

Instructions for SAP Hybris Commerce Developer Training - Part I

Introduction

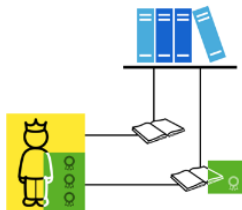
Welcome to the SAP Hybris Commerce Developer Training, which will help you better understand and remember basic SAP Hybris concepts, such as data modeling, content management, and security.

This document contains the demos and instructions we (the SAP Hybris Education Team) have created to guide you to such an understanding. The theory and concepts necessary to complete each exercise will first be presented by your instructor, who will then be available to assist you as you work through the instructions.

The exercises will have you build a simple bookstore that will not only sell, but also rent books. Your customers will accumulate reward points useable for future rentals. You can feel the repeat business already, no?

The aim of the exercises is to familiarize you not only with the core features of the SAP Hybris Commerce Suite, but also with the suite's tools and best practices. Given how much we'll be covering, it's not realistic to expect you to write and configure everything you need to build your bookstore. We will provide setup scripts that copy all the files you need to complete each exercise, but with critical code missing; it will be your job to write this code, applying what you've learned during the corresponding lecture.

The bookstore will serve to frame these exercises into a coherent whole, and it is built atop the SAP Hybris B 2C Accelerator. An Accelerator is a set of extension templates many of our customers use as a starting point for their sites, just as we will use one as the basis for your bookstore. You will create Accelerator-based extensions, following a best-practice, layered approach, extend the data model, localize your data, configure a catalog, set up import/export of its data, etc. Exciting stuff!



The files you will need to complete each exercise will be copied into your Accelerator-based extensions by ant scripts we've written for you. Located in the Training Lab Tools directory, these ant scripts also create verification scripts you will run within the Hybris Administration Console (more on that later). And if you don't have enough time to complete an exercise and wish to move on to the next one, solution ant scripts copy in a working solution that allows you to do so.

Training Labs Tool

This automated tool both sets the groundwork for working on an exercise, and defines scripts allowing you to verify your solution. The core of this tool are two Ant files, which defines targets for each exercise; before working on an exercise (other than the first one), you will invoke the appropriate Ant target to prepare your environment, and then complete the exercise. The second Ant file defines targets for each exercise that *overwrite* your solution with our (correct) one — you should use this only if you wish to move on to the next exercise without completely finishing the current one. You may verify your solution at every step using Groovy scripts defined by an ant target (of course, these only test those parts of the solution that can be verified automatically).

Exercises

The exercises will lead you through the development of a simple bookstore. You will edit Java code, create and modify configuration files, configure import/export (ImpEx) files, and learn to use the following SAP Hybris tools: the Hybris Administration Console (HAC), Hybris Management Console (HMC), Backoffice, Web Content Management Console (WCMS), and Administration Cockpit.

The first exercise simply covers the installation of the SAP Hybris Accelerator. In the exercises that follow, you will develop and configure the accelerator to meet the requirements of our bookstore.

Verification

You may verify the solution that you develop for each exercise by invoking a Groovy script. These scripts are imported into the commerce suite by

the **Start_the_Training** Ant task. In a few cases, it is not possible to check your solution programmatically; we'll let you know in the exercise instructions, and you should then verify the solution manually.

Contents

- Introduction
 - Training Labs Tool
 - Exercises
 - Verification
- Prerequisites
 - Installing Java
 - Installing IDE (Eclipse)
- Exercise 1 - Installation
 - Goal
 - Instructions
 - 1.1 Installing the Hybris Accelerator
 - 1.2 Generating New Storefront Based on the Hybris Accelerator
 - 1.3 Set Up the Test Environment
 - 1.4 Development Environment Setup
 - Verify
 - Recap
- Exercise 2 - Data Modeling
 - Goal
 - Instructions
 - Preparation
 - Exercise 2.1
 - Exercise 2.2
 - Exercise 2.3
 - Exercise 2.4
 - Update Hybris
 - Verify
 - Recap
- Exercise 3 - Product Modeling (Classification)
 - Goal
 - Instructions
 - Preparation
 - Exercise 3.1
 - Exercise 3.2
 - Exercise 3.3
 - Exercise 3.4
 - Verify
 - Recap
- Exercise 4 - ImpEx
 - Goal
 - Instructions
 - Preparation
 - Exercise 4.1
 - Exercise 4.2
 - Exercise 4.3
 - Verify
 - Recap
- Exercise 5 - Flexible Search
 - Goal
 - Instructions
 - Preparation
 - Test-Driven Development
 - Exercise 5.1
 - Exercise 5.2
 - Exercise 5.3
 - Exercise 5.4
 - Exercise 5.5
 - Test
 - Verify
- Exercise 6 - Services
 - Goal
 - Instructions
 - Preparation
 - Test-Driven Development
 - Exercise 6.1

- Exercise 6.2
 - Exercise 6.3
 - Exercise 6.4
 - Test
 - Verify
 - Recap
 - Exercise 7 - Commerce Services and Facades
 - Goal
 - Instructions
 - Preparation
 - Exercise 7.1
 - Exercise 7.2
 - Verify
 - Recap
 - Exercise 8 - Validation
 - Goal
 - Instructions
 - Preparation
 - Exercise 8.1
 - Exercise 8.2
 - Verify
 - Recap
 - Exercise 9 - WCMS
 - Goal
 - Instructions
 - Preparation
 - Exercise 9.1
 - Exercise 9.2
 - Exercise 9.3
 - Exercise 9.4
 - Verify
 - Recap
 - Exercise 10 - Search and Navigation
 - Goal
 - Instructions
 - Preparation
 - Exercise 10.1
 - Exercise 10.2
 - Exercise 10.3
 - Exercise 10.4
 - Verify
 - Recap
 - Exercise 11 - Security
 - Goal
 - Instructions
 - Preparation
 - Exercise 11.1
 - Exercise 11.2
 - Verify
 - Recap
-

Prerequisites

These are the necessities!

The Accelerator training is very intensive so in order to participate in it **every student must have Java installed**:

- Installing Java
- Installing IDE (Eclipse)

Installing Java

If you do not have a JDK installed, download the latest version of **JDK 8 - 64 bit** from Oracle. Make sure you're installing a JDK, and not a JRE.

1. Select the JDK for your operating system and follow the installation instructions

2. If not already done, set your JAVA_HOME environment variable to point to the JDK directory
3. Open a command line and type **javac -version**. You should see something similar to

```
~ $ java -version
java version "1.8.0_74"
Java(TM) SE Runtime Environment (build 1.8.0_74-b02)
Java HotSpot(TM) 64-Bit Server VM (build 25.74-b02, mixed mode)
```

Figure: Java Version in Console

For a list of other system requirements, please refer to [System Requirements - Release 6](#).

Installing IDE (Eclipse)

We have provided you with an IDE called STS, based on the Eclipse IDE and tuned for using Spring. We highly recommend you use this IDE for your exercises.

If you insist on using your own IDE, then you're responsible for configuring it appropriately, as your instructor may not have experience with it.

A few pointers:

On Windows, make sure your IDE is x64. (Assuming you're running Windows x64. If you're not, then your machine doesn't have enough memory for this class, so get one that can access 8 Gb of RAM.)

Allocate enough memory for the IDE in its configuration file. If you were using Eclipse, then you'd modify eclipse.ini to use -Xmx1024m (instead of 256m).

Configure your JAVA_HOME environment variable to point to your Java 8 installation.

[Back to Contents](#)

Exercise 1 - Installation

Goal

In this exercise, you will set up your first storefront based on the Hybris B2C Accelerator.

After installing and configuring its projects, you will import them into an IDE.

Instructions

If you have not yet installed java 1.8 on your machine, here are the [instructions](#). The Hybris Commerce Suite and the Spring Source Tool (STS) will be provided to you on a USB stick at the start of the training. STS is an Eclipse-based IDE well-suited to implementing Spring-based applications.

You don't have to use the IDE we provide for you. However, if you choose to use another IDE, such as IntelliJ, you have to execute training-related ant targets (such as the 'preparation' steps) from the command line. You will get an error if you try to run the target from within IntelliJ, since it cannot correctly resolve Hybris project dependencies.

That's why we strongly recommend you use the pre-configured STS we provide for you; you'll have an easier time with the training.

You should now follow the instructions in the following **three** sub-exercises.

Java 1.8

Note that the Hybris platform currently runs on Java 1.8. If you have other versions installed, your IDE might be configured to compile and run using a version other than 1.8. Change the settings of your IDE so that it compiles using Java 1.8.

Before we start, a quick word of caution:

Read Instructions Carefully

The following exercise may be a long one, but it's very important you carefully follow each and every step. You should also follow the proposed naming conventions in all exercises *to the letter*. You wouldn't have to do that in a real-life project, but our exercise setup and verification scripts assume you've named your classes and components exactly as given in the exercises. And since it's really important that you pay attention to this warning, let's say it again, this time in a really big font:

Carefully follow each and every step of the exercises!

Okay, so you got that, right? 😊

1.1 Installing the Hybris Accelerator

1. Unzip the content of **hybrisCommerceDevTrainingPart1.zip** into a new directory, which we will refer to as **YOURPATH** from now on. If you're using Windows, make sure you unzip the suite in a directory close to the root of your filesystem, and avoid spaces in the name; for example **C:\training**. (Windows has a maximum path length of 256 characters, and both the STS IDE and the Hybris platform contain files whose total path lengths are close to that value, so please unzip to a directory as close to the drive root as possible.)

You are going to install the Hybris Accelerator using a recipe that we have already created for you. Open a terminal window and navigate to **YOURPATH/workspace/ installer** and execute the following command:

- On Windows: **install.bat -r commerce_developer_1**
- On Linux or Mac: **./install.sh -r commerce_developer_1**

The recipe contains all the information needed to install and set up our environment appropriately, such as necessary plugins, required extensions, and the properties we use in our configuration. For further information about recipes, and guides how to create your own, please refer to [CreatingInstallerRecipes](#).

This recipe is not part of the Hybris Commerce Suite. It was created specifically for this training class.

Alternatively, we could perform this installation using an **ant** build. The steps are the following:

1. Set Ant and Hybris platform environment variables – open a terminal window and navigate to **YOURPATH/workspace/hybris/bin/platform** and execute:
./setantenv.sh (on OSX or Linux) or **setantenv.bat** (on Windows)
2. Run **ant clean**. This will prompt you for a configuration template – press enter to specify the default setting (**develop**).
3. Modify, according to your needs, **localextensions.xml** and **local.properties**, located in the **config** folder. The **build.gradle** file under **YOURPATH/workspace/ installer /recipes/commerce_developer_1** directory contains all the information about local extensions and local properties you need to add to your files.
4. Execute **ant all** in the **platform** directory to apply your changes to the server.

1.2 Generating New Storefront Based on the Hybris Accelerator

1. Set Ant and Hybris platform environment variables in the current terminal by navigating to **YOURPATH/workspace/hybris/bin/platform** directory and executing **setantenv.bat** (on Windows) or **./setantenv.sh** (on OSX or Linux – the period operator indicates the script should run in the current shell). Execute the **ant modulegen** command, select the **accelerator** template (it's the default, so hit **Enter**), give your module the name **bookstore** and put it in package **my.bookstore**

```

modulegen:
  [input]
  [input] Please choose a template for generation.
  [input] Press [Enter] to use the default value ([accelerator],
b2baccelerator, telcoaccelerator, commercewebservices)
accelerator
  [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]
bookstore
  [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]
my.bookstore

```

Naming

Remember that this training's setup scripts and verification mechanism all assume that you have entered **bookstore** as the extension module name and **my.bookstore** as the package in the above step.

Please do not choose different names, as this will cause you problems later in the training.

