Commerce 123

Generated on: 2024-11-08 09:45:34 GMT+0000

SAP Commerce | 2205

PUBLIC

Original content: https://help.sap.com/docs/SAP_COMMERCE/3fb5dcdfe37f40edbac7098ed40442c0?locale=en-US&state=PRODUCTION&version=2205

Explore SAP Commerce Tour

In the Explore SAP Commerce tour, you use the core features of SAP Commerce to create a website for a fictitious company called the Little Concert Company.

ℹ Note

If you are going through multiple guided tours, delete any unzipped SAP Commerce folders and restart your computer between tours to be sure that there are no SAP Commerce threads still running. Then perform the steps in Before You Start and start your next tour.

The goal is to meet the realistic but fictitious business demands of the Little Concert Company. The Little Concert Company sells tickets for an eclectic assortment of small, independent concert tours. Each tour consists of a number of concerts performed at a small set of locations throughout Europe. Tickets are currently sold by phone, but the company now wants to add this capability to its website by replacing the old, bespoke website and manual back-office booking process with a modern, SAP Commerce-based commerce system.

Go through the sections in order, read a little about the introduced concepts, and put them into practice.

1. Version Control

   Version control systems are an essential part of any non-trivial software development project. The primary purpose of a version control system is to centrally track and store changes made to software as it is developed.

2. Installer Recipes

   SAP Commerce ships with a set of pre-configured installation scripts, called installer recipes. Installer recipes simplify the installation and setup of SAP Commerce for development or demonstration purposes.

3. Extensions

   SAP Commerce has a modular architecture, where new business logic is developed in separate, function-specific modules called extensions.

4. The localextensions.xml File

   The `localextensions.xml` file contains the list of extensions your specific SAP Commerce configuration includes at compile- and run-time.

5. Data Models

   The data model underlying SAP Commerce is defined in XML files. New data types for extensions, called item types, are defined in *<extension-name>*`-items.xml` files.

6. Database Design

   SAP Commerce stores its data in a relational database. A large range of third-party databases are supported, allowing you to choose a database best suited to your solution.

7. ImpEx

   SAP Commerce ships with a powerful text-based import and export functionality called ImpEx.

8. Essential and Project Data by Convention

   You can prepare ImpEx files containing essential and project data that the system imports on initialization. This functionality follows the Convention over Configuration principle (CoC).

9. Essential and Project Data by Code

   You can hook into the system initialization and update process to load project data during platform initialization.

10. Essential and Project Data by Convention and Code

    SAP Commerce looks for and loads data from ImpEx files that follow a specific naming convention. This behavior supports the convention over configuration software design paradigm.

11. The ServiceLayer

    When implementing new business logic, you separate the business code into java classes called services. Each service implements a specific, well-defined requirement.

12. Integration Tests

    Integration tests are essential for demonstrating that your new functionality works as expected. They notify you when you break existing behavior, and can therefore help reduce bugs.

13. Unit Tests

    You can simulate dependencies to execute unit tests that run independently of the SAP Commerce platform.

14. The Facade Layer

    A facade is an abstraction layer that provides a simplified interface to the underlying implementation.

15. The Front End

    Once you have a model and business logic in place, you can develop a suitable front-end web application. When building your front end, use the Spring MVC framework to separate the model, the view, and the controller parts.

16. Dynamic Attributes

    Dynamic attributes enable you to add attributes to your model, and to create custom logic for them, without touching the model class itself. They provide a way to generate new data, and access it without calling a separate service to do so. Dynamic attributes are transient data that is not persisted to the database.

17. Dynamic Attributes Integration

    It is good practice to always run an integration test when introducing new features, to test your new classes in context.

18. Web Page Update

    Update the relevant parts of your extension to use the new dynamic attribute.

19. Events and Listeners

    The Event System is a framework provided by the `ServiceLayer` that allows you to send and receive events within SAP Commerce.

20. Interceptors and Custom Events

    Interceptors intercept model object lifecycle transitions and, depending on the conditions of that transition, may publish an event when they do so.

21. Cluster-Aware Events

    SAP Commerce supports cluster-aware events. With cluster-aware events, SAP Commerce can process events in separate threads, or on particular nodes of a cluster.

22. Cron Jobs

    SAP Commerce provides a means to set up regular tasks. With these tasks, or cron jobs, you can repeatedly perform complex business logic at particular times and intervals.

23. Triggers

    With your business logic successfully factored into a job class, you can trigger its execution with the use of a cron job.

24. Groovy

    You can use the Groovy scripting language to write jobs for execution.

25. Backoffice Administration Cockpit

    Backoffice Administration Cockpit is a user-friendly, browser-based, user interface for viewing, creating, and manipulating data within SAP Commerce.

26. Localization

    Localization is intended to adapt SAP Commerce to multiple languages.

27. Localization in Backoffice Administration Cockpit

    You can define localized values for item type attributes directly in the Backoffice Administration Cockpit.

28. Validation

    The SAP Commerce data validation framework ensures clean, correct, and useful data. The validation framework is based on the Java validation specification, JSR 303. It offers an easy and extensible way to validate data before it is passed on to the persistence layer.

29. Validation Constraints in Backoffice

    You can create and define validation constraints in the SAP Commerce Backoffice Administration Cockpit.

30. Validation Constraints in ImpEx

    You can define validation constraints in ImpEx files, making it easy to reload constraints after initializing the database.

31. Custom Validation Constraints

    Although the validation framework provides all constraints from the JSR 303 specification, sometimes you need other constraint types that are specific to your project.

32. Integration Test for the Custom Constraint

    Become familiar with how to use the SAP Commerce validation service in code.

33. Media Files

    SAP Commerce supports media files. A media file can be anything that can be saved on a file system.

34. Properties Files

    SAP Commerce relies on two essential configuration files: project.properties and local.properties. Project properties are the SAP Commerce defaults, while local properties is where you may define your own configuration for your extension.

Version Control

Version control systems are an essential part of any non-trivial software development project. The primary purpose of a version control system is to centrally track and store changes made to software as it is developed.

As a developer, you will typically *commit* your changes several times a day, upon completing some small increment of work. These commits should be made only when your software is in a green state. Other developers in your team can then *pull* these changes from the version control system, thus allowing the team to efficiently share any changes. A version control system also allows you to roll your code back to an earlier commit, should you need to do so.

In SAP Commerce 123, you will use the version control system Git to commit your code as you progress. This allows you to revert to a previously completed step, should you wish to do so.

Parent topic: Explore SAP Commerce Tour

Next: Installer Recipes

Related Information

Git

Set Up a Git Repository

Create a Git repository to which you will commit your code as you proceed through SAP Commerce 123. By committing your changes to Git, you can retrieve the code of previously completed steps should you need to do so.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void gitRepoOk() {
  String output = CommandLineHelper.runCmd("git --git-dir ../hybris/.git log");
  assertTrue("Git Repo has not been set up correctly", output.contains("Set Up a Git Repository"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#gitRepoOk test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#gitRepoOk test
```

Procedure

1. Set up a Git repository in the folder *<HYBRIS_HOME_DIR>*`/hybris`

   ```
   cd $HYBRIS_HOME_DIR/hybris; git init
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris & git init
   ```

2. To limit the files saved to your Git repository, add a file named `.gitignore` to the *<HYBRIS_HOME_DIR>*`/hybris` folder with the content:

   ```
   /log
   /temp
   /bin/
   ```

```
!/bin/custom
/roles
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the file yourself or you want to skip this step, move *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/.gitignore into *<HYBRIS_HOME_DIR>*/hybris

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/.gitignore $HYBRIS_HOME_DIR/hybris
```

```
xcopy /h /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\.gitignore %HYBRIS_HOME_DIR%\hybris
```

3. Make an initial commit to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Set Up a Git Repository"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Set Up a Git Repository"
```

4. Run `git log` and confirm that you see your first git commit listed.

```
git log
```

```
git log
```

5. Run the `gitRepoOk` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#gitRepoOk test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#gitRepoOk test
```

Installer Recipes

SAP Commerce ships with a set of pre-configured installation scripts, called installer recipes. Installer recipes simplify the installation and setup of SAP Commerce for development or demonstration purposes.

Installer recipes create directories, move files, update properties and configuration settings, and initialize the system for a specific SAP Commerce configuration. With a single call to an installer recipe, you can configure, initialize, and run SAP Commerce in a number of variations, including the Accelerators, SAP integrations, or a minimal SAP Commerce platform-only configuration.

i Note

Do not use the installer to run one recipe after another on the same SAP Commerce setup. The installer does not uninstall recipes, and does not restore your SAP Commerce file system to its original settings. To install another recipe, use the original SAP Commerce files and directories.

Parent topic: Explore SAP Commerce Tour

Previous: Version Control

Next: Extensions

Related Information

Installing SAP Commerce Using Installer Recipes

Install, Build, and Run SAP Commerce

Install SAP Commerce using the `platform_only` recipe.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testSuiteIsOnline() {
assertTrue( canLoginToHybrisCommerce());
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSuiteIsOnline test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSuiteIsOnline test
```

Procedure

1. Create a new recipe file in a new `platform_only` folder called build.gradle.

```
apply plugin: 'installer-platform-plugin'
def platformOnly = platform {
    afterSetup {
        ensureAdminPasswordSet()
    }
```

```
}

task setup {
    doLast {
        platformOnly.setup()
    }
}

task buildSystem(dependsOn: setup) {
    doLast {
        platformOnly.build()
    }
}

task initialize(dependsOn: buildSystem) {
    doLast {
        platformOnly.initialize()
    }
}

task start {
    doLast {
        platformOnly.start()
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the file yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/installer/recipes/platform_only/build.gradle` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/platform_only/build.gradle`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/platform_only/build.gradle $HYBRIS_HOME_DIR/installer/recipes/platform_only/build.gr
```

```
xcopy /h /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\platform_only\build.gradle %HYBRIS_HOME_DIR%\installer\recipes\
```

