guide

The next sections describe typical configuration tasks and how they can be achieved.

##### Adding a New Import

To add a new import follow these steps:

1. Define the file name schema for the new import, for example: customer\- <sequenceId> .csv .
2. Set up a new **ImpexConverter**configuration describing the ImpEx header and the row expression.
3. Add the file prefix and the converter reference to the **converterMap** property of the **ImpexTransformerTask**.

##### Restricting to a Certain Catalog Version

Typically, the version is defined as a ImpEx variable in the ImpEx header. This variable can then referenced easily in the ImpEx import statements. The sample configuration imports into the **staged** catalog only with the exception of price imports importing into both **staged** and **online** catalogs.

##### Applying Updates to Multiple Catalog Versions

To achieve updates to multiple catalogs, an input line is written to multiple transformed lines with differing catalog version. An example of this approach is shown in the price converter configuration **batchPriceConverter**:

|  |
| --- |
| <property name="impexRow">  <value>;{+0}:$catalog:Staged;{+1};{+2};{3};;;;$catalog:Staged;{S} ;{+0}:$catalog:Online;{+1};{+2};{3};;;;$catalog:Online;{S}</value> </property> |

Table 23 acceleratorcore-spring-integration.xml

##### Sequencing Support

Sequencing, which is also known as version support, defines the ability to associate updates with a unique ID to prevent stale updates. The recommended way to provide this ID is to use **UNIX** timestamps while exporting data. In the hybris Multichannel Accelerator, this ID is called **sequence ID**. Support for this feature is available for **Product** and **PriceRow** updates. The data model for these two **Items** was extended with an additional property **sequenceId**.

The **sequence ID** is associated to an input file via a configurable naming convention **SequenceIdParser**). This sequence ID is then provided for ImpEx in every row by using standard converter row syntax.

The sequence ID is then imported by means of a specialized ImpEx translator **SequenceIdTranslator** comparing the provided with the current value and rejecting the row if the value is out dated. Rejected lines can be viewed as unresolved lines of the responsible cron job in the hybris Management Console (hMC).

Out of the box, tax and media import is configured to validate the sequence ID while tolerating an equal sequence ID (**GreaterSequenceIdTranslator**).

#### Guide to Sample Imports

The hybris Multichannel Accelerator provides sample import feeds which you can easily adjust to different requirements. This section describes the available file formats.

##### New Product Feed

New products are imported from files named using following convention: base\_product- <language>-<sequenceID> .csv . If the product variant type is not empty, the product is imported as **ApparelProduct**, otherwise as **Product**. This functionality is achieved by using row filtering and configuring multiple converters for **base\_product**.

The optional language part of the file name is added to the ImpEx header by using a header substitution. Thereby it is possible to update localized attributes like **name** or **description** for this language only.

Find the description of file format in the table below:

| **Column** | **Attribute** | **Type** | **Optional** | **Localized** | **Default** | **Values** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | code | String |  |  |  |  |  |  |
| 1 | variantType | String | yes |  |  | ApparelSizeVariantProduct ApparelStyleVariantProduct |  |  |
| 2 | name | String | yes | yes |  |  |  |  |
| 3 | description | String | yes | yes |  |  |  |  |
| 4 | ean | String | yes |  |  |  |  |  |
| 5 | manufacturerName | String | yes |  |  |  |  |  |
| 6 | manufacturerAID | String | yes |  |  |  |  |  |
| 7 | unit | String | yes |  | pieces |  |  |  |
| 8 | approved | String | yes |  | check | check approved |  |  |
| 9 | product tax group | String | yes |  | eu-vat-full | eu-vat-full eu-vat-half |  |  |

##### Stock Update Feed

Stock updates are imported from files named like stock- <sequenceID> .csv by calling the **StockService** for each line. This is achieved by declaring a dedicated **StockTranslator** delegating to a **StockImportAdapter** configured in acceleratorcore-spring-integration.xml . Thereby it is possible to configure a different warehouse for the stock updates and also leverage the stock history maintained by the platform **StockService**.

Find the description of file format in the table below:

| **Column** | **Attribute** | **Type** | **Optional** | **Localized** | **Default** | **Values** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | product code | String |  |  |  |  |  |  |
| 1 | stock | Integer |  |  |  |  | >= 0 |  |

##### Variant Feed

Product variants are imported from files named like variant- <language>-<sequenceID> .csv . The priority assigned to these files in the configuration of **FileOrderComparator** is lower than for **base\_product** meaning base products are imported first.

Variant products are imported as **ApparelSizeVariantProduct** or **ApparelStyleVariantProduct** depending on the column value. If no product variant type is given, the product is imported as **ApparelSizeVariantProduct**, if the size column is not empty, otherwise as **ApparelStyleVariantProduct**. This can be easily adapted to different requirements by changing the filter and converter configuration.

The optional language part of the file name is added to the ImpEx header by using a header substitution. Thereby it is possible to update localized attributes, like **style** or **size**, for this language only.

Find the description of file format in the table below:

| **Column** | **Attribute** | **Type** | **Optional** | **Localized** | **Default** | **Values** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | parent product code | String |  |  |  |  |  |  |
| 1 | code | String |  |  |  |  |  |  |
| 2 | variantType | String | yes |  |  | ApparelSizeVariantProduct ApparelStyleVariantProduct |  |  |
| 3 | style | String | yes | yes |  |  |  |  |
| 4 | size | String | yes | yes |  |  |  |  |

##### Price Update Feed

Price rows are imported from files named like price- <sequenceID> .csv . Prices are imported to both the **online** and **staged** catalog. **Unit**, **unitFactor** and **minqtd** attributes are set to default values.

Find the description of file format in the table below:

| **Column** | **Attribute** | **Type** | **Optional** | **Localized** | **Default** | **Values** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | product code | String |  |  |  |  |  |  |
| 1 | price | Double |  |  |  |  | >= 0 |  |
| 2 | currency | String |  |  |  |  | EUR USD JPY | Isocode of the price currency |
| 3 | net | Boolean | yes |  | configurable per site | true false |  | net/gross setting |

##### Merchandising Feed

Product references are imported from files named like merchandise- <sequenceID> .csv .

Find the description of file format in the table below:

| **Column** | **Attribute** | **Type** | **Optional** | **Localized** | **Default** | **Values** | **Constraints** | **Description** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | source product code | String |  |  |  |  |  | Source |
| 1 | reference type | String |  |  |  | see enum **ProductReferenceType** |  |  |
| 2 | target product code | String |  |  |  |  |  | Target |

##### Media Feed

Media is imported from files named like media- <sequenceID> .csv . Images have to be provided in the subdirectory images/ <media format> in the following formats: 30Wx30H, 65Wx65H,96Wx96H,300Wx300H,515Wx515H,1200Wx1200H.

The import is processed in three steps:

1. Import media in all specified formats.
2. Import a media container for the product gallery.
3. Assign the media and media container to the product.

Find the description of file format in the table below:

| **Column** | **Attribute** | **Type** | **Optional** | **Localized** | **Default** | **Description** |
| --- | --- | --- | --- | --- | --- | --- |
| 0 | product code | String |  |  |  | Code of product |
| 1 | code | String |  |  |  | Code of media |

##### Modifying File Mappings

Adding additional CSV columns can be achieved by updating the corresponding converter:

* Update the header property to import the new column.
* Update the **impexRow** to supply the CSV column in the transformed file.

For example, importing the unit as a fourth column in price- <sequenceId> .csv can be configured by changing this line

|  |
| --- |
| <value>;{+0}:$catalog:Staged;{+1};{+2};{3};;;;$catalog:Staged;{S}</value> |

to

|  |
| --- |
| <value>;{+0}:$catalog:Staged;{+1};{+2};{3};{4};;;$catalog:Staged;{S}</value> |

## Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator

This document describes how emails are defined and generated in the system. The hybris WCMS Module was extended to define the emails to be raised by the system in the same way as WCMS pages are defined. An email generated by this system contains WCMS components rendered within the email text. The WCMS contents, such as the logo and the banners of an email are managed through WCMS slot names, slots, components and so on and can be configured using ImpEx or the WCMS Cockpit. The actual email text, that is subject and body of an email, are managed through velocity templates and can be configured using ImpEx or the hybris Management Console (hMC).

* [WCMS Integration](#291)
  + [Email Page Template](#292)
  + [Email Page](#293)
  + [Renderer Template](#294)
  + [Adding an Email Template Structure Using ImpEx](#295)
  + [Turning off Preview](#296)
* [Email Processes](#297)
  + [Generating an Email](#298)
    - [DefaultContextResolutionStategy](#299)
    - [DefaultEmailContextFactory](#300)
    - [DefaultEmailGenerationService](#301)
  + [Sending Emails](#302)
    - [DefaultEmailService](#303)
  + [Deleting Emails](#304)
  + [Starting](#305)
  + [Audit](#306)
* [Data Model](#307)
  + [WCMS Integration Data Model](#308)

|  |
| --- |
| **About this Document**  This document describes the email integration within the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [commerceservices Extension - Technical Guide](#65) * [Business Processes and Eventing in the hybris Multichannel Accelerator](#35) * [Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator](#37) * [cms2 Extension - Technical Guide](#78) * [processengine Extension - Technical Guide](#63) * [Website Configuration in the hybris Multichannel Accelerator](#309), section **Managing Email Templates** |

### WCMS Integration

#### Email Page Template

The **EmailPageTemplate**, which extends **PageTemplate** type, represents an email template. An email is generated based on one email page template. The email page template, like the WCMS page template, can have one or more **ContentSlotNames**, which define the names and positions of the **ContentSlots** along with WCMS components that are allowed in the slots. The email page template can have predefined **ContentSlots** with Components exactly like the WCMS page template, which is available across all email pages that are associated with that template. In addition the email page templates also have a subject and a **htmlTemplate**, both of which are of type **RendererTemplate** which captures scripts for generating the subject and body of an email.

The page types that can be associated with **EmailPageTemplate** are restricted to only **EmailPage** type.

#### Email Page

The **EmailPage**, which extends **AbstractPage** type, represents an email such as **Customer Registration** email, **Order Confirmation** email, and so on. The email page has an email page template, **masterTemplate**, which defines **ContentSlotNames** and any predefined **ContentSlots** with components. Like the WCMS page, the email page can have **ContentSlots** with components which are specific to that particular email.

Preview of **EmailPage** type is disabled.

#### Renderer Template

The content of WCMS components and the subject and body of an email are generated using velocity templates. The **RendererTemplate** holds the velocity script along with the context class name. The velocity script contains the velocity tags which are replaced with the values from the context object provided during the rendering process.

The velocity script representing the subject is normally the simplest and may have very few velocity tags. For example an **Order Confirmation** email subject can have a tag referring order number. The script representing the body is normally more complex and may have many velocity tags including tags referring to WCMS components. For example, an email body can have tags referring to the site logo, top and bottom banners and so on. The script representing a WCMS component can have tags referring to the attributes of the WCMS component.

The velocity tags are in the form of **${ctx.parameterName}**, **${ctx.CMSSlotContents.slotName}** and so on, where **ctx** refers to the email context object passed to the renderer. The **parameterName** can be **firstName**, **lastName** and **orderNumber**, and the **slotName** can be **siteLogo**, **topBanner** and **bottomBanner**. More about the context object is in the email generation section of this document.

#### Adding an Email Template Structure Using ImpEx

You can configure the email page templates and the email pages in the WCMS Cockpit or using ImpEx. The configuration is very similar to the WCMS page configuration with an addition of providing renderer templates for the subject and body of the email.

The steps involved are:

1. Defining **EmailPageTemplate** which provides the subject and layout/contents of the email body

|  |
| --- |
| # Email page Template INSERT\_UPDATE EmailPageTemplate;$contentCV[unique=true];uid[unique=true];name;active;frontendTemplateName;subject(code);htmlTemplate(code);restrictedPageTypes(code) ;;CustomerRegistrationEmailTemplate;Customer Registration Email Template;true;customerRegistrationEmail;electronics\_Email\_Customer\_Registration\_Subject;electronics\_Email\_Customer\_Registration\_HTML;EmailPage |

1. Adding velocity template for the **EmailPageTemplate** which provides a layout for the email template in the WCMS Cockpit

|  |
| --- |
| # Templates for WCMS Cockpit Page Edit UPDATE EmailPageTemplate;$contentCV[unique=true];uid[unique=true];velocityTemplate[translator=de.hybris.platform.acceleratorcore.setup.FileLoaderValueTranslator] ;;CustomerRegistrationEmailTemplate;$jarResource/cmscockpit/structure-view/structure\_customerRegistrationEmailTemplate.vm |

1. Defining **ContentSlotNames** along with allowed WCMS components for the **EmailPageTemplate**. Currently **EmailGenerationService** supports **CMSParagraphComponent**, **CMSImageComponent**, **CMSLinkComponent** and **CMSBannerComponent**.

|  |
| --- |
| INSERT\_UPDATE ContentSlotName;name[unique=true];template(uid,$contentCV)[unique=true][default='CustomerRegistrationEmailTemplate'];validComponentTypes(code) ;SiteLogo;;CMSImageComponent,BannerComponent ;TopContent;;$wideContent; ;BottomContent;;$wideContent; ;Footer;;CMSLinkComponent,CMSParagraphComponent |

1. Defining **RenedererTemplates** with velocity template scripts for email subject and body.

|  |
| --- |
| # Email velocity templates INSERT\_UPDATE RendererTemplate;code[unique=true];contextClass;rendererType(code)[default='velocity'] ;electronics\_Email\_Customer\_Registration\_HTML;de.hybris.platform.commerceservices.process.email.context.CustomerEmailContext ;electronics\_Email\_Customer\_Registration\_Subject;de.hybris.platform.commerceservices.process.email.context.CustomerEmailContext  # Email velocity templates UPDATE RendererTemplate;code[unique=true];description[lang=$lang];templateScript[lang=$lang,translator=de.hybris.platform.acceleratorcore.setup.FileLoaderValueTranslator] ;electronics\_Email\_Customer\_Registration\_HTML;Customer Registration HTML Email;$emailResource/email-customerRegistrationHTML\_de.vm ;electronics\_Email\_Customer\_Registration\_Subject;Customer Registration Email Subject;$emailResource/email-customerRegistrationSubject\_de.vm |

1. Adding velocity templates for the subject and body as example show below:
   * Order confirmation email subject:

|  |
| --- |
| Order Confirmation ${ctx.order.code} |

* + Customer registration email body:

|  |
| --- |
| <html>  <head>  </head>  <body bgcolor="#e4e7e8">  <table width="100%" border="0" align="center" cellpadding="0" cellspacing="0" bgcolor="#e4e7e8">  <tr>  <td>&nbsp;</td>  </tr>  <tr>  <td align="center" valign="top">  <table width="610" border="6" align="center" cellpadding="0" cellspacing="0" bordercolor="#ebedee">  <tr>  <td align="center" valign="top" bgcolor="#FFFFFF">  <table width="570" cellpadding="0" cellspacing="0" border="0" align="center">  <tr>  <td valign="middle">  ${ctx.cmsSlotContents.SiteLogo}  </td>  </tr>  <tr>  <td height="30" align="right" valign="middle" bgcolor="#51585c">  <font color="#FFFFFF" size="2" face="Arial, Helvetica, sans-serif"><a href="${ctx.secureBaseUrl}/my-account"><font color="#FFFFFF">My Account</font></a> | <a href="${ctx.baseUrl}/store-finder"><font color="#FFFFFF">Store Finder</font></a> &nbsp;&nbsp;</font>  </td>  </tr>  <tr>  <td align="center" valign="middle">  <a href="${ctx.baseUrl}" style="display:block; margin-top:10px;margin-bottom:10px;">${ctx.cmsSlotContents.TopContent}</a>  </td>  </tr>  <tr>  <td align="left" valign="top">  <p><font color="#666666" size="2" face="Arial, Helvetica, sans-serif"><b>Dear ${ctx.title} ${ctx.displayName}</b>,</font></p>  <p><font color="#666666" size="2" face="Arial, Helvetica, sans-serif">Thank you for registering with hybris, we hope you&rsquo;ll enjoy shopping with us.</font></p>  <p><font color="#666666" size="2" face="Arial, Helvetica, sans-serif">Many Thanks </font></p>  <p><font color="#666666" size="2" face="Arial, Helvetica, sans-serif">Customer Services</font></p>  </td>  </tr>  <tr>  <td align="center" valign="middle">  <a href="${ctx.baseUrl}" style="display:block; margin-top:10px;margin-bottom:10px;">${ctx.cmsSlotContents.BottomContent}</a>  </td>  </tr>  <tr>  <td height="30" align="right" valign="middle" bgcolor="#51585c">  <font color="#FFFFFF" size="2" face="Arial, Helvetica, sans-serif"><a href="${ctx.baseUrl}"><font color="#FFFFFF">Help</font></a> | <a href="http://www.hybris.com/hybris/en/Contact.html"><font color="#FFFFFF">Contact Us</font></a> | <a href="${ctx.baseUrl}"><font color="#FFFFFF">Terms &amp; Conditions</font></a> &nbsp;&nbsp;</font>  </td>  </tr>  </table>  </td>  </tr>  </table>  </td>  </tr>  <tr>  <td>&nbsp;</td>  </tr>  </table> </body> </html> |

1. Defining **EmailPage**:

|  |
| --- |
| # Customer Registration Email Page INSERT\_UPDATE EmailPage;$contentCV[unique=true];uid[unique=true];name;masterTemplate(uid,$contentCV);defaultPage;approvalStatus(code)[default='approved'] ;;CustomerRegistrationEmail;Customer Registration Email;CustomerRegistrationEmailTemplate;false; |

1. Defining WCMS components and content slots

|  |
| --- |
| # WCMS Image Components INSERT\_UPDATE CMSImageComponent;$contentCV[unique=true];uid[unique=true];name ;;EmailBannerSaleNowOnImage;Email Banner Sale Now On Image  # Content Slots INSERT\_UPDATE ContentSlot;$contentCV[unique=true];uid[unique=true];name;active;cmsComponents(uid,$contentCV) ;;EmailTopSlot;Default Email Top Slot;true;EmailBannerSaleNowOnImage |

1. Binding reusable content slots to the **EmailPageTemplate** using **ContentSlotForTemplate**.

|  |
| --- |
| # Bind Content Slots to Email Page Templates INSERT\_UPDATE ContentSlotForTemplate;$contentCV;uid[unique=true];position[unique=true];pageTemplate(uid,$contentCV)[unique=true]  [default='CustomerRegistrationEmailTemplate'];contentSlot(uid,$contentCV)[unique=true];allowOverwrite ;;SiteLogo-CustomerRegistrationEmail;SiteLogo;;SiteLogoSlot;true ;;Footer-CustomerRegistrationEmail;Footer;;FooterSlot;true ;;TopContent-CustomerRegistrationEmail;TopContent;;EmailTopSlot;true ;;BottomContent-CustomerRegistrationEmail;BottomContent;;EmailBottomSlot;true |

1. Binding any specific content slots to the **EmailPage** using **ContentSlotForPage**.
2. Adding common WCMS component velocity templates, for example for WCMS image component:

|  |
| --- |
| <img src="${ctx.parentContext.mediaBaseUrl}${ctx.media.url}" alt="${ctx.media.altText}" border="0"/> |

#### Turning off Preview

In a code snippet below you find an example showing how to disable the preview of an email page:

|  |
| --- |
| # Disable preview for email pages UPDATE CMSPageType;code[unique=true];previewDisabled ;EmailPage;true |

### Email Processes

The **Email** process is a business process containing a sequence of actions and is defined in a process definition XML file. The business process object holds the information required for the actions within the process.

In the case of **Email** process the following actions have been configured:

1. **Generate Email Action** - during this action, the contents of the email are generated and an **EmailMessage** object is created. The **EmailMessage** objects create are attached to the business process so that the next action can act on them. This action depends on **ContextResolutionStrategy** to impersonate the WCMS site and to initialize session context, and **EmailGenerationService** to create **EmailMessage** objects.
2. **Send Email** - during this action all the emails attached to the business process are sent using **EmailService**. If the emails are sent successfully, then **EmailMessage** object is updated with sent details.
3. **Remove Sent Email Action** - if an **EmailMessage** was sent successfully, during this action it is removed from the system.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> <process xmlns="http://www.hybris.de/xsd/processdefinition" start="generateCustomerRegistrationEmail" name="customerRegistrationEmailProcess"  processClass="de.hybris.platform.commerceservices.model.process.StoreFrontCustomerProcessModel" onError="error">   <action id="generateCustomerRegistrationEmail" bean="generateCustomerRegistrationEmail">  <transition name="OK" to="sendEmail"/>  <transition name="NOK" to="error"/>  </action>   <action id="sendEmail" bean="sendEmail">  <transition name="OK" to="removeSentEmail"/>  <transition name="NOK" to="failed"/>  </action>   <action id="removeSentEmail" bean="removeSentEmail">  <transition name="OK" to="success"/>  <transition name="NOK" to="error"/>  </action>   <end id="error" state="ERROR">Something went wrong.</end>  <end id="failed" state="FAILED">Could not send customer registration email.</end>  <end id="success" state="SUCCEEDED">Sent customer registration email.</end>  </process> |

#### Generating an Email

The first step during this process is to impersonate the site and initialize the session context with language and currency required. This was done by **ContextResolutionStategy**. Then an email context object containing all the required values to render a velocity script is created. This was done by **EmailContextFactory**. And finally an **EmailMessage** is created using **EmailGenerationService** and is attached it to the business process.

##### DefaultContextResolutionStategy

The strategy resolves the site to be used from the business process. If the business process is a **StorefrontProcess**, then the site is retrieved from the process itself, however if the process is an **OrderProcess** then the site is retrieved from the order that is associated with the process. The strategy also tries to resolve the language and the currency to be used based on session language and session currency of the customer associated with the business process either directly or indirectly in the case of **OrderProcess**, where it uses customer of the order associated with the process. If it can not resolve the language and currency to be used then it uses the default session currency and language of the site.

##### DefaultEmailContextFactory

The **EmailContextFactory** is responsible for creating the email context object given the **EmailPage**, **RendererTemplate** and email business process. It retrives the name of the email context type from the renderer template, which holds the velocity script referening the email context object, associated with the email template and then it tries to get the bean instance of the email context type from the application context.

There are **CustomerEmailContext**, **ForgottenPasswordContext** and **OrderNotificationContext** in the system. All the email context objects contain general parameters like **baseUrl**, **secureBaseUrl**, **mediaBaseUrl**, replay email address details, and so on. The **CustomerEmailContext** includes **CustomerData** object containing customer related parameters like first name, last name, email address, and so on. The **ForgottenPasswordContext** includes password token, expiry time along with **CustomerData** object. And the **OrderNotificationContext** includes **OrderData** object containing order related parameters like order code, delivery address, payment information, order entries, and so on.

In addition, the email context object also includes the content of WCMS components of all WCMS slots configured for an email. The contents of all WCMS component of a slot are generated by using **RendererService**. For that, the default factory determines the renderer template, which contains velocity script, for a WCMS component by a simple logic where it finds a template with an ID same as the WCMS component class name prefixed with site UID and appended with 'template'. Then it creates a context object required for that WCMS component and fills it with component attribute values. And finally, with the renderer service providing the script and the context object, it generates the content of that WCMS component. These steps are repeated for all the WCMS components of a slot and the contents of these are included in the main context object as map **CMSContentSlots** of **slotName** and its contents. The WCMS components content thus can be accessed through their **contentSlotName**, so, for example, to access top banner content in the velocity script, the velocity tag is **${ctx.CMSSlotContents.topContent}** with **topContent** being the **slotName**.

Finally the context factory adds additional parameters to the email context object based on **EmailContextVariables** which provides a facility to define variables based on existing parameters in the email context with a simple syntax. For example a variable defined:

|  |
| --- |
| 'ThemeResourceUrl = {baseUrl}/\_ui/{theme}/' |

becomes a parameter **themeResourceUrl** with a value substituting baseUrl and theme parameter values.

##### DefaultEmailGenerationService

The **EmailGenerationService** determines the subject and body renderer templates. Given email page template and using the renderer service, it generates the subject and body contents of the email thus providing email context object created by the factory described in the above section. It then creates an **EmailMessage** object.

#### Sending Emails

The **SendEmailAction** sends emails and uses **EmailService**. The action retrieves all the **EmailMessages** attached to the business process and tries to send them.

##### DefaultEmailService

The default email service uses Apache commons mail wrapped in **MailUtils** class to send **EmailMessages** and updates the object details of send status.

#### Deleting Emails

The **RemoveEmailAction** removes the emails from the system that are sent successfully. The action retrieves all the **EmailMessages** attached to the business process and tries to remove from the system if they are sent successfully.