After this step you should have 7 new extensions in the **YOURPATH/workspace/hybris/bin/custom** directory that adhere to your project's namespace.

- .../custom/bookstore/bookstorefulfilmentprocess
- .../custom/bookstore/bookstorecore
- .../custom/bookstore/bookstoreinitialdata
- .../custom/bookstore/bookstorefacades
- .../custom/bookstore/bookstoretest
- .../custom/bookstore/bookstorestorefront
- .../custom/bookstore/bookstorecockpits

2. Follow the instructions from the modulegen output about adding the new bookstore extensions. However, you do not need to include the full path to each extension; instead, follow the format used by the existing extensions in localextensions.xml.

There is one exception! Please do not add the **bookstoretest** extension to localextensions.xml – this extension contains large data sets, such as storefronts (electronics and apparel), and importing these would significantly lengthen the time needed for initialization. Not including it in localextensions.xml will save you a lot of time during this training.

```

[echo] Next steps:
[echo] 1) Add your extension to your C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\config\localextensions.xml
[echo] <extension dir="C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\bin\custom\bookstore\bookstorefulfilmentprocess">
[echo] <extension dir="C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\bin\custom\bookstore\bookstorecore">
[echo] <extension dir="C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\bin\custom\bookstore\bookstoreinitialdata">
[echo] <extension dir="C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\bin\custom\bookstore\bookstorefacades">
[echo] <extension dir="C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\bin\custom\bookstore\bookstoretest">
[echo] <extension dir="C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\bin\custom\bookstore\bookstorecockpits">
[echo] <extension dir="C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\bin\custom\bookstore\bookstorestorefront">
[echo] 2) Remove the following extensions from your C:\hybrisTraining\hyCon-dev1-win64-6.0.0.0\workspace\hybris\config\localextensions.xml
[echo] yacceleratorfulfilmentprocess,yacceleratorcore,yacceleratorinitialdata,yacceleratorfacades,yacceleratoretest,yacceleratorstorefront
[echo] 3) Make sure the applicationserver is stopped before you build the extension the first time.
[echo] 4) Perform 'ant' in your hybris/platform directory.
[echo] 5) Restart the applicationserver
[echo]
BUILD SUCCESSFUL
Total time: 1 minute 13 seconds

```

Figure: modulegen Instructions

Did you remember to remove the **y extensions** from the localextensions.xml file? That was step 2 in the instructions above... however, since you're using our recipe, you'll have found only **yacceleratorstorefront**. Remove it (and don't remove **ycommer**

cewebservices – it wasn't in the list).

3. Add following entries to **local.properties** file

```
website.bookstore.http=http://bookstore:9001
website.bookstore.https=https://bookstore:9002
```

This will (among other things) assign your application to the main context, i.e. when you eventually (many steps later) enter **localhost:9001/** in your web browser, the bookstore homepage will appear instead of the Hybris Administration Console (HAC). Our recipe already rebound the HAC to *localhost:9001/hac*.

4. To simplify your development with the accelerator, let's modify your operating system's **hosts** file:

Modifying the hosts file allows Hybris to determine which site to send the request to based on the URL.

If you are on a Windows machine, the file is located here: **C:\Windows\System32\drivers\etc\hosts**

On a Unix machine (Mac or Linux), the file is located here: **/etc/hosts**

Add the following alias to the hosts file (As the last line of the file)

```
127.0.0.1 bookstore
```

Access to the hosts file

A permission issue may prevent you from directly modifying your hosts file. If you have administrator access on your computer, you may:

On Windows, open notepad (or even better, notepad++) in administrator mode by right-clicking on the application's icon and selecting "Run as administrator". Then open the hosts file from within the editor and perform the change.

On a Mac or Linux computer, you can edit the file directly in the terminal window with `sudo nano /etc/hosts`. After responding to the prompt with your administrator password, save your change with `ctrl-O` and quit with `ctrl-X`. (Of course, you could also just use `vi`...)

No Administrator rights on your machine?

You will have to live with the following uncomfortable workaround - change these properties in the local.properties file:

```
website.bookstore.http=http://localhost:9001?site=bookstore
website.bookstore.https=https://localhost:9002?site=bookstore
```

5. Execute **ant all initialize** in the platform directory to compile your new extensions with the Hybris Commerce Suite and initialize it.

It's going to take time (about 6 min) so go for a short coffee break 😊

6. Start the server – open a terminal and navigate to **YOURPATH/workspace/hybris/bin/platform** and execute:

./hybrisserver.sh (on OSX or Linux) or **hybrisserver.bat** (on Windows)

Once the commerce suite has started successfully, you'll see an info message in your console telling you that the server has started up and how long it took.

Check that you can access the admin console at bookstore:9001/hac/ (if you couldn't modify your hosts file, use localhost:9001/hac/). You should see the **Hybris Administration Console (HAC)**. You can log into it with username **admin** and password **nimda**.

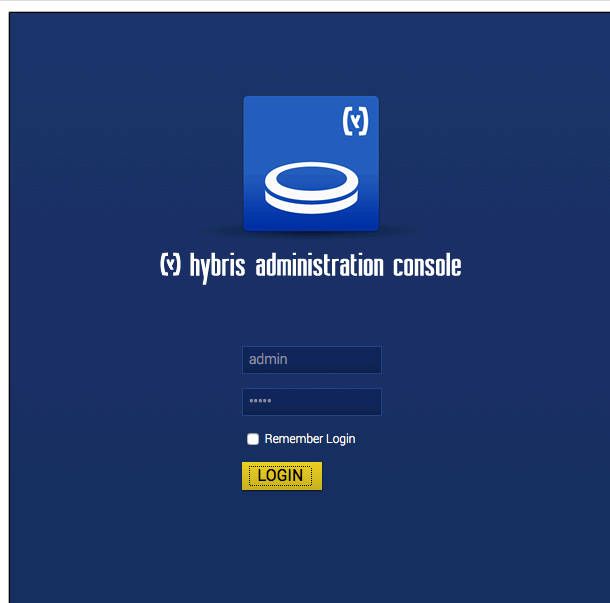


Figure: hybris Administration Console

You can use the HAC to check if you have installed the right extensions in your platform: go to Platform and then Extensions, where you can see a list of the extensions currently used by the Hybris installation on your system.

 The image shows the 'Extensions' page within the Hybris Administration Console. The top navigation bar includes 'Platform', 'Monitoring', 'Maintenance', and 'Console'. The 'Platform' tab is selected, and the 'Extensions' sub-tab is highlighted with a red box. A red arrow points from the 'Platform' tab to the 'Extensions' sub-tab. The page displays a table of installed extensions with columns for Name, Version, Core, HMC, and Web. The table lists 15 extensions, all with version '6.0.0.0-SNAPSHOT'. The 'Core' column shows green checkmarks for all extensions. The 'HMC' column shows green checkmarks for 'acceleratorcms', 'acceleratorweb', 'basecommerce', 'bmecat', and 'bookstorecore', and red X marks for others. The 'Web' column shows webroots for several extensions. A search bar is located at the top right of the table. On the right side of the page, there is a sidebar with a search bar and a list of details for each extension: Version, Contained extension module, and Webroot. Below this, there is a section titled 'See also in the hybris Wiki' with a link to 'About Extensions'.

Name	Version	Core	HMC	Web
acceleratorcms	6.0.0.0-SNAPSHOT	✓	✓	
acceleratorfacades	6.0.0.0-SNAPSHOT	✓	✗	
acceleratorservices	6.0.0.0-SNAPSHOT	✓	✓	/acceleratorservices
acceleratorstorefrontcommons	6.0.0.0-SNAPSHOT	✓	✗	
acceleratorwebservicesaddon	6.0.0.0-SNAPSHOT	✓	✓	
addonsupport	6.0.0.0-SNAPSHOT	✓	✗	
admincockpit	6.0.0.0-SNAPSHOT	✓	✗	/admincockpit
advancedsavedquery	6.0.0.0-SNAPSHOT	✓	✗	
backoffice	6.0.0.0-SNAPSHOT	✓	✗	/backoffice
basecommerce	6.0.0.0-SNAPSHOT	✓	✓	
bmecat	6.0.0.0-SNAPSHOT	✓	✗	/bmecat
bmecat-hmc	6.0.0.0-SNAPSHOT	✓	✓	
bookstorecockpits	6.0.0.0-SNAPSHOT	✓	✗	
bookstorecore	6.0.0.0-SNAPSHOT	✓	✓	

Figure: installed extensions inside the HAC

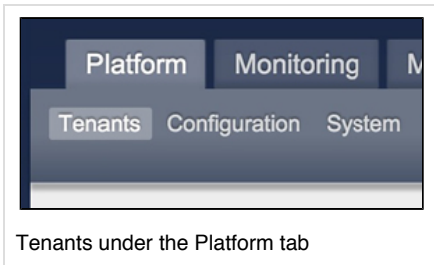
7. Don't stop the server yet... there's more!

1.3 Set Up the Test Environment

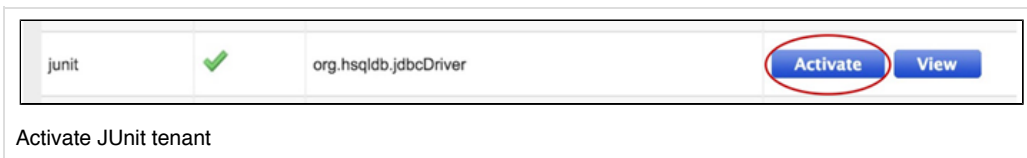
We will be applying a Test-Driven Development (TDD) methodology to code development in these exercises. That means we need to set up the **JUnit tenant**, where we'll be running our tests from Eclipse.

Still in the HAC, follow the following steps to set up the tenant:

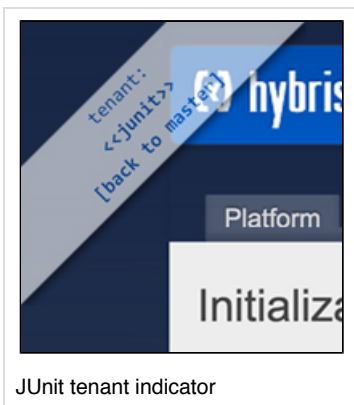
1. Select the Tenant tab under Platform.



2. Activate the JUnit tenant.



3. You'll be taken directly to the Initialize page for the JUnit tenant. The HAC lets you know you're looking at the tenant with a diagonal banner (which has the link to return to the master tenant's HAC).



4. To initialize the JUnit tenant, just click on the **Initialize** button. When that's done, you're ready to test.
5. You may now stop the server (**Ctrl+C**).

1.4 Development Environment Setup

Before doing the exercises, there are some steps that you'll need to take to prepare your development environment.

Path to the Provided IDE

The STS provided for you is at **YOURPATH/sts-bundle/STS.app** (or **STS.exe** on Windows). Please do **not** relocate the sts-bundle directory (for example, to **C:\Program Files** or **/Applications**).

If you're using OS X and you see an error when you try to open the STS IDE, saying that the STS IDE is damaged and can't be opened, you should try either of the following workarounds:

1. Open **System Preferences** and then go to **Security & Privacy**. Under the **General** tab, select the **Anywhere** radio button below *Allow apps downloaded from:*.

or

2. Open a terminal window and
 - a. run the following command:

```
sudo spctl --master-disable
```

- b. Open the STS IDE. It should open successfully.
- c. Run the following command to return your system to its normal state:

```
sudo spctl --master-enable
```

Set up your development environment:

1. First, manually import the **platform** and **config** projects into STS. All the projects you will be importing are located under **YOURPATH/hybris/bin**, except **config**, which is located here: **YOURPATH/hybris/config**.

Import Into Eclipse

If you don't know how to import a project into Eclipse (or STS), please take a look at [ImportIntoEclipse](#).