2. Navigate to `<HYBRIS_HOME_DIR>/installer` and install, build, and run SAP Commerce with the `platform_only` recipe.

   This may take up to 20 minutes.

```
cd $HYBRIS_HOME_DIR/installer; ./install.sh -r platform_only setup -A local_property:initialpassword.admin=$INITIAL_ADMIN;
./install.sh -r platform_only initialize -A local_property:initialpassword.admin=$INITIAL_ADMIN;
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

ℹ Note

If you get an error saying that the wrapper-macosx-universal-64 or libwrapper-macosx-universal-64.jnilib cannot be opened because the developer cannot be verified, proceed as follows:

a. Go to System Preferences > Security and Privacy and click the General tab.

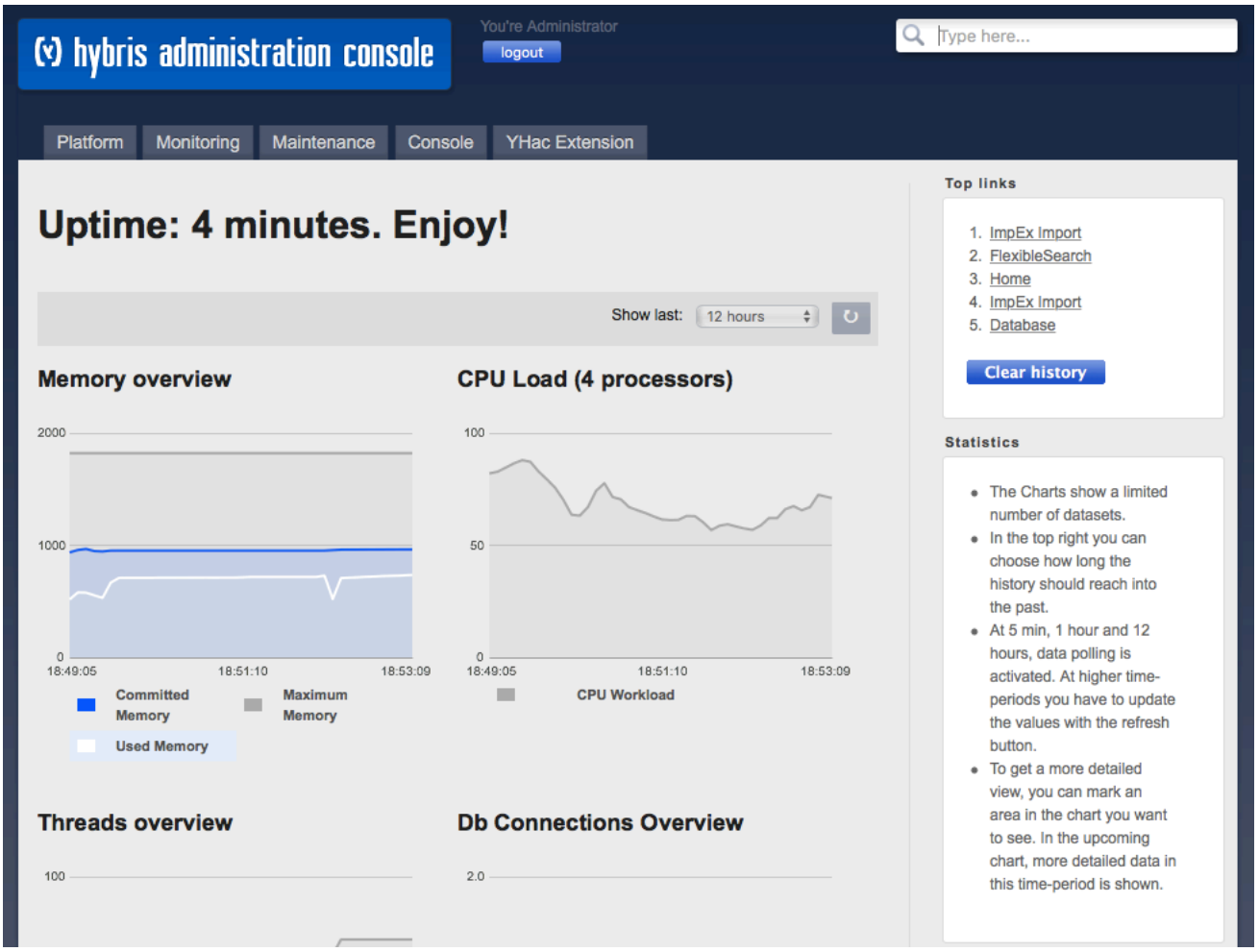
b. In the list, select the executable that you tried to run, and unblock it.

c. Run the command again.

```
rem setup and initialize hybris
cd %HYBRIS_HOME_DIR%\installer & install.bat -r platform_only setup -A local_property:initialpassword.admin=%INITIAL_ADMIN% & install.bat -r pl
rem setup tomcat as a windows service
set _YWRAPPER_CONF=%HYBRIS_HOME_DIR%\hybris\bin\platform\tomcat\conf\wrapper.conf
cd %HYBRIS_HOME_DIR%\hybris\bin\platform\tomcat\bin & InstallTomCatService.bat
rem start hybris
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

3. Wait for the server to start and then open the SAP Commerce backend at `https://localhost:9002`

   This opens the SAP Commerce Administration Console (Administration Console), the root web page in SAP Commerce. You can log in and explore the Administration Console using the default login (*admin*) and the password that you defined as an environment variable.



4. Run the `testSuiteIsOnline` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSuiteIsOnline test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSuiteIsOnline test
```

5. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Install, Build, and Run Hybris"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Install, Build, and Run Hybris"
```

Extensions

SAP Commerce has a modular architecture, where new business logic is developed in separate, function-specific modules called extensions.

An extension is an encapsulated piece of SAP Commerce functionality, that can contain business logic, type definitions, a web application, and back-office configuration functionality. Depending on your business needs, your solution will have a varying number of extensions, all wired into the core SAP Commerce platform via the Spring dependency injection model.

SAP Commerce ships with a number of extension templates, and an ant-based tool (called `extgen`) for generating new extensions based upon these templates. In this and subsequent sections, you extend SAP Commerce by adding and developing your own custom extension.

For your Concert Tours project, you create a custom extension called `concerttours`. To begin, you create a default extension from a template. In later steps, you add the custom functionality that you require.

Parent topic: Explore SAP Commerce Tour

Previous: Installer Recipes

Next: The localextensions.xml File

Related Information

Extension Concept

Create a New Extension

Create a new extension so you can start developing your own SAP Commerce functionalities.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testExtensionCreatedOk() {
        assertTrue("New constants are not there",
                FileHelper.fileExists("../hybris/bin/custom/concerttours/src/concerttours/constants/ConcerttoursConstants.java"));
        assertTrue("New services are not there",
                FileHelper.fileExists("../hybris/bin/custom/concerttours/src/concerttours/service/ConcerttoursService.java"));
        assertTrue("New default services are not there",
                FileHelper.fileExists("../hybris/bin/custom/concerttours/src/concerttours/service/impl/DefaultConcerttoursService.java"));
        assertTrue("New setup is not there",
                FileHelper.fileExists("../hybris/bin/custom/concerttours/src/concerttours/setup/ConcerttoursSystemSetup.java"));
        assertTrue("New standalone is not there",
                FileHelper.fileExists("../hybris/bin/custom/concerttours/src/concerttours/ConcerttoursStandalone.java"));
    }
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testExtensionCreatedOk test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testExtensionCreatedOk test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Navigate into the Platform folder.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform
   ```

3. Run the `setantenv` command to configure the Ant build tool delivered with SAP Commerce.

   ```
   . ./setantenv.sh
   ```

   ```
   setantenv.bat
   ```

4. Execute the Ant task `extgen` to create a new empty SAP Commerce extension, with name `concerttours` and package `de.hybris.platform.concerttours`.

   ```
   ant extgen -Dinput.template="yempty" -Dinput.name="concerttours" -Dinput.package="concerttours"
   ```

   ```
   ant extgen -Dinput.template="yempty" -Dinput.name="concerttours" -Dinput.package="concerttours"
   ```

5. Ignore the steps that the ant command then prints in the console (for example `[echo] 1) Add your extension...`), as these are given more explicitly below.
6. Start SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
   ```

7. Run the `testExtensionCreatedOk` acceptance test again and confirm that it now passes.

   ```
   cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testExtensionCreatedOk test
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testExtensionCreatedOk test
   ```

8. Commit the changes to your local Git repository.

   ```
   cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Create a New Extension"
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Create a New Extension"
   ```

The localextensions.xml File

The `localextensions.xml` file contains the list of extensions your specific SAP Commerce configuration includes at compile- and run-time.

The default list of extensions found in this file is determined by the installer recipe you use. The minimal `platform_only` recipe contains the shortest list of extensions. Other recipes, such as `b2b_accelerator`, contain much longer lists, providing a richer functionality.

To extend the business functionality of SAP Commerce, add other extensions to this list. You can add a combination of the extensions provided with SAP Commerce, and your own custom extensions.

When you first build SAP Commerce, the `localextensions.xml` file lists only the essential extensions. As you decide which extensions you need or want to use, add them to this file. In this procedure, you notify SAP Commerce of the new `concerttours` extension by adding it to the `localextensions.xml` file.

Parent topic: Explore SAP Commerce Tour

Previous: Extensions

Next: Data Models

Related Information

Installation Based on Specified Extensions

Add your Extension

Add the `concerttours` extension to the `localextensions.xml` file and rebuild SAP Commerce.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void loginAndCheckForConcertToursExtension() {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/platform/extensions") ;
assertTrue( waitForExtensionListing("concerttours"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#loginAndCheckForConcertToursExtension test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#loginAndCheckForConcertToursExtension test
```

Procedure

1. Navigate to *<HYBRIS_HOME_DIR>*`/hybris/config/localextensions.xml` and add `<extension dir="../custom/concerttours">` to the file.

   ```
   <hybrisconfig xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' xsi:noNamespaceSchemaLocation='../bin/platform/resources/schemas/extensions
     <extensions>
       <path dir='${HYBRIS_BIN_DIR}' autoload='false' />

       <!-- Your new extension -->
       <extension dir="../custom/concerttours"/>

     </extensions>
   </hybrisconfig>
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to update the `localextensions.xml` file yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/config/localextensions.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/localextensions.xml`

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/localextensions.xml $HYBRIS_HOME_DIR/hybris/config/localextensions.xml
   ```

   ```
   copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\localextensions.xml %HYBRIS_HOME_DIR%\hybris\config\localextensions.xml
   ```

2. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

3. Rebuild SAP Commerce with Ant.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
   ```

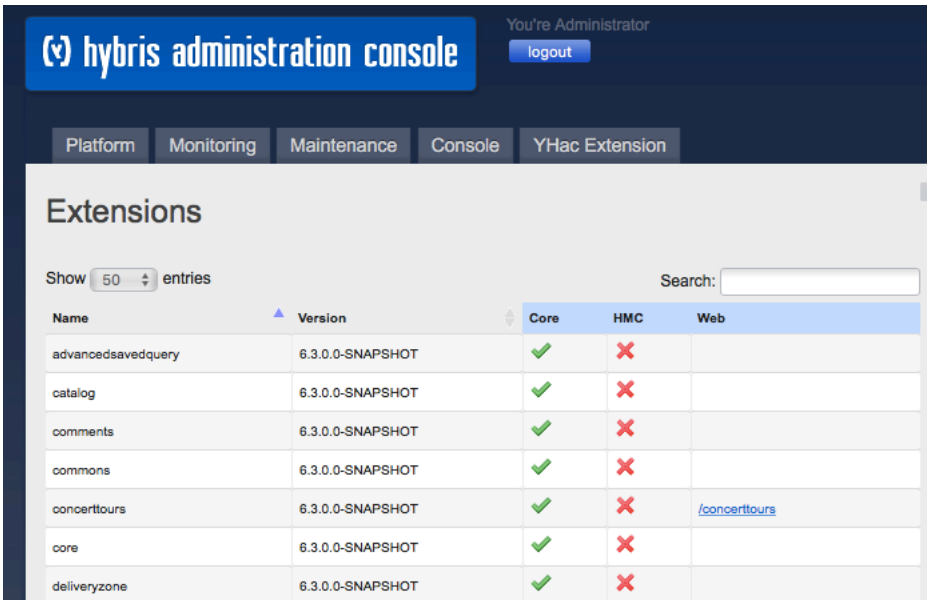
4. Start SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
   ```

5. Log into the SAP Commerce Administration Console (Administration Console) at `https://localhost:9002` using the login, *admin*, and the password that you defined as an environment variable.
6. Navigate to **Platform** > **Extensions** or go directly to `https://localhost:9002/platform/extensions`.

   This shows you an overview of the available extensions, as well as if any of those extensions include a core extension module, a webmodule and/or a core + extension module.



7. Run the `loginAndCheckForConcertToursExtension` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#loginAndCheckForConcertToursExtension test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#loginAndCheckForConcertToursExtension test
```

8. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Add your Extension"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Add your Extension"
```

Data Models

The data model underlying SAP Commerce is defined in XML files. New data types for extensions, called item types, are defined in *<extension-name>*`-items.xml` files.

New functionality you add to SAP Commerce may require additions to the SAP Commerce data model. For example, you need bands, tours, and concerts items to model the Little Concert company's business requirements.

The `core-items.xml` file contains the basic definitions of the most fundamental item types and relations provided and used by the platform. These include definitions for different kinds of users, orders, and products, among other entities.

If an extension contributes new item types, extend existing item types, modify relationships, or add relationships to the overall data model, it must do so in a file named *<extension-name>*`-items.xml`, located in its own top-level resource directory.

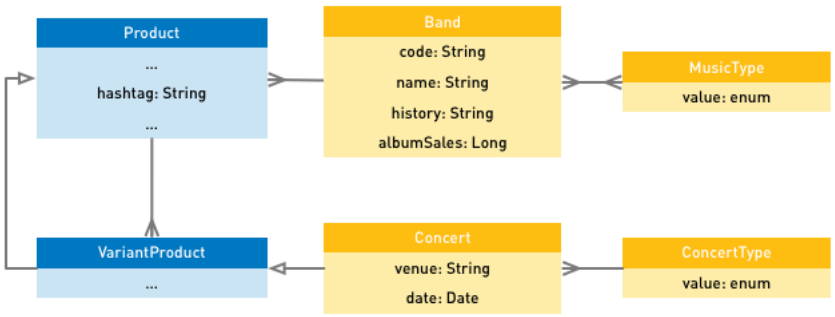
SAP Commerce refers to data types as itemtypes, each defined in an itemtype XML element. You define a new item type by adding a new `itemType` element to the *<extension-name>*`-items.xml` file. Similarly, SAP Commerce refers to a one-to-many and many-to-many relationship between itemtypes as a relation, and you define new relations by adding a relation XML element to the *<extension-name>*`-items.xml` file.

At build time and database-initialization time, the platform combines all the XML declarations from the extensions being used, and generates Java classes and a database schema.

For the little concert tours online shop, you represent each concert tour as a base product and each concert within a tour as a variant of a base product. You also extend the core data model to handle a few additional requirements:

- each concert tour must be associated with a hashtag for social media marketing purposes, so you need to add a string attribute to the Product item type for this
- each concert needs a venue and date associated with them, so you will create a sub-type that extends the VariantProduct item type with these attributes
- each concert can be held either outside or indoors so you will create an enumeration that you can use to indicate this
- the bands giving the concerts need to be represented in the data model so you will define a new item type, Band, to do this and a new relation between the Product and Band item types.
- you want to tag bands with the types of music they play so you will create another enumeration to provide this

The following diagram demonstrates the new data model. The yellow items are what you need to add.



i Note

You don't define any values here for the `MusicType` enumeration because you will do that when you initialize the running system. If you did define some values for `MusicType` here, then you would be stuck with them at runtime because, while you can remove any new values added at runtime, you cannot remove values that are defined in the `*-items.xml` files.

Parent topic: Explore SAP Commerce Tour

Previous: The localextensions.xml File

Next: Database Design

Related Information

Product and Data Modeling

Extend the Data Model

Extend the SAP Commerce data model by declaring new data types and relations, and extending data types declared by other extensions.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testExtensionModelOk() throws ClassNotFoundException, IOException {
    assertTrue("ProductModel has not been extended to support Hashtag and Band",
                FileHelper.fileContains("../hybris/bin/platform/bootstrap/gensrc/de/hybris/platform/core/model/product/ProductModel.java",
                "getHashtag", "getBand",
                "setHashtag", "setBand"));

    assertTrue("The new BandModel does not support Code, Name, History, AlbumSales",
                FileHelper.fileContains("../hybris/bin/platform/bootstrap/gensrc/concerttours/model/BandModel.java",
                "getName","getHistory","getCode", "getAlbumSales",
                "setName","setHistory","setCode", "setAlbumSales"));

    assertTrue("The new ConcertModel does not extend VariantProductModel or does not support Venue and Date",
                FileHelper.fileContains( "../hybris/bin/platform/bootstrap/gensrc/concerttours/model/ConcertModel.java",
                "ConcertModel extends VariantProductModel",
                "getVenue","getDate",
                "setVenue","setDate"));

    assertTrue("The new Band does not extend GenericItem or does not support Code, Name, History, AlbumSales",
                               FileHelper.isBandCreated());

    assertTrue("The new Concert does not extend VariantProduct or does not support Venue, Date",
                               FileHelper.isConcertCreated());
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testExtensionModelOk test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testExtensionModelOk test
```

Procedure

1. Update the data model in the *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-items.xml` file.
    a. Within the `itemtypes` tags, add a new attribute to an existing type, making sure to set the autocreate attribute to false here because you are adding to an itemtype that is defined elsewhere.

    ```
    <itemtype generate="true" code="Product" autocreate="false">
      <attributes>
        <attribute qualifier="hashtag" type="java.lang.String">
          <description>hashtag of concert tour for social media</description>
          <persistence type="property" />
        </attribute>
      </attributes>
    </itemtype>
    ```

    b. Within the `itemtypes` tags, create a new type, making sure to set the autocreate attribute to true because you are creating a new itemtype.

    ```
    <itemtype generate="true" code="Band" autocreate="true">
        <deployment table="Bands" typecode="30268" />
        <attributes>
            <attribute qualifier="code" type="java.lang.String">
                <description>short unique code of band</description>
                <persistence type="property" />
            </attribute>
            <attribute qualifier="name" type="java.lang.String">
                <description>name of band</description>
                <persistence type="property" />
            </attribute>
            <attribute qualifier="history" type="java.lang.String">
                <description>history of band</description>
    ```

```
            <persistence type="property" />
        </attribute>
        <attribute qualifier="albumSales" type="java.lang.Long">
            <description>official global album sales</description>
            <persistence type="property" />
        </attribute>
    </attributes>
</itemtype>
```

c. At the top of the `items` section, define two new enumerations.

```
<enumtypes>
    <enumtype code="ConcertType" autocreate="true" generate="true" dynamic="false">
        <value code="openair" />
        <value code="indoor" />
    </enumtype>

    <enumtype code="MusicType" autocreate="true" generate="true" dynamic="true">
    </enumtype>
</enumtypes>
```

d. Within the `itemtypes` tags, create a new type by extending another.

```
<itemtype generate="true" code="Concert" extends="VariantProduct" autocreate="true">
        <attributes>
            <attribute qualifier="venue" type="java.lang.String">
                <description>venue of concert</description>
                <persistence type="property" />
            </attribute>
            <attribute qualifier="date" type="java.util.Date">
                 <description>date of concert</description>
                <persistence type="property" />
            </attribute>
            <attribute qualifier="concertType" type="ConcertType">
                <description>type of concert (indoors or open air)</description>
                <persistence type="property" />
            </attribute>
         </attributes>
         </itemtype>
```

e. Beneath the `enumtypes` section, create two new relations.

```
<relations>
    <relation code="Product2RockBand" autocreate="true" generate="false" localized="false">
        <sourceElement qualifier="tours" type="Product" collectiontype="set" cardinality="many" ordered="false">
            <modifiers read="true" write="true" search="true" optional="true" />
        </sourceElement>
        <targetElement qualifier="band" type="Band" cardinality="one">
            <modifiers read="true" write="true" search="true" optional="true" />
         </targetElement>
    </relation>
    <relation code="Band2MusicType" autocreate="true" generate="false" localized="false">
        <deployment table="Band2MusicType" typecode="30269" />
        <sourceElement qualifier="bands" type="Band" collectiontype="set" cardinality="many" ordered="false">
            <modifiers read="true" write="true" search="true" optional="true" />
        </sourceElement>
        <targetElement qualifier="types" type="MusicType" cardinality="many">
            <modifiers read="true" write="true" search="true" optional="true" />
        </targetElement>
    </relation>
    </relations>
```

The final file will look similar to the following `concerttours-items.xml` file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- ~ [y] hybris Platform ~ ~ Copyright (c) 2000-2016 SAP SE ~ All rights
  reserved. ~ ~ This software is the confidential and proprietary information
  of SAP ~ Hybris ("Confidential Information"). You shall not disclose such
  ~ Confidential Information and shall use it only in accordance with the ~
  terms of the license agreement you entered into with SAP Hybris. -->
<!-- ATTENTION: This is just an example file. You have to edit it according
  to your needs. -->

<items xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="items.xsd">

  <!-- Hybris123SnippetStart concerttours-items.xml_enum -->
  <enumtypes>
      <enumtype code="ConcertType" autocreate="true" generate="true" dynamic="false">
          <value code="openair" />
          <value code="indoor" />
      </enumtype>

      <enumtype code="MusicType" autocreate="true" generate="true" dynamic="true">
      </enumtype>
  </enumtypes>
  <!-- Hybris123SnippetEnd -->

  <!-- Hybris123SnippetStart concerttours-items.xml_relations -->
  <relations>
      <relation code="Product2RockBand" autocreate="true" generate="false" localized="false">
          <sourceElement qualifier="tours" type="Product" collectiontype="set" cardinality="many" ordered="false">
              <modifiers read="true" write="true" search="true" optional="true" />
          </sourceElement>
          <targetElement qualifier="band" type="Band" cardinality="one">
              <modifiers read="true" write="true" search="true" optional="true" />
           </targetElement>
      </relation>
      <relation code="Band2MusicType" autocreate="true" generate="false" localized="false">
          <deployment table="Band2MusicType" typecode="30269" />
          <sourceElement qualifier="bands" type="Band" collectiontype="set" cardinality="many" ordered="false">
              <modifiers read="true" write="true" search="true" optional="true" />
          </sourceElement>
          <targetElement qualifier="types" type="MusicType" cardinality="many">
              <modifiers read="true" write="true" search="true" optional="true" />
          </targetElement>
      </relation>
  </relations>
```

```
        <!-- Hybris123SnippetEnd -->

        <itemtypes>

           <!-- Hybris123SnippetStart concerttours-items.xml_concert -->
            <itemtype generate="true" code="Concert" extends="VariantProduct" autocreate="true">
              <attributes>
                  <attribute qualifier="venue" type="java.lang.String">
                      <description>venue of concert</description>
                      <persistence type="property" />
                  </attribute>
                  <attribute qualifier="date" type="java.util.Date">
                       <description>date of concert</description>
                      <persistence type="property" />
                  </attribute>
                  <attribute qualifier="concertType" type="ConcertType">
                      <description>type of concert (indoors or open air)</description>
                      <persistence type="property" />
                  </attribute>
               </attributes>
            </itemtype>
            <!-- Hybris123SnippetEnd -->

          <!-- Hybris123SnippetStart concerttours-items.xml_hashtag -->
          <itemtype generate="true" code="Product" autocreate="false">
            <attributes>
              <attribute qualifier="hashtag" type="java.lang.String">
                <description>hashtag of concert tour for social media</description>
                <persistence type="property" />
              </attribute>
            </attributes>
          </itemtype>
          <!-- Hybris123SnippetEnd -->