#### Starting

Once an email process has been defined in an XML file, and a resource bean has been created as shown below, then an email can be raised by starting the process.

|  |
| --- |
| <!-- Process resourcess definition --> <bean id="customerRegistrationEmailProcessDefinitionResource" class="de.hybris.platform.processengine.definition.ProcessDefinitionResource" scope="tenant"> <property name="resource" value="classpath:/acceleratorcore/processes/customerRegistrationEmailProcess.xml"/> </bean> |

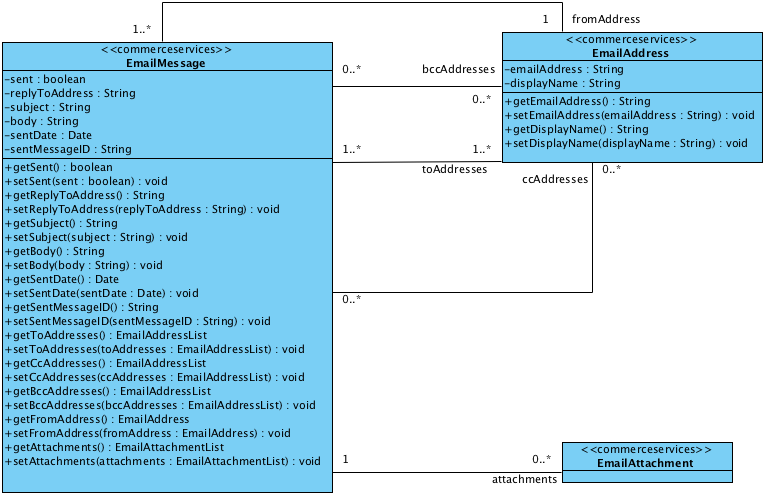
An email process can be started as follows:

|  |
| --- |
| final StoreFrontCustomerProcessModel storeFrontCustomerProcessModel = (StoreFrontCustomerProcessModel) getBusinessProcessService()  .createProcess("customerRegistrationEmailProcess" + System.currentTimeMillis(), "customerRegistrationEmailProcess"); storeFrontCustomerProcessModel.setCMSSite(registerEvent.getCMSSite()); storeFrontCustomerProcessModel.setCustomer(registerEvent.getCustomer()); getModelService().save(storeFrontCustomerProcessModel); getBusinessProcessService().startProcess(storeFrontCustomerProcessModel); |

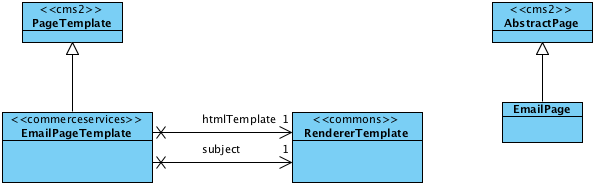
#### Audit

|  |  |
| --- | --- |
|  | **Audit History**  The email process object being a business process maintans all the process history. Details regarding the email content and recipients can also be saved, or optionally deleted when the email has been sent. All of this is captured in the following data model. |

### Data Model



#### WCMS Integration Data Model



## Essential and Project Data in the hybris Multichannel Accelerator

The hybris Multichannel Accelerator provides example sites that can be loaded as required via the Essential and Project Data load processes. This document describes these processes and the options that are available to control the data that is loaded.

* [General concept of core vs sample data](#310)
* [Features](#311)
* [Acceleratorcore](#312)
  + [Acceleratorcore Essential Data](#313)
  + [Acceleratorcore Project Data](#314)
    - [Sync Jobs](#315)
* [Acceleratorsampledata](#316)
  + [Acceleratorsampledata Project Data](#317)
  + [Resource folder structure](#318)
  + [Products](#319)
  + [CMS Content](#320)
  + [Points of sale](#321)
  + [Promotions](#322)
  + [Btg segments](#323)

|  |  |  |
| --- | --- | --- |
| **About this Document**  This documents provides the information about the essential and project data load processes in the hybris Multichannel Accelerator.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Technical deciders, software architects, developers, consultants  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [acceleratorcore Extension - Technical Guide](#81) * [acceleratorsampledata Extension - Technical Guide](#101) |

### General concept of core vs sample data

The Essential/Project data setup for the **acceleratorcore** extension is designed to:

* create BaseStores;
* create empty Catalogs;
* create CMSSites, and bind the Basestores and Catalogs to them;
* set Sites' languages, currencies, delivery options, theme and UrlBindings;
* create Solr search indexes and triggers;
* create CMSComponents, PageTemplates and Pages and configure CMSRestrictions;
* configure system user groups and roles;

These will need to be altered and customised for any Site based on the Accelerator but provide a starting point for a project. Loading the acceleratorcore essential and project data will create fully functional, but empty, web sites.

The Essential/Project data setup for the **acceleratorsampledata** extension is designed to:

* Load Sample Product Data, Categorisation, Media, Suppliers, Reviews, Promotions and Stock;
* Load Sample Store data (locations, facilities, imagery);
* Load and configure Sample CMS Pages and Components;

The acceleratorsampledata helps to boost productivity during development, but should not be loaded into Production systems. It should be removed from localextensions.xml and functionality / tests should not rely upon it being present. The acceleratorsampledata is primarily an example of how complex data sets can be set up in hybris.

### Features

The Accelerator provides the class AbstractSystemSetup to asist with adding new extensions and additional impex files. This features:

* A simplified mechanism for adding yes/no options for controlling Project data setup, using "createBooleanSystemSetupParameter";
* Automatic loading of localised impex files following a naming convention i.e., loading "my\_products.impex" will cause the files "my\_products\_en.impex", "my\_products\_de.imex" etc to be loaded - the list of languages that are detected is taken from the **CommonI18NService**;
* Optimised configuration of Catalog Sync Jobs for Product and Content Catalogs;
* Configuration of Solr Indexer Jobs;

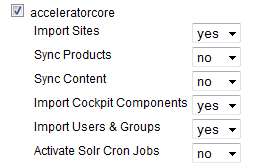
### Acceleratorcore

#### Acceleratorcore Essential Data

This stage imports:

* essential-data.impex
  + Languages
  + Currencies
  + Titles
  + Vendors
  + Warehouses
  + Supported Credit/Debit cards
  + User Groups
  + DistanceUnits for Storelocator
* countries.impex
  + Full country list conforming to ISO 3166-1 alpha-2
* delivery-modes.impex
  + Example delivery modes to be replaced / changeed as required
* mcc-sites-links.impex
  + Links for the MultiChannelCockpit

#### Acceleratorcore Project Data

Project Data setup for **acceleratorcore** gives the following options:  


**Import Sites** will import the Electronics, ApparelUK and ApparelDE sites, including:

* Site and Store Configuration (catalogs , classifications, countries, languages, currencies, delivery options, themes, media formats, tax groups)
* Base CMS Content (PageTemplates, Page Types, Pages, Components, Cockpit Configurations)
* Email Content (similar to CMS)
* Solr Search Configuration (Index Configurations, Indexed Types / Properties, Ranges, Queries, Sorts)

**Import Cockpit Components** will load customised cockpit configurations for the CmsCockpit and ProductCockpit:

* Configurations for Editors, Wizards and Lists for the components added in acceleratorcore and customisations of existing components;
* Configurations for Editors, Wizards and Lists for customisations made to Products and Categories and configurations for Variant sub-types;

These customisatos are created in XML configuration files stored in "resources/acceleratorcore-config".

**Import Users and Groups** will load:

* Users and Groups for the CmsCockpit and ProductCockpit
* Access Rights for the Cockpit Users and Groups

##### Sync Jobs

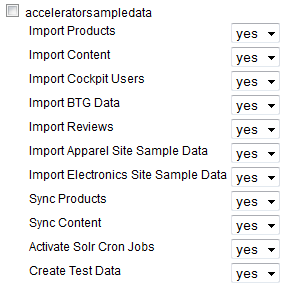
**Sync Product** and **Sync Content** can be used at this stage to sync the Product or Content catalogs, which is useful for development.

**Activate Solr Cron Jobs** can be used to configure and enable the cron jobs that will extract product data to Solr.

By default these are all set to "No" as there will be no data to sync during an initialisation.

### Acceleratorsampledata

#### Acceleratorsampledata Project Data

Project Data setup for **acceleratorsampledata** gives the following options:  


**Import Products** will import:

* Products
* Product Categories
* Promotions
* Product and Category Media
* Product Stock Levels

**Import Content** will import:

* CMS Content, including sample components, pages, images, paragraphs
* Email Content
* Points of Service (Stores) and associated Media

**Import Cockpit Users** will create a sample user for the Cms, Product and Customer Service Cockpits.

**Import BTG Data** will import some sample BTG rules for the Electronics site.

**Import Reviews** will import some sample product reviews for the Electronics site.

**Import Apparel Site Sample Data** and **Import Electronics Site Sample Data** control the load of each sites data, if one is disabled then the sample data for products and content for that site will not be loaded.

**Sync Products** and **Snc Content** will start SyncJobs for Product and COntent catalogues respectively.

**Activate Solr Cron Jobs** will activate the Solr Index update jobs.

**Create Test Data** will create sample orders and payments.

#### Resource folder structure

The Sample Data Import folders are structured as:

* Common, common impex / media
* Cockpit folders (cmscocpit, productcockpit, cscockpit) folders containing configuration and media to support the cockpits
* Site folder, which contains the sites to be imported
  + SiteName
    - Impex
      * Localised Site
    - Images
      * MediaFormat (1200Wx1200H, etc)
      * Cms Images (banners, etc)

#### Products

The Accelerator Sample Data demonstrates two approaches to defining a Product Catalog in Hybris:

* Using [Classifications](https://wiki.hybris.com/display/release4/Classification+System+API) in Electronics
* Using [Variants](https://wiki.hybris.com/display/release4/Variants+Extension) in Apparel

#### CMS Content

The Sample Cms Content for both sites is almost identical but using different media and themes the two sites look totally different.

#### Points of sale

Points of Sale are loaded into Elctronics, Apparel UK and Apparel DE seperately so that searching in one Apparel site, for example, does not return results from the other.

#### Promotions

Sample Promotions have been included that will apply for specific products, specific categories and across entire orders.

#### Btg segments

Some sample rules for Electronics have been loaded. See the [btg Extension - Technical Guide](#80)

## Search and Navigation in the hybris Multichannel Accelerator

The hybris Multichannel Accelerator uses the **solrfacetsearch** extension to provide all product results for the Search and Category navigation pages. This ensures that the navigation to products on the site is consistent across all the search and navigation pages.

* [Solr Configuration](#324)
* [Commerce Services](#325)
  + [Indexing](#66)
  + [Category Hierarchies](#326)
  + [Facet Value Display Name Providers](#327)
  + [Sorting Product Results](#328)
  + [Configuration in the hMC](#329)
* [Commerce Facades](#330)
  + [Starting a Search](#331)
  + [SearchQueryData](#332)
  + [Category Landing Pages](#333)
  + [Free Text Search](#334)
  + [Paging](#335)
  + [Sorting](#336)
  + [Building the Product Results](#337)
  + [Available Facets](#338)
  + [Filtering Results](#339)
  + [Search Breadcrumbs](#340)
  + [URL Generation](#341)
* [Storefront](#104)
  + [SearchPageController](#342)
  + [CategoryPageController](#343)
  + [Example Free Text Search URL and Search Refinements](#344)
  + [Example Search Result Paging URLs](#345)
  + [Example Category URL and Category Filtering URLs](#346)

|  |
| --- |
| **About this Document**  This document describes the implementation and customization of search and navigation mechanisms in the hybris MultichannelAccelerator.  **Audience**: Consultants, developers, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [Solr Facet Search Configuration in the hybris Multichannel Accelerator](#347) * <http://lucene.apache.org/solr/>: Apache Solr platform |

### Solr Configuration

The hybris Multichannel Accelerator is configured to use the embedded Solr instance provided by the **solrfacetsearch** extension. This provides the simplest setup for the development and evaluation. For production environments we recommend using a standalone Solr instance in its own process. Refer to the [SolrFacetSearch - Installation Guide](#348) document for details on how to configure standalone Solr instance.

|  |
| --- |
| **See Also**   * [solrfacetsearch Extension - Technical Guide](#68) |

The embedded Solr instance uses the **solrconfig** and **schema** defined within the **solrfacetsearch** extension. These files are in the $[HYBRIS\_BIN\_DIR]/ext-hybris/solrfacetsearch/resources/solr/embedded/conf folder. These are the standard configuration files shipped by hybris and provide functionality to support the **solrfacetsearch** extension. Solr provides additional capabilities beyond that. For detailed description of Solr configuration and customization options, refer to the [solrfacetsearch Extension - Technical Guide](#68) document.

The **solrfacetsearch** extension provides two ways of configuring the types and properties to index into Solr. In the hybris Multichannel Accelerator you create the entire search configuration in the database as hybris items. This allows you to extend the configuration more easily. See the [SolrFacetSearch - Configuration Guide](#349) document, section **Configuration in Version 4.1.1 and Higher** for details on how to configure the search in the database.

In the hybris Multichannel Accelerator you create a separate Solr index for each product catalog, therefore you have two indexes, one for the **Apparel** product catalog, and one for the **Electronics** product catalog. The configuration for each of these ImpExe instances are imported via ImpEx using following two files:

* $[HYBRIS\_BIN\_DIR]/ext-hybris/acceleratorcore\resources\acceleratorcore\import\site\apparel\solr.impex
* $[HYBRIS\_BIN\_DIR]/ext-hybris/acceleratorcore\resources\acceleratorcore\import\site\electronics\solr.impex

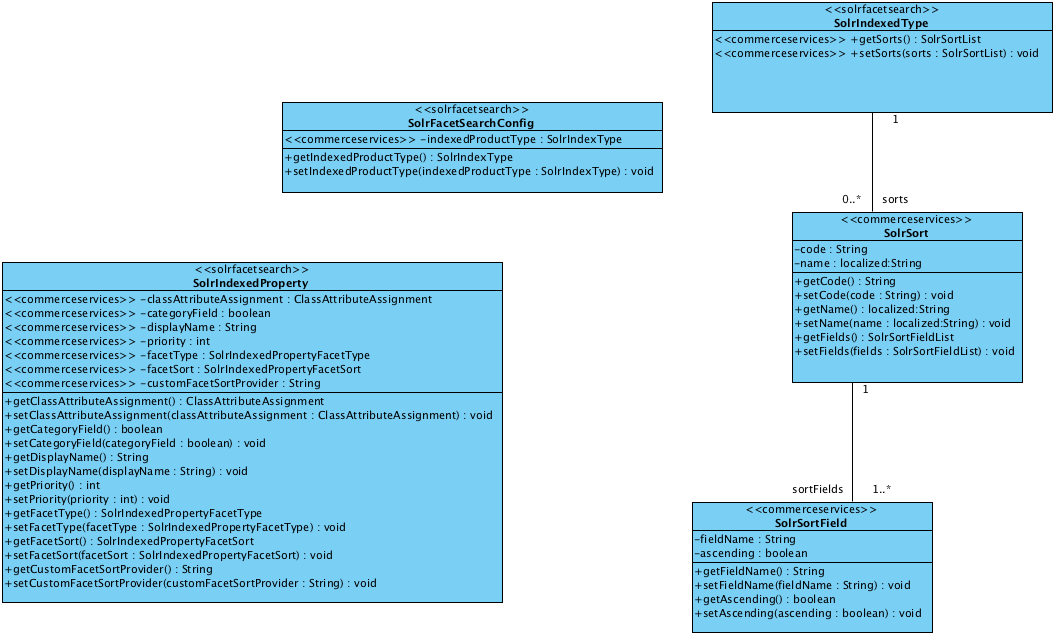


Figure: Extended data model

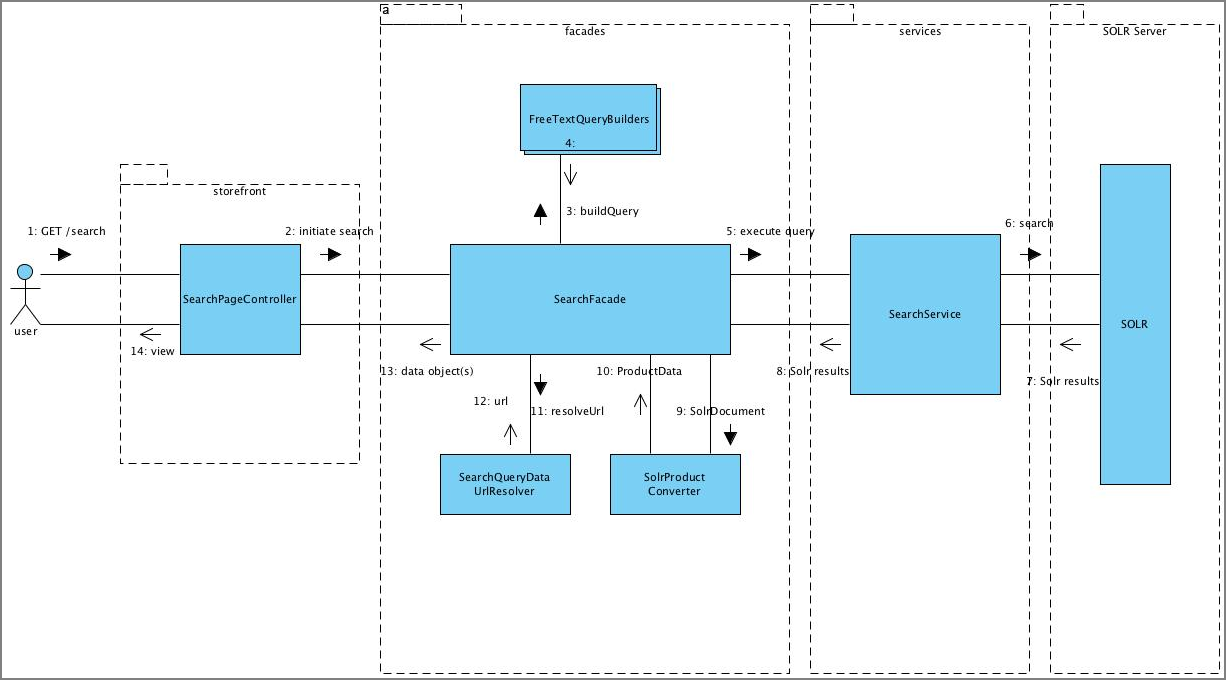


Figure: Search interaction diagram

### Commerce Services

The **commerceservices** extension contains extended the **solrfacetsearch** extension item types to provide the following additional functionality:

|  |
| --- |
| **See Also**   * [commerceservices Extension - Technical Guide](#65) |

* **Product Results Ordering**  
  The **solrfacetsearch** extension provides minimal support for configuring the order in which results are returned, only allowing the default sort order to be set. The **commerceservices** extension allows all the supported ordering to be configured on the **SolrIndexedType**. Each sort option has a display name and a list of the Solr fields to sort against, where ascending or descending sort is supported. This ensures that all results sorting is done in the Solr, and that the available orderings are part of the configuration not the business logic.
* **Display Name for Each Facet:**  
  The **commerceservices** extension added a localized display name to the **SolrIndexedProperty**. This is used in the frontend when displaying the facet.
* **Facet Ordering Priority:**  
  The **commerceservices** extension has added a priority for each facet. This is used to provide an ordering of the facets in the display, with the highest priority facet appearing first. Facets with a negative priority are not displayed. This allows for internal filtering facets that are not displayed to the user.
* **Multi-select Facets:**  
  The **solrfacetsearch** extension supports refinement facets, where only one facet value can be selected within the facet. This is fine for most drill-down search refinement requirements. The **commerceservices** extension has added support for multi-select facets where more than one value can be selected in the same facet, and these multiple values can either be **ANDed** or **ORed** together. For example this allows multiple shoe sizes to be selected as facet filters at the same time.
* **Facet Value Sorting:**  
  The **solrfacetsearch** extension does not support different sorts per facet. The **commerceservices** extension has added support for sorting the values within each facet. This can either be by the facet value count, with the highest count first, or sorted alphabetically A to Z, or with a custom sort provider.
* **Link Indexed Properties to Classification Attributes:**  
  The **commerceservices** extension links indexed properties to the classification system. This allows two additional features:
  + If the **ClassAttributeAssignment.searchable** property is set then the free text query is also matched against this classification feature.
  + If the **ClassAttributeAssignment.listable property** is set then the classification feature values are included in the product data shown on the search results and category navigation pages.
* **Category Facet Flag:**  
  Category facets are typically treated differently on search and navigation pages. The **commerceservices** extension has added a flag to the indexed property to indicate if it should be treated as a category facet.

#### Indexing

Indexing the correct products into Solr is critical. The Solr index must include only purchasable products. It is not possible to programmatically filter the products returned from the Solr search as this would invalidate all the total results count, the facet counts, and the paging. The Solr facet search supports full index rebuild as well as incremental index updates. The products to index are retrieved via **Flexible Search** query. It is important to ensure that all purchasable products are included in the full index rebuild query.

The query for incrementally updating the index is often much more complex than the full rebuild query as it has to take into account any modified products, and any items related to the product that may also have been modified.

|  |
| --- |
| **See Also**   * [FlexibleSearch](#350) |

#### Category Hierarchies

hybris categories are very flexible and are often used to represent different categorizations. For example a category hierarchy could be used to represent product categories, and another to represent brands.

These separate category hierarchies need to be indexed into separate indexed properties. The **commerceservices** extension allow these hierarchies to be configured via a **CategorySource** interface that is injected into the category **FieldValueProviders**. The **CategorySource** implementation is configured with the code of the root category and finds all the category paths between the product and the root category. Furthermore you can configure it to ignore classification class categories.

|  |
| --- |
| **See Also**   * [commerceservices Extension - Technical Guide](#65) |

#### Facet Value Display Name Providers

In certain specific cases the facet value indexed into Solr is not the value that you would like to display to the user. An example of this is with categories, where the indexed value is the unique category code, but the value that you would like to display is the category localized name.

The **solrfacetsearch** extension provides just such a facet display name provider that looks up the localized category name to show instead of the facet value indexed into Solr. When sorting the facet values alphabetically it is the display name that is sorted not the facet value.

#### Sorting Product Results

The ordering of the product results returned from the Solr is critical to correct operation of search. The ordering must be achieved within the Solr, the results cannot be re-sorted programmatically once they have been returned from the Solr as this invalidates the paging support.

Typically results ordering is not overly complex with the user only able to select from a number of pre-defined orderings. This is precisely what the **comerceservices** extension provides. The supported sorts are configured for each indexed type via database items. Each **SolrSort** item has a localized name to display to the user, an order list of **SolrSortField** items which indicate which field and direction to sort on.

The **comercefacade** extension ensures that the results are always finally sorted by the Solr score. The Solr score is the result measure of relevance to the search terms. If you have a sort that orders based on the price field then it is very likely that there are many results with the same price value, and the sort does not specify what the relative ordering of these results with the same price value should be, so to ensure that there is a defined and repeatable ordering the score field is automatically added as a final ordering field.

#### Configuration in the hMC

You can configure Solr facet search options using the hMC. The process of configuration is described in the [SolrFacetSearch - Configuration Guide](#351) document.

### Commerce Facades

The **SearchFacade** provides the interface to execute free text searches and category searches, as well as the ability to page, sort, and refine these searches.

The users search journey is essentially a stateful one, they run a search, review the results, select a refinement option, review the results, and repeat as required. The **SearchFacade** and its data model is designed to ensure that all the state is held by the client and not on the server. This ensures performance and scalability.

|  |
| --- |
| public interface SearchFacade {  SearchPageData search(String text);  SearchPageData search(CategoryData category, boolean categoryJumpMode);  SearchPageData searchAgain(SearchQueryData searchQueryData, PageableData pageableData); } |

#### Starting a Search

Each search must be initiated by calling one of the two **search** methods in the **SearchFacade**. These two search methods are used in the following scenarios:

* **Free text search:**  
  The search is triggered by the user entering a search string. The search string is translated into Solr search terms. The fields to search can be configured, which is explained below. The products that match the free text search string are returned.
* **Category browse:**  
  The search is triggered for a category. The products that are in the specified category are returned.

The search returns the matching products in the default sort ordering. All the search methods return the same type the **SearchPageData**.

The **SearchPageData** contains the following data:

* The search query that was executed.
* The matching products.
* The pagination data, that is: total number of results, page size, current page number, current sort.
* The currently applied filters.
* The available facets and facet values that can be applied to filter the results.
* The available sorts.

#### SearchQueryData

The search query data contains the data required to run a search. The result of running a search, **SearchPageData**, contains **SearchQueryData** instances for applying and removing search filters, see sections below for more details.

Each **SearchQueryData** has a URL that allows the search to be performed.

#### Category Landing Pages

Category search is used to browse the products within a category. The products in the category are returned in the default sort order.

The category search also supports category landing pages where products are not shown, and selecting a sub-category navigates directly to the selected category rather than filtering the results to only show the products that are in the selected category. This category navigation mode can be requested by setting the Boolean parameter to true in the category search method in the **SearchFacade**.

#### Free Text Search

The free text search string is turned into Solr query terms which are then executed to select the matching products. The **SearchFacade** is configured via spring with a list of **FreeTextQueryBuilders** that are responsible for building up the Solr query terms from the free text search.

#### Paging

The **SearchPageData** includes a **PaginationData** data that includes information on the total number of products matched, the page size, the current page and the total number of pages.

In order to get a specific page of search results the **SearchFacade** method **searchAgain** must be called, passing in the **currentQuery** from the **SearchPageData** and a **PageableData** with the **currentPage** set to the requested page. If the requested page is beyond the range of available pages then the last page is returned.

#### Sorting

The **PaginationData** also includes the current sort option, which is one of the available sorts in the **SearchPageData**. The sort order can be changed by calling the **SearchFacade** **searchAgain** method passing in the **currentQuery** from the **SearchPageData** and a **PageableData** with the sort set to the requested sort. If the requested sort is valid then the sort is changed, the search rerun and the results returned.

#### Building the Product Results

The Solr search returns Solr documents. These are converted into **ProductData** by a specialized converter. The converter has access to all the data about the product that has been indexed into Solr. The intention of this converter is that it should be able to create and populate the **ProductData** without loading the **Product** item from the database, which improves the performance and scalability of the search functionality.

It is important to ensure that the entire product data required for display on the search results page is indexed into the Solr index, even if it is not required for search queries.

#### Available Facets

The **SearchPageData** has a list of **FacetData**. Each **FacetData** has a list of **FacetValueDatas** which represents the available values within each of the available facets. The **FacetData** has a user displayable name. The **FacetValueData** also has a user displayable name as well as the facet count, that is the total number of results if this facet value were applied as a filter. The facet values are sorted in the configured order.

The **FacetData** also has another list of **FacetValueDatas** in the **topValues** property. This list holds the top five facet values ordered by count. The number of held values is configurable. This top values list is not always populated. If the total number of facet values is less than the top facet count then the top values are not populated, also if the facet is configured as a multi-select facet then the top values list is not populated either.

#### Filtering Results

Each **FacetValueData** has a **query** property holding a **SearchQueryData** which is preconfigured with the required filters. To perform the same search query but also filter the results by a facet value simply take the query value and pass to the **SearchFacade** **searchAgain** method.

#### Search Breadcrumbs

The **SearchPageData** holds the list of search breadcrumbs. These are the filters that have been applied to the search in the order in which they were applied.