While importing, make sure that the *Copy projects into workspace* checkbox is **not** checked.

2. You then need to import all the extensions listed in `localextensions.xml`. While you can do so manually (using the same procedure as for **platform** and **config**), we encourage you to save some time and instead use the **hybris** plug-in in STS to do so.

We have added a plug-in to the STS IDE in order to make this importing task easier for you. This plugin is developed by SAP Hybris. At this point all you need to do is to go to the menu bar and under **[y] hybris**, select **Synchronize Projects with yPlatform**. It will take a while to import and build all these projects, but it saves you a lot of effort. However, it **only works** if you've previously imported the **config** and **platform** extensions!

If you are using your own IDE,

- you will need to manually import every extension listed in `localextensions.xml`
- you will also need to import the **TrainingLabsTool** project. It's located at **YOURPATH/workspace/TrainingLabsTool**
- configure your IDE to use the Ant provided in the Hybris Commerce Suite. You can find it at **YOURPATH/workspace/hybris/bin/apache-ant-1.9.1**

Here are all the projects that will be imported by the **hybris** plugin. The astute student will notice that there are more projects here than show up in `localextensions.xml`. That's because some of them are referenced by other projects, and loaded dynamically. If you're importing projects manually, you need at least those in `localextensions.xml`, but it won't hurt to import the extra ones below:

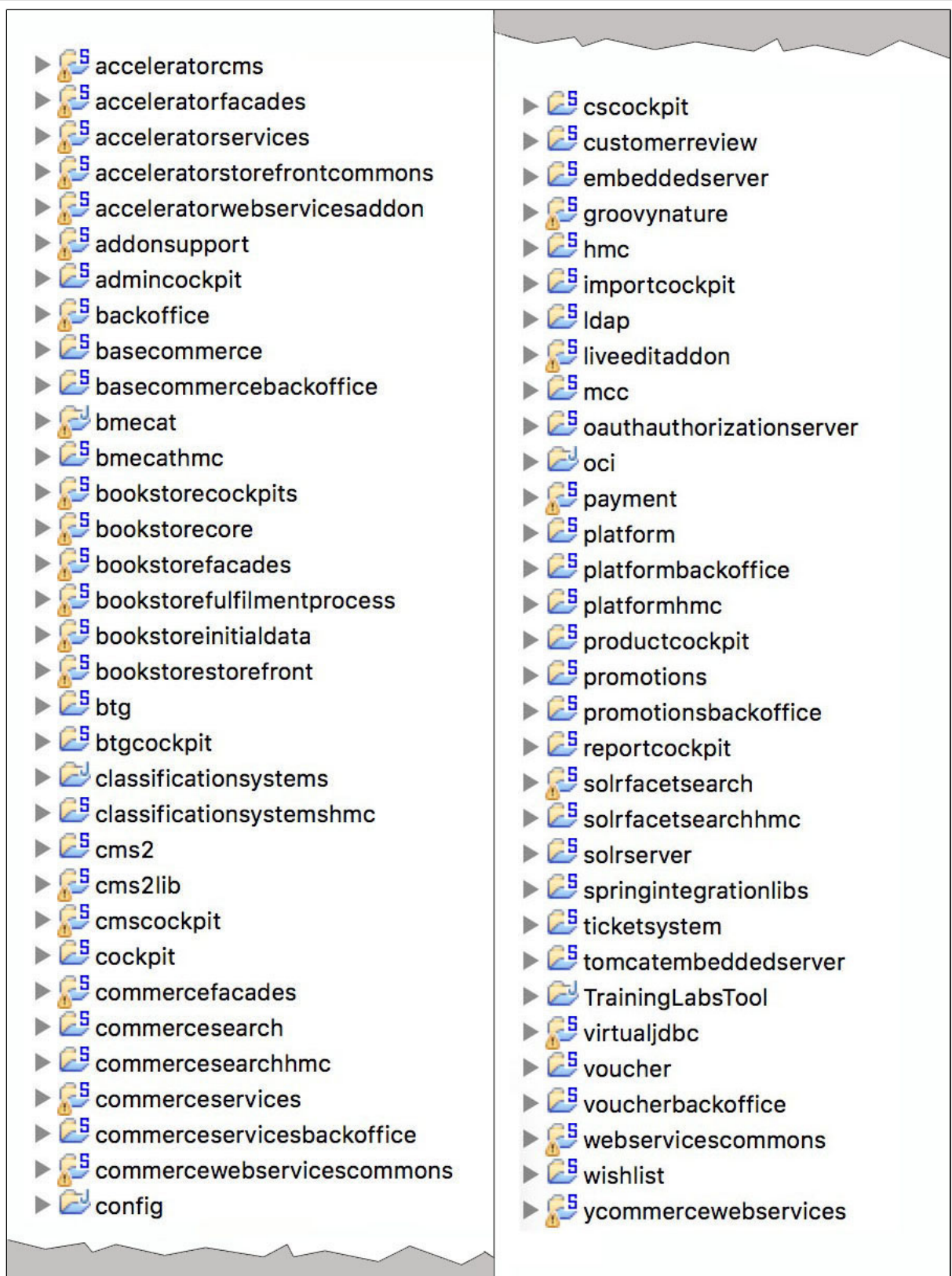


Figure: projects to import

When you open STS, you should see one project called TrainingLabsTool in the Project Explorer. If you cannot see the content of this project, you just need to refresh it.

You should set the text file encoding in STS to UTF-8. To do that go to Window -> Preferences -> General -> Workspace and in "Text file encoding" section set Other: UTF-8.

3. Run **Start_the_Training** ant target, under the **TrainingLabsTool - preparations** Ant build file, to set up the training tool.

If you don't see the Ant view (perhaps you've switched to the Java perspective), then remember to display it with *Window -> Show View -> Ant*. If the TrainingLabsTools build files don't correctly appear there, you might need to refresh TrainingLabsTool in the Project Explorer. You can also manually add the two build files from TrainingLabsTools into the Ant view: **lab_exercises_preparation.xml** and **lab_exercises_solution.xml**

Verify

By running **Start_the_Training**, you have, among other things, installed verification scripts into your Hybris Commerce Suite. Now that you have completed the first exercise, you can use one of these scripts to verify your the state of your Hybris installation.

Do the following in the Hybris Administration Console (HAC):

1. Start the Hybris server.
2. Open a browser and navigate to bookstore:9001/hac. This is the URL for the Hybris Administration Console in this training (the default URL for HAC is localhost:9001/). Enter **admin** as username and **nimda** as password to log in.
3. Select the **Console** tab and click on **Scripting Languages**. Under the **Browse** tab, double-click on **groovy** to display the exercises. Right-click on **verifyExercise1** and select **load**.
4. That automatically sends you to the **Edit Statement** tab, populated with the contents of the script, which you may now run by clicking on the **execute** button.
5. You're now shown the **Result** tab, which unfortunately doesn't actually have anything useful to say; you need to click on the **Output** tab to see the script's output. Before proceeding, make sure everything is **OK**.

Solution

If you created the bookstore modules by running the solution, remember that you still have to manually **import** your projects into STS.

Recap

In this exercise, you learned how to install an accelerator in Hybris Commerce Suite. You have installed the Training Labs Tool and verified your accelerator installation. You are now prepared to do the rest of the exercises.

[Back to Contents](#)

Exercise 2 - Data Modeling

Goal

This exercise is about the basics of data modeling in the Hybris Commerce Suite. Through this exercise, you will learn how to define items and relations based on an already-defined data model.

Instructions

First take a look at the UML diagram below, showing the data model you will be working with. The colors represent the status of a data type (item type) in the system.

	the corresponding item is available OOTB in the Hybris platform.
	a complete definition of this item is provided as part of the exercise setup.
	items and relations you will have to define.

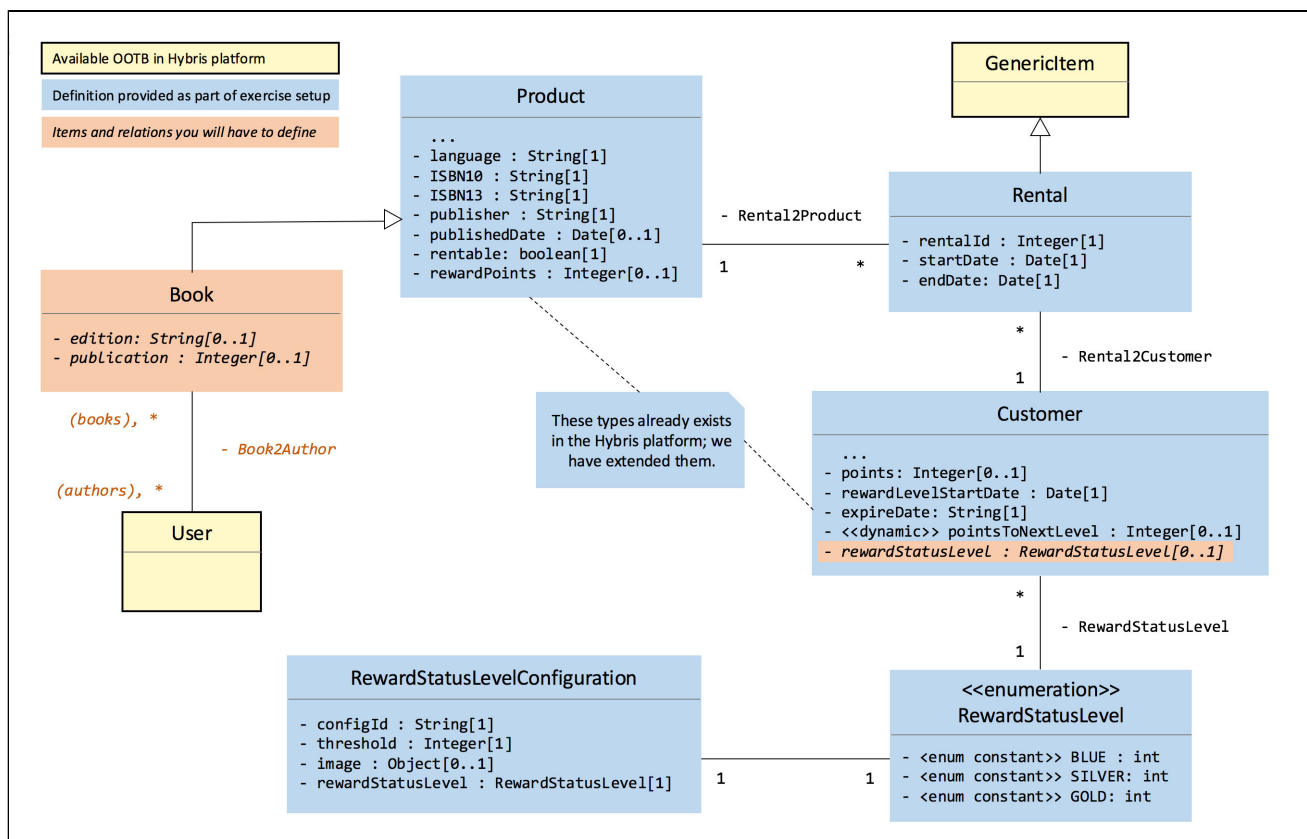


Figure: UML Diagram of Bookstore's Data Model

Preparation

To prepare your system for doing this exercise, run the **Exercise02_Data_Modeling-prepare** Ant target.