          <!-- Hybris123SnippetStart concerttours-items.xml_Band -->
          <itemtype generate="true" code="Band" autocreate="true">
              <deployment table="Bands" typecode="30268" />
              <attributes>
                  <attribute qualifier="code" type="java.lang.String">
                      <description>short unique code of band</description>
                      <persistence type="property" />
                  </attribute>
                  <attribute qualifier="name" type="java.lang.String">
                      <description>name of band</description>
                      <persistence type="property" />
                  </attribute>
                  <attribute qualifier="history" type="java.lang.String">
                      <description>history of band</description>
                      <persistence type="property" />
                  </attribute>
                  <attribute qualifier="albumSales" type="java.lang.Long">
                      <description>official global album sales</description>
                      <persistence type="property" />
                  </attribute>
              </attributes>
          </itemtype>
          <!-- Hybris123SnippetEnd -->

        </itemtypes>

      </items>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to update the data model yourself or you want to skip these steps, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/concerttours-items.xml` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours-items.xml`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-items.xml $HYBRIS_HOME_DIR/hybris/bin/custom/concer
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-items.xml %HYBRIS_HOME_DIR%\hybris\bin\custom
```

2. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

3. Rebuild SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

The `ant clean all` command ensures that you first remove all the previously generated Java classes including any that are now no longer needed.

4. Run the `testExtensionModelOk` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testExtensionModelOk test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testExtensionModelOk test
```

5. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Extend the Data Model"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Extend the Data Model"
```

Database Design

SAP Commerce stores its data in a relational database. A large range of third-party databases are supported, allowing you to choose a database best suited to your solution.

When you define a new data model, you must specify how that model is to be persisted in the database. Specifically, whether each new item type should be persisted in its own database table or in a parent table. When you define new item types, you can include a deployment XML attribute to specify the name of a new table specific to that type. If you do not include the deployment XML attribute in a new item type, SAP Commerce persists that item in the table used by the parent of the new type.

For example, if you have not specified a deployment tag for the `Concert` item type, SAP Commerce stores items of that type in the same database table as items of the `VariantProduct` item type because you defined `Concert` as extending `VariantProduct`. The `VariantProduct` type does not have a deployment clause either, but it extends the `Product` item type, which does have a deployment clause. This means that items of `Concert`, `VariantProduct`, and `Product` are all stored in the same database table, which is called `Products`, as specified in the `Product` item type definition in `core-items.xml`. This is generally what you want, because it makes it easy to search and retrieve subsets of all kinds of products. No database table joins are required.

In contrast, you have not defined the `Band` item type as extending any other item type. Therefore, items of our `Band` item type will be stored in the table specified by the root item type called `GenericItem`. All types extend `GenericItem` unless specified otherwise. This is almost certainly not what you want, so when you are not specifically extending another item type, you should always include a deployment clause.

Parent topic: Explore SAP Commerce Tour

Previous: Data Models

Next: ImpEx

Related Information

The Type System
Third-Party Databases

Update the Database

Initiate a system update to reflect the changes that you made to the data model, and confirm that you have created the new `Band` table.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testDatabaseSetup() throws Exception {
        HsqlDBHelper hsqldb = new HsqlDBHelper(); // Note test will fail if the suite is running on this DB at the same time.
        try {
                String res = hsqldb.select("SELECT TABLE_NAME FROM INFORMATION_SCHEMA.SYSTEM_COLUMNS WHERE TABLE_NAME NOT LIKE 'SYSTEM_%'");
                        assertTrue("Could not find the table BANDS", res.contains("BANDS") );
                }
                        catch (Exception e) {
                        fail("testDatabaseSetup", "HsqlDBTest failed: " + e.getMessage());
                }
                finally {
                hsqldb.shutdown();
        }
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDatabaseSetup test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDatabaseSetup test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Update the database.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant updatesystem
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant updatesystem
   ```

3. Open the database with the default hsqldb DatabaseManager.

   ```
   cd $HYBRIS_HOME_DIR/hybris; java -cp ./bin/platform/lib/dbdriver/hsqldb*.jar org.hsqldb.util.DatabaseManager --url jdbc:hsqldb:file:./data/hsql
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris & start /B java -cp .\bin\platform\lib\dbdriver\* org.hsqldb.util.DatabaseManager --url jdbc:hsqldb:file:.\data\hsq
   ```

   There should be a new table called *Bands*.

   → **Tip**

When Hybris is running, you can also view the database in the SAP Commerce Administration Cockpit at `https://localhost:9002` by going to Monitoring > Database > TableSize > CalculateTableSizes and searching for *BANDS* in the list of tables found in the database.

4. Close the hsqldb DatabaseManager.

```
pkill -f "DatabaseManager"

taskkill /FI "WINDOWTITLE eq HSQL*"
```

5. Run the `testDatabaseSetup` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDatabaseSetup test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDatabaseSetup test
```

6. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Update the Database"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Update the Database"
```

ImpEx

SAP Commerce ships with a powerful text-based import and export functionality called ImpEx.

ImpEx files are essentially comma-separated files (CSVs) that allow for compact, human-readable, import and export of data to and from SAP Commerce. They can be manually executed through the SAP Commerce Administration Console, or automatically executed every time you initialize the system by saving the ImpEx file according to a simple convention, and in a specific location.

The SAP Commerce Administration Console provides an interface, in the Impex Import tab of the Console, through which you can manually import small amounts of ImpEx data.

Parent topic: Explore SAP Commerce Tour

Previous: Database Design

Next: Essential and Project Data by Convention

Load Data Through the Administration Console

Import your Band and Group data using the SAP Commerce Administration Console.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testManualImpex() throws Exception {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/console/flexsearch");
waitForFlexQueryFieldThenSubmit("SELECT {pk}, {code}, {history} FROM {Band}");
assertTrue( waitFor("td","A cappella singing group based in Munich"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testManualImpex test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testManualImpex test
```

Procedure

1. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

2. Import some bands and groups.
   a. Log into the SAP Commerce Administration Console at `https://localhost:9002` using the login *admin* and the password that you defined as an environment variable.
   b. Go to the Console tab and select the ImpEx Import option.

      The Impex Import page appears.

   c. Copy and paste the following text into the Import content text box.

```
# ImpEx for Importing Bands into Little Concert Tours Store

INSERT_UPDATE Band;code[unique=true];name;albumSales;history
;A001;yRock;1000000;Occasional tribute rock band comprising senior managers from a leading commerce software vendor
;A006;yBand;;Dutch tribute rock band formed in 2013 playing classic rock tunes from the sixties, seventies and eighties
;A003;yJazz;7;Experimental Jazz group from London playing many musical notes together in unexpected combinations and sequences
```

```
;A004;Banned;427;Rejuvenated Polish boy band from the 1990s - this genre of pop music at its most dubious best
;A002;Sirken;2000;A cappella singing group based in Munich; an uplifting blend of traditional and contemporaneous songs
;A005;The Choir;49000;Enthusiastic, noisy gospel choir singing traditional gospel songs from the deep south
;A007;The Quiet;1200;English choral society specialising in beautifully arranged, soothing melodies and songs
```

d. Click the Validate Content button to execute some basic syntax checking of the content.

A message appears above Import content box confirming that the content is valid.

e. Click the Import Content button to launch the import.

SAP Commerce 123 Interactive Shortcut: If you are unable to import the bands and groups data yourself or you want to skip this step, run the following command.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#simulateManualImpex test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#simulateManualImpex test
```

3. Confirm that the import was successful.

a. In the SAP Commerce Administration Console, go the Console tab and select the FlexibleSearch option.

The FlexibleSearch page appears.

b. Retrieve all the Band data items from the database by entering the query `select {pk},{code},{history} from {Band}` into the FlexibleSearch query text box, and click the Execute button.

→ Tip

You could also click on the All products link in the Query Samples box on the right of the page, and replace the `name[de]` attribute with `history` and `Product` with `Band` in the contents of the query that appear.

c. Check that the imported Band data items are all present and correct in the Search results tab.

The Band data items are listed in the search results with a PK (primary key) attribute that is added for each item by the SAP Commerce persistence engine upon import.

4. Run the `testManualImpex` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testManualImpex test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testManualImpex test
```

5. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Load Data Through the HAC"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Load Data Through the HAC"
```

Essential and Project Data by Convention

You can prepare ImpEx files containing essential and project data that the system imports on initialization. This functionality follows the Convention over Configuration principle (CoC).

Using ImpEx to import data through the SAP Commerce Administration Console (Administration Console) is useful for adhoc imports, but not for repeatedly loading large amounts of ImpEx content every time you initialize or update the database. To automatically load this data, you can place your ImpEx in text files in the `/resources/impex` directory of your extension. A naming convention then ensures that SAP Commerce automatically loads them as part of the initialization or update process.

Impex files fall into two categories: those describing essential data, and those describing project or product data.

Essential data ImpEx file

Contains fundamental reference data that is required by your extension. Essential data is always imported when you initialize the platform with your extension. Essential data ImpEx files have names in the form `essentialdata-*.impex`.

Project data ImpEx file

Contains data that is optional your extension, such as sample data. Project data is included only when you check the project data checkbox for your extension in the Administration Console during initialization. Project data ImpEx files have names in the form `projectdata-*.impex`.

If there are multiple files found that fit the naming convention, SAP Commerce loads and processes them in an unspecified order. If the order is important, one simple solution is to create a single file meeting the naming convention. You then use the ImpEx include feature to include the content from the other files in the required order, and rename the included files so they no longer fit the naming convention.

In this example, the ImpEx files are deliberately designed to separate all the content for each tour into a separate file, and use `INSERT_UPDATE` commands. Therefore, if there are many tours and concerts to import, then the order in which the files are processed is not important.

Parent topic: Explore SAP Commerce Tour

Previous: ImpEx

Next: Essential and Project Data by Code

Related Information

ImpEx Import for Essential and Project Data

Import Essential Data into Your Data Model

Use ImpEx to set up essential and project data for your extension.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testCoCImpex() throws Exception {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/platform/init");
waitForThenClickButtonWithText("Initialize");
waitForThenClickOkInAlertWindow();
waitForInitToComplete();
closeBrowser();

canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/console/flexsearch");
waitForFlexQueryFieldThenSubmit("SELECT {pk}, {code}, {history} FROM {Band}");
assertTrue( waitFor("td","A cappella singing group based in Munich"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCoCImpex test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCoCImpex test
```

Procedure

1. Create the `essentialdata-bands.impex` and `projectdata-yBandTour.impex` files in *<HYBRIS_BIN_DIR>*`/custom/concerttours/resources/impex`

   a. Create the essential data ImpEx file.

   ```
   # ImpEx for Importing Bands into Little Concert Tours Store

   INSERT_UPDATE Band;code[unique=true];name;albumSales;history
   ;A001;yRock;1000000;Occasional tribute rock band comprising senior managers from a leading commerce software vendor
   ;A006;yBand;;Dutch tribute rock band formed in 2013 playing classic rock tunes from the sixties, seventies and eighties
   ;A003;yJazz;7;Experimental Jazz group from London playing many musical notes together in unexpected combinations and sequences
   ;A004;Banned;427;Rejuvenated Polish boy band from the 1990s - this genre of pop music at its most dubious best
   ;A002;Sirken;2000;A cappella singing group based in Munich; an uplifting blend of traditional and contemporaneous songs
   ;A005;The Choir;49000;Enthusiastic, noisy gospel choir singing traditional gospel songs from the deep south
   ;A007;The Quiet;1200;English choral society specialising in beautifully arranged, soothing melodies and songs
   ```

   b. Create the project data ImpEx file.

   ```
   # ImpEx for Importing Tour and dates for a band