Each **BreadcrumbData** has a **removeQuery** which can be used to run a search with the specific applied filter removed. Also each **BreadcrumbData** has a **truncateQuery** which can be used to remove all the filters applied after the specified filter.

#### URL Generation

Each **SearchQueryData** has a URL. These URLs are created by a **UrlResolver** for the **SearchQueryData** type. The default implementation is **DefaultSearchQueryDataUrlResolver** which identifies if the **SearchQueryData** is a free text search or a category browse. If it is a category browse then it uses the category **UrlResolver** to build the URL to the category page, if it is a free text search then it uses a spring configured path, by default /search . In both cases the **DefaultSearchQueryDataUrlResolver** appends the **SearchQueryData** as a query string parameter **q**.

A spring converter is used to serialize the **SearchQueryData** query string parameter. There are two spring converters for the **SearchQueryData** to support to and from query string parameters. In the storefront these converters are bound into Spring MVC to allow automatic conversion of request parameters.

### Storefront

#### SearchPageController

The **SearchPageController** handles initiating a free text search, and refining a previously run search. Both of these are handled as **GET** requests and use bookmarkable URLs that the user can persist. Reloading once of these bookmarked URLs would re-run the search.

The **SearchPageController** also loads the appropriate CMS page to display content managed content with the search results.

#### CategoryPageController

The **CategoryPageController** decodes the URL path and identifies the relevant category. It then loads the CMS page for the category, which is used to display content managed content on the category page. The controller uses the **SearchFacade** to either initiate a new category search or to refine the existing search depending on which query parameters are passed. Again the URLs that it uses are **GET** requests and bookmarkable.

#### Example Free Text Search URL and Search Refinements

1. Search for shirt .

|  |
| --- |
| /search?text=shirt |

1. Refine by category Playboard .

|  |
| --- |
| /search?q=shirt:relevance:brand:Playboard |

1. Refine by style black .

|  |
| --- |
| /search?q=shirt:relevance:brand:Playboard:style:black |

1. Refine by size L .

|  |
| --- |
| /search?q=shirt:relevance:brand:Playboard:style:black:size:L |

#### Example Search Result Paging URLs

1. Search for shirt .

|  |
| --- |
| /search?text=shirt |

1. Navigate to the second page of results.

|  |
| --- |
| /search?q=shirt:relevance&page=1 |

|  |  |
| --- | --- |
|  | **Note**  The page request parameter is zero based, so 0 is the first page, and 1 is the second page of results. |

#### Example Category URL and Category Filtering URLs

1. Navigate to the **Shirts** category.

|  |
| --- |
| /Categories/Clothes/Shirts/c/shirts |

1. Refine by style red .

|  |
| --- |
| /Categories/Clothes/Shirts/c/shirts?q=:relevance:style:red |

1. Refine by size S .

|  |
| --- |
| /Categories/Clothes/Shirts/c/shirts?q=:relevance:style:red:size:S |

## SEO URLs in the hybris Multichannel Accelerator

This document describes the implementation and customization of SEO URL generation and resolution in the hybris Multichannel Accelerator.

* [SEO URL Implementation in Accelerator](#353)
* [Example SEO URLs from Accelerator](#354)

|  |  |  |
| --- | --- | --- |
| **About this Document**  This document describes the implementation and customization of SEO URL generation and resolution in the hybris Multichannel Accelerator.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Consultants, developers, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [btg Extension - Technical Guide](#80) |

### SEO URL Implementation in Accelerator

In the Accelerator Storefront SEO URLs for Products and Product Categories are generated by Resolvers configured via Spring as follows:

|  |
| --- |
| <alias name="defaultCategoryDataUrlResolver" alias="categoryDataUrlResolver" /> <bean id="defaultCategoryDataUrlResolver" class="de.hybris.platform.commercefacades.url.impl.DefaultCategoryDataUrlResolver" scope="tenant">  <property name="commerceCategoryService" ref="commerceCategoryService"/>  <property name="pattern" value="/{category-path}/c/{category-code}"/> </bean>  <alias name="defaultProductDataUrlResolver" alias="productDataUrlResolver" /> <bean id="defaultProductDataUrlResolver" class="de.hybris.platform.commercefacades.url.impl.DefaultProductDataUrlResolver" scope="tenant">  <property name="commerceCategoryService" ref="commerceCategoryService"/>  <property name="productService" ref="productService"/>  <property name="pattern" value="/{category-path}/{product-name}/p/{product-code}"/> </bean> |

These resolver classes are used to build internal links and for SEO. Changing the generated URLs would usually involve:

* extending DefaultProductDataUrlResolver or DefaultCategoryDataUrlResolver
* changing the configured pattern
* changing the annotated binding in the Page Controller (ProductPageController or CategoryPageController)

In the Accelerator Storefront requests are bound to Controllers via the configured RequestMapping annotation:

|  |
| --- |
| /\*\*  \* Controller for product details page  \*/ @Controller @Scope("tenant") @RequestMapping(value = "/\*\*/p/{productCode}") public class ProductPageController extends AbstractPageController {  private static final Logger LOG = Logger.getLogger(ProductPageController.class); ... |

Table 24 de.hybris.platform.acceleratorstorefront.controllers.pages.ProductPageController.java

SEO URLs are also used by BTG so any changes would need to be mirrored in the BTG filter configuration.

### Example SEO URLs from Accelerator

| **Type** | **Name** | **URL** |
| --- | --- | --- |
| Product | Cyber-shot W55 | /acceleratorstorefront/Cameras/Digital-Cameras/Digital-Compacts/Cyber-shot-W55/p/676442 |
| Category | Digitial Compacts | /acceleratorstorefront/Cameras/Digital-Cameras/Digital-Compacts/c/576 |

## Setting Up the Store Locator in the hybris Multichannel Accelerator

The hybris Store Locator helps consumers to find stores in the proximity of a postcode, town or using the Global Positioning System (GPS) information and to provide directions to the selected store. From a more technical point of view, the hybris Store Locator functionality provides services that can work with standard on-line interfaces, as well as with the mobile solutions.

* [Adding Points of Sale to a Base Store](#356)
  + [Adding a POS using the hMC](#357)
  + [Adding a POS using the ImpEx](#358)
  + [Store Content](#359)
    - [Picture](#360)
    - [Opening Times](#361)
    - [Features](#362)
    - [Store Specific Content](#363)
    - [Address](#364)
    - [Extended Data Model](#365)
* [Storefront API Key Configuration](#366)
* [Setting Site Distance Unit](#367)
* [CMS Storelocator Pages](#368)
* [Geolocation Cron Job](#369)

|  |
| --- |
| **About this Document**  This document introduces the Store Locator usage in the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [Store Locator - Business Guide](#370) * [basecommerce Extension - Technical Guide](#73)   + [Store Locator](#371) |

### Adding Points of Sale to a Base Store

Each base store object that represents the store in the system contains the collection of points of sale. A point of sale represents a single place with geographical representation of the store. It contains several pieces of information, including some descriptive texts, images, a geographical position and an address. A point of sale can be added to the base store using the hybris Management Console (hMC). that is a quick way to add or edit a small number of data, or using ImpEx file, that can be a better choice when you want to add larger amount of data. Find more information on how to add a point of sale in [Using Store Locator Service - Tutorial](#372) document, section **Adding PointsOfService in the hMC**. For information on how to use ImpEx import, refer to [ImpEx - User Guide](#373) document, section **Import**.

#### Adding a POS using the hMC

To add a point of sale using the hMC, navigate to the **Base Commerce** node and locate the base store where you want to add a point of sale. In the tab **Locations** you find an attribute **Locations** that contains points of sale. Here you can create a new point of sale by right clicking at the **Locations** table and selecting **Create new**. The only required attribute is point of sale type, that can be one of **STORE**, **WAREHOUSE** or **POS**. It is recommended to fill other attributes like address and a store name.

When an address is entered you can use geocoding service to determine latitude and longitude of the store and these values are later used by services to display stores on the map and calculating distances. It is possible to manually trigger geocoding service by pressing **Geocode POS location now** button, or the alternative way is to create geolocation job to calculate those values periodically, which is described later in this document.

#### Adding a POS using the ImpEx

Other way of adding new points of sale is using the ImpEx. To do so, you have to prepare script that contains all required information for creating a point of sale in the system. For details on how to create ImpEx script files, refer to [ImpEx Syntax](#374) document, section **ImpEx Syntax**.

Below you find an example script that imports two sample points of sale and then fills it with address and assigns them to store with **basestore**:

|  |
| --- |
| INSERT\_UPDATE PointOfService;name[unique=true];type(code);address(&addrID);latitude;longitude;geocodeTimestamp[dateformat=dd-MM-yyyy] ;Hybris Electronics - Nakano;STORE;addr1;35,7091;139,6732;29-04-2011 ;Hybris Electronics - Shinbashi;STORE;addr2;35,66730;139,75429;29-04-2011 ; INSERT\_UPDATE Address;&addrID;streetname;streetnumber;postalcode[unique=true];town[unique=true];country(isocode);owner(PointOfService.name)[unique=true] ;addr1;Waseda Dori;13;;Tokio;jp;Hybris Electronics - Nakano ;addr2;Hibiya Dori;20;;Tokio;jp;Hybris Electronics - Shinbashi ; INSERT\_UPDATE PointOfService;name[unique=true];basestore(uid) ;Hybris Electronics - Nakano;electronics ;Hybris Electronics - Shinbashi;electronics |

If you do not provide geographic coordinates please make sure they are calculated manually or with geolocation cron job.  
Basing on stored in the system locations of points of service proper distances are calculated and displayed at frontend pages. Distance units are localized and can be configured so after switching to japanese locale *km* will be displayed as *キロ*.

#### Store Content

There are some attributes of each point of sale that are visible at the frontend and can be updated by a user.  
This section contains description of these attributes.

##### Picture

Each point of sale can contain two types of graphics. The first one is represented by **mapIcon** attribute and it is displayed at the Store Locator map of found results. Second one is **storeImage** that is media container that contains store image in different formats that are displayed in the search result list or store details page. If no store image is provided default **coming soon** image is displayed.

##### Opening Times

A point of sale object contains attribute **openingHours**. It is localized text that should present information about opening hours of this specific place. This information is displayed both in search result list and in store details page.

##### Features

Another attribute is **features**. It holds the collection of store features. A store **feature** is a dynamic enumeration value, and you can add new **features** to the system. A **feature** holds some store specific information that you can reuse for the other stores as well, for example if a store is open on Sundays or has wheelchair access. This information is displayed only at store details page.

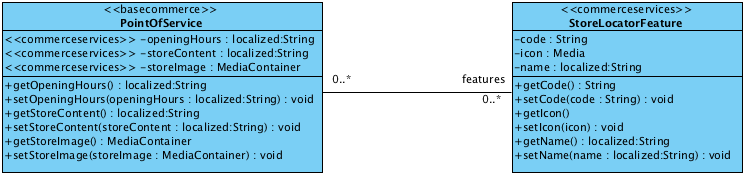
##### Store Specific Content

**storeContent** is another attribute that holds some descriptive content. This attribute is localized and its value is displayed only at store details page.

##### Address

An important attribute for each point of sale is its **address**. An address should be formatted in a way that allows geolocation service to determine geographical coordinates. It is possible to put latitude and longitude directly to the point of sale only using an ImpEx script. In all other cases these values are not editable, and are determined automatically basing on information that are placed to the point of sale **address**.

##### Extended Data Model



### Storefront API Key Configuration

The hybris Multichannel Accelerator project is ready to maintain multiple stores and each store can operate at different address. It requires Google Maps API key to be configurable because each key is valid only for one URL. Project contains **GoogleAPIKeyInterceptor** that is able to hold many references to different API keys and matches proper API key for URL. However to work properly it must be configured using platform project.properties files. The file can contain multiple entries for different registered URL addresses. Properties file contains data as key to value pairs. Google API key in property file should start with **googleApiKey.** followed with url that this particular key should match. Here is sample piece of property file for generated google API keys:

|  |
| --- |
| googleApiKey.apparel.uk.local=ABQIAAAAv(...)4LmJ7bMEeRQ googleApiKey.apparel.de.local=ABQIAAAAv(...)Ma6XGgaJvyQ |

|  |
| --- |
| **See Also**   * <http://code.google.com/apis/maps/signup.html> - you can get the API keys from Google |

### Setting Site Distance Unit

In Store Locator search results page and store details page distance between point of service and searched location is displayed. Depending on the actual address of the point of sale, distance can be displayed in different units. Currently two different unit systems are supported: metric and imperial. You can assign each country ISO code to be displayed in specific unit system using project.properties file in the system. Sample entries in project.properties file may look like this:

|  |
| --- |
| acceleratorstorefront.imperial.isocodes=us,gb acceleratorstorefront.metric.isocodes=de,ja |

The distances of countries with ISO codes listed for **acceleratorstorefront.imperial.isocodes** property key are displayed with imperial units, and countries listed for **acceleratorstorefront.metric.isocodes** are displayed with metric units. Distances in the system are always calculated in the metric system and the actual ratio for an imperial system is hardcoded in the **DistanceHelper** helper class.

### CMS Storelocator Pages

A Store Locator search results and store locator details pages are placed in the CMS page context. With except for the Store Locator related dynamic content, displayed page contains also several other elements that need to be configured in the CMS. For both Store Locator related pages, the CMS page with **storefinderPage** UID is used. So to display Store Locator pages correctly, there must be such a page present in the content catalog version that is used in the current website. The template for this page is **StoreFinderPageTemplate** and it already contains a proper page template and all required content slots that can be filled with required data using **cmscockpit** extension. Most of the content slots are filled by default with proper data (like **SiteLogo**, **HeaderLinks**, **MiniCart**, **NavigationBar**, **Footer**) but they can be easily edited and adapted to the required specification. Top and side content that can contain some commercial banners can also be updated to display desired content.

You can find more information about customizing CMS pages in [cms2 Extension Tutorial](#210) and [cmscockpit Extension - Technical Guide](#59) documents.

### Geolocation Cron Job

At the system initialization or updating of a page you can select check box that creates geocoding cron job. When this cron job is created, it periodically traverses all points of service and tries to determine actual geographical coordinates for the entered address. When the coordinates are determined, they are then used for determining distances between searched location and points of sale.

You can find more information on geocoding cron job in [Using Store Locator Service - Tutorial](#375), section **Adjusting Geocoding Cron Job**.

## Spring Usage in the hybris Multichannel Accelerator

Like the hybris Multichannel Suite itself the hybris Multichannel Accelerator is based on the Spring Framework. It not only uses the IoC container to manage its components, but also other frameworks that are built on top of the Spring Framework. This document provides an overview of which frameworks the hybris Multichannel Accelerator uses.

* [Using Spring as IoC Container](#376)
* [Spring MVC](#377)
* [Spring Integration](#378)
* [Spring Security](#379)

|  |
| --- |
| **About this Document**  This document introduces Spring usage in the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, software architects, technical deciders  **Related concept**: the hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [Spring in the hybris Multichannel Suite](#149)   + [Spring Integration](#247)     - [How To Start processengine Process with a Service Activator - Tutorial](#380) * [Spring Security](#145) |

### Using Spring as IoC Container

The Spring Framework is used to manage components like hybris services, facades, MVC controllers and so on. They IoC (Inversion of Control) container is responsible for creating these components and wiring the interdependencies. In this regard the hybris Multichannel Accelerator is no different than the rest of the hybris Multichannel Suite. Please refer to the document about [Spring in the hybris Multichannel Suite](#149) for information about how in the application context is set up, where to find the bean definitions and which pattern and practices are used.

See also the following documents:

* [commerceservices Extension - Technical Guide](#65)
* [commercefacades Extension - Technical Guide](#95)
* [acceleratorfacades Extension - Technical Guide](#86)

|  |
| --- |
| **See Also**   * [ServiceLayer](#44) * [Using Facades and DTOs - Best Practice](#243) |

### Spring MVC

In the **acceleratorstorefront** extension the frontend logic is implemented using Spring MVC. It follows the patterns explained in the document [Spring in the hybris Multichannel Suite](#150) section **Using Spring MVC**. The Spring MVC related configuration files are located in the web/webroot/WEB-INF/config folder of the **acceleratorstorefront** extension. The controllers themselves are configured by annotation as per hybris conventions.

|  |
| --- |
| **See Also**   * [acceleratorstorefront Extension - Technical Guide](#97) * [http://static.springsource.org/spring/.../mvc.html](http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/mvc.html): Web MVC framework |

### Spring Integration

A hot folder import capability is built into the hybris Multichannel Accelerator. It uses the Spring Integration framework to implement Enterprise Integration Patterns, like watching the hot folder and service activators to invoke the necessary ImpEx import infrastructure.

|  |
| --- |
| **See Also**   * [acceleratorcore Extension - Technical Guide](#83) section **Hot Folder Batch Data Import Capability** * [Spring Integration](#247) * <http://www.springsource.org/spring-integration>: Spring Integration |

### Spring Security

Spring security-related issues are described in the document [Spring Security in the hybris Multichannel Accelerator](#108).

## Spring Security in the hybris Multichannel Accelerator

Like the hybris Multichannel Suite itself the hybris Multichannel Accelerator is based on the Spring Framework. It not only uses the IoC container to manage its components but also other frameworks that are built on top of the Spring Framework. This document explains which spring security related topics in the hybris Multichannel Accelerator project.

* [Spring Security](#381)
* [Configuration](#382)

|  |
| --- |
| **About this Document**  This document introduces Spring usage in the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, software architects, technical deciders  **Related concept**: the hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [Spring in the hybris Multichannel Suite](#149)   + [Spring Integration](#247)     - [How To Start processengine Process with a Service Activator - Tutorial](#380) * [Spring Security](#145) |

### Spring Security

The hybris Multichannel Accelerator leverages the built in Spring Security support of the hybris Platform, for details see [Spring Security](#145) document. The main goal is to deliver a secure, reliable and best-practices driven solution. Therefore, the existing **CoreAuthenticationProvider** was reused providing a clean separation of the authentication and authorization by allowing to configure access restrictions by means of security interceptors. Instead of relying on session fixation protection, an even more restrictive secure GUID cookie was introduced offering this functionality:

* After successful authentication, a GUID is generated and stored in a secure session cookie.
* For every secure request, the presence of this cookie is ensured by means of an interceptor.
* Exclusion URLs are configurable to allow accessing secure pages prior to authentication.

|  |
| --- |
| **See Also**   * <http://static.springsource.org/spring-security/site/>: springsource Spring Security * [Wikipedia on GUID](http://en.wikipedia.org/wiki/GUID) |

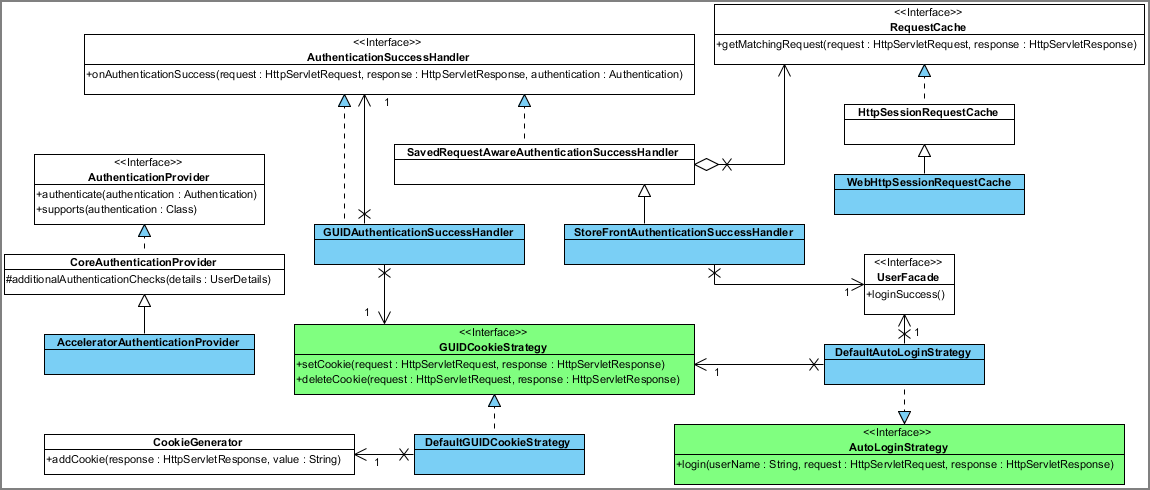


Figure: Classes relevant for the authentication.

During authentication, these steps are processed:

1. The user is authenticated by **AcceleratorAuthenticationProvider** overriding **additionalAuthenticationChecks**. This has two purposes:
   * Preventing administrators from authentication in the hybris Multichannel Accelerator storefront. This is required since **SearchRestrictions** do not apply for administrators leading to catalog version ambiguities.
   * Preventing users created in the CS Cockpit without password from authenticating.
2. If authentication was successful, following steps are performed by the system:
   * The secure GUID cookie is added by **GUIDAuthenticationSuccessHandler**.
   * **StoreFrontAuthenticationSuccessHandler** is activated, delegating initialization of the JaloSession for the user to UserFacade.
   * Deep linking is supported by **SavedRequestAwareAuthenticationSuccessHandler**.
   * **WebHttpSessionRequestCache** wraps the **SavedRequest** to supply the newly created GUID cookie to the **GUIDInterceptor**.

Automatic login is supported during registration. In this case reusing **AuthenticationSuccessHandler** is not possible due to the redirecting responsibility implied by Spring Security.

### Configuration

All security related configuration except for **GUIDInterceptor** is contained in the spring-security-config.xml file which is located in $[HYBRIS\_BIN\_DIR]/ext-hybris/acceleratorstorefront/web/webroot/WEB-INF/config . Redirecting to a secure channel and access restrictions are maintained using the security namespace, for example:

|  |
| --- |
| <security:intercept-url pattern="/checkout/\*\*" access="ROLE\_CUSTOMERGROUP" requires-channel="https" /> |

Thereby, access to checkout and all subpages is granted to users in the customergroup ony. Additionally, if the page is called from an unsecure channel, Spring Security will redirect using HTTPS. Static files should remain unrestricted, for example:

|  |
| --- |
| <security:intercept-url pattern="/\_ui/\*\*" filters="none" /> |

The **GUIDInterceptor** is configured in the spring-mvc-config.xml file which is located in $[HYBRIS\_BIN\_DIR]/ext-hybris/acceleratorstorefront/web/webroot/WEB-INF/config :

|  |
| --- |
| <bean class="de.hybris.platform.acceleratorstorefront.security.GUIDInterceptor" scope="tenant">  <property name="redirectStrategy" ref="redirectStrategy"/>  <property name="loginUrl" value="/login"/>  <property name="excludeUrls">  <set>  <value>/login</value>  <value>/login/register</value>  <value>/login/checkout</value>  <value>/login/checkout/register</value>  <value>/login/pw/request</value>  <value>/login/pw/change</value>  <value>/s/slang</value>  <value>/s/scurrency</value>  <value>/search</value>  <value>/rolloverCartPopup/MiniCart</value>  </set>  </property>  <property name="cookieGenerator" ref="guidCookieGenerator"/>  <property name="urlPathHelper" ref="urlPathHelper"/> </bean> |

Table 25 spring-mvc-config.xml

For paths contained in **excludeUrls**, the existence of the GUID cookie is not enforced. All remaining paths trigger a redirect to the **loginUrl** if called on a secure channel without GUID cookie set.

## Stock Management in the hybris Multichannel Accelerator

An important part of the hybris Multichannel Accelerator system is **stock**. This functionality comes from the **basecommerce** extension and no additional facade is implemented in the hybris Multichannel Accelerator. Stock information is used in two main places - first one is **ProductStockPopulator** where stock information is injected into product data, that is then presented in the frontend. Stock information is also used independently during cart operations. When a product is added to the cart, or a cart entry amount is updated, the new quantity is checked against current stock level and proper actions are performed.

* [Stock Service](#383)
  + [Stock in Multichannel Accelerator](#384)
* [Warehouses](#385)
  + [Vendors](#386)
* [Stock Levels](#387)
  + [Creating Stock Level Information](#388)
  + [Retrieving Stock Level Information](#389)
  + [Turning Stock Functionality On and Off](#390)

|  |
| --- |
| **About this Document**  This document describes the stock service and warehouse integration in the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [basecommerce Extension - Technical Guide](#73)   + [Stock Service](#391) * [Warehouse Integration](#392) |

### Stock Service

Most of stock operations are performed using the **stock service**. Using this service you are able to determine current stock level of the desired product. It can be both a global stock level, or a stock level in the chosen warehouse. The stock service also exposes methods that allow you to update current stock level or check stock status for a specified product.

#### Stock in Multichannel Accelerator

As it was mentioned at the beginning of the document stock is used in accelerator in two main places. **ProductStockPopulator** uses **stock service** to retrieve information about the current stock level of the product and this information is displayed later on at the front end. There are few stock level displaying scenarios in accelerator:

1. no warehouse exists in the system or there is no default warehouse set. In this case all products are displayed as *available* and user is able to add any amount to the cart.
2. at least one warehouse exists in the system and there is default warehouse set. At least one stock level is assigned to the product and no assigned stock levels has set **inStockStatus** attribute. In this case numeric stock level is displayed for the product and user is able to add to the cart amount that is no larger than available stock level.
3. at least one warehouse exists in the system and there is default warehouse set. At least one stock level is assigned to the product and at least one stock level has set **inStockStatus** to *FORCEINSTOCK*. In this case product is displayed as *available* regardless of real stock level and user is able to add any amount to the cart.
4. at least one warehouse exists in the system and there is default warehouse set. At least one stock level is assigned to the product and at least one stock level has set **inStockStatus** to *FORCEOUTOFSTOCK*. In this case product not available regardless of real stock level and user is not able to add any amount to the cart.

Another place where stock levels are used in the system is the cart. When user requests to change cart entry amount the current stock levels are taken into the consideration in the same scenarios as it was described above.