Now you can start working on exercise 2. You will be editing two files, **bookstorecore-items.xml** and **bookstore-locales_en.properties**, located here:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/resources/bookstorecore-items.xml
```

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/resources/localization/bookstorecore-locales_en.properties
```

Irrelevant Data Types

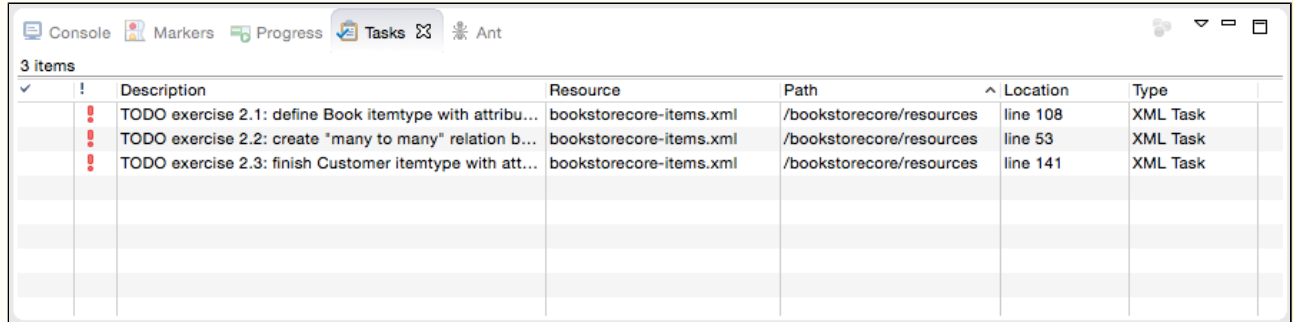
The **bookstorecore-items.xml** file defines item types that are not in the diagram above. These types are not relevant to this exercise, so you may ignore them.

Naming Convention

The item types and their qualifiers defined here will be used in all the exercises for this class. It is therefore very important that you follow the naming convention used in the **bookstorecore-items.xml** file provided for you in this exercise. For example, following the convention, the relation "Book2Author" will have the code "Book2AuthorRelation" and the qualifier for "Book", in this relation, is "books". If you don't follow these conventions, the preparation steps for later exercises will not function correctly, and our verification scripts will incorrectly report errors.

TODO

If an exercise is supposed to be done in a file, it is denoted as a comment tagged *TODO exercise* in the same file. Eclipse's Task view captures such comments. Unfortunately, Task view does not support all types of files. In this view, you will see the tasks in `.java` or `.xml` files but not `.properties` and `.impex` files.



The screenshot shows the Eclipse IDE's Task view. At the top, there are tabs for Console, Markers, Progress, Tasks (selected), and Ant. Below the tabs, it says '3 items'. The main area contains a table with the following data:

	Description	Resource	Path	Location	Type
✓	! TODO exercise 2.1: define Book itemtype with attribu...	bookstorecore-items.xml	/bookstorecore/resources	line 108	XML Task
!	! TODO exercise 2.2: create "many to many" relation b...	bookstorecore-items.xml	/bookstorecore/resources	line 53	XML Task
!	! TODO exercise 2.3: finish Customer itemtype with att...	bookstorecore-items.xml	/bookstorecore/resources	line 141	XML Task

Figure: task view after running Exercise02-prepare

Exercise 2.1

Define the *Book* item type in `bookstorecore-items.xml`. Note that *Book* is a subtype of *Product*.

Exercise 2.2

Create a many-to-many relation between *Book* and *User* as shown in the UML diagram. Use **Book2AuthorRelation** as the code for this relation, and **books** and **authors**, respectively, as qualifiers for *Book* and *User* in this relation.

Exercise 2.3

The *Customer* item type is available out-of-the-box in the Hybris Commerce Suite, and we have added attributes to it. You will see these additional attributes defined in `bookstorecore-items.xml` (which was copied to your resources folder by the lab setup Ant task), and you need to complete the definition of this item type, using the UML diagram as a reference.

Exercise 2.4

Provide a localized name and description for the *edition* attribute of *Book*. (Be careful: you're localizing the edition property's meta-data, not making *edition* a localized property.)

Update Hybris

1. In the terminal, navigate to the Hybris directory and run **ant**. Then restart the Hybris server.
2. Go to the **HAC** on your browser and update the platform. Note that you need to check only two options: "Update running system" and "Localize types". You should update the **junit** tenant, with the same options as the master tenant, in order to keep your testing environment updated about the changes on Type System.

Update Options

If you don't know why you should choose these options, please take a look at [Initializing and Updating the Hybris Commerce Suite](#) before proceeding.

View Your Changes in backoffice

In case you are curious, you may view some of the changes that you made to an already-existing type, like *Customer*, in the Hybris Backoffice. To do so, log in to the backoffice, and go to *Types*, under *System*. Then search for the type *Customer*. Click on *Customer* and navigate to the *Properties* tab. The new attributes should appear here, using the labels you defined in Exercise 2.3.

Verify

To verify your solution, go to the **HAC** and run the "verifyExercise2" script to check your changes.

Solution

You can find the solutions to every exercise inside the **TrainingLabsTool** project. There, under **resources**, you will see a directory for each exercise, and inside each one, a **solution** directory containing the solution files.

Recap

In this exercise you have learned to define new item types and relations based on a data model. You have also learned how you can localize item types and their attributes.

[Back to Contents](#)

Exercise 3 - Product Modeling (Classification)

Goal

In this exercise, you will see how to use the Hybris platform's classification system.

Instructions

The classification we will use in this exercise is based on the following diagram. The blue boxes already exist; you will create or complete the red ones.

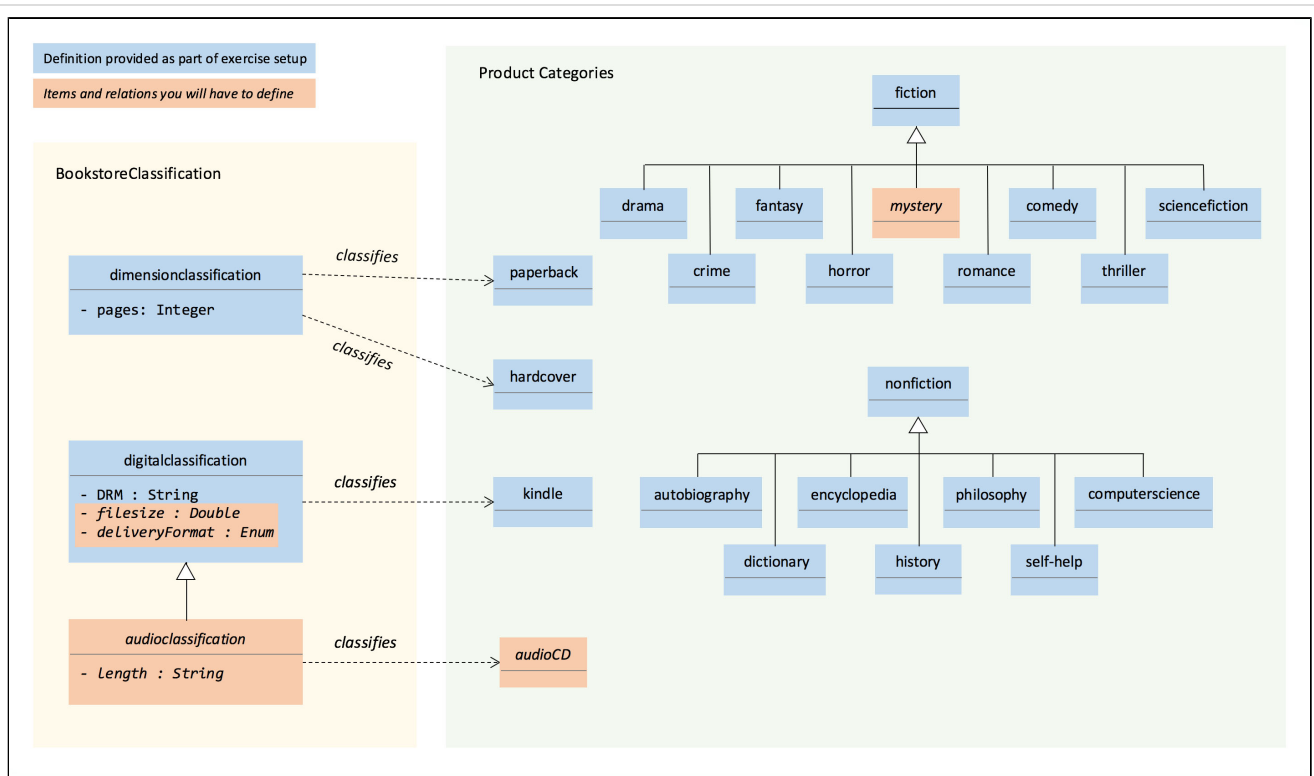


Figure: Classification UML Diagram

Preparation

To prepare your system for this exercise, run the **Exercise03_Product_Modeling-prepare** Ant target. In short, it creates all the blue part of the diagram above.

First Exercise 2

The previous exercise is a prerequisite of this exercise. So if you want to solve without finishing exercise 2, please run the solution of exercise 2 first, before running preparation for exercise 3

Now you can start working on exercise 3. This exercise will differ from the previous one in that you will use the **backoffice** to make your changes instead of modifying code or properties files.

Exercise 3.1

Create the **mystery** category as shown in the diagram. You should add it to the staged bookstore product catalog (do you know why?).

Synchronize the category from staging to online. Expand **Catalog**, click on **Catalog Versions**, select **Staged Bookstore Catalog** and click on the **Catalog Synchronization** icon located above the tabs (it's the one on the right).

Exercise 3.2

Complete the **digitalclassification** category. That is, do the following:

1. Create the **filesize** category feature.
2. Create the **MB** classification unit. When creating a classification unit (here and in the next steps), please provide **all** the attributes including **Symbol** and **Type**. Type is a free-form field, but it is still required; use Number for this unit (and String for format, below).
3. Assign the **filesize** feature to **digitalclassification**, using **MB** as its unit.
4. Now create the **deliveryFormat** category feature.
5. Create the **format** classification unit it will use. Don't forget to fill in all the fields!
6. Then create the following feature values: **MP3**, **MP4**, and **PDF**.
7. Assign the **deliveryFormat** feature to **digitalclassification**, using **format** as its unit, **Valuelist** as its type, and **MP3**, **MP4**, and **PDF** as its Feature Descriptor Values.

Exercise 3.3

Create the **audioclassification** classifying category and its features, taking into account the relation it has to **digitalclassification**. The **length** feature will need a **duration** unit, whose symbol is **HH:mm:ss**. Remember to set its type (String).

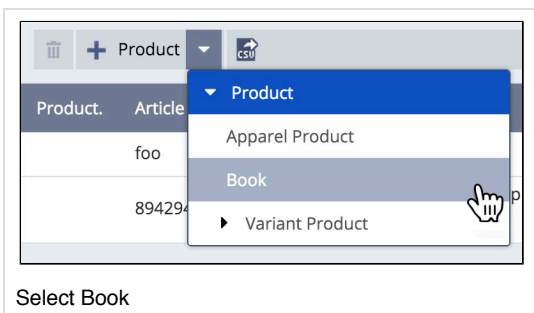
Create the **audioCD** product category, and assign **audioclassification** as its Supercategory (parent category). Synchronize the categories again.

Exercise 3.4

Create a book in the system, using the table below.

If you're using the backoffice, you will notice that after selecting Products in the left-hand pane, you will see at the top of the page: **+ Product**.

1. Next to the word Product, you will see a downward-facing triangle. Click on it to get a small pop-up window showing a right-facing triangle and the word **Product**.
2. Click on this triangle to expand the list of Product subtypes, and select **Book**. A dialog box will appear, titled *Create New Book*.



You should use the following information:

Property	Value
Article Number (Code)	8942944779
Approval	approved
Catalog version	bookstoreProductCatalog : Staged
Identifier (Title)	The Strange Case of the Disappearing Hybris Website
ISBN10	8942944779
ISBN13	3338942944779
Publisher	Pinniped
Rentable	true
Language	en
Rental Points (reward points)	10
Product Category	mystery, audioCD
Length	1:08:20
DRM	none
Delivery Format	MP3
File Size	620 MB

Verify

To verify your solution, go to the **HAC** and run the script **verifyExercise3**.