   # Macros / Replacement Parameter definitions
   $productCatalog=concertToursProductCatalog
   $supercategories=supercategories(code, $catalogVersion)
   $baseProduct=baseProduct(code,$catalogVersion)
   $approved=approvalstatus(code)[default='approved']
   $catalogVersion=catalogversion(catalog(id[default=$productCatalog]),version[default='Online'])[unique=true,default=$productCatalog:Online

   # Product catalog
   INSERT_UPDATE Catalog;id[unique=true]
   ;$productCatalog

   # Product cataog version
   INSERT_UPDATE CatalogVersion;catalog(id)[unique=true];version[unique=true];active;languages(isoCode);readPrincipals(uid)
   ;$productCatalog;Online;true;en;employeegroup

   # Insert Products
   INSERT_UPDATE Product;code[unique=true];name;band(code);$supercategories;manufacturerName;manufacturerAID;unit(code);ean;variantType(code
   ;201701;The Grand Little x Tour;A001;;x;y;pieces;;Concert

   # Insert Products
   INSERT_UPDATE Concert;code[unique=true];name;date[dateformat=dd.MM.yyyy];venue;concertType(code);baseProduct(code);$catalogVersion;$appro
   ;20170101;Grand Tour - Munich;03.02.2017;"hybris Munich, Germany";openair;201701;
   ;20170102;Grand Tour - London;21.03.2017;"hybris London, UK";openair;201701;
   ;20170103;Grand Tour - Montreal;15.06.2017;"hybris Montreal, Canada";indoor;201701;
   ;20170104;Grand Tour - Gliwice;04.11.2017;"hybris Gliwice, Poland";indoor;201701;
   ;20170105;Grand Tour - Boulder;07.01.2018;"hybris Boulder, USA";indoor;201701;
   ;20170106;Grand Tour - Boston;;"hybris Boston, USA";;201701;
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the ImpEx files yourself or you want to skip these steps, copy *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/impex/*.impex` into *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/impex/`

   ```
   ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/impex/essentialdata-bands.impex $HYBRIS_HOME_DIR/hybris/bin/custom/concerttours/reso
   ```

   ```
   xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\impex\*.impex %HYBRIS_HOME_DIR%\hybris\bin\custom\concerttours\resources\impex\
   ```

2. Re-initialize the database using the SAP Commerce Administration Console.

   a. Position the cursor over the **Platform** tab to display the sub-menu, and then click on the **Initialization** option, to take you to `https://localhost:9002/platform/init`

   b. In the **Project data** settings area, ensure that all the check boxes are seleced.

   c. Press one of the **Initialize** buttons.

3. Check the log file to see if both files were processed without errors.

   a. Open the latest console log file in the `/hybris/log/tomcat` directory with a plain text editor or a log file browsing tool.

   b. Locate in the log file where the two ImpEx files were processed and confirm that no errors were reported for the imports.

4. Check that the imports were successful by using the Administration Console FlexibleSearch console to run some queries and confirm that the data from both files was imported correctly.
5. Run the `testCoCImpex` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCoCImpex test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCoCImpex test
```

6. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Import Essential Data into Your Data Model"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Import Essential Data into Your Data Model"
```

Essential and Project Data by Code

You can hook into the system initialization and update process to load project data during platform initialization.

After some consideration, you decide that band data is not really essential data and should be loaded as project data. Essential data is reserved for data that the system cannot work without. For this reason, move your band data out of essential data and set it up as project data instead. While the results are not exciting, all your code still works if there are no bands stored in the database.

As your project data may grow significantly and become more complex, you can take the sophisticated approach of hooking into the initialization and update process.

Your simple Java class uses the ImpEx service layer API to explicitly load your two ImpEx files. The main work is done in the `impexImport` method. You set up an `ImportConfig` object with various import options, including the name of the file you want to import. You then supply it as a parameter to the `importData` call. The system checks the resulting `ImportResult` for errors.

The code also demonstrates the logging facilities that are built in to the platform.

Parent topic: Explore SAP Commerce Tour

Previous: Essential and Project Data by Convention

Next: Essential and Project Data by Convention and Code

Related Information

Hooks for Initialization and Update Process

Hook into the Initialization and Update Process

Create one or more Java classes to hook into the initialization/update process and use the Impex ServiceLayer API to manipulate and load your data.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testHookImpex() throws Exception {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/platform/init");
waitForThenClickButtonWithText("Initialize");
waitForThenClickOkInAlertWindow();
waitForInitToComplete();

long timeSinceHookLogsFound = LogHelper.getMSSinceThisWasLogged("Custom project data loading for the Concerttours extension completed");
assertTrue("Did not find the expected logs "+ timeSinceHookLogsFound,
timeSinceHookLogsFound < 10000);
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testHookImpex test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testHookImpex test
```

Procedure

1. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

2. Rename `essentialdata-bands.impex` to `concerttours-bands.impex` and rename `projectdata-yBandTour.impex` to `concerttours-yBandTour.impex` to ensure that the files are no longer being loaded by convention.

```
mv $HYBRIS_HOME_DIR/hybris/bin/custom/concerttours/resources/impex/essentialdata-bands.impex $HYBRIS_HOME_DIR/hybris/bin/custom/concerttours/re
```

```
move %HYBRIS_HOME_DIR%\hybris\bin\custom\concerttours\resources\impex\essentialdata-bands.impex %HYBRIS_HOME_DIR%\hybris\bin\custom\concerttour
```

3. Create a setup class called `ConcerttoursCustomSetup.java` in your *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/setup` folder.

   The following code uses annotations at class and method level to hook the class into the initialization/update process and to indicate where in that process particular methods should be invoked. For example, the SystemSetup annotation on the `putInMyEssentialData()` method has a type attribute to specify that the method should be called as part of the essential data loading phase of the initialization/update process.

```
package concerttours.setup;
import de.hybris.platform.core.initialization.SystemSetup;
import de.hybris.platform.servicelayer.impex.ImportConfig;
import de.hybris.platform.servicelayer.impex.ImportResult;
import de.hybris.platform.servicelayer.impex.ImportService;
import de.hybris.platform.servicelayer.impex.impl.StreamBasedImpExResource;
import java.io.InputStream;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

@SystemSetup(extension = "concerttours")
public class ConcerttoursCustomSetup
{
    private static final Logger LOG = LoggerFactory.getLogger(ConcerttoursCustomSetup.class);
    private ImportService importService;
    public ImportService getImportService()
    {
        return importService;
    }
    public void setImportService(final ImportService importService)
    {
        this.importService = importService;
    }
    @SystemSetup(type = SystemSetup.Type.ESSENTIAL)
    public boolean putInMyEssentialData()
    {
        LOG.info("Starting custom essential data loading for the Concerttours extension");
        LOG.info("Custom essential data loading for the Concerttours extension completed.");
        return true;
    }
    @SystemSetup(type = SystemSetup.Type.PROJECT)
    public boolean addMyProjectData()
    {
        LOG.info("Starting custom project data loading for the Concerttours extension");
        impexImport("/impex/concerttours-bands.impex");
        impexImport("/impex/concerttours-yBandTour.impex");
        LOG.info("Custom project data loading for the Concerttours extension completed.");
        return true;
    }
    protected boolean impexImport(final String filename)
    {
        final String message = "Concerttours impexing [" + filename + "]...";
        try (final InputStream resourceAsStream = getClass().getResourceAsStream(filename))
        {
            LOG.info(message);
            final ImportConfig importConfig = new ImportConfig();
            importConfig.setScript(new StreamBasedImpExResource(resourceAsStream, "UTF-8"));
            importConfig.setLegacyMode(Boolean.FALSE);
            final ImportResult importResult = getImportService().importData(importConfig);
            if (importResult.isError())
            {
                LOG.error(message + " FAILED");
                return false;
            }
        }
        catch (final Exception e)
        {
            LOG.error(message + " FAILED", e);
            return false;
        }
        return true;
    }
}
```

   SAP Commerce 123 Interactive Shortcut: If you are unable to create a `ConcerttoursCustomSetup` setup class in the `concerttours.setup` package or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/setup/ConcerttoursCustomSetup.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/setup/ConcerttoursCustomSetup.java`

   UNIX/Mac

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/setup/ConcerttoursCustomSetup.java $HYBRIS_HOME_DIR/hy
```

   Windows

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\setup\ConcerttoursCustomSetup.java %HYBRIS_H
```

4. Register this class as a Spring bean in *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-spring.xml`

```
<bean id="ConcerttoursCustomSetup" class="concerttours.setup.ConcerttoursCustomSetup" >
  <property name="importService" ref="importService"/>
  </bean>
```

   SAP Commerce 123 Interactive Shortcut: If you are unable to register the class as a Spring bean or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-spring.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withCustomSetup.xml`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withCustomSetup.xml $HYBRIS_HOME_DIR/hybris/
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring-withCustomSetup.xml %HYBRIS_HOME_DIR%\
```

5. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

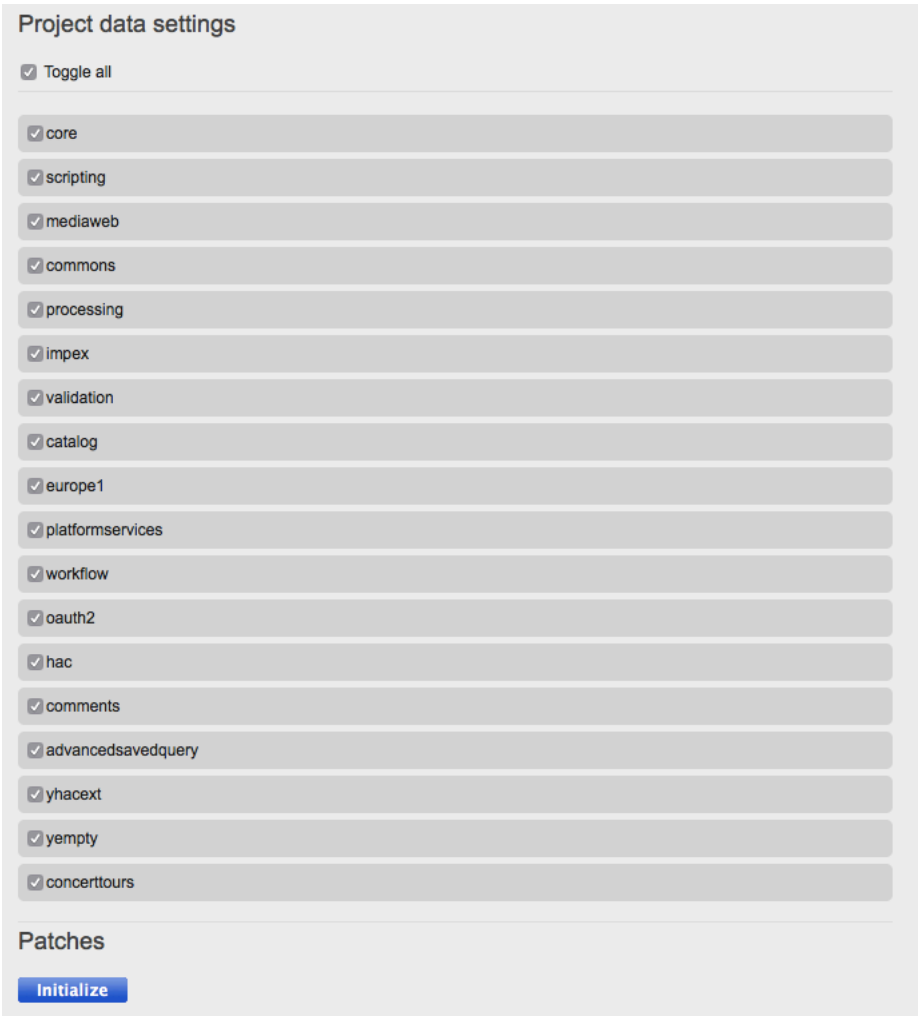
```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

6. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

7. Reinitialize in the SAP Commerce Administration Console (Administration Console).
   a. Log into the SAP Commerce Administration Console (Administration Console) at `https://localhost:9002` using the login, *admin*, and the password that you defined as an environment variable.
   b. Navigate to Platform > Initialization or go directly to `https://localhost:9002/platform/init`.
   c. In the Project data settings area ensure all the check boxes are selected.
   d. Click the Initialize button.



8. In the console output or the console log file in the *<HYBRIS_HOME_DIR>*`/hybris/log/tomcat` directory, locate the output produced by `ConcerttoursCustomSetup` to confirm that the new class was used.
9. Run the `testHookImpex` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testHookImpex test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testHookImpex test
```

10. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Hook into the Initialization and Update Process"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Hook into the Initialization and Update Process"
```

Essential and Project Data by Convention and Code

SAP Commerce looks for and loads data from ImpEx files that follow a specific naming convention. This behavior supports the convention over configuration software design paradigm.

Any ImpEx files that follow the SAP Commerce naming convention are automatically loaded whenever you initialize the system. You can use this to set up essential data that your extension needs to run. You can also optionally provide some initial data such as sample data, or initial values in a categories table for example, that the system also loads for convenience. The latter data type is known as project data.

The naming convention for these two data types is as follows:

- `essentialdata-*.impex`

- `projectdata-*.impex`

In each case, the * can be any name you choose, but it is convential to use your project name or data type.

Parent topic: Explore SAP Commerce Tour

Previous: Essential and Project Data by Code

Next: The ServiceLayer

Use The Naming Convention to Load Data

Create two ImpEx files using the SAP Commerce ImpEx naming convention to load project data to add music style tags to your bands.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testHookAndCoC() throws Exception {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/platform/init");
waitForThenClickButtonWithText("Initialize");
waitForThenClickOkInAlertWindow();
waitForInitToComplete();
long timeSinceHookLogsFound = LogHelper.getMSSinceThisWasLogged("importing resource : /impex/projectdata-musictypes.impex");
assertTrue("Did not find the expected logs",
timeSinceHookLogsFound < 10000);
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testHookAndCoC test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testHookAndCoC test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Create two ImpEx files: `projectdata-musictypes.impex` and `essentialdata-musictypes.impex` in `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/impex/`

   ```
   # ImpEx for Importing Bands music type assignments
   INSERT_UPDATE Band;code[unique=true];types(code)
   ;A001;rock,eighties
   ;A006;rock,sixties,seventies,eighties,maleVocals
   ;A003;jazz,femaleVocals
   ;A004;nineties,maleVocals,pop
   ;A002;choral,pop
   ;A005;gospel
   ;A007;choral,classical
   ```

   ```
   # ImpEx for Importing Music Type Enum values into Little Concert Tours Store
   INSERT_UPDATE MusicType;code[unique=true]
   ;rock
   ;jazz
   ;classical
   ;pop
   ;gospel
   ;choral
   ;sixties
   ;seventies
   ;eighties
   ;nineties
   ;maleVocals
   ;femaleVocals
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the ImpEx files or you want to skip this step, move `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/impex/hookwithcoc/*.impex` into `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/impex/`

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/impex/hookwithcoc/*.impex $HYBRIS_HOME_DIR/hybris/bin/custom/concerttours/resources/imp
   ```

   ```
   copy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\impex\hookwithcoc\*.impex %HYBRIS_HOME_DIR%\hybris\bin\custom\concerttours\resources
   ```

3. Start SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
   ```

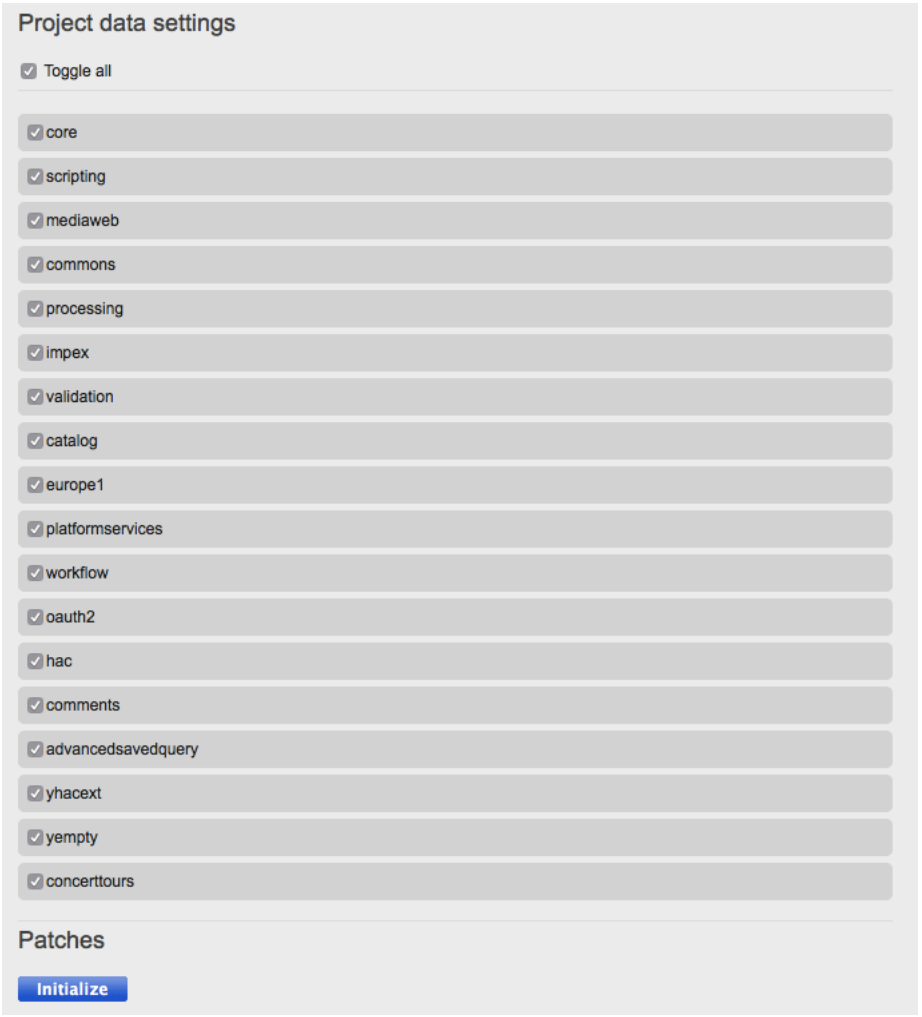
4. Reinitialize in the SAP Commerce Administration Console (Administration Console)

    a. Log into the SAP Commerce Administration Console (Administration Console) at `https://localhost:9002` using the login, *admin*, and the password that you defined as an environment variable.

    b. Navigate to Platform > Initialization or go directly to `https://localhost:9002/platform/init`.

    c. In the Project data settings area ensure all the check boxes are selected.

    d. Click the Initialize button.



5. In the console output or the console log file in the *<HYBRIS_HOME_DIR>*`/hybris/log/tomcat` directory, locate the output produced by `ConcerttoursCustomSetup` to confirm that the new class was used.

6. Run the `testHookAndCoC` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testHookAndCoC test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testHookAndCoC test
```

7. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Use The Naming Convention to Load Data"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Use The Naming Convention to Load Data"
```

The ServiceLayer

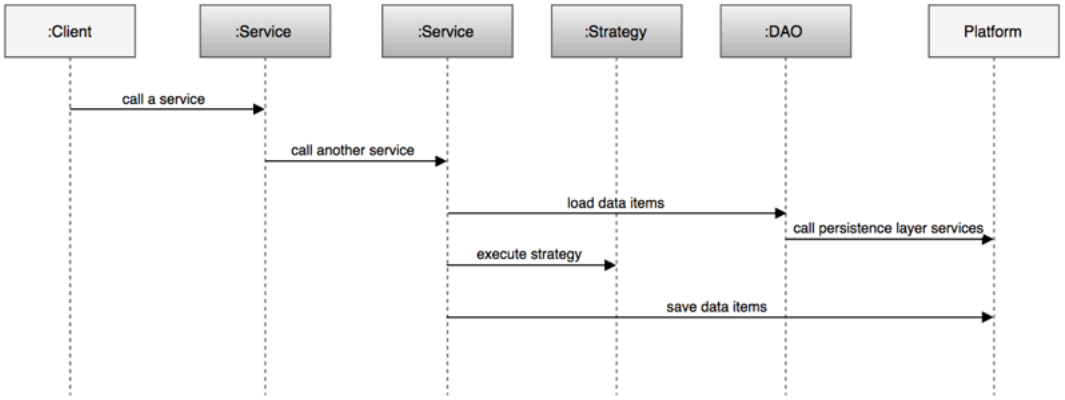
When implementing new business logic, you separate the business code into java classes called services. Each service implements a specific, well-defined requirement.

Services form part of the `ServiceLayer`. This layer is a logical tier between the client and persistence layer. Each service has its own java interface that lists the public methods that can be called by clients and other services.

A service may contain all required business logic, but more commonly will delegate to one or more of the following:

- Other services providing part of the behavior required
- Strategy objects that provide swappable behavior for different requirements, e.g. different cart calculation algorithms
- Data access objects that handle locating and retrieving data items from the database

These other services, strategy objects and data access objects are also defined as Spring beans, with respective interfaces and implementations.



To build your service, you create the following files in the `src` folder of the `concerttours` extension:

- `BandDAO`: The DAO interface, which describes the CRUD functionality that you require, in this case only read functionality.
- `DefaultBandDAO`: The DAO implementation
- `BandService`: The band service interface
- `DefaultBandService`: The band service implementation

Parent topic: Explore SAP Commerce Tour

Previous: Essential and Project Data by Convention and Code

Next: Integration Tests

Related Information

ServiceLayer

Extend the ServiceLayer

Add new functionality to your extension to list and find Bands by adding a new service interface, service implementation, and supporting DAOs.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testServiceLayerClassesExist() throws IOException {
  // If you have correctly added an extension there should be some new folders and files
  assertTrue("You should have added concerttours.daos.BandDAO.java", FileHelper.fileExistsAndContains(
      "../hybris/bin/custom/concerttours/src/concerttours/daos/BandDAO.java", "public interface BandDAO"));
  assertTrue("You should have added concerttours.daos.impl.DefaultBandDAO.java", FileHelper.fileExistsAndContains(
      "../hybris/bin/custom/concerttours/src/concerttours/daos/impl/DefaultBandDAO.java", "public class DefaultBandDAO implements BandDAO"));
  assertTrue("You should have modified concerttours-spring.xml", FileHelper.fileExistsAndContains(
      "../hybris/bin/custom/concerttours/resources/concerttours-spring.xml", "<context:component-scan base-package=\"concerttours\"/>"));
  assertTrue("You should have added concerttours.service.impl.DefaultBandService.java", FileHelper.fileExistsAndContains(
      "../hybris/bin/custom/concerttours/src/concerttours/service/impl/DefaultBandService.java", "public class DefaultBandService implements BandSer
  assertTrue("You should have added concerttours.service.BandService.java", FileHelper.fileExistsAndContains(
      "../hybris/bin/custom/concerttours/src/concerttours/service/BandService.java", "public interface BandService"));
  }
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerClassesExist test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerClassesExist test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Create a new `BandDAO` DAO interface under the `concerttours` extension's `src/concerttours/daos` folder.