### Warehouses

A warehouse collects stock levels of the specified vendor. Using stock service you are able to retrieve a product stock level for the specific warehouse. You can find more information about warehouses in the [Warehouse Integration](#392) document. There are no accelerator specific changes made to the warehouse functionality. If there is no warehouse defined in the system or there is no default warehouse set, all products are displayed as *available* and the user is able to add them to the cart, but no real stock levels are displayed.  
Warehouses and vendors are not related in any way to website or store. When system is multistore and the requirement is to assign warehouse to a specific store or website it will require client specific customization.

#### Vendors

Each vendor collects one or more warehouses, that can be later used for retrieving a product stock levels. The stock service does not expose methods that you use to retrieve product stock levels for all warehouses of the specified vendor, so a vendor object itself is more like information field. There is no vendor specific changes made in accelerator to vendor area. It is not displayed anywhere and used to retrieve information, however at least one vendor must exist in the system to create a warehouse.

### Stock Levels

Stock levels are the most important objects of the whole stock framework. Each product can have one or more stock levels assigned. Stock levels are described in details in the [Stock Service - Technical Description](#393) document, section **Stock Level**. Each stock level contains information about the actual stock level of a product in the specific warehouse and several other attributes that helps to control stock related operations, like **reserved amount**, **overselling amount**, **stock status** that can be one of **force in stock**, **force out of stock**, and so on. In Mutlichannel Accelerator only two of them are used: **available** that returns actual stock level amount and **inStockStatus** that is enumeration value and is used to override available amount if this attribute is set to any value.

#### Creating Stock Level Information

To create the stock level information for a specific product, we need to assure following conditions are fulfilled:

1. The vendor must be created that contains one ore more warehouses.
2. At least one warehouse must be created, and there must exist the default warehouse.
3. The product must have assigned at least one stock level with filled up required attributes.

When we have all these data in the system we can then display the stock level information for a specific product.

#### Retrieving Stock Level Information

When we have the stock related data in the system we can use the stock service to retrieve the required information. The stock service contains several methods that allow us to fetch the product stock information data for a specific warehouse, but also the global stock level for all available warehouses.

|  |
| --- |
| **See Also**   * [Stock Service - Technical Description](#394) |

#### Turning Stock Functionality On and Off

If you do not have or want to provide any stock information to the running system, you have all products just available without any stock information provided per product. To set the system to this state you need to have no default warehouse existing. If there is no default warehouse, all products are presented as **available** and no stock information is displayed. A customer is able to add products to the cart and proceed with the checkout.

## Top Navigation and Link Management in the hybris Multichannel Accelerator

The hybris Multichannel Accelerator offers a customizable top navigation and link management functionality. The customization is done through the WCMS components.

* [Top Navigation](#395)
  + [Examples](#396)
* [Link Management](#397)

|  |  |  |
| --- | --- | --- |
| **About this Document**  This document describes the mechanisms of the top navigation and link management in the hybris Multichannel Accelerator.   |  |  | | --- | --- | |  | **Note**  This document is a draft, which has not yet been reviewed. Thus you might recognize mistakes or defects. |   **Audience**: Consultants, developers  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [acceleratorcore Extension - Technical Guide](#82), section **WCMS Components** |

### Top Navigation

The Top Navigation on an Accelerator site is controlled though the configuration of a number of CMS Components:

* **NavigationBarComponents** which represent sections of the Navigation Bar, each can have **one** CMSLinkComponent and **one** CMSNavigationNode considered to be the root node.
* **CMSNavigationNodes** which represent the structure of the navigation tree and have titles which are rendered in the drop down menu, each can have **many** CMSLinkComponent and **many** child CMSNavigationNodes.
* **CMSLinkComponents** which are actually rendered on the page and sit at each point in the navigation tree.

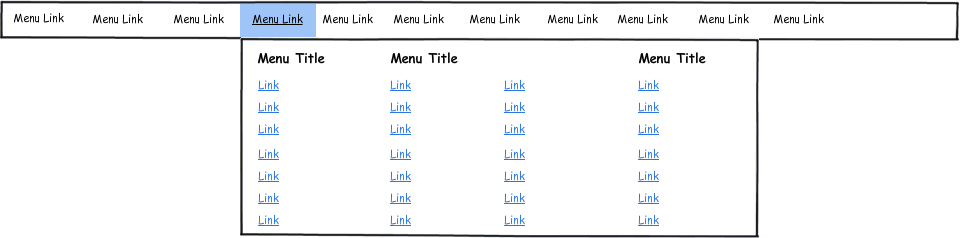
The process for adding a new NavigationBarComponent to a site so that a menu is shown is:

1. Create a NavigationBarComponent
2. Create a CMSLinkComponent, give it a Title and a Url and assign it to the NavigationBarComponent
3. Create a root CMSNavigationNode and add this to the NavigationBarComponent
4. Create as many children CMSNavigationNodes as required and add these as children of the root CMSNavigationNode, each child node will create an additional column in the menu
5. Create as many CMSLinkComponents as required and add each to a child node, these will appear as a list of links in the column

Additional configuration on NavigationBarComponent is available:

* **wrapAfter** controls the wrapping of links into additional columns
* **styleClass** allows an additional CSS style to be apended to the NavigationBarComponent so that it can be right justified, or highlighted etc
* **dropDownLayout** allows configuration of the justification of the Drop Down Menu, so that it can be made to appear on the right of the screen for example, to improve layout

#### Examples



* This shows 11 NavigationBarComponents.
* The highlighted NavigationBarComponent's root node has 3 child nodes.
* The 1st and 3rd child nodes have 7 CMSLinkComponents, the 2nd has 14.
* wrapAfter is set to 7



* This shows a similar configuration with dropDownLayout set to RIGHT

### Link Management

Links in the Top Navigation are CMSLinkComponents. These can be used to provide links to:

* A Url
* A ContentPage
* A Category
* A Product

Using Url directly can be used to provide any internal link such as to a category. If the CMSLinkComponent has more than one of the above set, then the order of preference is also as above.

Links can also be configured to open in new tabs/windows via the **target** attribute.

## WCMS Integration in the hybris Multichannel Accelerator

This document contains all the necessary information about the WCMS integration of the **acceleratorstorefront** extension in order to enable all the content management, preview and live edit functionality based on the **cms2** extension. This integration is implemented reusing existing classes of the **storetemplate** extension while adding and modifying existing functionality which is described in detail.

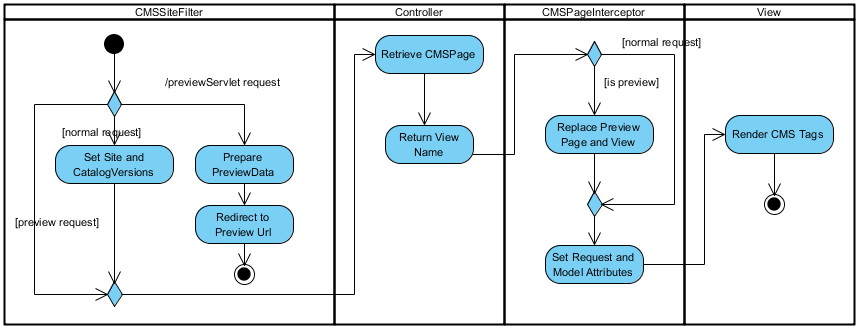
* [WCMS Integration Overview](#398)
  + [CMSSiteFilter](#399)
  + [DefaultCMSPageURLMappingHandler](#400)
  + [AbstractPageController](#401)
    - [Example for Use of Switchable Template](#402)
    - [Example for Use of Static Template](#403)
  + [CMSPageInterceptor](#404)
* [View](#405)
  + [Example Using Default Controller](#406)
  + [Example Using a Custom Controller](#407)
  + [Tag Files](#408)
    - [Tag Files Example](#409)
* [CMS Page Configuration](#410)
  + [CMS Page Configuration Example](#411)
    - [PageTemplate Definition](#412)
    - [ContentSlotName Definition](#413)
    - [Page Definition](#414)
    - [ContentSlot Definition](#415)
    - [ContentSlotForTemplate Definition](#416)
    - [ContentSlotForPage Definition](#417)

|  |
| --- |
| **About this Document**  This document provides information on the hybris Multichannel Accelerator storefront WCMS integration.  **Audience**: Consultants, developers, software architects  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [cms2 Extension Tutorial](#210) * [CMS Data Model](#418) * [How To Set Up the Preview in the WCMS Cockpit - Tutorial](#419) * [ImpEx Syntax](#163) * [acceleratorcore Extension - Technical Guide](#81) * [Email WCMS and Process Engine Integration in the hybris Multichannel Accelerator](#37) * [WCMS Components for the hybris Multichannel Accelerator](#89) |

### WCMS Integration Overview

The general page processing in the **acceleratorstorefront**extension regarding WCMS is shown in the next diagram:



As shown, there are four major steps during processing:

1. General session setup and preview handling, that is session setup and redirect.
2. Page and view resolution in the Spring MVC Controller.
3. Further population of request and model attributes in the Interceptor with additional page and view replacement functionality during preview.
4. View rendering using CMS tags, for example **cms:body**, **cms:slot**, **cms:component**).

The classes executing these steps are described in the next section.

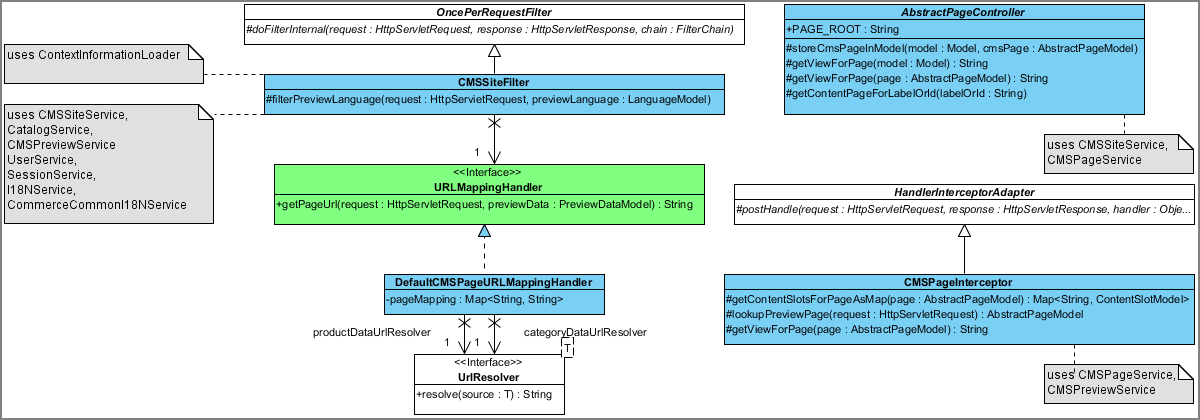


Figure: A diagram of classes for the WCMS integration within the hybris Multichannel Accelerator.

#### CMSSiteFilter

This filter is responsible for session setup and preview handling and was reused from the **storetemplate** extension. For details see [How To Set Up the Preview in the WCMS Cockpit - Tutorial](#419) document. The changes made to these classes include:

* Using **CommerceCommonI18NService** instead of **CommonI18NService**.
* Adding a method **filterPreviewLanguages** to restrict preview languages to languages supported by the **CMSSite**.
* Replacing the **URLMappingHandler** with an implementation supporting page mappings and product and category pages.

|  |
| --- |
| **See Also**   * [storetemplate Extension - Technical Guide](#420) |

For initializing, the JaloSession **CMSSiteFilter** collaborates with **ContextInformationLoader** which was also reused without relevant changes from the **storetemplate** extension.

#### DefaultCMSPageURLMappingHandler

The responsibility lies in retrieving a redirect URL during previewing and supports URLs for:

* A previewed **Product** or **Category** by delegating to**UrlResolver** implementations in the **commercefacades** extension.
* Content pages using the **DefaultPageController** and using the lookup path as UID, for example **/about**
* Content pages using custom controllers configured in a **pageMapping**.

|  |
| --- |
| **See Also**   * [commercefacades Extension - Technical Guide](#95) |

Page mappings are added to spring-mvc-config.xml :

|  |
| --- |
| <bean id="defaultCMSPageURLMappingHandler" class="de.hybris.platform.acceleratorstorefront.preview.DefaultCMSPageURLMappingHandler" scope="tenant">  <property name="pageMapping">  <map>  <entry key="homepage" value="/"/>  <entry key="basketPage" value="/basket"/>  <entry key="search" value="/search"/>  <entry key="account" value="/my-account"/>  <entry key="profile" value="/my-account/profile"/>  <entry key="addressBook" value="/my-account/addressBook"/>  <entry key="addEditAddress" value="/my-account/addEditAddress"/>  <entry key="paymentDetails" value="/my-account/paymentDetails"/>  <entry key="order" value="/my-account/order"/>  <entry key="orders" value="/my-account/orders"/>  </map>  </property> </bean> |

Table 26 spring-mvc-config.xml

#### AbstractPageController

This abstract class is the base class of all controllers in the **acceleratorstorefront** extension and has the responsibility to provide:

* Lookup functionality for CMS pages.
* Lookup functionality for retrieving the view name for controllers supporting switchable views, for example **HomePageController**).
* Convenient methods to set basic model attributes, that is **user**, **currencies**, **languages**, **currentCurrency**, **user**, **request**.

##### Example for Use of Switchable Template

**HomePageController** is an example of a controller supporting switchable views. Thereby, the view is manageable in the WCMS Cockpit by setting the **frontendTemplateName**. To support this behavior, only two lines of code are necessary:

|  |
| --- |
| @Controller @Scope("tenant") @RequestMapping("/") public class HomePageController extends AbstractPageController {  @RequestMapping(method = RequestMethod.GET)  public String home(final Model model) throws CMSItemNotFoundException  {  storeCmsPageInModel(model, getContentPageForLabelOrId(null));  return getViewForPage(model);  } } |

Table 27 HomePageController

First, the method **home** is mapped to the correct path and request method. Then, **getContentPageForLabelOrId** is called which in turn performs following calls:

* If the parameter **labelOrId** is **null** or empty retrieves the label for the homepage from **CMSPageService**.
  + If no page is marked as homepage, retrieves the label for the starting page for the current site.
* Returns the page for the resolved key by calling **getPageForLabelOrId** on **CMSPageService**.

The retrieved page is stored as model attribute **CMS\_PAGE\_MODEL**. In a second step, the view is retrieved by calling **getViewForPage** which is achieved by receiving the the previously set page from the model and accessing the attribute **frontendTemplateName** of master template as shown here:

|  |
| --- |
| protected String getViewForPage(final AbstractPageModel page) {  if (page != null)  {  final PageTemplateModel masterTemplate = page.getMasterTemplate();  if (masterTemplate != null)  {  final String targetPage = cmsPageService.getFrontendTemplateName(masterTemplate);  if (targetPage != null && !targetPage.isEmpty())  {  return PAGE\_ROOT + targetPage;  }  }  }  return null; } |

Table 28 AbstractPageController

This allows you to exchange the landing page layout easily by switching to a different landing page template in the WCMS Cockpit. Besides, the **frontendTemplateName** could also be updated directly to a different JSP page supporting the given template layout. Nevertheless, the recommended approach followed in the hybris Multichannel Accelerator is to define a new template instead of updating the template name directly.

##### Example for Use of Static Template

For some controllers supporting switchable views is not a requirement. For example, the login and account page do not support multiple layouts and therefore take care of returning the view name directly after defining the CMS page. An example of this is the method to show the page to update the password:

|  |
| --- |
| @RequestMapping(value = "/update-password", method = RequestMethod.GET) public String updatePassword(final Model model) throws CMSItemNotFoundException {  final UpdatePasswordForm updatePasswordForm = new UpdatePasswordForm();  model.addAttribute("updatePasswordForm", updatePasswordForm);  storeCmsPageInModel(model, getContentPageForLabelOrId(PROFILE\_PAGE));  return ACCOUNT\_PAGE\_ROOT + PASSWORD\_PAGE; } |

Table 29 AccountPageController

This simple example shows how to perform following actions:

* Map a controller method to a request by using annotations.
* Populate the model with a form bean.
* Retrieve the CMS page by calling **storeCmsPageInModel** providing a page UID.
* Return the view name to Spring MVC using constants.

#### CMSPageInterceptor

This interceptor is required to keep the controller agnostic of preview related functionality and to populate **Model** and **HttpServletRequest** attributes for the view. First, in the presence of the request parameter **PREVIEW\_TICKET\_ID\_PARAM**, the preview context is evaluated. If the page attribute was set in the WCMS Cockpit and differs from the page retrieved by the controller, then, the page is replaced in the model or, if there is view associated to the template and this is different from the view returned by the controller, than the view is replaced in the **ModelAndView**.

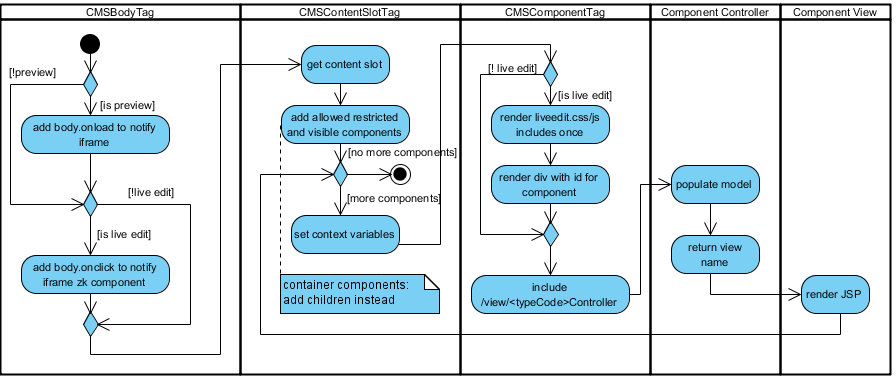
In a second step, **Model** and **HttpServletRequest** attributes are defined. Find a a list of attributes to be taken care of below:

| **Class** | **Attribute Name** | **Attribute Type** | **Description** |
| --- | --- | --- | --- |
| **HttpServletRequest** | currentPage | **AbstractPageModel** | The page request attribute required by **CMSBodyTag**. |
| **Model** | cmsPage | **AbstractPageModel** | The page model attribute. Currently, only referenced in master.tag to retrieve the page title and provide debug information. |
| **Model** | cmsSite | **CMSSiteModel** | The current site. Currently, only referenced in master.tag to provide debug information. |
| **Model** | slots | **Map<String, ContentSlotModel>** | The slots attribute to allow for referencing CMS content slots in JSPs and tags. |

### View

The view rendering using the **cms2** extension tags and Spring MVC controllers is shown in the next diagram:

|  |
| --- |
| **See Also**   * [cms2 Extension - Technical Guide](#78) |



The classes involved in this processing are as follows:

| **Class Name** | **Description** |
| --- | --- |
| **CMSBodyTag** | Adds specific scripts for preview and live edit to integrate preview in CMS cockpit. |
| **CMSContentSlotTag** | * Retrieves the content slot either by UID or as an tag attribute * Initializes a list with all components which are visible or allowed restricted. For restricted components this decision is delegated to **CMSRestrictionService**. * Iterates over all components, setting the required page context attributes: **contentSlot**, **elementPos**, **isFirst**/**LastElement**, **numberOfElements**, tag attribute **var**. |
| **CMSComponentTag** | * Handles Live Edit perspective by adding the liveedit.js and liveedit.css once and rendering a **div** around the component. * Renders the component by calling **pageContext.include("/view/" + controllerName)** using:   + **<componentTypeCode>Controller** as a controller name if a bean with this name is defined in the Spring application context.   + **DefaultCMSComponentController** otherwise. |
| Component Controller | * Populates the model   + the implementation of **DefaultCMSComponentController** simply puts all frontend properties of the component into the model   + returns the view name to use, following the convention to append the component type in lowercase to     - **/WEB-INF/components** for specific components or     - **/WEB-INF/components/cms2** for components using **DefaultCMSComponentController** |
| Component View | Finally, renders the component. |

Above, you find examples of components using a default and a specific controller. As an example of a specific controller see also the [Search And Navigation](#352) document, section **Storefront**.

#### Example Using Default Controller

A simple example of a CMS component using the default controller is the **CMSParagraphComponent** containing only a content attribute to manage rich text content. To display this component, a JSP is provided in the directory /WEB-INF/components/cms2 :

|  |
| --- |
| <div class="content">${content}</div> |

Table 30 cmsparagraphcomponent.jsp

This component is used in the header to display the contact information. To achieve this, the **cms:slot** tag is added to  header.tag , referencing a previously configured content slot:

|  |
| --- |
| <cms:slot var="link" contentSlot="${slots.HeaderLinks}">  <li><cms:component component="${link}"/></li> </cms:slot> |

Table 31 header.tag

This example shows how to:

* Add a JSP view for a simple CMS component.
* Use the CMS tags to render the component in a JSP or tag file.

#### Example Using a Custom Controller

You can use more advanced CMS component controllers to call:

* Facade implementations in the **acceleratorfacades** and **commercefacades** extension.
* Services in the **commerceservices** extension.
* Other platform services.

An example of this is shown below:

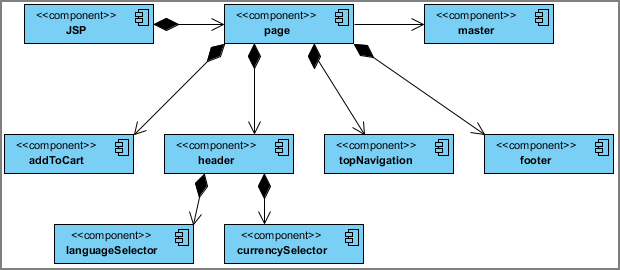
|  |
| --- |
| @Controller("PurchasedProductReferencesComponentController") @Scope("tenant") @RequestMapping(value = ControllerConstants.Actions.Components.PURCHASED\_PRODUCT\_REFERENCES\_COMPONENT) public class PurchasedProductReferenceComponentController extends  AbstractCMSComponentController<PurchasedProductReferencesComponentModel> {  @Autowired  @Qualifier("simpleSuggestionFacade")  private SimpleSuggestionFacade simpleSuggestionFacade;   @Override  public String renderComponent(final HttpServletRequest request, final HttpServletResponse response, final Model model,  final PurchasedProductReferencesComponentModel component)  {  final List<ProductData> products = simpleSuggestionFacade.getReferencesForPurchasedInCategory(component.getCategory()  .getCode(), component.getProductReferenceType(), component.getFilterPurchased().booleanValue(), component  .getMaximumNumberProducts());  model.addAttribute("title", component.getTitle());  model.addAttribute("productReferences", products);  model.addAttribute("comp", component);   return ControllerConstants.Views.Components.PRODUCT\_REFERENCES;  } } |

Table 32 ProductFeatureComponentController

This CMS component is used to retrieve and display product references of a product purchased by a customer. These references are returned by calling a **SimpleSuggestionFacade** in the **acceleratorfacades** extension. Additionally, component attributes are populated in the Spring **Model**. Finally, the view name to use is returned using a constant in **ControllerConstants**. This example shows how to:

* Map a custom controller to a request URL using a Spring MVC **RequestMapping** annotation.
* Autowire a dependency using Spring annotations.
* Retrieve additional data by calling a facade method.
* Populate the model.
* Return a view name to render the component.

#### Tag Files

The view is rendered using JSPs and tag files. JSPs include several tag files to impose a general, reusable structure shown in the next diagram:  


The following tag components are provided in /WEB-INF/tags folder:

| **Component** | **Filename** | **Description** |
| --- | --- | --- |
| page | template/page.tag | General page structure |
| master | template/master.tag | Master HTML template |
| header | common/header/header.tag | Header bar |
| languageSelector | common/header/languageSelector.tag | Language selector |
| currencySelector | common/header/currencySelector.tag | Currency selector |
| topNavigation | nav/topNavigation.tag | Top navigation |
| footer | common/footer/footer.tag | Footer |
| addToCart | cart/addToCart.tag | Ajax cart popup |

##### Tag Files Example

The **electronics** storefront landing page uses a dedicated JSP landingPage2.jsp referenced in the CMS master template. This example shows how to:

* Include the page tag in a JSP file.
* Access a content slot by name using the predefined **slots** attribute set up in **CMSPageInterceptor**
* Render CMS components within content slots.

|  |
| --- |
| <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %><c:set var="ignore-white-space">  <%@ taglib tagdir="/WEB-INF/tags/template" prefix="template" %>  <%@ taglib prefix="cms" uri="/cms2lib/cmstags/cmstags.tld" %> </c:set>  <template:page pageTitle="${title}">  <cms:slot var="feature" contentSlot="${slots.Section1}">  <div class="span-24 section1 advert">  <cms:component component="${feature}"/>  </div>  </cms:slot>  <div class="span-24 section2">  <div class="span-6 zone\_a thumbnail\_detail">  <cms:slot var="feature" contentSlot="${slots.Section2A}">  <cms:component component="${feature}"/>  </cms:slot>  </div>  <div class="span-6 zone\_b advert">  <cms:slot var="feature" contentSlot="${slots.Section2B}">  <cms:component component="${feature}"/>  </cms:slot>  </div>  <div class="span-12 zone\_c advert last">  <cms:slot var="feature" contentSlot="${slots.Section2C}">  <cms:component component="${feature}"/>  </cms:slot>  </div>  </div>  <div class="span-24 section3 advert">  <cms:slot var="feature" contentSlot="${slots.Section3}">  <cms:component component="${feature}"/>  </cms:slot>  </div>  <div class="span-24">  <cms:slot var="feature" contentSlot="${slots.Section4}">  <div class="span-6 section4 small\_detail ${(elementPos%4 == 3) ? 'last' : ''}">  <cms:component component="${feature}"/>  </div>  </cms:slot>  </div>  <div class="span-24 section5 advert">  <cms:slot var="feature" contentSlot="${slots.Section5}">  <cms:component component="${feature}"/>  </cms:slot>  </div>  </template:page> |

Table 33 landingPage2.jsp

### CMS Page Configuration

You can configure templates and pages in the WCMS Cockpit or using an ImpEx script. Due to the complexity of the data, the more common approach is using ImpEx. To do so, perform these steps :

1. Define **PageTemplate**.
2. Define **ContentSlotName**.
3. Define page.
4. Define **ContentSlot**.
5. Assign reusable content slots to the template in **ContentSlotsForTemplate**.
6. Assign specific content slots to the page in **ContentSlotsForPage**.

|  |
| --- |
| **See Also**   * [Data Model Overview](#418) |

#### CMS Page Configuration Example

As an example you find the relevant scripts for configuring the homepage on the **electronics** storefront site. The **acceleratorcore** extension provides the general setup for steps one to three in cms-content.impex :

|  |
| --- |
| # # Import the CMS content for the Electronics site # $contentCatalog=electronicsContentCatalog $contentCV=catalogVersion(CatalogVersion.catalog(Catalog.id[default=$contentCatalog]),CatalogVersion.version[default=Staged])[default=$contentCatalog:Staged] $jarResource=jar:de.hybris.platform.acceleratorcore.setup.AcceleratorCoreSystemSetup& # These define sets of components that can fit into similar slots $wideContent=CMSImageComponent,SimpleBannerComponent,BannerComponent,ImageMapComponent,RotatingImagesComponent,ProductCarouselComponent,CMSParagraphComponent $narrowContent=ProductFeatureComponent,CategoryFeatureComponent,CMSImageComponent,SimpleBannerComponent,BannerComponent,ImageMapComponent,RotatingImagesComponent,ProductCarouselComponent,CMSParagraphComponent |

This defines some macro definitions namely the catalog version to use and the class used for loading resources from the classpath in **FileLoaderValueTranslator** by calling **getResourceAsStream**. Besides, component types fitting into the wide and narrow content slots are assigned.