Recap

In this exercise, you learned how to define a classification system and how to create classifying categories and their features. Moreover, you have associated classifying categories to product categories, and created a product that uses the properties gained through the classification system.

[Back to Contents](#)

Exercise 4 - ImpEx

Goal

This exercise will show you how to import data using ImpEx, the out-of-the-box import/export tool of the Hybris Commerce Suite. You will learn about the basics of creating and updating data in the platform using ImpEx files.

Instructions

Preparation

To prepare your system for this exercise, run the **Exercise04_ImpEx-prepare** Ant target.

Now you can start exercise 4; each step will have you modify the files **products.impex** and **products-classifications.impex**. Look for the comments labelled *TODO exercise*.

The exact paths of these files are:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/
import/sampledatalogs/productCatalogs/bookstoreProductCatalog/products.impex

${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/
import/sampledatalogs/productCatalogs/bookstoreProductCatalog/products-classifications.impex
```

To see the tasks and make the required modifications for each exercise, open the relevant ImpEx file with your favorite text editor or within Eclipse (STS). If you're using Eclipse, remember to refresh your view after running Ant.

Exercise 4.1

Modify "**products.impex**" so that it will add/update 2 new authors with following information:

1. uid:Charlotte T. Bron, name:Charlotte T. Bron
2. uid:Dale Carnaby, name:Dale Carnaby

Exercise 4.2

Add the appropriate header and value lines to **products.impex** to create a book with the following values:

code	1013742419
name	Withering Heights
categories	drama,audioCD
\$approved	[default] (look at the macros)
catalog version	Staged (look at the macros)
rentable	true
reward points	12
ISBN10	1013742419
ISBN13	3331013742419
language	\$lang
publisher	hybris Classics Library
authors	use the authors added in exercise 4.1

Now go to the **HAC** and import the contents of this ImpEx file.

Exercise 4.3

Since the book we are creating in ImpEx is assigned to at least one classification category (audioCD), we need to supply values to the properties it gains from this association. We will do so inside the **products-classifications.impex** file, updating the book with the following *attribute:value* information:

DRM	8675210DRM
filesize	4
format	MP3
length	2:00:00

Now go to the **HAC** to import the contents of this ImpEx file.

By now, you added the book to the Staged catalog. To apply the changes you made to the online (active) catalog, go to the **backoffice** and synchronize the product catalog.

Verify

To verify your solution, go to the **HAC** and run the **verifyExercise4** Groovy script.

Recap

In this exercise you saw how to insert/update data in the Hybris Commerce Suite using ImpEx. ImpEx can also be used to export data, but this is outside the scope of this training.

[Back to Contents](#)

Exercise 5 - Flexible Search

Goal

In this exercise, you will learn how to define a DAO class to encapsulate your data access. You will also practice using Flexible Search to create queries that retrieve the data you need.

Instructions

You should already be familiar with the syntax and features of Flexible Search. As we will be working with our project's data, you should go back and look at the data model provided in the second exercise. Armed with this knowledge, you will first prepare your system before tackling this exercise. (Not quite slaying dragons, but still exciting, right?)

Preparation

Run the **Exercise05_Flexible_Search-prepare** Ant target. Among other things, this will populate your database with books for you to skim through later.

Now you can start working on exercise 5. You will complete some steps using the **HAC**, and others by editing the **bookstorecore** project. Two steps are supported by TODO comments in **DefaultRentalDao.java** and **bookstorecore-spring.xml**, which you will find here:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/src/my/bookstore/core/daos/impl/DefaultRentalDao.java
```

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/resources/bookstorecore-spring.xml
```

You can also see the description of these exercises in Eclipse's Task view.

Test-Driven Development

We will apply the principles of Test-Driven Development (TDD) in this exercise. Your first task would then be to write a test class that calls the relevant methods in **DefaultRentalDao.java**, and makes sure they return the correct data. To save time, we've done that for you; see the **DefaultRentalDaoTest.java** test class here:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/testsrc/my/bookstore/core/daos/impl/DefaultRentalDaoTest.java
```

Refresh Project Explorer

To see the changes made by the preparation Ant task in Eclipse, you may have to refresh **bookstorecore** in the project explorer.

Exercise 5.1

Define the **customerDao** bean, using the **DefaultGenericDao** class in **bookstorecore-spring.xml**. We will use **customerDao** in later exercises.

Exercise 5.2

Write a Flexible Search query that finds all the currently active rentals for a customer with `uid='john@hybris.com'`. The **uid** is a unique attribute of the **Customer** itemtype. Run this query in the **HAC**.

SQL Query

You can see the actual SQL generated by the Flexible Search query in the HAC, under **SQL Query** tab.

Exercise 5.3

Look at the method `void testGetActiveRentalsForCustomer()` in class **DefaultRentalDaoTest**. It tests that the `getActiveRentalsForCustomer()` method below correctly returns the active rentals for the given customer. By active, we mean that the rental's start date is before the current date, and its end date is in the future. The class populates the database with test data, and provides two lists used in the test: **activeRentalIds** and **inactiveRentalIds** contain the **rentalId** of the customer's active and inactive rentals, respectively. The variable **customer** refers to the test customer the class creates for you. The variable **rentalDao** is wired to the **DefaultRentalDao** instance defined in Spring.

Now complete the implementation of the method `List<RentalModel> DefaultRentalDao.getActiveRentalsForCustomer(CustomerModel customer)`. This method returns all the active rentals for *customer*. Use the query you wrote in the previous exercise as a starting point. It's actually simpler here, since you have the customer item on hand (it is a parameter to the method!).

We want to return only *active* rentals, so we'll compare each rental's start and end dates to today's dates. Remember that if we want to benefit from query caching, we'll need to truncate the date. The `getActiveRentalsForCustomer()` method provides the variables `dayStart` and `dayEnd`, which you can compare to the start and end dates of each rental.

You can access the Flexible Search service inside **DefaultRentalDao**. Take a look at the entry for the **defaultRentalDao** bean in **bookstorecore-spring.xml**. The Flexible Search service is injected into the **defaultRentalDao** bean from the parent bean through inheritance as the variable **flexibleSearchService**.

When you're done writing the code, run the test. In Eclipse, right-click on **DefaultRentalDaoTest**, and select **Run As->JUnit Test**. If it works, you're looking good! If not, fix the problem.

Recall that the slides showed you how to use static constants from the Model class to refer to property names, ensuring that changes to the model will cause a compilation error, alerting you to the problem. While we won't ask you to do this here, keep in mind that we did so in the solution.

Exercise 5.4

Write another Flexible Search query that finds the 5 most-rented books. In other words, the 5 books that were rented out to customers the most often. Run this query in the **HAC** to make sure it produces the expected result.

Exercise 5.5

Look at the method `void testGetMostRentedBooks()` in class **DefaultRentalDaoTest**. It tests that the `getMostRentedBooks()` method below correctly returns the books with the most rentals. It asks for the top 5 books. The class populates the database with test data, and provides a list, **topRentedBooks**, containing the **ISBN10** of the five books with the most rentals. The variable **rentalDao** is wired to the **DefaultRentalDao** instance defined in Spring.

Now complete the implementation of the method `List<BookModel> DefaultRentalDao.getMostRentedBooks(int numberOfBooks)`. It returns the same result as the query of the previous exercise, with the minor difference that the size of this list of most-rented books is passed as an input

argument to the method.

When you're done writing the code, it's time to test, using **DefaultRentalDaoTest**. Keep working until the test works.

Test

In Eclipse, run the JUnit tests you completed in this lab. Right-click on **DefaultRentalDaoTest.java** and select **Run As... JUnit Test**. The success (or failure) of your test will appear in Eclipse's JUnit view.

Verify

To verify your solution, go to the **HAC** and run the script **verifyExercise5**. Note that the verification script only checks for exceptions and does not check the logic of the methods you just implemented.

Recap

This exercise taught you how to write Flexible Search queries to access data inside the hybris Commerce Suite. You have also made use of Flexible Search inside code to retrieve data.

[Back to Contents](#)

Exercise 6 - Services

Goal

This exercise covers the hybris Commerce Suite's service layer. You will learn how to create a service containing business logic pertaining to customer rewards and rentals.

Instructions

Preparation

Begin by running the **Exercise06_Services-prepare** Ant target.

Then proceed with exercise 6; your tasks are outlined in the **bookstorecore** java files **DefaultBookstoreCustomerAccountService.java**, **DefaultRentalService.java**, and configuration file **bookstorecore-spring.xml**, located here:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/src/my/bookstore/core/services/impl/DefaultBookstoreCustomerAccountService.java
```

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/src/my/bookstore/core/services/impl/DefaultRentalService.java
```

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/resources/bookstorecore-spring.xml
```

You can also see the description of these exercises in the Task view of Eclipse.

Test-Driven Development

We will continue using a TDD approach in this exercise. Before writing the code, you will complete the test class provided for you; look for the TODO comments in **DefaultBookstoreCustomerAccountServiceTest.java**, which you will find here:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorecore/testsrc/my/bookstore/core/services/impl/DefaultBookstoreCustomerAccountServiceTest.java
```

Refresh Project Explorer

Remember that, in Eclipse, you may need to refresh the **bookstorecore** project (in the project explorer) to see the files added by the preparation step.

Exercise 6.1

Look at the interface *my.bookstore.core.services.BookstoreCustomerAccountService*, and at its implementing class, *my.bookstore.core.services.impl.DefaultBookstoreCustomerAccountService*. Since the **BookstoreCustomerAccountService** class extends *de.hybris.platform.commerceservices.customer.CustomerAccountService*, we want to take advantage of the configuration of the bean of the same name (**CustomerAccountService**).

Inside **bookstorecore-spring.xml**, make the appropriate changes to the definition of the **defaultBookstoreCustomerAccountService** bean so that it inherits configuration from **defaultCustomerAccountService**.

Exercise 6.2

Complete the implementation of the method *void testUpdateRewardStatusPoints()* in the **DefaultBookstoreCustomerAccountServiceTest** class. You want to test that the *updateRewardStatusPoints()* method below correctly changes the points property of the customer based on the books in the order. The class you're modifying will populate the database with test data: a customer with zero reward points, and an order containing two order entries: one copy of a book with 40 reward points, and 2 copies of a book worth 15 points. You'll need to make sure that the customer has 70 reward points after calling the *updateRewardStatusPoints()* method. The test class provides the variables **customer** and **order**, which refer to the test customer and order created for you. The variable **bookstoreCustomerAccountService** is wired to the **DefaultBookstoreCustomerAccountService** instance defined in Spring.

When you have finished writing the test method, you are ready to code:

Complete the implementation of the partially written method *updateRewardStatusPoints* in **DefaultBookstoreCustomerAccountService**. You should use the total value to update the customer's points.

When you're done writing the code, run the test. In Eclipse, right-click on **DefaultBookstoreCustomerAccountServiceTest**, and select *Run As->JUnit Test*. If it works, you're looking good! If not, fix the problem.

Exercise 6.3

Complete the implementation of the method *void testGetAllCustomersForLevel()* in the **DefaultBookstoreCustomerAccountServiceTest** class. You want to test that the *getAllCustomersForLevel()* method below correctly returns all the customers who have a particular reward level. In your test, you will look for the customers with a reward level of **gold**. The class provides three variables (blue, silver, and gold) which refer to the corresponding enumeration values, and the variable **goldCustomerIds**, which contains the **Uids** of the customers with a gold reward level.