   ```
   package concerttours.daos;
   import java.util.List;
   import concerttours.model.BandModel;

   /**
    * An interface for the Band DAO including various operations for retrieving persisted Band model objects
    */
   ```

```
public interface BandDAO
{
    /**
     * Return a list of band models that are currently persisted. If none are found an empty list is returned.
     *
     * @return all the bands in the system
     */
    List<BandModel> findBands();
    /**
     * Finds all bands with given code. If none is found, an empty list will be returned.
     *
     * @param code
     *           the code to search for bands
     * @return All bands with the given code.
     */
    List<BandModel> findBandsByCode(String code);
}
```

- The interface consists of methods required by the test `DefaultBandDAOIntegrationTest.java`.
- The comments describe the behavior for both success and failure of each method.
- You do not need a `save` method. The saving mechanism is done by `modelService`.
- `findBandsByCode` returns a list, but because you have set `unique=true` in `items.xml`, the list will have either 1 or 0 elements.

SAP Commerce 123 Interactive Shortcut: If you are unable to create a new DAO interface yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/daos/BandDAO.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/daos/BandDAO.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/daos/BandDAO.java $HYBRIS_HOME_DIR/hybris/bin/custom/c
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\daos\BandDAO.java %HYBRIS_HOME_DIR%\hybris\b
```

3. Create the `DefaultBandDAO` DAO implementation under the `src/concerttours/daos/impl` folder of the `concerttours` extension.

```
package concerttours.daos.impl;
import de.hybris.platform.servicelayer.search.FlexibleSearchQuery;
import de.hybris.platform.servicelayer.search.FlexibleSearchService;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import concerttours.daos.BandDAO;
import concerttours.model.BandModel;

@Component(value = "bandDAO")
public class DefaultBandDAO implements BandDAO
{
    /**
    * Use SAP
                                Commerce FlexibleSearchService for running queries against the database
     */
    @Autowired
    private FlexibleSearchService flexibleSearchService;
    /**
     * Finds all Bands by performing a FlexibleSearch using the {@link FlexibleSearchService}.
     */
    @Override
    public List<BandModel> findBands()
    {
        // Build a query for the flexible search.
        final String queryString = //
                "SELECT {p:" + BandModel.PK + "} "//
                        + "FROM {" + BandModel._TYPECODE + " AS p} ";
        final FlexibleSearchQuery query = new FlexibleSearchQuery(queryString);
        // Note that we could specify paginating logic by providing a start and count variable (commented out below)
        // This can provide a safeguard against returning very large amounts of data, or hogging the database when there are
        // for example millions of items being returned.
        // As we know that there are only a few persisted bands in this use case we do not need to provide this.
        //query.setStart(start);
        //query.setCount(count);
        // Return the list of BandModels.
        return flexibleSearchService.<BandModel> search(query).getResult();
    }
    /**
     * Finds all Bands by given code by performing a FlexibleSearch using the {@link FlexibleSearchService}.
     */
    @Override
    public List<BandModel> findBandsByCode(final String code)
    {
        final String queryString = //
                "SELECT {p:" + BandModel.PK + "}" //
                        + "FROM {" + BandModel._TYPECODE + " AS p} "//
                        + "WHERE " + "{p:" + BandModel.CODE + "}=?code ";
        final FlexibleSearchQuery query = new FlexibleSearchQuery(queryString);
        query.addQueryParameter("code", code);
        return flexibleSearchService.<BandModel> search(query).getResult();
    }
}
```

- The DAO uses SAP Commerce Flexible Search Query to construct its queries. The `DefaultBandDAO` class uses the SAP Commerce Flexible Search Query for the queries. If you want to persist new Band items in the DAO, use the `save` method of the model service. However, as shown in the diagram in the introduction, you can call the model service `save` method in the service class instead.
- By separating the DAO logic into its own POJO, it is easier to test and clearer to develop.
- At the top of the `DefaultBandDAO` class, there is an annotation that is provided by the Spring framework: `@Component (value = "bandDAO")`. This annotation tells Spring to define a bean, an instance of this class, called `bandDAO` that it can use for wiring dependencies. However, you need to tell Spring that it should scan this package to look for annotation-defined beans. To do so, add a `context:component-scan` tag to the application context.

SAP Commerce 123 Interactive Shortcut: If you are unable to create the DAO implementation yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/daos/impl/DefaultBandDAO.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/daos/impl/DefaultBandDAO.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/daos/impl/DefaultBandDAO.java $HYBRIS_HOME_DIR/hybris/
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\daos\impl\DefaultBandDAO.java %HYBRIS_HOME_D
```

4. Create the `BandService` band service interface under the `src/concerttours/service` folder of the `concerttours` extension.

```
package concerttours.service;
import java.util.List;
import concerttours.model.BandModel;

public interface BandService
{
   /**
    * Gets all bands in the system.
    *
    * @return all bands in the system
    */
   List<BandModel> getBands();
   /**
    * Gets the band for the given code.
    *
    * @throws de.hybris.platform.servicelayer.exceptions.AmbiguousIdentifierException
    *            in case more then one band is found for the given code
    * @throws de.hybris.platform.servicelayer.exceptions.UnknownIdentifierException
    *            in case no band for the given code can be found
    */
   BandModel getBandForCode(String code);
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the band service interface yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/service/BandService.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/service/BandService.java`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/service/BandService.java $HYBRIS_HOME_DIR/hybris/bin/cust
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\service\BandService.java %HYBRIS_HOME_DIR%\hybris\b
```

5. Create the `DefaultBandService` band service implementation under the `src/concerttours/service/impl` folder of the `concerttours` extension.

```
package concerttours.service.impl;
import de.hybris.platform.servicelayer.exceptions.AmbiguousIdentifierException;
import de.hybris.platform.servicelayer.exceptions.UnknownIdentifierException;
import java.util.List;
import org.springframework.beans.factory.annotation.Required;
import concerttours.daos.BandDAO;
import concerttours.model.BandModel;
import concerttours.service.BandService;

public class DefaultBandService implements BandService
{
    private BandDAO bandDAO;
    /**
     * Gets all bands by delegating to {@link BandDAO#findBands()}.
     */
    @Override
    public List<BandModel> getBands()
    {
        return bandDAO.findBands();
    }
    /**
     * Gets all bands for given code by delegating to {@link BandDAO#findBandsByCode(String)} and then assuring
     * uniqueness of result.
     */
    @Override
    public BandModel getBandForCode(final String code) throws AmbiguousIdentifierException, UnknownIdentifierException
    {
        final List<BandModel> result = bandDAO.findBandsByCode(code);
        if (result.isEmpty())
        {
            throw new UnknownIdentifierException("Band with code '" + code + "' not found!");
        }
        else if (result.size() > 1)
        {
            throw new AmbiguousIdentifierException("Band code '" + code + "' is not unique, " + result.size() + " bands found!");
        }
        return result.get(0);
    }
    @Required
    public void setBandDAO(final BandDAO bandDAO)
    {
        this.bandDAO = bandDAO;
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the band service implementation yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/service/impl/DefaultBandService.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/service/impl/DefaultBandService.java`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/service/impl/DefaultBandService.java $HYBRIS_HOME_DIR/hyb
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\service\impl\DefaultBandService.java %HYBRIS_HOME_D
```

6. Configure Spring to register the DAO and `DefaultBandService` as Spring beans in `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/concerttours-spring.xml`

Replace the previously used `<beans xmlns=...>` declaration with:

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.1.xsd"
```

```
>
<context:component-scan base-package="concerttours"/>
```

Add:

```
<alias name = "defaultBandService" alias = "bandService" />
<bean id = "defaultBandService" class = "concerttours.service.impl.DefaultBandService" >
<property name = "bandDAO" ref = "bandDAO" />
</bean>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to register the DAO and `DefaultBandService` yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-spring.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring.xml`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring.xml $HYBRIS_HOME_DIR/hybris/bin/custom/conce
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring.xml %HYBRIS_HOME_DIR%\hybris\bin\custo
```

7. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

8. Run the `testServiceLayerClassesExist` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerClassesExist test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerClassesExist test
```

9. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Extend the ServiceLayer"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Extend the ServiceLayer"
```

Integration Tests

Integration tests are essential for demonstrating that your new functionality works as expected. They notify you when you break existing behavior, and can therefore help reduce bugs.

Create an integration test that tests and demonstrates some of the following expected behavior of your DAO:

- Calling the `findBands` method and what it should return when it does and does not find data
- Calling the `findBands(String code)` method and what it should return when it does and does not find data.
- Persisting a Band

A typical integration test tests a wide range of possibilities. You could expand the test to demonstrate and test that the method succeeds correctly when the parameters are within range, and also that the method fails correctly, and gracefully, when the parameters are out of range. Your test parameters could include any or all of the following:

- An incorrect upper and lower case combination
- A null value
- An empty string
- A space, or invalid characters

Parent topic: Explore SAP Commerce Tour

Previous: The ServiceLayer

Next: Unit Tests

Related Information

The SAP Commerce Testweb Front End

Test Your Service

Create an integration test for your new service, then run it and view the test results.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testServiceLayerIntegrationTest() throws Exception {
assertTrue( checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"DefaultBandDAOIntegrationTest\" package=\"concerttours.da
checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"DefaultBandServiceIntegrationTest\" package=\"concerttours.service.i
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerIntegrationTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerIntegrationTest test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Create the `DefaultBandDAOIntegrationTest.java` DAO integration test in `concerttours\testsrc\concerttours\daos\impl`

   ```
   package concerttours.daos.impl;
   import static org.junit.Assert.assertTrue;
   import de.hybris.bootstrap.annotations.IntegrationTest;
   import de.hybris.platform.core.Registry;
   import de.hybris.platform.servicelayer.ServicelayerTransactionalTest;
   import de.hybris.platform.servicelayer.model.ModelService;
   import java.lang.InterruptedException;
   import java.util.List;
   import java.util.concurrent.TimeUnit;
   import javax.annotation.Resource;
   import org.junit.Assert;
   import org.junit.After;
   import org.junit.Before;
   import org.junit.Test;
   import org.springframework.jdbc.core.JdbcTemplate;
   import concerttours.daos.BandDAO;
   import concerttours.model.BandModel;