##### PageTemplate Definition

For better readability, the **PageTemplate** definition is split up into two steps:

* Name, UID and **frontendTemplateName** definition.
* Velocity script used for displaying the template in the WCMS Cockpit.

|  |
| --- |
| INSERT\_UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];name;frontendTemplateName;active[default=true] ;;LandingPage2Template;Landing Page 2 Template;landingPage2  # Add Velocity templates that are in the CMS Cockpit. These give a better layout for editing pages # The FileLoaderValueTranslator loads a File into a String property. The templates could also be inserted in-line in this file. UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];velocityTemplate[translator=de.hybris.platform.acceleratorcore.setup.FileLoaderValueTranslator] ;;LandingPage2Template;$jarResource/acceleratorcore/import/cmscockpit/structure-view/structure\_landingPage2Template.vm |

##### ContentSlotName Definition

Content slot names are defined according to the wireframe definition in **Landing Page 2** layout:

|  |
| --- |
| INSERT\_UPDATE ContentSlotName;name[unique=true];template(uid,$contentCV)[unique=true][default='LandingPage2Template'];validComponentTypes(code) ;SiteLogo;;CMSImageComponent,BannerComponent ;HeaderLinks;;CMSLinkComponent,CMSParagraphComponent ;MiniCart;;MiniCartComponent ;NavigationBar;;NavigationBarComponent ;Section1;;$wideContent ;Section2A;;$narrowContent ;Section2B;;$narrowContent ;Section2C;;$wideContent ;Section3;;$wideContent ;Section4;;$narrowContent ;Section5;;$wideContent ;Footer;;CMSLinkComponent,CMSParagraphComponent,FooterComponent |

##### Page Definition

Finally in the **acceleratorcore** extension, the CMS page is defined:

|  |
| --- |
| INSERT\_UPDATE ContentPage;$contentCV[unique=true];uid[unique=true];name;masterTemplate(uid,$contentCV);defaultPage[default='true'];approvalStatus(code)[default='approved'];homepage[default='true'] ;;homepage;Homepage;LandingPage2Template |

The remaining steps are accomplished in cms-content.impex in the **acceleratorsampledata** extension.

##### ContentSlot Definition

First, content slots are defined using the document ID syntax meaning the referenced components have to be defined in the same file. Alternatively, components could also be referenced by UID.

|  |
| --- |
| **See Also**   * [ImpEx Syntax](#163) |

|  |
| --- |
| # commonly used content slots INSERT\_UPDATE ContentSlot;$contentCV[unique=true];uid[unique=true];name;active;cmsComponents(&componentRef) ;;SiteLogoSlot;Default Site Logo Slot;true;SiteLogoImage ;;HeaderLinksSlot;Header links;true;contactInfo ;;MiniCartSlot;Mini Cart;true;MiniCart ;;NavigationBarSlot;Navigation Bar;true;DigitalCamerasBarComponent,FilmCamerasBarComponent,HandheldCamcordersBarComponent,PowerSuppliesBarComponent,FlashMemoryBarComponent,CameraAccessoriesBarComponent,SpecialOffersBarComponent ;;FooterSlot;Footer;true;FooterComponent  # content slots used for the home page INSERT\_UPDATE ContentSlot;$contentCV[unique=true];uid[unique=true];name;active;cmsComponents(&componentRef) ;;Section1Slot-Homepage;Section1 Slot for Homepage;true;BrandsGaloreBanner ;;Section2ASlot-Homepage;Section2A Slot for Homepage;true;CanonPowershotBannerComponent,SonyCybershotBannerComponent,KodakEasyshareBannerComponent,SonyDSCBannerComponent ;;Section2BSlot-Homepage;Section2B Slot for Homepage;true;CamcordersBanner,LensesBanner ;;Section2CSlot-Homepage;Section2C Slot for Homepage;true;ElectronicsHomepageCarouselComponent ;;Section3Slot-Homepage;Section3 Slot for Homepage;true;ElectronicsHomepageProductCarouselComponent,PurchasedCategoryCrossSelling ;;Section4Slot-Homepage;Section4 Slot for Homepage;true;DigitalCompactCamerasBannerComponent,DigitalSLRCamerasBannerComponent,MemoryBannerComponent,AccessoriesBannerComponent ;;Section5Slot-Homepage;Section5 Slot for Homepage;true;NextDayDeliveryBanner |

##### ContentSlotForTemplate Definition

In the next step, content slots are assigned to a template in a previously named position. Typically, content slots that can be reused across multiple pages are assigned to the page template:

|  |
| --- |
| INSERT\_UPDATE ContentSlotForTemplate;$contentCV;uid[unique=true];position[unique=true];pageTemplate(uid,$contentCV)[unique=true][default='LandingPage2Template'];contentSlot(uid,$contentCV)[unique=true];allowOverwrite ;;SiteLogo-LandingPage2;SiteLogo;;SiteLogoSlot;true ;;HomepageLink-LandingPage2;HomepageNavLink;;HomepageNavLinkSlot;true ;;NavigationBar-LandingPage2;NavigationBar;;NavigationBarSlot;true ;;MiniCart-LandingPage2;MiniCart;;MiniCartSlot;true ;;Footer-LandingPage2;Footer;;FooterSlot;true ;;HeaderLinks-LandingPage2;HeaderLinks;;HeaderLinksSlot;true |

##### ContentSlotForPage Definition

Finally, specific content slots are assigned to the page:

|  |
| --- |
| INSERT\_UPDATE ContentSlotForPage;$contentCV[unique=true];uid[unique=true];position[unique=true];page(uid,$contentCV)[unique=true][default='homepage'];contentSlot(uid,$contentCV)[unique=true] ;;Section1-Homepage;Section1;;Section1Slot-Homepage ;;Section2A-Homepage;Section2A;;Section2ASlot-Homepage ;;Section2B-Homepage;Section2B;;Section2BSlot-Homepage ;;Section2C-Homepage;Section2C;;Section2CSlot-Homepage ;;Section3-Homepage;Section3;;Section3Slot-Homepage ;;Section4-Homepage;Section4;;Section4Slot-Homepage ;;Section5-Homepage;Section5;;Section5Slot-Homepage |

# Getting Started with the hybris Multichannel Accelerator

The hybris Multichannel Accelerator enables you to easily build a multichannel e-Commerce solution. It has all the functionality and business tools necessary to create an engaging customer experience, improve conversion and simplify management. The hybris Multichannel Accelerator is built on the hybris Multichannel Suite. Below you can find the information how to start the successful hybris Multichannel Accelerator implementation.

* [1. Sample Store Set-Up](#422)
* [2. Source Code Adaptation](#423)
* [3. Product Catalog Data Modeling](#424)
* [4. Sample Data Scripts Creation](#425)

|  |
| --- |
| **About this Document**  This document provides information on the how to start with the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, software architects, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Child Pages**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **See Also**   * [hybris Multichannel Accelerator - End User Guide](#426) * [hybris Multichannel Accelerator - Delivery Framework Guide](#427) * [hybris Multichannel Accelerator - Functional Specification Guide](#428) * [hybris Multichannel Accelerator - Integration Guide](#429) |

## 1. Sample Store Set-Up

First, you should setup the sample stores to see what you have as a starting point

Start by running up the accelerator with its **acceleratorsampledata** extension included. This will be very useful as a reference point and will ensure that there are no environment/setup issues in your development environment. Be sure to try out all of the functionality of the web store.

Follow the [Multichannel Accelerator Quick Installation Guide](#430) to do this.

Read through all the documentation about the accelerator and identify the areas of the accelerator code base the documentation refers. This will help you to familiarize with all the available technical features as well as where they are implemented in the code base.

## 2. Source Code Adaptation

Use **modulegen** and **extgen** to make a copy of the accelerator extensions to a version using your project's naming as a starting point for adapting the source code.

Accelerator is delivered completely as source code. You can edit the source code directly but your project would probably prefer some kind of project specific package naming. Therefore we have provided the module generation functionality with **extgen** so you are able to quickly create a copy of the accelerator source code but with your projects naming.

See the [Customizing the hybris Multichannel Accelerator with Extgen and Modulegen](#9) document for more information.

Throughout the rest of the documentation **acceleratorcore**, **acceleratorfacades**, **acceleratorstorefront** and **acceleratorcockpits** are interchangeable with your project specific extensions you have generated using **extgen** and **modulegen**.

## 3. Product Catalog Data Modeling

In this step you should identify and apply any necessary changes to your product catalog model.

It is always a good milestone to have your customers products available for sale on a web store, regardless of web store branding and complete functionality. Modeling your customers products will enable you to identify any gaps that might be present in the out of the box data model shipped with the Multichannel Accelerator extensions and the rest of the Multichannel Suite. Deciding which additional attributes need to be exposed on the Storefront will also help you decide whether you need to customize the out of the box Data Objects provided in the Facade Layer. You'll also identify which attributes need to be indexed for Search and Navigation purposes.

This process will also help you to identify how Product Data is supposed to be provided to the system and therefore identify any integration points such as data feeds from third-party ERP systems. It may be a lot of additional attributes are simply only necessary for reference in the Product Content Management tools or for back-office reporting.

During this stage you will do many of the following steps in the suggested order :

1. Make changes to **acceleratorcore** extension items.xml file.
2. Create some basic impex scripts to create you product catalog and catalog versions.
3. Create a basic category taxonomy using impex .
4. Create a classification structure using impex, if applicable.
5. Extend and customize facades, converters, populators and data objects defined in **commercefacades** in the projects **accelectorfacades** extension.
6. Configure a basic SOLR Index using impex.

|  |
| --- |
| **See Also**  **Product Catalog Modeling**   * [Catalog Guide](#431) * [Structure Your Collection with Products and Categories](#432) * [items.xml](#45) * [Classification Guide](#433).   **Search and Navigation**   * [SolrFacetSearch - Configuration Guide](#351). * [Solr Facet Search Configuration in the hybris Multichannel Accelerator](#347)   **Facades**   * [commercefacades Extension - Technical Guide](#95) |

## 4. Sample Data Scripts Creation

now you should create some sample data scripts that run whenever the system is initialized.

In the early stages of development the data model is likely to be changed a lot. This will result in many re-initializations of each developers database. For productivity purposes it is highly desirable for the system to automatically execute scripts that load sample content as part of a project extensions project data. This saves developers from constantly having to add in sample products and/or WCSM content to test the functionality they have developed. It is of course also useful to have a set of automated data provisioning scripts in place if you wish to provide a system in a ready state for QA or have a continuous integration system that runs smoke tests.

The sample Multichannel Accelerator storefronts are all built using impex scripts that are triggered with switches whilst running project data through on the update running system page. You can use these scripts and templates/guides to code up your own scripts that all developers can run during a system initialization.

You may also want to setup a series of essential scripts that need to be run in all environments including even production. However it is advised to keep these separate from Sample Data and the Multichannel Accelerator provided a clear example of how to do this. Sample data scripts can be separated into a different extension to make it very easy to remove, that is simply don't include the extension in your localextensions.xml .

|  |
| --- |
| **See Also**   * [Initialization and Update Documentation](#434) * [Essential and Project Data in the hybris Multichannel Accelerator](#167) * [acceleratorsampledata Extension - Technical Guide](#101) * [ImpEx Import for Essential and Project Data](#435) |

## Multichannel Accelerator Quick Installation Guide

This document runs you through how to setup the hybris Multichannel Accelerator including its sample data demonstration storefronts on your local machine. It is an appendix to the [hybris Multichannel Suite Quick Installation Guide](#436) adding additional setup instructions to install the hybris Multichannel Accelerator with its sample storefronts.

* [Basic Installation](#437)
* [Accessing the Sample Storefronts](#438)
  + [Hosts File Approach - Recommended](#439)
  + [Site Parameter Approach](#440)
  + [Setting Up Email](#441)
  + [Setting Up Store Locator](#442)

|  |
| --- |
| **About this Document**  This document gives a quick overview of how to download, set up, and start the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, software architects, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [Quick Installation](#436) * [Adding Extensions](#443) * [hybris Multichannel Accelerator Extensions](#8) |

### Basic Installation

1. Complete the following steps from the [hybris Multichannel Suite Quick Installation Guide](#436)
   * [Review System Requirements](#444)
   * [Download and Unpack the Package](#445)
   * [Build the hybris Multichannel Suite](#446)
2. Add the subset of Multichannel Commerce extensions that the accelerator requires to your HYBRIS\_BIN\_DIR /config/localextensions.xml file:

|  |
| --- |
| <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/basecommerce"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/btg"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/btgcockpit"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/cscockpit"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/cms2"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/cms2lib"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/cmscockpit"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/customerreview"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/fulfilmentprocess"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/payment"/>  <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/productcockpit"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/promotions"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/solrfacetsearch"/>  <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/ticketsystem"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-hybris/voucher"/> |

1. Add the hybris Multichannel Accelerator-related extensions to the localextensions.xml file:

|  |
| --- |
| <extension dir="${HYBRIS\_BIN\_DIR}/ext-accelerator/commerceservices"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-accelerator/commercefacades"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-accelerator/acceleratorfacades"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-accelerator/acceleratorcore"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-accelerator/acceleratorsampledata"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-accelerator/acceleratorstorefront"/> <extension dir="${HYBRIS\_BIN\_DIR}/ext-accelerator/acceleratorcockpits"/> |

More context regarding enabling functionality with extensions can be found in the [hybris Multichannel Suite Full Installation Documentation](#447), section **Set Up the Functional Range**.

1. Perform **ant clean all<ENTER>** from the **platform** directory again as previously outlined in the [hybris Multichannel Suite Quick Installation Guide](#446), section **Build the hybris Multichannel Suite**. This ensures all the new extensions are compiled and included into your build.
2. Complete the final two steps from the [hybris Multichannel Suite Quick Installation Guide](#436)
   * [Start the hybris Server](#448)
   * [Initialize the hybris Multichannel Suite](#449)

### Accessing the Sample Storefronts

To access the sample storefronts you must add a localhost mapping to your computers hosts file. The hosts file approach requires this extra setup step on your computer but importantly it more accurately represents how Storefronts will most likely be organized in terms of domain URLs in a production environment.

|  |
| --- |
| **See Also**   * [Wikipedia on hosts file](http://en.wikipedia.org/wiki/Hosts_(file)) * [Just One Storefront](#450) |

#### Hosts File Approach - Recommended

You should add the following lines to your file once you have located it. Note that system admin however often restrict access to this file, at least write access, so you may not have the necessary access rights to edit the file.

|  |
| --- |
| 127.0.0.1 apparel.uk.local apparel.de.local electronics.local |

You can then access the sites as follows:

* <http://apparel.uk.local:9001/acceleratorstorefront>
* <http://apparel.de.local:9001/acceleratorstorefront>
* <http://electronics.local:9001/acceleratorstorefront>

#### Site Parameter Approach

There is an alternative shortcut approach of using a 'site' parameter. This however only allows you to visit one site in the same browser session, otherwise you'll run into session sharing issues when you visit a second site caused by the fact that your domain is the same and the application server will not provide you with a second fresh session. This approach is not recommended but is available as a convenience feature.

* <http://localhost:9001/acceleratorstorefront?site=apparel-uk&clear=true>
* <http://localhost:9001/acceleratorstorefront?site=apparel-de&clear=true>
* <http://localhost:9001/acceleratorstorefront?site=electronics&clear=true>

|  |  |
| --- | --- |
|  | **Implementation Note**  If it is not possible to use the hosts file approach, you should consider creating a servlet filter and wrapping the *HttpServletRequest* object in a bespoke *HttpServletRequestWrapper* sub-class which uses a bespoke *HttpSession* to internally segregate all session attributes by a site-id key. The filter should essentially setup the request wrapper and initialise the current active site using the site-id parameter from the request or if not available from the session. This code however would never be recommended for a production setup. |

#### Setting Up Email

The **MailUtils** Utility from the hybris Platform **core** extension is used to send emails. Typical configuration parameters to place in your local.properties file are as follows:

|  |
| --- |
| mail.smtp.server=<your\_mail\_server> mail.smtp.port=25 mail.smtp.user=<your\_user> mail.smtp.password=<your\_users\_password> |

You need some kind of mail server installed on your local machine or accessible on a shared network to receive emails.

Links and Images in emails require a fully qualified path to resolve the resource correctly in the storefront. To setup email links to work correctly, configure the **UrlResolutionService** in your local.properties . For more information on the **UrlResolutionService**, go to the [commerceservices Extension - Technical Guide](#190), section **Site Base URL Resolution Service**.

Assuming you are using the hosts file approach, the following configuration properties need to be declared in the local.properties file:

|  |
| --- |
| website.apparel-uk.http=http://apparel.uk.local:9001/acceleratorstorefront website.apparel-uk.https=https://apparel.uk.local:9002/acceleratorstorefront website.apparel-de.http=http://apparel.de.local:9001/acceleratorstorefront website.apparel-de.https=https://apparel.de.local:9002/acceleratorstorefront website.electronics.http=http://electronics.local:9001/acceleratorstorefront website.electronics.https=https://electronics.local:9002/acceleratorstorefront  media.apparel-uk.http=http://apparel.uk.local:9001 media.apparel-uk.https=https://apparel.uk.local:9002 media.apparel-de.http=http://apparel.de.local:9001 media.apparel-de.https=https://apparel.de.local:9002 media.electronics.http=http://electronics.local:9001 media.electronics.https=https://electronics.local:9002 |

#### Setting Up Store Locator

If you are using a different domain, then you need to generate API keys and configure the system for the Store Locator maps feature to work on the storefront. More information can be found in the [Setting Up the Store Locator in the hybris Multichannel Accelerator](#366), section **Storefront API Key Configuration**.

## Customizing the hybris Multichannel Accelerator with Extgen and Modulegen

The **modulegen** is a tool delivered with the hybris Multichannel Suite that can be used to clone and adapt modules to meet the client requirements from the very beginning. It enables you to create clones of the extensions that will be renamed with the given schema, also package name can be changed, extension class prefix and some other extension properties. The **modulegen** is run as **ant** task, so it requires **ant** libraries to be present in the system. The Java runtime libraries also need to be installed.  
The hybris Multichannel Suite comes with an extension generator system called **extgen**, short for extension generator. Using **extgen**, you can create new extensions based on extension templates. For more information about the **extgen** tool, go to the [Creating a New Extension](#72) document.

* [Modulegen Configuration and Usage](#451)
* [Creating customised accelerator system using modulegen](#452)

|  |
| --- |
| **About this Document**  This document presents you the tools for module and extension generation.  **Audience**: Consultants, developers, software architects, technical deciders  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * [About Extensions](#453) * [Creating a New Extension](#72) * [hybris Multichannel Accelerator Extensions](#8) |

### Modulegen Configuration and Usage

The **modulegen** configuration is divided into two parts.

First one is in the platform/extgen/project.properties file and is represented with following properties:

| **Property** | **Descripton** |
| --- | --- |
| **modulegen.module.list** | This is a comma separated list of the available modules that can be selected in the **modulegen** task. |
| **modulegen.module.default** | This is the preselected module that will be chosen if user entered no value for module. |
| **modules.<module name>.root** | This is the root directory that should contain all source extensions that given module contains. In the process all these extensions will be cloned and adapted. |
| **modules.<module name** | This is a comma separated list of all source extensions of the given module. |

Second part of configuration is placed in the extgen.properties file of each source extension. It contains following properties:

| **Property** | **Descripton** |
| --- | --- |
| **YEXTNAME\_TOKEN** | This is token that is used during cloning process to rename extension with the value entered by user. |
| **YMODULE\_TOKEN** | This is token that is used during cloning process to give new name to the module. |
| **YMODULE\_PACKAGE\_ROOT** | This property is added to the root of the cloned extension package name for all classes. |
| **YPACKAGE\_TOKEN** | This token in package name will be replaced in the cloned extensions with new package path. |
| **YMANAGER\_TOKEN** | This manager token will be replaced in the cloned version with new value entered by user. |
| **YCLASSPREFIX\_TOKEN** | The class prefix token will be replaced in the cloned version with the new value entered by user. |
| **YGENERATED\_TOKEN** | The generated prefix will be used for classes that will be generated during extension build process. |

When **modulegen** ant task is started it requires user to enter some values:

1. **Name of new extension module** - user here should enter new module name. The default value is set to **training**.
2. **Base package name** - all packages defined with the token in the properties file will be renamed to the new value.
3. **Template for generation** - user needs to select among the templates defined in the platform/extgen/project.properties file.

When information is gathered and validated the process begins. Extensions are cloned with given parameters changed and after whole process is completed proper message is displayed that informs us what to do next. We need to add our cloned extensions to localextensions.xml file and rebuild the system.

### Creating customised accelerator system using modulegen

Using the **modulegen** tool it is easy to create customized version of accelerator module. There are a few steps that need to be fulfilled.

1. First, you need to take care of the platform/extgen/project.properties file and make sure it contains required information for creating clone of all required extensions. It may look like this:

|  |
| --- |
| modulegen.module.list=accelerator,acceleratordata  modulegen.module.default=accelerator  modules.accelerator.root=${HYBRIS\_BIN\_DIR}/ext-accelerator modules.accelerator=acceleratorcore,acceleratorfacades,acceleratorstorefront,acceleratorcockpits modules.acceleratordata.root=${HYBRIS\_BIN\_DIR}/ext-accelerator modules.acceleratordata=acceleratorcore,acceleratorfacades,acceleratorstorefront,acceleratorcockpits,acceleratorsampledata |

In the example above two new templates are defined - one with the **acceleratorsampledata** extension and one without.

1. Modify, if you need to, the extgen.properties in the all extensions that will be cloned.  
   Sample extgen.properties file from the **acceleratorcore** extension may look like this:

|  |
| --- |
| YEXTNAME\_TOKEN=acceleratorcore YMODULE\_TOKEN=accelerator YMODULE\_PACKAGE\_ROOT=ac YPACKAGE\_TOKEN=de.hybris.platform.acceleratorcore YMANAGER\_TOKEN=AcceleratorCoreManager YCLASSPREFIX\_TOKEN=AcceleratorCore YGENERATED\_TOKEN=Generated |

1. Start the ant task called **modulegen**. In the example below, you can see the script with the following information:

|  |
| --- |
| modulegen:  [input]  [input] Please choose the name of your module extension. It has to start with a letter followed by letters and/or numbers.  [input] Press [Enter] to use the default value [training] megastore  [input]  [input] Please choose the base package name of your extensions. It has to fulfill java package name convention. Each extension in the module will add its name to this package.  [input] Press [Enter] to use the default value [org.training] org.megastore  [input]  [input] Please choose a template for generation.  [input] Press [Enter] to use the default value ([accelerator], acceleratordata) acceleratordata |

And after a few minutes we have a copy of customized accelerator project. In our example we got:

* + All cloned extensions are placed in the **megastore** directory.
  + The extensions are renamed as follows:
    - **megastoreacceleratorcockpits**
    - **megastoreacceleratorfacades**
    - **megastoreacceleratorsampledata**
    - **megastorecore**
    - **megastorestorefront**.
  + Package name changes and all other renaming was applied as they were configured in properties files.

1. When script completes work, at the end it displays information what to do next:

|  |
| --- |
| [echo] Next steps: [echo] [echo] 1) Add your extension to your D:\src\trunk\config/localextensions.xml [echo] [echo] <extension dir="(...)/bin/custom/megastore/megastorecore"/> [echo] <extension dir="(...)/bin/custom/megastore/megastoreacceleratorfacades"/> [echo] <extension dir="(...)/bin/custom/megastore/megastorestorefront"/> [echo] <extension dir="(...)/bin/custom/megastore/megastoreacceleratorcockpits"/> [echo] <extension dir="(...)/bin/custom/megastore/megastoreacceleratorsampledata"/> [echo] [echo] 2) Make sure the applicationserver is stopped before you build the extension the first time. [echo] [echo] 3) Perform 'ant' in your hybris/platform directory. [echo] [echo] 4) Restart the applicationserver |

# Storefront and Catalog Modelling in the hybris Multichannel Accelerator

The intention of this document is to give the reader a fairly high-level overview of how the accelerator and its sample stores are configured and modelled with respect to Storefront setup and distribution of customer data, product and content catalogs between multiple Storefronts. A multiple Storefront setup in the document is deemed to be a single hybris deployment that serves multiple web sites, rather than separate hybris deployments for each web site.

This document focusses only on the configuration and modelling aspects used by the 3 sample stores that ship with the **acceleratorsampledata** extension. These configuration aspects are effectively the components in the hybris system data model that can be tweaked to alter a Storefront's offering and behaviour. As with other hybris modules, its possible to customise and adapt all of this, however the accelerator is modelled and to meet more 'typical' B2C Storefront requirements.