When you have finished writing the test method, you are ready to code:

The method *getAllCustomersForLevel* (in **DefaultBookstoreCustomerAccountService**) needs to return a list of *customers* who have a certain **rewardStatusLevel**. Recall that in exercise 5 you defined **customerDao** bean in **bookstorecore-spring.xml**. You should use this DAO to implement *getAllCustomersForLevel*.

When you're done writing the code, run the **DefaultBookstoreCustomerAccountServiceTest** as a JUnit test again. Your second test needs to pass before you move on.

Exercise 6.4

Implement *getActiveRentalsForCustomer* in **DefaultRentalService**. This method finds all the active rentals for a *customer*. This one will be quick to code; you have already done most of the required work in exercise 5.

Also add the necessary properties (bean injections) to the **defaultRentalService** bean in **bookstorecore-spring.xml**.

Test

In Eclipse, run the JUnit tests you completed in this lab. Right-click on **DefaultBookstoreCustomerAccountServiceTest.java** and select **Run As... JUnit Test**. The success (or failure) of your tests will appear in Eclipse's JUnit view.

Verify

To verify your solution, go to the **HAC** and run the script **verifyExercise6**. Please note that the verification script does not check the business logic, but only the existence of required methods.

Recap

In this exercise, you learned how to create a service that uses existing beans to access data.

[Back to Contents](#)

Exercise 7 - Commerce Services and Facades

Goal

You already have a page that shows the product details. In this exercise, you'll configure your store to show the reward points on this page.

Instructions

Preparation

First, run the **Exercise07_CommerceServicesFacades-prepare** Ant target. It sets the stage for your exercises, including altering the view so that you can eventually see the changes you make to the store.

In this exercise, you'll be working with the following files:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorefacades/resources/bookstorefacades-beans.xml
```

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorefacades/src/my/bookstore/facades/populators/BookstoreProductPopulator.java
```

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorefacades/resources/bookstorefacades-spring.xml
```

Your IDE's Task View will show links to the following steps.

Exercise 7.1

We want to show the reward points on the product details page. Based on the accelerator's architecture, a DTO is used to transfer the data from the facade to its view. Your task is to add **rewardPoints** to this DTO and make sure that it's passed to the storefront.

Add **rewardPoints** to the already-defined DTO *de.hybris.platform.commercefacades.product.data.ProductData*, in **bookstorefacades-beans.xml**. You must run **ant all** to generate the code supporting this integer property in the auto-generated *ProductData* class.

You should now make sure that this property is populated. The **BookstoreProductPopulator** has already been created for you by the preparation step.

Modify this populator so that it also populates the **rewardPoints** property, then run **ant all**.

Exercise 7.2

BookstoreProductPopulator is now complete and ready to use. But it won't be used unless it's associated with a converter. Like the DTO in the previous part, we want to reuse what's already there. In this case it's the **productConverter** bean. Recall how to hook a populator into an existing converter. (And if you can't recall, refer to the lecture slides!)

Make the necessary changes in **bookstorefacades-spring.xml** to add **BookstoreProductPopulator** to the appropriate converter.

To make your changes take effect, **restart your server**.

Verify

To verify your solution, go to the **HAC** and run the script **verifyExercise7**. You will also want to visit your site and view the book product detail page; the reward points are now displayed.

Recap

In this exercise, you learned how to use facades and populators to pass data from the services layer to the front end layer.

[Back to Contents](#)

Exercise 8 - Validation

Goal

In this exercise, you will learn how to validate data in the hybris platform using constraints.

Instructions

You will carry out the exercise tasks below in the **Backoffice**.

Preparation

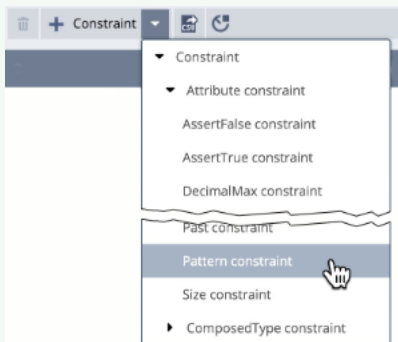
Log into the **Backoffice** (<http://localhost:9001/backoffice/>) using administrator credentials (user **admin**, password **nimda**). Then navigate to **Validation** in the left-side menu (System -> Validation -> Constraints).

Exercise 8.1

Create a Pattern Constraint to check that the ISBN13 attribute has exactly 13 digits. Please use **ISBN13SizeValidator** as the ID for this constraint.

In the Backoffice, notice that after selecting Constraints, the **+** button assumes you want to create an Abstract Constraint. To create a Pattern Constraint:

1. Click on the downward-facing triangle next to the word Constraint, which displays a pop-up showing > Constraint.
2. Click on the right-facing triangle to the left of Constraint, which displays the subtypes of the Abstract Constraint type.
3. Select Pattern Constraint.



Regular Expression

You will need a regular expression to express this constraint. **\d{13}** is such a pattern which means a sequence of only 13 digits. You're also asked to set a pattern flag; select **Case-insensitive**.

Validation Engine

To be able to see and use a newly created constraint, you need to reload the *validation engine*. In the Validation perspective, you can find the **Reload validation engine** button at the top of the main pane (below the search bar).

Exercise 8.2

Create a Dynamic Constraint to check whether the ISBN10 attribute satisfies the following formula:

$$(X_1 + 2X_2 + 3X_3 + 4X_4 + 5X_5 + 6X_6 + 7X_7 + 8X_8 + 9X_9 + 10X_{10}) \equiv 0 \pmod{11},$$

where X_1, X_2, \dots, X_{10} denote the digits of ISBN10 from left to right. That is to say, calculate the numeric value of the equation on the left of the symbol, then divide by 11. If the result of the division is a whole number, the ISBN10 number is valid.

Please use **ISBN10FormatValidator** for the constraint's ID.

Following is the script you can use for this purpose.

Script body

```
char[] digits = getISBN10().toCharArray();
if (digits.length != 10) {
    return false;
}

int sum = 0;
for (int i = 1; i <= digits.length; i++) {
    sum += i * digits[i - 1];
}

if (sum % 11 == 0) {
    return true;
}
return false;
```

Verify

To verify your solution, go to the **HAC** and run the script **verifyExercise8**.

We recommend you to also see the effect of the changes you made in practice. Try to create a new book with an ISBN10 or an ISBN13 that is invalid according to the constraints you created in this exercise.

Recap

In this exercise, you learned how to create two different types of constraints for data validation in the hybris platform.

[Back to Contents](#)

Exercise 9 - WCMS

Goal

This exercise covers defining a base store, and providing a website to act as its storefront.

Your website will need a homepage, which we'll configure in a later exercise to display a banner to certain customers. The WCMS Cockpit will provide you with a structural view of the page, shown in the image below. (You will only be able to see it in the CMS Cockpit once you've created the homepage, later in this exercise.)

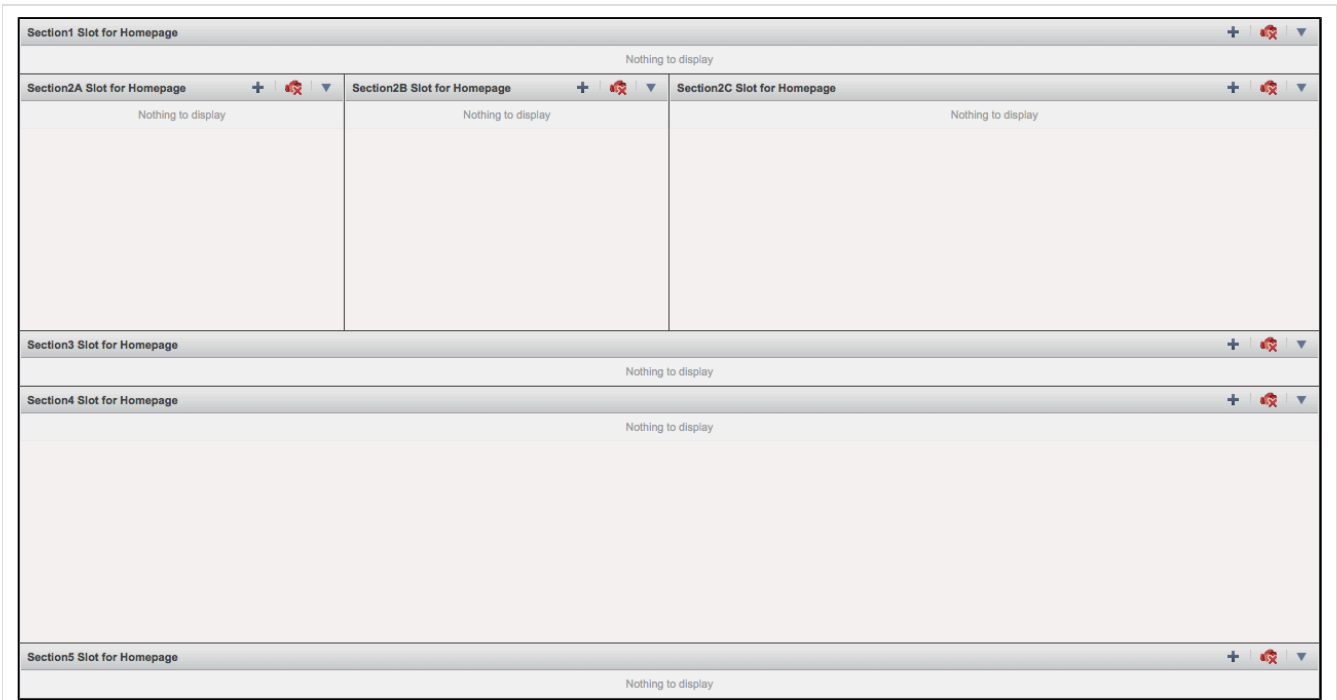


Figure: Layout of a page template (LandingPage2Template) as seen in CMS cockpit

Instructions

Preparation

First, run the **Exercise09_WCMS-prepare** Ant target. It imports core and sample data into the system so you don't have to create everything from scratch; we don't have enough time to do that in the time we have.

A Carousel, defined as part of the preparation process, will appear in **Section 5** of the homepage as soon as you create it. This carousel displays the books in the store.

In this exercise, you'll be working with the following ImpEx files:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/import/coredata/stores/bookstore/store.impex

${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/import/coredata/stores/bookstore/store_en.impex

${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/import/coredata/contentCatalogs/bookstoreContentCatalog/cms-content.impex

${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/import/sampledata/contentCatalogs/bookstoreContentCatalog/cms-content.impex

${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/import/coredata/stores/bookstore/site.impex
```

Unfortunately, your IDE's Task View doesn't display TODO comments in ImpEx files.

Exercise 9.1

Go to **store.impex** and create a **BaseStore** for your bookstore, using the information in the table below. Become comfortable with using macros instead of placing values directly in the ImpEx script value rows. For example, defining *\$storeId=bookstore* and using the macro everywhere you want to refer to your store's id allows you to change it in one location should you ever want to use the script for another store.

Attribute	Reference	Value
uid		bookstore
catalogs	id	bookstoreProductCatalog, BookstoreClassification
defaultCurrency	isocode	EUR
defaultLanguage	isocode	en
currencies	isocode	EUR
languages	isocode	en

Remember that attributes which are references to other Hybris items must be referenced by their business key. For instance, the header *catalogs(id)* tells ImpEx that you will be passing the id property of the target catalog, and it's up to Hybris to find its PK. Likewise, the header *defaultCurrency(isocode)* tells Hybris you'll be passing the iso code of the currency item you want referenced. And yes, for the default language, the header should be *defaultLanguage(isocode)*, and so on.

A pluralized property (like *catalogs*, as opposed to *catalog*) is most likely a collection, so you can provide multiple values, separated by commas. But you don't have to. In this exercise, we will provide a single currency and language for our store, even though Hybris supports having as many as you need.