The **acceleratorcore** and **acceleratorsampledata** extensions contain all the configuration scripts for multiple Storefronts and are for developers the ultimate reference.

* [Sample Storefront Diagram](#454)
* [Storefront Setup](#455)
  + [CMS Site](#456)
  + [Base Store](#457)
* [Catalog Data](#458)
  + [Content Catalog](#459)
  + [Product Catalog](#460)
    - [Products](#461)
    - [Variants](#462)
    - [Categorisation](#463)
    - [Pricing](#464)
  + [Classification System](#465)
  + [Solr Facet Search](#466)
* [Customer Data](#467)
* [Order Data](#468)
* [Stock](#469)

|  |
| --- |
| **About this Document**  **Audience**:  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

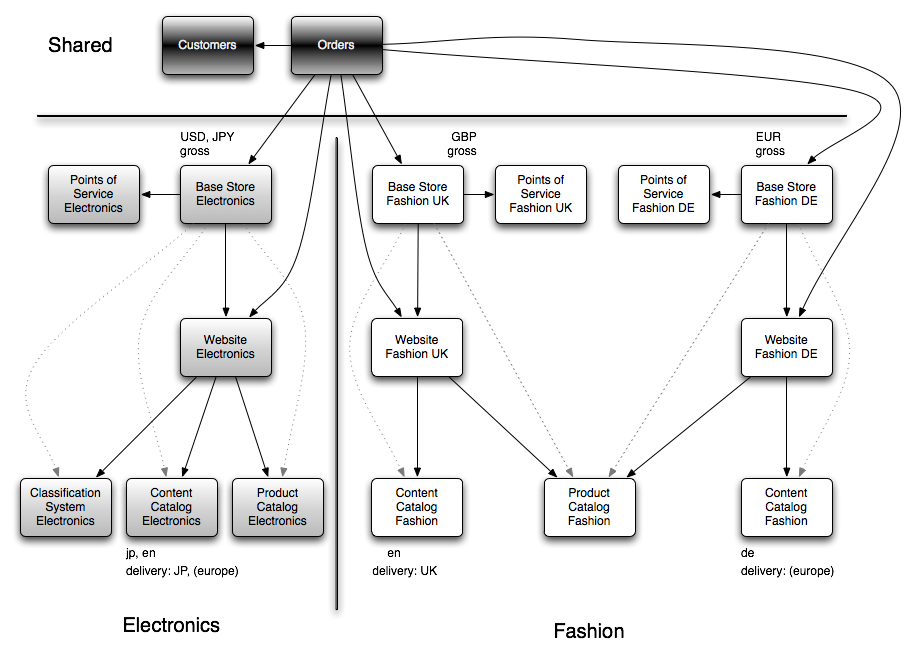
|  |
| --- |
| **Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **Parent Page's Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **See Also**   * [Essential and Project Data in the hybris Multichannel Accelerator](#167) * [acceleratorcore Extension - Technical Guide](#81) * [acceleratorsampledata Extension - Technical Guide](#101) |

## Sample Storefront Diagram

Lets start with a high-level diagram that visualises how the **acceleratorsampledata** Storefront and Customer data is separated.



## Storefront Setup

Each Storefront will have a counterpart CMS Site and Base Store. These are the central objects for configuring a new website.

|  |
| --- |
| **See Also**   * [commerceservices Extension - Technical Guide](https://wiki.hybris.com/display/acc/commerceservices+Extension+-+Technical+Guide#commerceservicesExtension-TechnicalGuide-StorefrontSetup) section **Storefront Setup** |

### CMS Site

A CMS Site configuration options include :

* The Base Store for the Storefront (see above)
* URL Pattern Regular Expressions that enable the **acceleratorstorefront** Web Application to decide which Storefront the user is requesting to visit
* The Content Catalog for the WCMS Content. Usually One Content Catalog serves just one CMSSite but it is possible to share between multiple CMS Site (for example a storefront wanting a different Theme or Base Store).
* The Product Catalog (this also implicitly configures which SOLR index to use on the Storefront)
* The Classification System (optional)
* The Theme (i.e. the CSS skin of the Storefront)
* The Homepage for the Storefront
* The Corresponding system Java Locale when a user selects a particular language (important for formatting currencies and numbers).
* A flag to enable or disable the entire Storefront

|  |
| --- |
| **See Also**   * [cms2 Extension - Technical Guide](https://wiki.hybris.com/display/release4/Data+Model+Overview#DataModelOverview-BaseStore) section **Data Model Overview - BaseStore** * [cms2 Extension - Technical Guide](https://wiki.hybris.com/display/release4/Data+Model+Overview#DataModelOverview-CMSSite) section **Data Model Overview - CMSSite** |

### Base Store

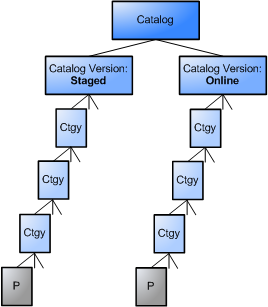
A Base Store configuration options include :

* The Currencies which are supported, the first currency in the list is deemed the default.
* The Points Of Service (aka Bricks and Mortar Stores) that can be found in the Store Locator.
* Specify the distance unit for the Store Locator (KM or Miles)
* The user tax group to allow for products to use different sales tax rules per Storefront
* Whether a Storefront shows prices net or gross of sales tax

|  |
| --- |
| **See Also**   * [BaseStore Type](#470) * [Using Store Locator Service - Tutorial](#471) |

## Catalog Data

Catalogs are used to store Product and WCMS content. Whilst setting up the CMS Site, you will link the Site to the appropriate Catalog's that hold the content for the Storefront. **accelerator** examples Catalogs which have one or more Catalog Version's to enable a preview to live publishing workflow. All sample data with the **accelerator** come with a recommended Staged and Online Catalog Version for each Catalog.



### Content Catalog

A Content Catalog stores all the WCMS pages, components and templates. All content is added first to the Staged Catalog version and then published to Online either piecemeal or in its entirety using the WCMS Cockpit or some automated synchronisation jobs.

* The languages stored on the Content Catalog Version identify which languages are available on the Storefront.
* The territories stored on the Content Catalog Version are used to identify which Zoned Delivery Modes (delivery options) will apply for the Storefront.

|  |
| --- |
| **See Also**   * [cms2 Extension - Technical Guide](https://wiki.hybris.com/display/release4/Data+Model+Overview#DataModelOverview-ContentCatalog) section **Data Model Overview - Content Catalog** |

### Product Catalog

A Product Catalog stores all product and category data. You assign your products and category data to one or more Catalog Version's to enable a preview-to-live publishing workflow. Staged content can then published to Online either piecemeal or in its entirety using the Product Cockpit or some automated synchronisation jobs.

|  |
| --- |
| **See Also**   * [Catalog Guide](#431) |

#### Products

Out of the box the accelerator supports the following product models without the need for further code or data model modification.

* 'hybris Vanilla' Products (standard hybris product model)
* 'hybris Vanilla' Products with Classification (standard hybris product model including classification system)
* Variants (base products with variant products)
* Variants with Classification (not exampled in **acceleratorsampledata** but supported)
* Apparel Product (extended Apparel product model using Variants)

|  |
| --- |
| **See Also**   * [Product Modeling](#472) * [items.xml](#45) * [acceleratorcore Extension - Technical Guide](https://wiki.hybris.com/display/acc/acceleratorcore+Extension+-+Technical+Guide#acceleratorcoreExtension-TechnicalGuide-ApparelProducts) section **Apparel Products** * [How To Extend the ProductFacade - Tutorial](#96) |

Projects are of course fully able to modify the data model in their own items.xml as well as customise facades to extend the template code provided by the accelerator to support a product data model tailored for the project requirements.

#### Variants

Variants are products that differ in some aspect from one another, but are based on the same basic model sharing many attributes. They are represented with a tree-style hierarchy where only the leaf nodes are purchasable (i.e. a complete SKU). The root (or base product) contains attributes common to the entire hierarchy or branch. Typically on a front-end the user starts at a base product or a branch product and applies further configuration of the product (such as picking a color and/or size) to arrive at a selected purchasable leaf node which can then be added to cart.



The **accelerator** supports the following variant models:

* Single depth or Double depth Product Variants are supported. Each depth can have multiple variant attributes.
* Special Apparel Data Model using Double depth Product Variants is available in the **acceleratorcore** extension.

#### Categorisation

Categorising products into a clean taxonomy hierarchy is fully supported. Adding other category hierarchies and linking either the complete hierarchy or a subset of it to facets in a Solr Search Index is also supported and exampled. The **acceleratorsampledata** includes both a taxonomy category hierarchy (i.e. Root->Cameras->Digital Cameras->SLR) as well as a brand hierarchy (i.e. brands->Nikon) to example this.

|  |
| --- |
| **See Also**   * [Structure Your Collection with Products and Categories](#432) |

Category facets can also be used to trigger merchandised landing pages in the WCMS.

#### Pricing

Pricing uses the **europe1** pricing extension. This includes a vast number of configuration options for pricing. The options demonstrated in the **acceleratorsampledata** extension are listed below.

* Multiple prices can be stored for a product in different currencies.
* If two currencies share the same base currency it is possible for automatic currency conversion to be performed in the event of a price being unavailable for a particular Storefront currency.
* A user tax group can be set on a tax row, this can be matched to the store tax group so different tax rates can be applied for the same product but on different Storefronts.

|  |
| --- |
| **See Also**   * [europe1 Pricing System Guide](#473) |

### Classification System

A Classification System can be added to enable the capability of assigning dynamic attributes to Product Categories from the counterpart Product Catalog. Classification Systems tend to be good fits for stores that either sell a lot of technical products with a vast number of attributes that would be too much effort to model, or stores with a very disparate taxonomy of products (often also technical product stores).

* Classification Attributes can be assigned to a Solr Search Index. This includes, facets, attributes displayed in search results, attributes in a free text search and result sorting. Using classification attributes, it's possible to create new taxonomy in a Solr Search Index without the need for a system restart.

|  |
| --- |
| **See Also**   * [Classification Guide](#433) |

### Solr Facet Search

Solr Facet Search Indexes are bound to the Online product catalog in all of the **accelerator** sample stores. The system determines which search index to use based on the active product catalog versions for the current site.

## Customer Data

Customer accounts are shared across all Storefronts. A user who registers in one Storefront will be registered in all other Storefronts. An address book is saved with the customer, addresses that don't reside in the current active storefront supported territories are filtered out of the box. Saved payment cards are also supported, however out of the box the accelerator uses a Payment Service Provider subscription service to store the full credit card number and only a masked number is saved in the "profile".

## Order Data

Orders *can* be shared across multiple Storefronts but the CMS Site and Base Store will be captured on the order when the order is placed. Services in the **commerceservices** extension are available to return only orders from the current Storefront.

* The order fulfilment process runs globally, but the CMS Site and Base Store are available if required to be used to effect behaviour of fulfilment tasks (demonstrated in the accelerator with the rendering of Storefront specific order confirmation and dispatch emails).

## Stock

It is possible to turn off stock check functionality by simply removing any default warehouse from the system. If you don't turn off stock check functionality the system will assume all products without Stock Levels are out of stock.

|  |
| --- |
| **See Also**  **accelerator**   * [Stock Management in the hybris Multichannel Accelerator](https://wiki.hybris.com/display/acc/Stock+Management+in+the+hybris+Multichannel+Accelerator#StockManagementinthehybrisMultichannelAccelerator-StockLevels) section **Stock Levels**   **basecommerce**   * [Stock Service - Technical Description](#394) |

# Developer Trails

|  |
| --- |
| **About this Document**  **Audience**:  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **Parent Page's Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **See Also**   * . |

## Add New CMS Component Trail

### Motivation

The hybris Web Content Management Cockpit provides a central facility for managing content. Content is laid out in Pages via CMS Components added to ContentSlots. The goal of this trail is to show how to create a custom CMS component within hybris Accelerator, add it to a page and view that page.

The example component that this trail will build aims to provide some text together with a picture. This trail will show how to create all the neccessary artifacts in order to add this component to a hybris Accelerator page.

#### Required/Helpful Reading

[hybris Type System](#45)

[Accelerator CMS Components](#477)

[Configuration of the Cockpit UI](#478)

[How to Import Cockpit Configuration - Tutorial](#162)

[ImpEx Syntax](#163)

[hybris Access Rights](#479)

[Spring MVC 3](http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html)

#### About Components

CMS components in hybris are composed of:

* A hybris Item extending AbstractCMSComponent, defined in the items.xml file for your extension similarly to any other hybris Item
* A Spring MVC Controller extending AbstractCMSComponentController, or using the DefaultCMSComponentController
* A JSP used to render the component on a page, with HTML, CSS, Tag Libraries as required

The CMSCockpit can be customised for the component via XML configuration:

* editorArea\_<typeName>.xml: configuration for CMSCockpit Editor Area
* contentEditor\_<typeName>.xml: configuration for LiveEdit mode and in-line editing
* wizardConfig\_typeName>.xml: wizard configuration if a component is added via a wizard dialogue

### Tasks

#### Creating the hybris Item

This is achieved by editing the items.xml provided within your extension. For the Accelerator, the component will be added to the acceleratorcore extension.

1. Create a new type extending the CMSParagraphComponent in the file "acceleratorcore/resources/acceleratorcore-items.xml". For a detailed description of the configuration of a hybris Item see [items.xml Element Reference](#483).

|  |
| --- |
| <itemtype code="CMSMediaParagraphComponent" generate="true"  jaloclass="de.hybris.platform.acceleratorcore.jalo.cms2.components.CMSMediaParagraphComponent"  extends="CMSParagraphComponent" autocreate="true">  <description>It represents paragraph component with an additional media attribute.</description>  <attributes>  <attribute qualifier="media" generate="true" autocreate="true" type="localized:Media">  <persistence type="property" />  <description>Attribute that stores the localized media of the paragraph.</description>  </attribute>  </attributes> </itemtype> |

Table 34 acceleratorcore-items.xml

1. Add localized attribute names and descriptions for this Item in English to the file "acceleratorcore/resources/localization/acceleratorcore\_locales\_en.properties", and to other localisation files as required.

|  |
| --- |
| type.CMSMediaParagraphComponent.name=Media Paragraph Component type.CMSMediaParagraphComponent.description=It represents paragraph component with an additional media attribute. type.CMSMediaParagraphComponent.content.name=Content type.CMSMediaParagraphComponent.content.description= type.CMSMediaParagraphComponent.media.name=Media type.CMSMediaParagraphComponent.media.description= |

Table 35 acceleratorcore\_locales\_en.properties

#### Creating the Controller

The DefaultCMSComponentController offers enough functionality out of the box to support this component. It will add all attributes of the Component to the Spring MVC Model (see [Spring MVC 3](http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html)). By default it will return a View that maps to the file "cmsmediaparagraphcomponent.jsp" (based on the component name) in **acceleratorstorefront/web/webroot/WEB-INF/components/cms2**:

|  |
| --- |
| <div class="content">  <img src="${media.url}" alt="${media.altText}"/>  ${content} </div> |

Table 36 cmsmediaparagraphcomponent.jsp

Changes might also be added to the appropriate CSS files in acceleratorstorefront.

#### Content Editing

Once the system has been Built and Updated, the component will be available through the CMSCockpit to add to pages. In the Accelerator the components that can be added to any PageTemplate are restricted via Impex configuration.

##### Add to Valid Component Types

Modify the file "cms-content.impex" in **acceleratorcore/resources/acceleratorcore/import/site/electronics**:

|  |
| --- |
| $narrowContent=ProductFeatureComponent,CategoryFeatureComponent,CMSImageComponent,SimpleBannerComponent,BannerComponent,ImageMapComponent,RotatingImagesComponent,ProductCarouselComponent,CMSParagraphComponent,CMSMediaParagraphComponent |

Table 37 cms-content.impex

This will allow the component to be added to all ContentSlots restricted to this list of components. It could also be added to the list marked $wideContent.

##### Modify Access Rights

Access rights for the new component have to be defined for the user to be able to access the component in the CMS cockpit. This is achieved by adding this line to the file "cmscockpit-access-rights.impex" in **acceleratorcore/resources/acceleratorcore/import/cmscockpit**:

|  |
| --- |
| ;;;;CMSMediaParagraphComponent;+;+;+;+;+; |

Table 38 cmscockpit-access-rights.impex

This will grant certain rights to users of the cmsmanagergroup for our new component:

* read component data;
* update component data;
* create a new component instance;
* remove an existing component instance;
* change permissions.

For a detailed description of access rights in hybris see [Access Rights](#479).

##### Cockpit Configuration

It is possible to add customised configuations to the CMS cockpit for creating / editing the new component. All configuration files are added to "acceleratorcore\resources\acceleratorcore-config\cmsmanagergroup". All XML files in this directory will be collected and inserted during the ProjectData initialization stage, if set.

WCMS Page View for component is configured in the file "editorArea\_CMSMediaParagraphComponent.xml":

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <editor>  <group qualifier="General" visible="true" initially-opened="true">  <label lang="de">Stammdaten</label>  <label lang="en">General</label>  <property qualifier="CMSItem.name"/>  <property qualifier="CMSItem.catalogVersion" editor="shortListEditor"/>  <property qualifier="AbstractCMSComponent.visible"/>  </group>   <group qualifier="editorial" visible="true" initially-opened="false">  <label lang="de">Redaktionell</label>  <label lang="en">Editorial</label>  <property qualifier="CMSParagraphComponent.content" editor="wysiwyg"/>  <property qualifier="CMSMediaParagraphComponent.media" editor="simpleMediaReferenceSelector">  <parameter>  <name>mimeTypes</name>  <value>  image/jpeg;image/gif  </value>  </parameter>  <parameter>  <name>mimeTypes</name>  <value>  image/jpeg;image/gif  </value>  </parameter>  <parameter>  <name>allowCreate</name>  <value>true</value>  </parameter>  <parameter>  <name>restrictToCreateTypes</name>  <value>Media</value>  </parameter>  <parameter>  <name>celumMediaFormat</name>  <value>picture</value>  </parameter>  </property>   </group>   <group qualifier="visibility" visible="true" initially-opened="false">  <label lang="de">Context Visibility</label>  <label lang="en">Context Visibility</label>  <property qualifier="AbstractCMSComponent.restrictions">  <parameter>  <name>allowCreate</name>  <value>true</value>  </parameter>  </property>  <property qualifier="AbstractCMSComponent.onlyOneRestrictionMustApply"/>  </group>   <group qualifier="admin" visible="true" initially-opened="false">  <label lang="de">Administration</label>  <label lang="en">Administration</label>  <property qualifier="CMSItem.uid"/>  <property qualifier="AbstractCMSComponent.slots"/>  <property qualifier="AbstractCMSComponent.container"/>  <property qualifier="Item.pk" />  <property qualifier="Item.creationTime" />  <property qualifier="Item.modifiedtime" />  </group>   <custom-group  class="de.hybris.platform.cockpit.services.config.impl.UnassignedEditorSectionConfiguration"  qualifier="unassigned"  initially-opened="false"  visible="false">  <label lang="de">Andere</label>  <label lang="en">Other</label>  </custom-group> </editor> |

Table 39 editorArea\_CMSMediaParagraphComponent.xml

LiveEdit mode is supported by adding "contentEditor\_CMSMediaParagraphComponent.xml":

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <content-editor hideEmpty="true" hideReadOnly="true" groupCollections="false">  <template>  <![CDATA[  <table width="100%" style="margin:0px;padding:0px">  <tbody>  <tr>  <td colspan="2"><div style="height:6px"/></td>  </tr>  <tr>  <td colspan="2"><div class="contentEditorHeader" style="background-color:#C3C3C3;font-size:11px;padding:3px;"><b>$label</b></div></td>  </tr>  <tr>  <td style="width:900px"><cockpit code="property" value="CMSParagraphComponent.content"/></td>  <td><div style="height:6px"/></td>  </tr>  <tr>  <td style="width:900px"><cockpit code="property" value="CMSMediaParagraphComponent.media"/></td>  <td><div style="height:6px"/></td>  </tr>  </tbody>  </table>  ]]>  </template>  <custom-editors>  <property qualifier="CMSParagraphComponent.content" editorCode="wysiwyg">  <parameter>  <name>inline</name>  <value>true</value>  </parameter>  <parameter>  <name>fckToolbarConfiguration</name>  <value>  [['Bold', 'Italic','Underline','-','JustifyLeft','JustifyCenter','JustifyRight','JustifyFull','-','OrderedList','UnorderedList','-','Outdent','Indent','-','Table'],'/',['FontName','FontSize','TextColor','-','Source']]  </value>  </parameter>  <parameter>  <name>editorWidth</name>  <value>700px</value>  </parameter>  <parameter>  <name>editorHeight</name>  <value>200px</value>  </parameter>  <parameter>  <name>labelWidth</name>  <value>120px</value>  </parameter>  </property>  <property qualifier="CMSMediaParagraphComponent.media" editorCode="simpleMediaReferenceSelector">  <parameter>  <name>mimeTypes</name>  <value>image/jpeg;image/gif</value>  </parameter>  <parameter>  <name>allowCreate</name>  <value>true</value>  </parameter>  <parameter>  <name>restrictToCreateTypes</name>  <value>Media</value>  </parameter>  <parameter>  <name>imageHeight</name>  <value>80px</value>  </parameter>  <parameter>  <name>celumMediaFormat</name>  <value>picture</value>  </parameter>  </property>  </custom-editors> </content-editor> |

Table 40 contentEditor\_CMSMediaParagraphComponent.xml

Finally, the Wizard is configured via "wizardConfig\_CMSMediaParagraphComponent.xml":

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?>  <wizard-config showPrefilledValues="false" selectMode="true" createMode="true" displaySubtypes="true">  <displayed-properties>  <group qualifier="General" visible="true" initially-opened="true">  <label lang="en">General</label>  <label lang="de">General</label>  <property qualifier="CMSItem.name"/>  </group>  <group qualifier="Properties" visible="true" initially-opened="true">  <label lang="en">Properties</label>  <label lang="de">Eigenschaften</label>  <property qualifier="CMSParagraphComponent.content" editorCode="wysiwyg" />  <property qualifier="CMSMediaParagraphComponent.media" editorCode="simpleMediaReferenceSelector" />  </group>  </displayed-properties> </wizard-config> |

Table 41 wizardConfig\_CMSMediaParagraphComponent.xml

You should now be able to log on as cmsmanager and add your new component to a Page.

### Summary

In this trail you have learned how to add a new CMS component to hybris accelerator.

* Adding a new CMS component type to the type system (Model)
* Creating a JSP (View) and ComponentController (Controller)
* Configuring the CMSCockpit editor views

## Add New CMS Page Template Trail

### Motivation

The hybris Web Content Management Cockpit provides a central facility for managing content. CMS Pages are laid out according to their PageTemplate. The goal of this trail is to show how to create a PageTemplate within the hybris Accelerator.

#### Required/Helpful Reading

[hybris Type System](#45)

[Accelerator CMS Components](#477)

[Configuration of the Cockpit UI](#478)

[How to Import Cockpit Configuration - Tutorial](#162)

[ImpEx Syntax](#163)

[Spring MVC 3](http://static.springsource.org/spring/docs/3.0.x/reference/mvc.html)

#### About Page Templates

A PageTemplate in the Accelerator project has:

* a list of ContentSlots that are predefined for the Template, for the Accelerator this includes the MiniCart, Homepage Image, and NavigationBars
* a FrontendTemplateName that corresponds to the name of a JSP that drives the layout of the page
* a list of PageTypes to which the template is restricted
* a VelocityTemplate used to drive the layout of the CmsCockpit Page View

### Tasks

This trail will focus on the creation of the PageTemplate via Impex. They can also be added via the HMC.

This trail will demonstrate creation of a Content Page Template, a slight variation on the current ContentPage1Template.

#### Define PageTemplate

The PageTemplate is configured using the following impex:

|  |
| --- |
| INSERT\_UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];name;frontendTemplateName;restrictedPageTypes(code);active[default=true] ;;ContentPage2Template;Content Page 2 Template;layout/contentLayout2Page;ContentPage |

This PageTemplate is restricted to ContentPages and is bound to the JSP at WEB-INF/views/pages/layout/contentLayout2Page.jsp.

#### Define Content Slots

First the ContenSlotNames for the template are setup:

|  |
| --- |
| INSERT\_UPDATE ContentSlotName;name[unique=true];template(uid,$contentCV)[unique=true][default='ContentPage2Template'];validComponentTypes(code) ;SiteLogo;;CMSImageComponent,BannerComponent ;HeaderLinks;;CMSLinkComponent,CMSParagraphComponent ;MiniCart;;MiniCartComponent ;NavigationBar;;NavigationBarComponent ;Section1;;$wideContent ;Section2;;$wideContent ;Section3;;$narrowContent ;Footer;;CMSLinkComponent,CMSParagraphComponent,FooterComponent |

Each ContentSlotName on the template is given a list of allowed CMSComponent Types. These are restricted to only one component type for the header slots.

The list of ContentSlotForTemplate bindings for the PageTemplate are added:

|  |
| --- |
| INSERT\_UPDATE ContentSlotForTemplate;$contentCV;uid[unique=true];position[unique=true];pageTemplate(uid,$contentCV)[unique=true][default='ContentPage2Template'];contentSlot(uid,$contentCV)[unique=true];allowOverwrite ;;SiteLogo-ContentPage2;SiteLogo;;SiteLogoSlot;true ;;HomepageLink-ContentPage2;HomepageNavLink;;HomepageNavLinkSlot;true ;;NavigationBar-ContentPage2;NavigationBar;;NavigationBarSlot;true ;;MiniCart-ContentPage2;MiniCart;;MiniCartSlot;true ;;Footer-ContentPage2;Footer;;FooterSlot;true ;;HeaderLinks-ContentPage2;HeaderLinks;;HeaderLinksSlot;true |

This preconfigures a number of the slots on the template to use shared ContentSlots. AllowOverwrite controls behaviour of the CMSCockpit PageView.

#### Create PageTemplate JSP

Create the following JSP:

|  |
| --- |
| <%@ page trimDirectiveWhitespaces="true" %> <%@ taglib prefix="template" tagdir="/WEB-INF/tags/template" %> <%@ taglib prefix="cms" uri="/cms2lib/cmstags/cmstags.tld" %> <%@ taglib prefix="breadcrumb" tagdir="/WEB-INF/tags/nav/breadcrumb" %>  <template:page pageTitle="${pageTitle}">  <breadcrumb:breadcrumb breadcrumbs="${breadcrumbs}" />  <cms:slot var="feature" contentSlot="${slots.Section1}">  <div class="span-24 section1 advert">  <cms:component component="${feature}"/>  </div>  </cms:slot>  <div class="span-20 section2 advert">  <cms:slot var="feature" contentSlot="${slots.Section2}">  <cms:component component="${feature}"/>  </cms:slot>  </div>  <div class="span-4 section3 advert last">  <cms:slot var="feature" contentSlot="${slots.Section3}">  <cms:component component="${feature}"/>  </cms:slot>  </div> </template:page> |

Table 42 acceleratorstorefront/web/webroot/WEB-INF/views/pages/layout/contentLayout2Page.jsp

#### Create Velocity Template

Create a Velocity Template that controls the layout of the CMSCockpit PageView. This is not mandatory.

|  |
| --- |
| UPDATE PageTemplate;$contentCV[unique=true];uid[unique=true];velocityTemplate[translator=de.hybris.platform.acceleratorcore.setup.FileLoaderValueTranslator] ;;ContentPage2Template;$jarResource/acceleratorcore/import/cmscockpit/structure-view/structure\_contentPage2Template.vm |

|  |
| --- |
| <div>  <table width="100%" cellspacing="0" style="margin:0;padding:0;border:1px solid #1E4EBF;">  <tbody>  <tr>  <td height="125px" width="25%" colspan="2" rowspan="2" class="structureViewSection">  <cockpit code="SiteLogo"/>  </td>  <td colspan="2" class="structureViewSection">  <cockpit code="HeaderLinks"/>  </td>  <td width="20%" rowspan="2" class="structureViewSection">  <cockpit code="MiniCart"/>  </td>  </tr>  <tr>  <td colspan="2" height="89px" class="structureViewSection">  <div>Header</div>  </td>  </tr>  <tr>  <td colspan="5" class="structureViewSection">  <cockpit code="NavigationBar"/>  </td>  </tr>  <tr>  <td colspan="5" height="58px" style="vertical-align:middle;" class="structureViewSection">  <div>Breadcrumb</div>  </td>  </tr>  <tr>  <td colspan="5" style="vertical-align:top;" class="structureViewSection">  <cockpit code="Section1"/>  </td>  </tr>  <tr>  <td width="80%" colspan="4" style="vertical-align:top;" class="structureViewSection">  <cockpit code="Section2"/>  </td>  <td width="20%" style="vertical-align:top;" class="structureViewSection">  <cockpit code="Section3"/>  </td>  </tr>  <tr>  <td height="270px" colspan="5" class="structureViewSection">  <cockpit code="Footer"/>  </td>  </tr>  </tbody>  </table>  <div style="width:100%; border-top: 2px solid #bbb">  <cockpit code="editor"/>  </div> </div> |

Table 43 acceleratorstorefront/resources/acceleratorcore/import/cmscockpit/structure-view/structure\_contentPage2Template.vm

### Summary

The ContentPage2Template will now be available in the CMSCockpit and HMC for configuration of Pages. It will have a number of preconfigured ContentSlots and three slots that can be setup on individual pages.

# Customization Possibilities in the hybris Multichannel Accelerator

The hybris Multichannel Accelerator is shipped as source code for you to edit. As described in the [Getting Started Guide](#421) you will start by creating a copy of the **accelerator** extensions with your project naming and package structure using [extgen and modulegen](#9). You will adapt the code and scripts to fit your projects requirements. Therefore it is of course infinitely customisable and it is out of the scope of this document to list every possible code change you could make.

The **commerce** extensions **commercefacades** and **commerceservices** are also shipped as source but intended to be customised by replacing bean implementations in the spring application context.

Likewise it will also be interesting to see how to remove certain unwanted functionality. Customization guides are available in this section for your perusal.

|  |
| --- |
| **About this Document**  **Audience**:  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

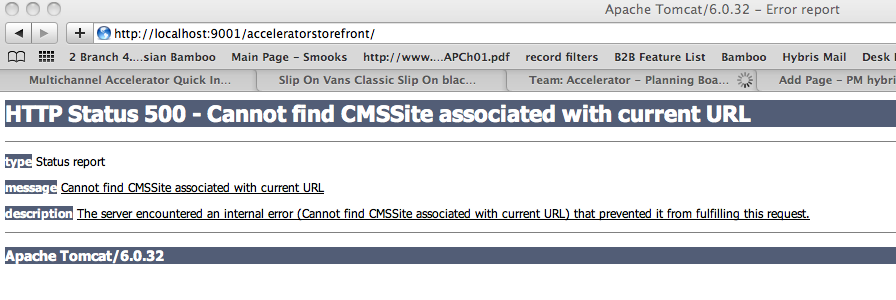
|  |
| --- |
| **Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **Parent Page's Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **See Also**   * [Getting Started with the hybris Multichannel Accelerator](#421) * [Customizing the hybris Multichannel Accelerator with Extgen and Modulegen](#9) * [acceleratorfacades extension Technical Guide](https://wiki.hybris.com/display/acc/acceleratorfacades+extension+-+Technical+Guide#acceleratorfacadesextension-TechnicalGuide-StepByStepGuidetoextendingtheProductFacade) section **Apparel - Step By Step Guide to extending the ProductFacade** * [Developer Trails](#474) |

## Just One Storefront

There is much functionality in the accelerator to handle multiple Storefronts, indeed a request to the servlet at its root context path directly will result in a 500 error! You need to use hosts file mappings or use the site-id parameter to avoid a 500 error (see [Getting Started with the hybris Multichannel Accelerator](#421)).



This document explains what changes need to be applied to the data to support a development installation for a project with just a single storefront. Even if you have multiple storefronts, you can apply the changes to this guide to the Storefront you wish to appear as the default when the users url doesn't provide sufficient information to discriminate between different Storefront's.

|  |
| --- |
| **About this Document**  **Audience**:  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **Parent Page's Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **See Also** |

The CMS Site has a an attribute called *urlPatterns*. This is a list of regular expressions that the Website CMS Filters will match to determine which Storefront a user is trying to access.

The simplest thing is to add a further regular expression to the CMS Site like so :

|  |
| --- |
| (?i)^https?://[^/]\*/acceleratorstorefront((?![\?\&]site=).)\* |

1. This can be done in the hMC under the Website->[Site]->Properties tab.

|  |
| --- |
| **Making electronics the default Website** |

2. To make the change more permanent, you should add this to a CMS Site setup impex script run during the project data phase of Initialization.

|  |
| --- |
| # CMS Site UPDATE CMSSite;uid[unique=true];urlPatterns; ;electronics;(?i)^https?://[^/]+(/[^?]\*)?\?(.\*\&)?(site=electronics)(|\&.\*)$,(?i)^https?://electronics\.[^/]+(|/.\*|\?.\*)$,(?i)^https?://api\.hybrisdev\.com(:[\d]+)?/rest.\*$,(?i)^https?://[^/]\*/acceleratorstorefront((?![\?\&]site=).)\*; |

For more information on setting up scripts that run during Initialization see: [Essential and Project Data in the hybris Multichannel Accelerator](#167)

## Removing Apparel and the Apparel Stores

If your project does not need Apparel support you may want to remove this functionality. This guide explains how to remove Apparel from the accelerator source code. This includes the apparel data model, any apparel services and facades as well as the apparel sample storefronts configuration.

You will most probably already have used **extgen** and **modulegen** to create versions of the **accelerator** extensions with your projects naming and packages. You will need to substitute the **accelerator** packages, filenames and extension names with your projects names.

* [acceleratorcore](#499)
  + [Data Model Removal](#500)
  + [Hot Folder Batch Import](#501)
  + [Store Content](#502)
  + [Solr](#503)
* [acceleratorfacades](#504)
  + [Facades](#505)
* [acceleratorstorefront](#506)
  + [spring application context](#507)
  + [Site resource files](#508)
* [acceleratorsampledata](#509)
  + [Store Content](#502)
  + [Permissions](#510)
  + [Recompile](#511)
* [Re-Initialize](#512)

|  |
| --- |
| **About this Document**  **Audience**:  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **Parent Page's Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **See Also**   * [acceleratorcore extension - Technical Guide](https://wiki.hybris.com/display/acc/acceleratorcore+extension+-+Technical+Guide#acceleratorcoreextension-TechnicalGuide-ApparelProducts) section **ApparelProducts** * [acceleratorfacades extension - Technical Guide](https://wiki.hybris.com/display/acc/acceleratorfacades+extension+-+Technical+Guide#acceleratorfacadesextension-TechnicalGuide-Apparel) section **Apparel** * [acceleratorstorefront Extension - Technical Guide](#97) * [acceleratorsampledata Extension - Technical Guide](#101) |

### acceleratorcore

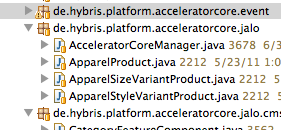
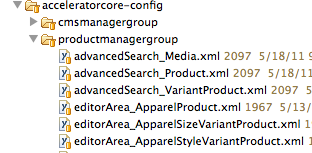
#### Data Model Removal

We should start by removing the Data Model changes and any associated configuration for cockpits.

* In **acceleratorcore**/resources/acceleratorcore-items.xml remove everything inside the typegroup apparel

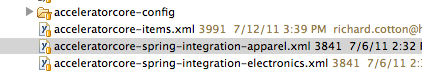
|  |
| --- |
| <typegroup name="Apparel">  <itemtype code="ApparelProduct" extends="Product"  autocreate="true" generate="true"  jaloclass="de.hybris.platform.acceleratorcore.jalo.ApparelProduct">  <description>Base apparel product extension that contains additional attributes.</description>  <attributes>  <attribute qualifier="genders" type="GenderList">  <description>List of genders that the ApparelProduct is designed for</description>  <modifiers />  <persistence type="property" />  </attribute>  </attributes>  </itemtype>   <itemtype code="ApparelStyleVariantProduct" extends="VariantProduct"  autocreate="true" generate="true"  jaloclass="de.hybris.platform.acceleratorcore.jalo.ApparelStyleVariantProduct">  <description>Apparel style variant type that contains additional attribute describing variant style.</description>  <attributes>  <attribute qualifier="style" type="localized:java.lang.String"  metatype="VariantAttributeDescriptor">  <description>Colour/Pattern of the product.</description>  <modifiers />  <persistence type="property" />  </attribute>   <attribute qualifier="swatchColour" type="java.lang.String">  <description>A normalised colour mapping to a standardised front-end navigable name.</description>  <modifiers />  <persistence type="property" />  </attribute>  </attributes>   </itemtype>   <itemtype code="ApparelSizeVariantProduct" extends="ApparelStyleVariantProduct"  autocreate="true" generate="true"  jaloclass="de.hybris.platform.acceleratorcore.jalo.ApparelSizeVariantProduct">  <description>Apparel size variant type that contains additional attribute describing variant size.</description>  <attributes>  <attribute qualifier="size" type="localized:java.lang.String"  metatype="VariantAttributeDescriptor">  <description>Size of the product.</description>  <modifiers />  <persistence type="property" />  </attribute>  </attributes>   </itemtype>  </typegroup> |

Table 44 items.xml extract to remove

* Remove *ApparelProduct*, *ApparelSizeVariantProduct* and *ApparelStyleVariantProduct* from **acceleratorcore**/src/de/hybris/platform/acceleratorcore/jalo  
  
* Remove the apparel specific productcockpit editor configuration xml files (editorArea\_ApparelProduct.xml, editorArea\_ApparelSizeVariantProduct.xml, editorArea\_ApparelStyleVariantProduct.xml) from **acceleratorcore**/resources/acceleratorcore-config/productmanagergroup  
  
* Remove the following test method **acceleratorcore**/testsrc/de/hybris/platform/acceleratorcore/batch/task/BatchIntegrationTest.testVariant
* Execute 'ant clean all' inside the **acceleratorcore** extension to ensure the extension still compiles.

#### Hot Folder Batch Import

We should remove the back import instance for the apparel store. This can be done with just 2 steps.

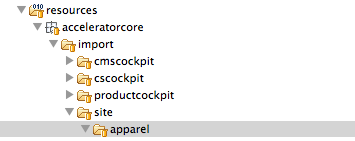
* Remove the file **acceleratorcore**/resources/acceleratorcore-spring-integration-apparel.xml  
  
* Remove the following line from **acceleratorcore**/resources/acceleratorcore-spring-integration.xml

|  |
| --- |
| <import resource="acceleratorcore-spring-integration-apparel.xml"/> |

Table 45 Line to remove from acceleratorcore-spring-integration.xml

#### Store Content

We should remove all scripts that load content and setup the Apparel stores. This can be done with just 2 simple steps.

* Remove the entire apparel folder under **acceleratorcore**/resources/acceleratorcore/import/site/apparel  
  
* Remove the following line in **acceleratorcore**/src/de/hybris/platform/acceleratorcore/setup/AcceleratorCoreSystemSetup.createProjectData

|  |
| --- |
| importSite(context, "apparel", countries, syncProducts, syncContent); |

#### Solr

Deleting the Store Content will remove the scripts that create the SOLR Index, but there are also some value provider spring beans that need to be removed from **acceleratorcore** application context.

* Remove the following *CategorySource* beans from the file **acceleratorcore**/resources/acceleratorcore-spring.xml

|  |
| --- |
| <bean id="apparelCategorySource" parent="abstractCategorySource">  <property name="rootCategory" value="categories"/> <!-- 'categories' is the root apparel category --> </bean> <bean id="apparelBrandCategorySource" parent="abstractCategorySource">  <property name="rootCategory" value="brands"/> <!-- 'brands' is the root of the brands hierarchy --> </bean> |

Table 46 Category Source Beans to Remove

* Remove the following *ValueProvider* beans from the file **acceleratorcore**/resources/acceleratorcore-spring.xml

|  |
| --- |
| <bean id="apparelCategoryCodeValueProvider" parent="abstractCategoryCodeValueProvider">  <property name="categorySource" ref="apparelCategorySource"/> </bean> <bean id="apparelBrandCategoryCodeValueProvider" parent="abstractCategoryCodeValueProvider">  <property name="categorySource" ref="apparelBrandCategorySource"/> </bean> <bean id="apparelCategoryNameValueProvider" parent="abstractCategoryNameValueProvider">  <property name="categorySource" ref="apparelCategorySource"/> </bean> <bean id="apparelBrandCategoryNameValueProvider" parent="abstractCategoryNameValueProvider">  <property name="categorySource" ref="apparelBrandCategorySource"/> </bean> |

Table 47 Value Provider Beans to Remove

* Remove the Size specific Sort Providers and Comparators from the file **acceleratorcore**/resources/acceleratorcore-spring.xml

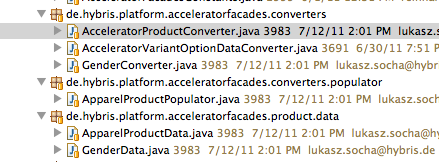
|  |
| --- |
| <bean id="sizeAttributeSortProvider" class="de.hybris.platform.acceleratorcore.comparators.DefaultFacetSortProvider">  <property name="comparator" ref="sizeFacetAttributeComparator"/> </bean>  <bean id="sizeFacetAttributeComparator" class="de.hybris.platform.acceleratorcore.comparators.SizeFacetAttributeComparator">  <property name="comparator" ref="sizeAttributeComparator"/> </bean>  <bean id="sizeAttributeComparator" class="de.hybris.platform.acceleratorcore.comparators.SizeAttributeComparator">  <property name="pattern" value="[0-9]+(\.[0-9])\*"/>  <property name="sizeSystems">  <list>  <value>XXS</value>  <value>XS</value>  <value>S</value>  <value>M</value>  <value>L</value>  <value>LXL</value>  <value>XL</value>  <value>XXL</value>  <value>3XL</value>  <value>4XL</value>  <value>5XL</value>  </list>  </property> </bean> |

Table 48 Sort Providers and Comparators to Remove

### acceleratorfacades

Most of the code in the **acceleratorfacades** template is provided for Apparel Purposes. This guide assumes you have not adapted any of the code we plan to delete for other purposes than just apparel.

#### Facades

* Remove *AcceleratorProductConverter*, *AcceleratorVariantOptionDataConverter*, *GenderConverter* from the **acceleratorfacades**/src/de/hybris/platform/acceleratorfacades/converters folder
* Remove *ApparelProductPopulator* from the **acceleratorfacades**/src/de/hybris/platform/acceleratorfacades/converters/populator folder
* Remove *ApparelProductData* and *GenderData* from the **acceleratorfacades**/src/de/hybris/platform/acceleratorfacades/product/data folder  
  
* Remove the spring bean configuration for all these classes in **acceleratorfacades**/resources/acceleratorfacades-spring.xml

|  |
| --- |
| <alias name="acceleratorVariantOptionDataConverter" alias="variantOptionDataConverter"/>  <bean id="acceleratorVariantOptionDataConverter" class="de.hybris.platform.acceleratorfacades.converters.AcceleratorVariantOptionDataConverter" parent="defaultVariantOptionDataConverter" scope="tenant">  <property name="mediaService" ref="mediaService"/>  <property name="mediaContainerService" ref="mediaContainerService"/>  <property name="typeService" ref="typeService"/>  <property name="imageFormatMapping" ref="imageFormatMapping"/>  <property name="variantAttributeMapping">  <map>  <entry key="ApparelStyleVariantProduct.style" value="styleSwatch"/>  </map>  </property>  </bean>   <alias name="acceleratorGenderConverter" alias="genderConverter"/>  <bean id="acceleratorGenderConverter" class="de.hybris.platform.acceleratorfacades.converters.GenderConverter" scope="tenant">  <property name="typeService" ref="typeService"/>  </bean>   <bean id="apparelProductPopulator" class="de.hybris.platform.acceleratorfacades.converters.populator.ApparelProductPopulator" scope="tenant">  <property name="genderConverter" ref="genderConverter"/>  </bean>   <alias name="acceleratorProductConverter" alias="productConverter"/>  <bean id="acceleratorProductConverter" class="de.hybris.platform.acceleratorfacades.converters.AcceleratorProductConverter" scope="tenant" parent="defaultProductConverter">  <property name="apparelProductPopulator" ref="apparelProductPopulator"/>  </bean> |

Table 49 Facade beans to remove from acceleratorfacades-spring.xml

* Execute 'ant clean all' inside the **acceleratorfacades** extension to ensure the extension still compiles.

### acceleratorstorefront

The Storefront Controllers are really quite agnostic of Apparel so there is little to do.

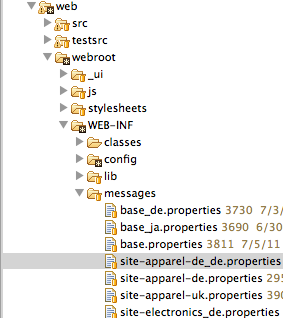
#### spring application context

* In **acceleratorstorefront**/web/webroot/WEB-INF/config/spring-mvc-config.xml remove the line:

|  |
| --- |
| <entry key="size" value-ref="sizeAttributeComparator"/> |

Table 50 Line to remove from spring-mvc-config.xml

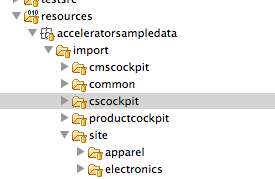
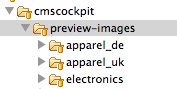
#### Site resource files

* Remove the apparel site resource files (site-apparel-de\_de.properties, site-apparel-de.properties and site-apparel-uk.properties) from the **acceleratorfacades**/web/webroot/WEB-INF/messages folder  
  

### acceleratorsampledata

The **acceleratorsampledata** extension adds content for the apparel store. We can remove this content if we still wish to include the electronics sample store.

#### Store Content

* Delete the **acceleratorsampledata**/resources/acceleratorsampledata/import/site/apparel folder  
  
* Remove sample cmscockpit preview image folders **acceleratorsampledata**/resources/acceleratorsampledata/import/cmscockpit/preview-images/apparel\_de and **acceleratorsampledata**/resources/acceleratorsampledata/import/cmscockpit/preview-images/apparel\_uk  
  
* Remove the following lines from **acceleratorsampledata**/src/de/hybris/platform/acceleratorsampledata/setup/AcceleratorSampleDataSystemSetup

|  |
| --- |
| params.add(createBooleanSystemSetupParameter(IMPORT\_SITE\_APPAREL, "Import Apparel Site Sample Data", true)); |

Table 51 method getInitialisationOptions()

|  |
| --- |
| if (getBooleanSystemSetupParameter(context, IMPORT\_SITE\_APPAREL)) {  importApparelProductData(context); } |

Table 52 method createProjectData(final SystemSetupContext context)

* Finally remove the whole method *importApparelProductData(final SystemSetupContext context)* from **acceleratorsampledata**/src/de/hybris/platform/acceleratorsampledata/setup/AcceleratorSampleDataSystemSetup

#### Permissions

Some scripts to give users permissions to edit a catalogs need to be removed.

* Remove the lines that give the user write permissions to the apparelProductCatalog in **acceletatorsampledata**resources/acceleratorsampledata/import/productcockpit/productcockpit-users.impex

|  |
| --- |
| ;apparelProductCatalog;Online;productmanagergroup,productmanager; ;apparelProductCatalog;Staged;productmanagergroup,productmanager; |

Table 53 Lines to remove from productcockpit-users.impex

* Remove the lines that give the user write permissions to the apparelContentCatalog in **acceletatorsampledata**resources/acceleratorsampledata/import/cmscockpit/cmscockpit-users.impex

|  |
| --- |
| ;apparel-ukContentCatalog;Online;cmsmanagergroup,cmsmanager; ;apparel-ukContentCatalog;Staged;cmsmanagergroup,cmsmanager; ;apparel-deContentCatalog;Online;cmsmanagergroup,cmsmanager; ;apparel-deContentCatalog;Staged;cmsmanagergroup,cmsmanager; |

Table 54 Lines to remove from cmscockpit-users.impex

#### Recompile

* Execute 'ant clean all' inside the **acceleratorsampledata** extension to ensure the extension still compiles.

### Re-Initialize

Finally we need to re-initialize the hybris Platform database to remove all the apparel data model from the type system and any orphaned data no longer added by the scripts. Re-initialise your local platform database as per [the initialisation documentation](https://wiki.hybris.com/display/release4/Initialization+and+Update+Documentation#InitializationandUpdateDocumentation-InitializeSystem).