You should now give your store a name, used by your front end and by Hybris cockpits and Backoffice applications. Since the Hybris UIs are designed to be localizable, the store's name is defined in a localization file. We'll only specify an English name here, by editing **store_en.impex** and setting the *bookstore*'s name to **Bookstore Store** using an ImpEx *UPDATE*.

Of course, you could give the store its name right there in *store.impex*. But Hybris best practice is to use a separate ImpEx file for localization in every language. This simplifies creating localization in other languages; you just duplicate the English one as a starting point.

Exercise 9.2

Your second task will have you edit **two** different instances of **cms-content.impex**. The first, which you'll find in `.../coredata/.../cms-content.impex`, defines page templates and content slots, as well as the pages that use them.

Use the following data to create a homepage as a *ContentPage*.

Attribute	Value
uid	homepage
name	Homepage
masterTemplate	LandingPage2Template
label	homepage
approvalStatus	approved
defaultPage	true
homepage	true

Remember to set the page's catalog version. (Hint: have a look at the `$contentCV` macro defined at the top of the ImpEx file.)

The localization for the *homepage* template, the content item you just created, has already been provided for you in the file **cms-content_en.impex**, located in the same directory as *cms-content.impex*. Take a look if you want to see how that's done.

You will find the second instance of **cms-content.impex** in `.../sampledata/.../cms-content.impex`, where you will import the homepage's navigation link component by **updating** the `ContentPage` you defined a moment ago. The data you need is in the following table:

Attribute	Value
<code>catalogVersion[unique=true]</code>	<code>\$contentCV</code> (use the macro provided in the impex file)
<code>uid[unique=true]</code>	<code>homepage</code>
<code>linkComponents(&linkRef)</code>	<code>HomepageNavLink</code> (use the document ID)

Exercise 9.3

To put content inside the *homepage*, you need to assign CMS components to its content slots and then assign each content slot to a position in the homepage. These assignments are defined in `.../sampledata/.../cms-content.impex`.

You can see similar assignments in neighboring rows, so pay attention to how the document ID is used to connect a link component to a page.

Now connect **Section1Slot-Homepage** to the position **Section1** on the **homepage**.

Exercise 9.4

Our last task involves adding rows to **site.impex**.

In `site.impex`, create a CMS site (with item type *CMSSite*) and associate it with the `BaseStore` you just created, *bookstore*, as well as *bookstoreProductCatalog* and *bookstoreContentCatalog*. Populate the CMS sit with the values given in the following table:

Attribute	Value
<code>uid</code>	<code>bookstore</code>
<code>theme(code)</code>	<code>blue</code>
<code>channel(code)</code>	<code>B2C</code>
<code>stores(uid)</code>	<code>bookstore</code>
<code>contentCatalogs(id)</code>	<code>bookstoreContentCatalog</code>
<code>defaultCatalog(id)</code>	<code>bookstoreProductCatalog</code>
<code>defaultLanguage(isocode)</code>	<code>en</code>
<code>urlPatterns</code>	<code>(?i)^https?://[^\s/]+(?:[/\?]*\?(.*&)?(site=\$siteUid)(\&.*))\$, (?i)^https?://\$siteUid[^\s/]+(?:[/\?]*\?(.*&)?(site=\$siteUid)(\&.*))\$, (?i)^https?://api\.hybrisdev\.com(?:[/\?]*\?(.*&)?(site=\$siteUid)(\&.*))\$, (?i)^https?://localhost(?:[/\?]*\?(.*&)?(site=\$siteUid)(\&.*))\$</code>
<code>active</code>	<code>true</code>
<code>previewURL</code>	<code>/bookstorestorefront/?site=\$siteUid</code>
<code>startingPage(uid,\$contentCV)</code>	<code>homepage</code>
<code>urlEncodingAttributes</code>	<code>language</code>

In the table above, *urlPatterns* are given in terms of regular expressions (or patterns). For example any valid *https* URL that has a parameter *site* which equals *\$siteUid* (*bookstore* in this case) matches the first regular expression.

To learn more about regular expressions in Java, please consult this Oracle Tutorial: [Regular Expressions](#)

To import these ImpEx files, perform a system update in the **HAC**. Make sure that you have selected only **bookstoreinitialdata** option with **Import**

t **Core Data** and **Import Sample Data** values set to **yes** (Activate Solr Cron Jobs value should be set to **no**). Do the same for the JUnit tenant.

•
Bookstore initial data update

You may now navigate to <http://bookstore:9001/> and visit the store you created 😊 It should look like the screenshot below. Click on any book link on the home page to see details on that product.



Verify

To verify your solution, go to the **HAC** and run the script **verifyExercise9**.

Recap

In this exercise, you learned how to create a basic store and its associated website.

[Back to Contents](#)

Exercise 10 - Search and Navigation

Goal

In this exercise, you will enable searching for products on your storefront. You will also see how to create facets to refine a product list.

Instructions

Preparation

First, run the **Exercise10_SearchNavigation-prepare** Ant target. It sets the stage for you to do your exercises, and modifies the view so that you can see the changes you later make in the store.

In this exercise, you'll be working with the following files:

```
${YOURPATH}/hybris/bin/custom/bookstore/bookstorefacades/resources/bookstorefacades-spring.xml  
  
${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/import/sam  
pledata/stores/bookstore/solr.impex  
  
${YOURPATH}/hybris/bin/custom/bookstore/bookstoreinitialdata/resources/bookstoreinitialdata/import/sam  
pledata/contentCatalogs/bookstoreContentCatalog/cms-content.impex  
  
${YOURPATH}/hybris/bin/custom/bookstore/bookstorefacades/src/my/bookstore/facades/populators/SearchRes  
ultProductRewardPointsPopulator.java  
  
${YOURPATH}/hybris/config/localextensions.xml
```

Note that the Task view won't show tasks inside impex files.

Exercise 10.1

We have already provided a populator for copying book data that comes from the database into `ProductData`, a DTO used by the front end to display product data. However, not all of this book data comes from the database; some of it is provided by the Solr index. That means we'll need to create a populator that passes Solr product data to the `ProductData` DTO.

We will call this populator the **SearchResultProductRewardPointsPopulator**. Since this exercise focuses only on `rewardPoints`, the populator is defined only for this attribute. This populator was already created for you during the preparation step, but its `populate` method is only partially implemented. Complete the implementation of this method so that `rewardPoints` are copied to `ProductData`.

Much as we did with the *BookstoreProductPopulator* in the previous exercise, we need to assign our new populator to the appropriate converter. In this case, the converter is **commerceSearchResultProductConverter**. Add the **SearchResultProductRewardPointsPopulator** to **commerceSearchResultProductConverter**'s list of populators, in `bookstorefacades-spring.xml`.

Exercise 10.2

Make `rewardPoints` a **SolrIndexedProperty** to enable Solr indexing. The best place to do that is in `solr.impex`. Since it's a simple integer (int) property, you won't need to provide any of the value providers (valueProvider, sortProvider, etc.). The TODO comments in the file will tell you what you need to add. The data type of the `rewardPoints` property is **int**.

Note that the import statement you will define in this exercise is, by itself, not sufficient to use a Solr indexed property in a storefront. Have a look at this snippet of `solr.impex`, which shows the other imports required to do so.

solr.impex

```
$prefix=bookstore  
$productCatalog=$prefixProductCatalog  
$catalogVersions=catalogVersions(catalog(id),version);  
$serverConfigName=$prefixSolrServerConfig  
$indexConfigName=$prefixSolrIndexConfig  
$searchConfigName=$prefixPageSize  
$facetSearchConfigName=$prefixIndex  
$facetSearchConfigDescription=Bookstore Solr Index
```

```

$searchIndexNamePrefix=$prefix
$solrIndexedType=$prefixProductType
$indexBaseSite=$prefix
$indexLanguages=en,de
$indexCurrencies=EUR,USD
$priceRangeUSD=$prefixPriceRangeUSD
$priceRangeEUR=$prefixPriceRangeEUR

...

# Setup the indexed types, their properties, and the update queries
#
# Declare the indexed type Product
INSERT_UPDATE
SolrIndexedType;identifier[unique=true];type(code);variant;sorts(&sortRefID)
;$solrIndexedType;Product;false;sortRef1,sortRef2,sortRef3,sortRef4,sortRef5,sortRef6,
sortRef7,sortRef8
INSERT_UPDATE
SolrFacetSearchConfig;name[unique=true];description;indexNamePrefix;languages(isocode)
;currencies(isocode);solrServerConfig(name);solrSearchConfig(description);solrIndexCon
fig(name);solrIndexedTypes(identifier);enabledLanguageFallbackMechanism;$catalogVersio
ns
;$facetSearchConfigName;$facetSearchConfigDescription;$searchIndexNamePrefix;$indexLan
guages;$indexCurrencies;$serverConfigName;$searchConfigName;$indexConfigName;$solrInde
xedType;true;$productCatalog:Online,$productCatalog:Staged
UPDATE BaseSite;uid[unique=true];solrFacetSearchConfiguration(name)

...

# Define the available sorts
INSERT_UPDATE
SolrSort;&sortRefID;indexedType(identifier)[unique=true];code[unique=true];useBoost
;sortRef1;$solrIndexedType;relevance;true
;sortRef2;$solrIndexedType;topRated;false
;sortRef3;$solrIndexedType;name-asc;false
;sortRef4;$solrIndexedType;name-desc;false
;sortRef5;$solrIndexedType;price-asc;false
;sortRef6;$solrIndexedType;price-desc;false
;sortRef7;$solrIndexedType;rewardPoints-asc;false
;sortRef8;$solrIndexedType;rewardPoints-desc;false
# Define the sort fields
INSERT_UPDATE
SolrSortField;sort(indexedType(identifier),code)[unique=true];fieldName[unique=true];a
scending[unique=true]
;$solrIndexedType:relevance;score;false
;$solrIndexedType:name-asc;name;true
;$solrIndexedType:name-desc;name;false
;$solrIndexedType:price-asc;priceValue;true
;$solrIndexedType:price-desc;priceValue;false
;$solrIndexedType:rewardPoints-asc;rewardPoints;true

```

```
; $solrIndexedType:rewardPoints-desc;rewardPoints;false
```

```
...
```

To update the Solr index, a frequently-triggered CronJob should be created. A CronJob that occasionally does a full index is also useful. These CronJobs are already defined in `solrtrigger.impex`, which was copied into `bookstoreinitialdata` by the preparation Ant task. Here is the content of this file:

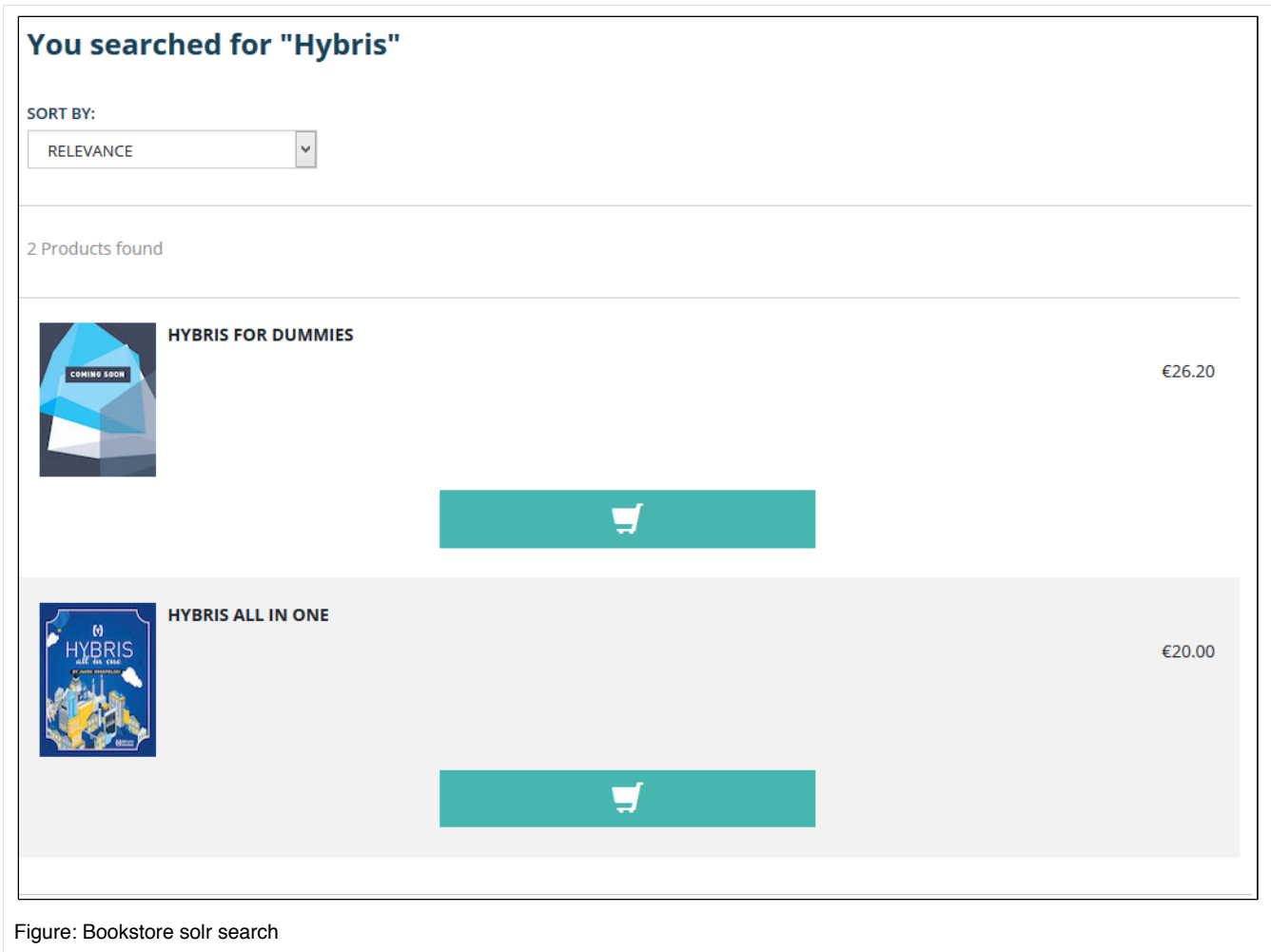
solrtrigger.impex

```
INSERT_UPDATE
Trigger;cronJob(code)[unique=true];second;minute;hour;day;month;year;relative;active;maxAcceptableDelay
# Run the full-bookstoreIndex-cronJob at 3:05 AM every day
;full-bookstoreIndex-cronJob;0;5;3;-1;-1;-1;false;false;-1

# Run the update-bookstoreIndex-cronJob every 1 minutes
;update-bookstoreIndex-cronJob;0;1;-1;-1;-1;-1;true;false;-1
```

Perform an **ant all** to apply your changes, and start your server. Then go to the **HAC** and update the system, selecting the following options under the **bookstoreinitialdata** project: **Import Sample Data** and **Activate Solr Cron Jobs** (you can leave all other checkboxes unchecked).

Let's go to our bookstore site to see our changes. If you try to look for books related to Hybris you will see that Solr search is enabled and returns the results for your request.



Exercise 10.3

The imports in `solr.impex` enabled Solr indexing and created facets for enhancing search in our storefront. But we're still missing the part that makes all of this useful to a user: the part where we display them on the storefront!

We have already defined a "search box" for you in the homepage that can be used to search for a product, and a "search" page that shows the search results. Now you need to define a facet component and connect it to the search page. The facet component we're creating is an instance of **ProductRefinementComponent**, which will display all the available facets for the item being searched for.

Define a **ProductRefinementComponent** using `cms-content.impex`. The data you need is in the following table:

Attribute	Value
uid	RefinementFacetComponent
name	Product Refinement Facet Component
&componentRef	RefinementFacetComponent

Then you need to create a ContentSlot and connect to a position in the `template` that the "search" page is using. Consider **SearchResultsListComponent** and how it is connected to the search `page` as an example; the only difference is that you should connect the content slot to a template rather than a page. The following table contains the data that you need to import the content slot:

Attribute	Value
uid	ProductLeftRefinements
name	Slot for facets

active	true
cmsComponents (&componentRef)	<i>provide reference to the facet component</i>

What remains to be done? Well, to connect the component with the template. We'll establish this relationship by creating a ContentSlotForTemplate item in cms-content.impex. The required data is contained on the following table:

Attribute	Value
uid	ProductLeftRefinements-SearchResultsList
position	ProductLeftRefinements <i>(name of position on the page)</i>
pageTemplate	SearchResultsListPageTemplate
contentSlot	<i>provide uid of the content slot</i>
allowOverwrite	true

Remember that this all works because the search page (bookstorestorefront/web/webroot/WEB-INF/views/responsive/pages/search/searchListPage.jsp) defines a pageSlot called ProductLeftRefinements:

Snippet from searchListPage.jsp

```
<cms:pageSlot position="ProductLeftRefinements" var="feature">
  <cms:component component="{feature}" />
</cms:pageSlot>
```

Similar to exercise 10.2, you should go to the **HAC** and update the system, selecting the following option under the **bookstoreinitialdata** project: **Import Sample Data**. Our search page should then look like the screenshot below:

Shop by Category

Kindle (1)

Paperback (1)

Shop by Price

☐ €10-€29.99 (2)

Shop by Reward Points


☐ 5 (1)

☐ 8 (1)

You searched for "hybris"


SORT BY: RELEVANCE

2 Products found



HYBRIS FOR DUMMIES

€26.20



HYBRIS ALL IN ONE

€20.00

Bookstore search with facet

Exercise 10.4

First, add **commercerearchbackoffice** to `localextensions.xml` to enable the manipulation of boost rules in the Backoffice.

Build your platform using the **ant all** command, then start the server. Go to **hAC**, and do a *project* update only for **commercerearchbackoffice**.

Open the Backoffice and go to the Commerce Search area. Create a boost rule that boosts books that are rentable. Use an arbitrary value for the boost factor.

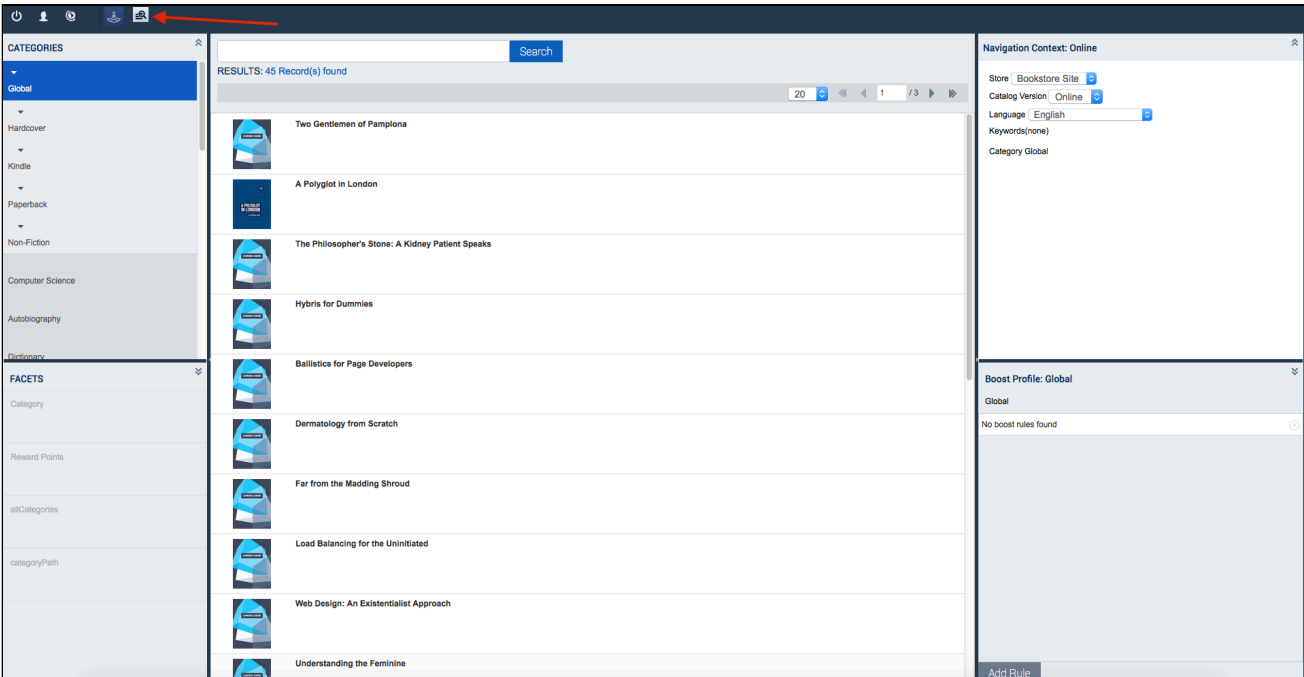


Figure: Backoffice when Commerce Search is enabled

Verify

If you want to check your changes manually, you can go to your bookstore site, search for a product and see the changes that you made. Additionally, you can verify your solution if you go to the **HAC** console and run the script **verifyExercise10**.

Recap

You saw and practiced how to enable Solr indexing and use it in the storefront.

[Back to Contents](#)

Exercise 11 - Security

Goal

In this exercise, you will learn how to create user groups, change access rights, and create restrictions in the Hybris Commerce Suite. Our scenario is to create a group of employees who can only see customers with a **BLUE** reward status level.

Instructions

This exercise will be done entirely in the **Backoffice**.

Preparation

Log in to the **Backoffice** as user **admin**, with password **nimda**. In the tree view, expand the **User** folder and examine the entries found there. For each one, perform a default search to get an idea of what items are there already.

Exercise 11.1

Create the user group **blueManagerGroup**, and make it a member of the **employee** group.

If you click on the **key** icon you will be navigated to **Access Rights** section. Add a type access right to grant **blueManagerGroup** access to **customers** (but grant only **read** and **change** access).

Exercise 11.2

Under **System** -> **Personalization**, create a new **Personalization Rule**, and call it **blueManagerRestriction**. This restriction is intended to restrict **blueManagerGroup** employees from seeing **Customers** who don't have a **BLUE** reward status level. Set the **Generate** parameter to **true** and use the following restriction rule:

Restriction Rule

```
{item.rewardstatuslevel} IN ({SELECT {rsl.PK} FROM {RewardStatusLevel as rsl} WHERE {rsl.code}='BLUE' })
```

Verify

To verify your work, go to the **HAC** and run the script **verifyExercise11**. Please note that the verification script checks only whether or not the user group and an associated restriction exists.

You can easily verify your work yourself. Navigate to **User -> Customers**, and perform a search. You should see four customers. Now create a **blueManager-employee** user who belongs to the **blueManagerGroup** group. Don't forget to set a password for this user, and set the **Disable backoffice Login** flag to **false** (under the Administration tab).

Log out of the **backoffice**, and log back in as the **blueManager-employee**. Navigate again to **User -> Customers** and perform another default search. This time, only customers with a **BLUE** reward status level should appear (only one: john@hybris.com). To verify the reward status level, look in the Administration tab; about halfway down, you'll see the Reward status level, with a drop-down allowing you to select Blue, Silver, and Gold.

Recap

In this exercise, you learned how to assign access rights to user groups and how to create restrictions in the backoffice.

[Back to Contents](#)