## Removing BTG

If behavioral targeting is not desired in your storefront the extension can be removed by following the steps outlined below:

|  |
| --- |
| **About this Document**  **Audience**:  **Related concept**: hybris Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **Parent Page's Resources**  Collapse all[Expand all](#)  [Collapse all](#) |

|  |
| --- |
| **See Also** |

* Remove all classes in de.hybris.platform.acceleratorstorefront.servlets.btg and de.hybris.platform.acceleratorstorefront.servlets.btg.impl
* Remove the BTG interceptor in spring-mvc-config.xml from mvc:interceptors

|  |
| --- |
| <bean class="de.hybris.platform.acceleratorstorefront.servlets.btg.ContentPageVisitedBtgInterceptor" scope="tenant">  <property name="eventService" ref="eventService"/> </bean> |

Table 55 spring-mvc-config.xml

* Comment out or remove all BTG filters in web.xml

|  |
| --- |
| <!-- BTG Filters --> <filter>  <filter-name>refererHeaderBtgFilter</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter> <filter-mapping>  <filter-name>refererHeaderBtgFilter</filter-name>  <servlet-name>DispatcherServlet</servlet-name> </filter-mapping>  <filter>  <filter-name>requestParamsBtgFilter</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter> <filter-mapping>  <filter-name>requestParamsBtgFilter</filter-name>  <servlet-name>DispatcherServlet</servlet-name> </filter-mapping>  <filter>  <filter-name>productVisitedBtgFilter</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter> <filter-mapping>  <filter-name>productVisitedBtgFilter</filter-name>  <servlet-name>DispatcherServlet</servlet-name> </filter-mapping>  <filter>  <filter-name>categoryVisitedBtgFilter</filter-name>  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class> </filter> <filter-mapping>  <filter-name>categoryVisitedBtgFilter</filter-name>  <servlet-name>DispatcherServlet</servlet-name> </filter-mapping>  <filter>  <filter-name>BTGSegmentFilter</filter-name>  <filter-class>de.hybris.platform.acceleratorstorefront.servlets.btg.BTGSegmentFilter</filter-class> </filter> <filter-mapping>  <filter-name>BTGSegmentFilter</filter-name>  <servlet-name>DispatcherServlet</servlet-name> </filter-mapping> |

Table 56 web.xml

* AbstractSystemSetup in acceleratorcore: remove reference to BTGItem  
  Change:

|  |
| --- |
| protected List<ComposedType> getContentSyncRootTypes() {  final TypeManager typeManager = TypeManager.getInstance();   return Arrays.asList(typeManager.getComposedType("CMSItem"), typeManager.getComposedType("CMSRelation"),  typeManager.getComposedType("Media"), typeManager.getComposedType("BTGItem")); } |

Table 57 AbstractSystemSetup.java

To:

|  |
| --- |
| protected List<ComposedType> getContentSyncRootTypes() {  final TypeManager typeManager = TypeManager.getInstance();   return Arrays.asList(typeManager.getComposedType("CMSItem"), typeManager.getComposedType("CMSRelation"),  typeManager.getComposedType("Media")); } |

Table 58 AbstractSystemSetup.java

* Remove the btg extension in acceleratorcore and acceleratorstorefront

|  |
| --- |
| <requires-extension name="btg"/> |

Table 59 extensioninfo.xml

* Remove or comment out all the beans in acceleratorstorefront/web/webroot/WEB-INF/config/acceleratorstorefront-spring-btg.xml

|  |
| --- |
| <bean id="abstractBtgFilter" abstract="true">  <property name="eventService" ref="eventService" /> </bean> <bean id="refererHeaderBtgFilter" class="de.hybris.platform.acceleratorstorefront.servlets.btg.RefererHeaderBtgFilter"  parent="abstractBtgFilter" scope="tenant" /> <bean id="requestParamsBtgFilter" class="de.hybris.platform.acceleratorstorefront.servlets.btg.RequestParamsBtgFilter"  parent="abstractBtgFilter" scope="tenant" /> <bean id="productVisitedBtgFilter" class="de.hybris.platform.acceleratorstorefront.servlets.btg.ProductVisitedBtgFilter"  parent="abstractBtgFilter" scope="tenant">  <property name="pkResolvingStrategy" ref="productPkResolvingStrategy" /> </bean> <bean id="categoryVisitedBtgFilter" class="de.hybris.platform.acceleratorstorefront.servlets.btg.CategoryVisitedBtgFilter"  parent="abstractBtgFilter" scope="tenant">  <property name="pkResolvingStrategy" ref="categoryPkResolvingStrategy" /> </bean> <bean id="productPkResolvingStrategy" class="de.hybris.platform.acceleratorstorefront.servlets.btg.impl.ProductPkResolvingStrategy"  scope="tenant">  <property name="urlParsingStrategy" ref="productUrlParsingStrategy" />  <property name="productService" ref="productService" /> </bean> <bean id="productUrlParsingStrategy" class="de.hybris.platform.acceleratorstorefront.servlets.btg.impl.DefaultUrlParsingStrategy"  scope="tenant">  <property name="regex" value="^/.+/p/(\w+)$" /> </bean> <bean id="categoryPkResolvingStrategy" class="de.hybris.platform.acceleratorstorefront.servlets.btg.impl.CategoryPkResolvingStrategy"  scope="tenant">  <property name="urlParsingStrategy" ref="categoryUrlParsingStrategy" />  <property name="categoryService" ref="commerceCategoryService" /> </bean> <bean id="categoryUrlParsingStrategy" class="de.hybris.platform.acceleratorstorefront.servlets.btg.impl.DefaultUrlParsingStrategy"  scope="tenant">  <property name="regex" value="^/.+/c/(\w+)$" /> </bean> |

Table 60 acceleratorstorefront-spring-btg.xml

# User Interface and Creatives in the hybris Multichannel Accelerator

|  |
| --- |
| TKZ 2011.06.29: Abstract (Introduction to the topic) Update: All topics regarding the visual frontend the enduser of the accelerator will see and how to change the design with Themes and CSS. |

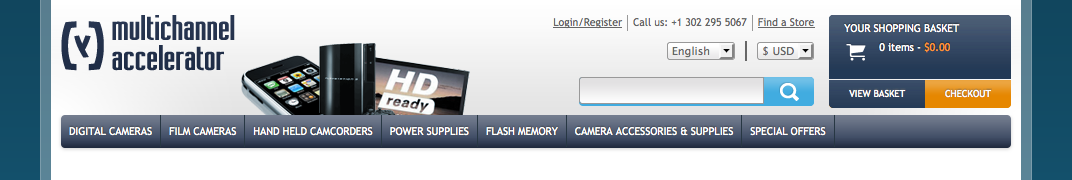
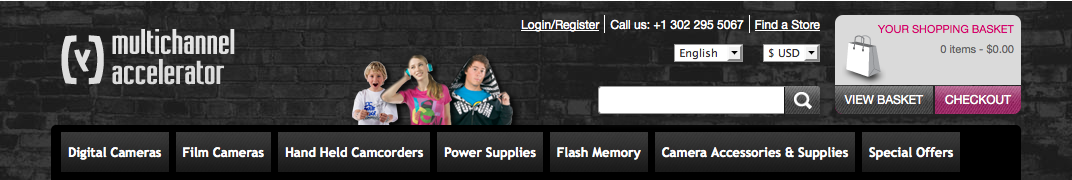
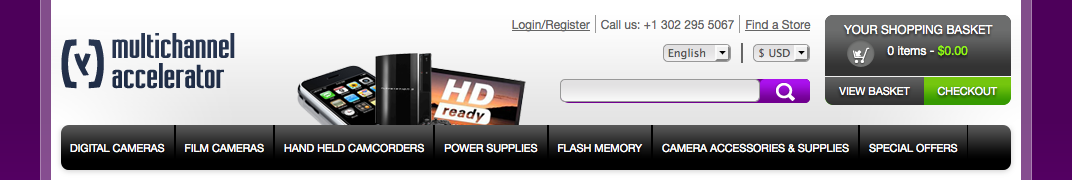
* [The Front End](#513)
* [Usability, Layout and Design Good Practice](#514)
  + [Colors](#515)
  + [Layout](#516)
    - [Dimensions](#517)
    - [Standardized Components](#518)
  + [Accessibility](#519)
* [Technical Structure](#520)
  + [HTML](#521)
  + [Theme Directory](#522)
    - [Change theme](#523)
  + [CSS Framework](#524)
  + [Javascript](#525)
  + [Browser Compatibility](#526)

|  |
| --- |
| **About this Document**  This document provides information on the user interface and creatives within the hybris Multichannel Accelerator.  **Audience**: Consultants, developers, web designers.  **Related concept**: hybris Multichannel Accelerator  **Validity**: 4.4.0 and higher.  **Based on hybris version**: 4.4.0 |

|  |
| --- |
| **See Also**   * . |

## The Front End

By default, the hybris Accelerator is shipped with three themes, which can be used in any storefront:

* Blue, the **electronics** storefront default one, applied during system initialization:  
  
* Black, the **apparel** storefront default one, applied during system initialization:  
  
* Purple:  
  

## Usability, Layout and Design Good Practice

### Colors

Each of the themes created for the hybris Accelerator storefronts uses minimized colors to keep customers focus on important elements.

Examples: <http://kuler.adobe.com/>

|  |
| --- |
| TKZ 2011.06.29: What is the purpose of the page above? Update: A good approach to a color theme is to select a small range of colors (5 is good choice) that fit together well with one or two complementary colors for highlighting purpose. A nice selection of color themes can be found here for inspiration. |

The reasons behind choosing colors for highlighting elements are as follows:

1. Using one specified color to lucid all graphical elements which have something to do with the shopping process, for example cart, shopping, prices, buying, check out and so on.
2. Using one specified color to highlight elements which are focused or hovered.
3. Using one color to show available functions, for example buttons.
4. Using one color as navigation elements background.
5. Using one color for special offers, that is text and buttons.

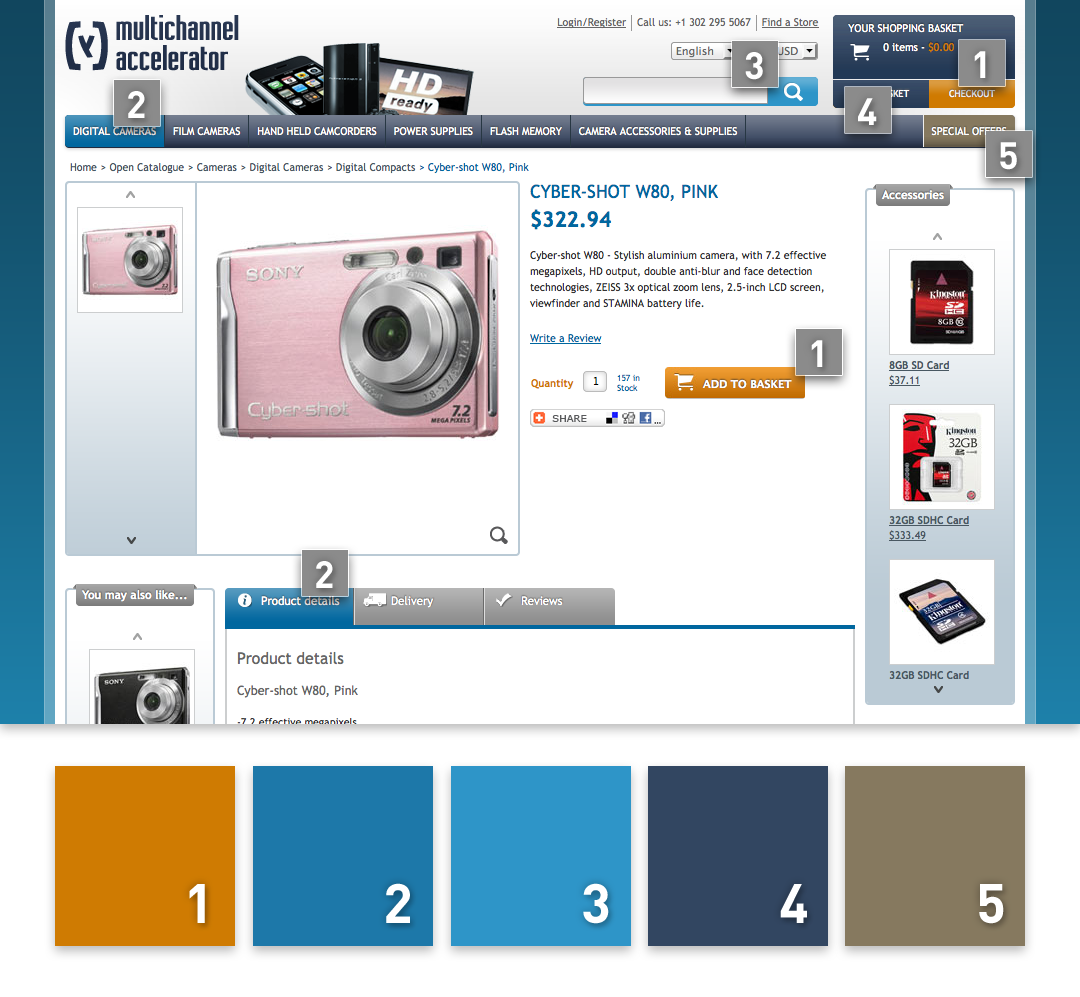


Figure:

### Layout

The elements of the most interest are placed above the page fold, based on a 1024x768 px screen resolution. The header contains mostly important elements consistent on every page.  
Cart / global search / main navigation / bread crumb navigation / login

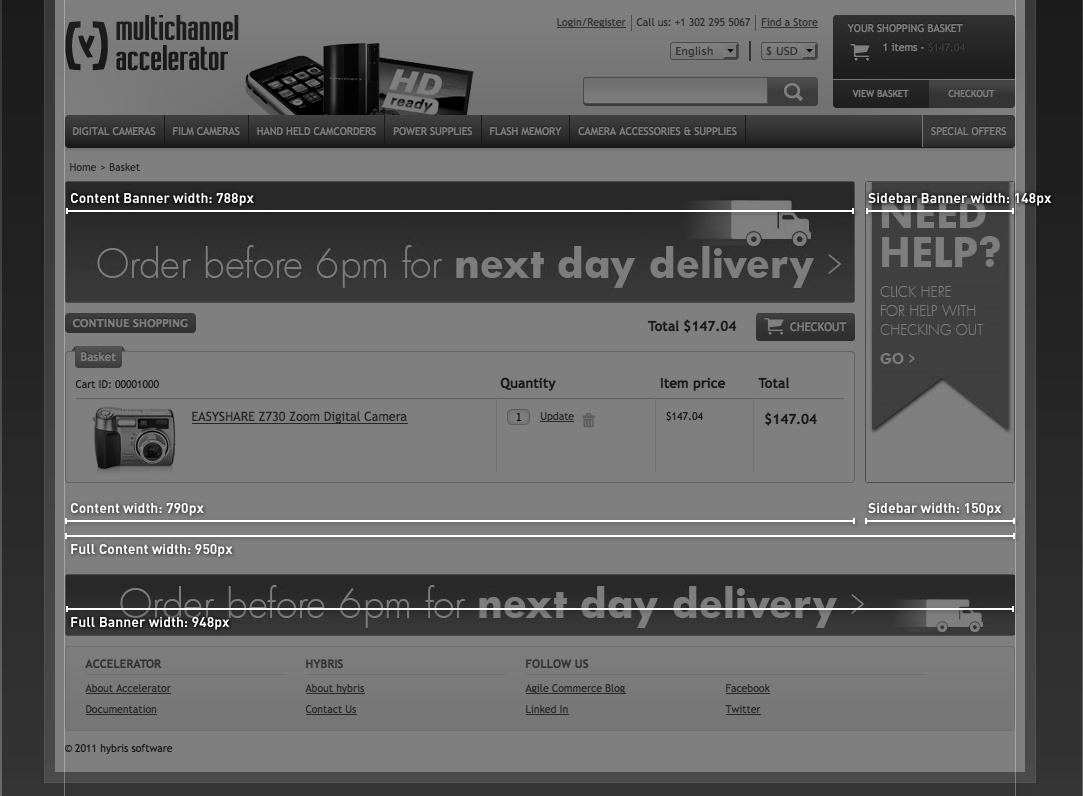
|  |
| --- |
| TKZ 2011.06.29: How to connect the above ideas (are "Cart / global search / main navigation / bread crumb navigation / login" elements consistent on every page?) Update: The elements of the most interest are placed above the page fold in general, based on a 1024x768 px screen resolution. Always needed elements of importance are placed above the pagefold in the header where these are available consistently on every page. |



The most important factors for design of the storefront templates involves:

* Consistent and only a few specified text sizes.
* Short product names.
* Short and concise product description.
* Consistent icon style and sizes.

#### Dimensions

Please note: Every Banner has applied a border of one pixel width. Therefor every banner is 2 px smaller than it´s surrounding content element  


|  |
| --- |
| TKZ 2011.06.29: For the section below:   * Where can I get "the corresponding JSP that holds the components HTML"? * Where can I get the "description to each"? Update: See below |

#### Standardized Components

Find below a list of standardized components

| **Component Name** | **Example** | **Description** | **"corresponding JSP that holds the components HTML** |
| --- | --- | --- | --- |
| **Slider Component** |  | common component for changing/rotating images/banner in one place via Javascript automatically or on user interaction | rotatingimagescomponent.jsp |
| **Banner Component** |  | place html/image banner all over across the site | bannercomponent.jsp |
| **Product Carousel** |  | promotes certain products in a special box that holds more products than in the visible area. rotate products on user interaction | productcarouselcomponent.jsp |
| **Product Tab** |  | Holds different types of content spacesaving in one place beneath the productdetailinformations: datasheet/shippinginfo/reviews | productPageTabs.tag, productDetailsTab.tag, productDetailsJavascript.tag, reviewsTab.jsp |
| **Product Category Component** |  | Displays all categories in seperate boxes on one page in the contentarea | categoryfeaturecomponent.jsp |
| **Product Grid Component** |  | Displays all products of a category in seperate boxes on one page in the contentarea (including display all and paging) | productListerGridItem.tag |
| **Product List Component** |  | Displays all products in seperate big boxes (list) on one page in the contentarea (including display all and paging) | productListerItem.tag |
| **Ajax Basket Lightbox** |  | Displays the product that the user recently added successfully to the cart | addToCartPopup.jsp |
| **Checkout Summary Component** |  | Displays all information you need to buy a product on one page (single page checout) | checkoutSummaryPage.jsp |
| **Customer Account Tile Component** |  | Displays all available Account Information the user likes to look up or change | accountHomePage.jsp |

Find a complete description of the listed components in the [WCMS Components for the hybris Multichannel Accelerator](#89) document.

### Accessibility

All non-text content that is presented to the user has an alternative text, that serves the equivalent purpose. The content can be presented in different ways, for example simpler layout, no stylesheet, without losing information or structure.

|  |
| --- |
| TKZ 2011.06.29: For the "All functionality is available from a keyboard. " - can you provide any examples? Update: It is possible to purchase a product without using the mouse as interface |

All functionality is available from a keyboard.

More Information:<http://www.w3.org/TR/WCAG20/>

## Technical Structure

|  |
| --- |
| TKZ 2011.06.29: Could you provide the full path of **\_ui** folder? Update: See below |

The common directory located within the \\_ui folder  
/bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/  
provides the basis and underlying structure for all themes and consists of the following sub directories:

* CSS  
  The directory contains all core CSS files relating to the hybris Accelerator. Almost all core CSS values are defined within the common.css file. This directory is required in all themes.  
  /bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/common/css/
* Images  
  This directory provides a location for all theme-independent imagery referenced by the common.css file.  
  /bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/common/images/
* JS  
  This directory provides a location for all JS libraries used globally within the hybris Accelerator.  
  /bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/common/js/

|  |  |
| --- | --- |
|  | **Note**  Note that all re branding should be placed in your theme directory. |

### HTML

|  |
| --- |
| TKZ 2011.06.29: The section below should contain some more explanation:   * What is the source of the HTML /bin/ext-accelerator/acceleratorstorefront/web/webroot/WEB-INF/tags/ /bin/ext-accelerator/acceleratorstorefront/web/webroot/WEB-INF/views/ * How to change the themes with CSS? eg. blue theme: /bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/blue/css/changes.css open the file with your preferred text editor and go ahead * Why tables are avoided? Tables have a predefined structure that can´t be changed via CSS. THerefor a DIV based HTML Structure was selected. It´s more generic and changeable. * Example usage of Firefox Extension Firebug. Let´s you select any item in the domtree by just clicking on it in the browser and displays all available styles for this item and which CSS files contains these informations * Full path for **webroot/webinf**, what's inside? /bin/ext-accelerator/acceleratorstorefront/web/webroot/WEB-INF As a designer you will need only the HTML (see above) and the localization files: /bin/ext-accelerator/acceleratorstorefront/web/webroot/WEB-INF/messages * What does "components, pages, tags" refer to? out of date... (just the folders, see above) |

There is one source of HTML.  
Themes are changed only with CSS.  
Tables are avoided for CSS based Styling and google optimization.  
We recommend the usage of the Firefox Extension Firebug to easily explore and analyze the structure of the Accelerator Web front end.  
The HTML is located in webroot/webinf  
components, pages, tags  
Considerations:  
Use of Headers within tables [more Input]  
Approach to Header logic - H1, H2 semantics [more Input]  
Forms used in conjuction with Labels [more Input]  
[more Topics]

### Theme Directory

|  |
| --- |
| TKZ 2011.06.29: Could you provide the full path of **\_ui** and **/data/media/sys\_master/** folders? Update: These images are changed within the WCMS. Please do so. |

The theme directory, which is also in the \\_ui folder consists the following sub directories:

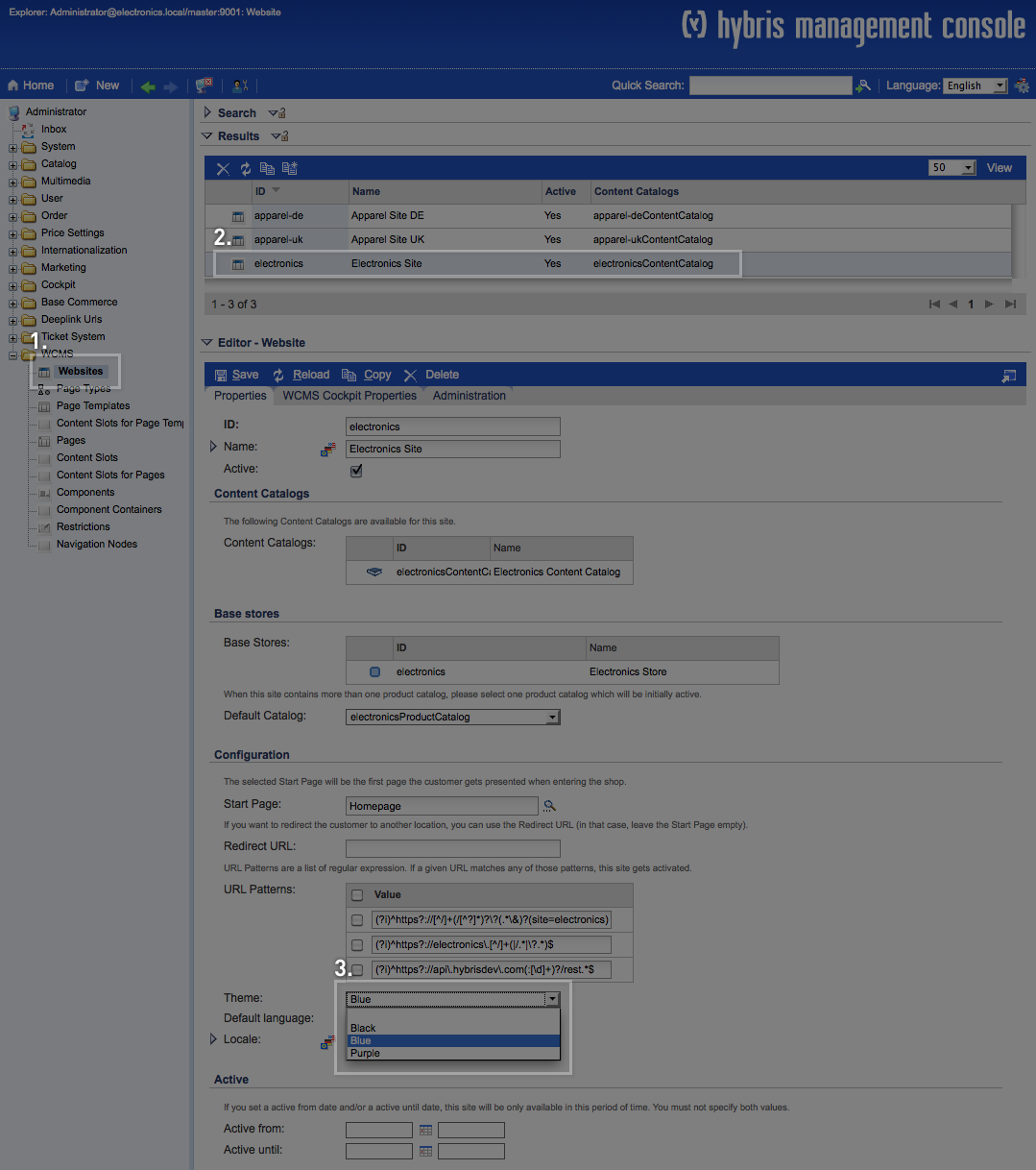
* CSS  
  The CSS directory contains all CSS files pertaining to your theme, in case of the hybris Accelerator this consists of at least 1 file: changes.css .  
    
  You can create the desired look-and-feel for your store by making changes to this stylesheet. All images, for example logos and banners, that are located in /data/media/sys\_master/ are **not** changed by CSS.
* Images

|  |
| --- |
| TKZ 2011.06.29: I think that the sentence below is not understandable. Update: All linked images in a theme specific CSS File should also reside in the theme specific image folder |

When defining an image URL within the changes.css file, the image should fit the critera of  
being .theme dependant. It should reside within this directory.  
  
When re branding the CSS to suit your new theme it is the following two stylesheets that you will be mostly concerned with: common.css and changes.css . In summary - the style in the common.css file provides the core CSS values for most of the classes within the hybris Accelerator project, but it is within the changes.css file, that all theming should take place and where the core styles are adjusted.

Feel free to add new styles, overwrite styles inherited from the common.css file and amend existing styles as necessary to achieve your re-branding.

#### Change theme

You can apply any theme to your store very easy in the hmc  


### CSS Framework

The framework used for the hybris Accelerator is called **Blueprint**. It provides a flexible, highly customizable CSS framework that allows for multiple column design layouts, based on a fixed, 960px width, container.

|  |
| --- |
| **See Also**   * [Wikipedia on Blueprint](http://en.wikipedia.org/wiki/Blueprint_%28CSS_framework%29) |

The Blueprint CSS framework also contains basic form and print stylesheets alongside a reset to help eliminate cross browser discrepancies. At its core, Blueprint consists of 24 width defined classes used to signify columns and dividers within the page increasing in width from span-1 to span-24.

The width values assigned to each span co-incide with their numbering, for example you could nest 24 span-1 columns within a span-24 parent container or 2 span-12's or 4 span-6's respectively:  
1 x 24 = 24  
2 x 12 = 24  
4 x 6 = 24

Alongside a width value, the CSS parameters for each span less than 24, also set a **marginright**  
value of 10px to ensure consistent column spacing.  
It is important to note that on the last column of any nested span the value of **.last'** should be  
added to remove the right margin. A span-24 column does not require this additional class, due to being the widest column, there is no margin to be accounted. You can find more detailed information on using Blueprint in the <http://www.blueprintcss.org/> page.

### Javascript

|  |
| --- |
| TKZ 2011.06.29: Where to get descriptions of libraries from? Update: see below |

Used JS Framework: <http://jquery.com/>

The following javascript libraries are currently in place and can be located from the following  
path :  
/bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/common/js/

* jquery-1.5.1.min.js  
  Provides the core jquery library functionality<http://jquery.com/>
* jquery-ui.min.js<http://jqueryui.com/>  
  Provides the interface for the jquery core

<http://plugins.jquery.com/:>

* jquery.jcarousel.js  
  Empowers the product carousel component.
* jquery.colorbox.js  
  Empowers all ajax layers (lightboxes) (like the Ajax Basket Lightbox component).
* jquery.slideviewer.1.2.js  
  Empowers the Slider component.
* jquery.easing.1.3.js  
  advanced easing options neede by other jquery libraries

### Browser Compatibility

Browsers to be supported are:  
Windows XP, Vista, 7: IE 7 series, IE 8 series, IE 9 series  
Windows XP, Vista, 7: Firefox 3.5+ series  
Windows XP, Vista, 7: Chrome 10  
Windows XP, Vista, 7 & Mac OS X: Safari 5+ series

The HTML Structure and the Javascript functionality fits the needs of all common browsers.  
CSS3 is only used for little visual improvements that makes the overall look fresher and smoother (eg shadows and round corners)

The special CSS needs of IE7 and IE8 can be adjusted within the ie7.css/ie8.css file that resides in each theme:  
/bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/common/css/ie7.css  
/bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/common/css/ie8.css  
/bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/yourTheme/css/ie7.css  
/bin/ext-accelerator/acceleratorstorefront/web/webroot/\_ui/yourTheme/css/ie8.css

IE6 is not supported