   /**
    * The purpose of this test is to illustrate DAO best practices and behaviour.
    *
    * The DAO logic is factored into a separate POJO. Stepping into these will illustrate how to write and execute
    * FlexibleSearchQueries - the basis on which most hybris DAOs operate.
    */
   @IntegrationTest
   public class DefaultBandDAOIntegrationTest extends ServicelayerTransactionalTest
   {
       /** As this is an integration test, the class (object) being tested gets injected here. */
       @Resource
       private BandDAO bandDAO;
       /** Platform's ModelService used for creation of test data. */
       @Resource
       private ModelService modelService;
       /** Name of test band. */
       private static final String BAND_CODE = "ROCK-11";
       /** Name of test band. */
       private static final String BAND_NAME = "Ladies of Rock";
       /** History of test band. */
       private static final String BAND_HISTORY = "All female rock band formed in Munich in the late 1990s";
       /** Albums sold */
       private static final Long ALBUMS_SOLD = Long.valueOf(1000L);
       @Before public void setUp() throws Exception {
         try {
           Thread.sleep(TimeUnit.SECONDS.toMillis(1));
           new JdbcTemplate(Registry.getCurrentTenant().getDataSource()).execute("CHECKPOINT");
           Thread.sleep(TimeUnit.SECONDS.toMillis(1));
         } catch (InterruptedException exc) {}
       }
       @Test
       public void bandDAOTest()
       {
           // check that our test band is not already present in the database
           List<BandModel> bandsByCode = bandDAO.findBandsByCode(BAND_CODE);
           assertTrue("No Band should be returned", bandsByCode.isEmpty());
           // retrieve all bands currently in the database
           List<BandModel> allBands = bandDAO.findBands();
           final int size = allBands.size();
           // add our test band to the database
           final BandModel bandModel = modelService.create(BandModel.class);
           bandModel.setCode(BAND_CODE);
           bandModel.setName(BAND_NAME);
           bandModel.setHistory(BAND_HISTORY);
           bandModel.setAlbumSales(ALBUMS_SOLD);
           modelService.save(bandModel);
           // check we now get one more band back than previously and our test band is in the list
           allBands = bandDAO.findBands();
           Assert.assertEquals(size + 1, allBands.size());
           Assert.assertTrue("band not found", allBands.contains(bandModel));
           // check we can locate our test band by its code
           bandsByCode = bandDAO.findBandsByCode(BAND_CODE);
           Assert.assertEquals("Did not find the Band we just saved", 1, bandsByCode.size());
           Assert.assertEquals("Retrieved Band's code attribute incorrect", BAND_CODE, bandsByCode.get(0).getCode());
           Assert.assertEquals("Retrieved Band's name attribute incorrect", BAND_NAME, bandsByCode.get(0).getName());
           Assert.assertEquals("Retrieved Band's albumSales attribute incorrect", ALBUMS_SOLD, bandsByCode.get(0).getAlbumSales());
           Assert.assertEquals("Retrieved Band's history attribute incorrect", BAND_HISTORY, bandsByCode.get(0).getHistory());
       }
       @Test
       public void testFindBands_EmptyStringParam()
       {
           //calling findBandsByCode() with an empty String - returns no results
           final List<BandModel> bands = bandDAO.findBandsByCode("");
           Assert.assertTrue("No Band should be returned", bands.isEmpty());
       }

       @Test(expected = IllegalArgumentException.class)
       public void testfindBands_NullParam()
       {
           //calling findBandByCode with null should throw an IllegalArgumentException
           bandDAO.findBandsByCode(null); //method's return value not captured
       }
   ```

```
    @After public void tearDown() {

    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the test class yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/testsrc/concerttours/daos/impl/DefaultBandDAOIntegrationTest.java with the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/daos/impl/DefaultBandDAOIntegrationTest.java

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/daos/impl/DefaultBandDAOIntegrationTest.java $HYBR
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\daos\impl\DefaultBandDAOIntegrationTest.
```

3. Create a second integration test to test the BandService by creating a test class called `concerttours.service.impl.DefaultBandServiceIntegrationTest` under the `testsrc` folder of the `concerttours` extension.

```
package concerttours.service.impl;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import de.hybris.bootstrap.annotations.IntegrationTest;
import de.hybris.platform.core.model.product.ProductModel;
import de.hybris.platform.servicelayer.ServicelayerTest;
import de.hybris.platform.servicelayer.exceptions.UnknownIdentifierException;
import de.hybris.platform.servicelayer.model.ModelService;
import de.hybris.platform.variants.model.VariantProductModel;
import java.lang.InterruptedException;
import java.util.Collection;
import java.util.Collections;
import java.util.List;
import java.util.Set;
import javax.annotation.Resource;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import concerttours.model.BandModel;
import concerttours.service.BandService;
import java.util.concurrent.TimeUnit;
import de.hybris.platform.core.Registry;
import org.springframework.jdbc.core.JdbcTemplate;

@IntegrationTest
public class DefaultBandServiceIntegrationTest extends ServicelayerTest
{
    @Resource
    private BandService bandService;
    @Resource
    private ModelService modelService;
    /** Test band */
    private BandModel bandModel;
    /** Name of test band. */
    private static final String BAND_CODE = "101-JAZ";
    /** Name of test band. */
    private static final String BAND_NAME = "Tight Notes";
    /** History of test band. */
    private static final String BAND_HISTORY = "New contemporary, 7-piece Jaz unit from London, formed in 2015";
    /** Albums sold by test band. */
    private static final Long ALBUMS_SOLD = Long.valueOf(10L);
    @Before
    public void setUp()
    {
        try {
          Thread.sleep(TimeUnit.SECONDS.toMillis(1));
          new JdbcTemplate(Registry.getCurrentTenant().getDataSource()).execute("CHECKPOINT");
          Thread.sleep(TimeUnit.SECONDS.toMillis(1));
        } catch (InterruptedException exc) {}
        // This instance of a BandModel will be used by the tests
        bandModel = modelService.create(BandModel.class);
        bandModel.setCode(BAND_CODE);
        bandModel.setName(BAND_NAME);
        bandModel.setAlbumSales(ALBUMS_SOLD);
        bandModel.setHistory(BAND_HISTORY);
    }
    @Test(expected = UnknownIdentifierException.class)
    public void testFailBehavior()
    {
        bandService.getBandForCode(BAND_CODE);
    }
    /**
     * This test tests and demonstrates that the Service's getAllBand method calls the DAOs' getAllBand method and
     * returns the data it receives from it.
     */
    @Test
    public void testBandService()
    {
        List<BandModel> bandModels = bandService.getBands();
        final int size = bandModels.size();
        modelService.save(bandModel);
        bandModels = bandService.getBands();
        assertEquals(size + 1, bandModels.size());
        assertEquals("Unexpected band found", bandModel, bandModels.get(bandModels.size() - 1));
        final BandModel persistedBandModel = bandService.getBandForCode(BAND_CODE);
        assertNotNull("No band found", persistedBandModel);
        assertEquals("Different band found", bandModel, persistedBandModel);
    }
    /**
     * This test tests and demonstrates that the Service's getAllBand method calls the DAOs' getAllBand method and
     * returns the data it receives from it.
     */
    @Test
    public void testBandServiceTours() throws Exception
    {
        createCoreData();
        importCsv("/impex/concerttours-bands.impex", "utf-8");
        importCsv("/impex/concerttours-yBandTour.impex", "utf-8");
        final BandModel band = bandService.getBandForCode("A001");
        assertNotNull("No band found", band);
        final Set<ProductModel> tours = band.getTours();
        assertNotNull("No tour found", tours);
```

```
            Assert.assertEquals("not found one tour", 1, tours.size());
            final Object[] objects = new Object[5];
            final Collection<VariantProductModel> concerts = ((ProductModel) tours.toArray(objects)[0]).getVariants();
            assertNotNull("No tour found", tours);
            Assert.assertEquals("not found one tour", 6, concerts.size());
        }
    }
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the test class yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceIntegrationTest.java with the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceIntegrationTest.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceIntegrationTest.jav

echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\service\impl\DefaultBandServiceIntegrati
```

4. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

5. Initialize your test tenant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant yunitinit

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant yunitinit
```

6. Run the integration tests and confirm that both new integration tests pass.

```
ant integrationtests -Dtestclasses.packages="concerttours.*"

ant integrationtests -Dtestclasses.packages=concerttours.*
```

7. View the test results at *<HYBRIS_HOME_DIR>*/hybris/log/junit/index.html
8. Run the `testServiceLayerIntegrationTest` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerIntegrationTest test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerIntegrationTest test
```

9. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Test Your Service"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Test Your Service"
```

Unit Tests

You can simulate dependencies to execute unit tests that run independently of the SAP Commerce platform.

An integration test demonstrates that you have successfully achieved the expected behavior of classes that implement the `BandService` interface. However it is not truly testing the expected functionality of your particular `DefaultBandService` class. To do that you should mock up the dependencies, the `BandDAO` in this case, using Mockito in a new unit test class called `DefaultBandServiceUnitTest`. As this simulates a real DAO, the test requires no access to the SAP Commerce persistence layer, and therefore does not need to extend `ServicelayerTransactionalTest`. Instead it is a simple POJO and will run very quickly.

You will test that the expected calls are made from the `BandService` to the `BandDAO` interface. No implementation of the `BandDAO` is needed when simulating the `BandDAO` interface.

Parent topic: Explore SAP Commerce Tour

Previous: Integration Tests

Next: The Facade Layer

Related Information

Spring Framework in SAP Commerce
Working with the ServiceLayer

Test Your New Service with a Unit Test

Use Mockito to create a simulation of the `BandDAO` implementation that the service would usually use.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testServiceLayerUnitTest() {
assertTrue( checkTestSuiteXMLMatches("(.*)<testsuite errors=\"0\" failures=\"0\" (.*) name=\"DefaultBandServiceUnitTest\" package=\"concerttours.ser
```

```
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerUnitTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerUnitTest test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Create the new `DefaultBandServiceUnitTest.java` unit test class in `testsrc/concerttours/service/impl`

   ```
   /*
    * [y] hybris Platform
    *
    * Copyright (c) 2000-2017 SAP SE
    * All rights reserved.
    *
    * This software is the confidential and proprietary information of SAP
    * Hybris ("Confidential Information"). You shall not disclose such
    * Confidential Information and shall use it only in accordance with the
    * terms of the license agreement you entered into with SAP Hybris.
    */
   package concerttours.service.impl;
   import static org.junit.Assert.assertEquals;
   import static org.mockito.Mockito.mock;
   import static org.mockito.Mockito.when;
   import de.hybris.bootstrap.annotations.UnitTest;
   import java.util.Arrays;
   import java.util.Collections;
   import java.util.List;
   import org.junit.Before;
   import org.junit.Test;
   import concerttours.daos.BandDAO;
   import concerttours.model.BandModel;

   /**
    * This test file tests and demonstrates the behavior of the BandService's methods getAllBand, getBand and saveBand.
    *
    * We already have a separate file for testing the Band DAO, and we do not want this test to implicitly test that in
    * addition to the BandService. This test therefore mocks out the Band DAO leaving us to test the Service in isolation,
    * whose behavior should be simply to wraps calls to the DAO: forward calls to it, and passing on the results it
    * receives from it.
    */
   @UnitTest
   public class DefaultBandServiceUnitTest
   {
       private DefaultBandService bandService;
       private BandDAO bandDAO;
       private BandModel bandModel;
       /** Name of test band. */
       private static final String BAND_CODE = "Ch00X";
       /** Name of test band. */
       private static final String BAND_NAME = "Singers All";
       /** History of test band. */
       private static final String BAND_HISTORY = "Medieval choir formed in 2001, based in Munich famous for authentic monastic chants";
       @Before
       public void setUp()
       {
           // We will be testing BandServiceImpl - an implementation of BandService
           bandService = new DefaultBandService();
           // So as not to implicitly also test the DAO, we will mock out the DAO using Mockito
           bandDAO = mock(BandDAO.class);
           // and inject this mocked DAO into the BandService
           bandService.setBandDAO(bandDAO);
           // This instance of a BandModel will be used by the tests
           bandModel = new BandModel();
           bandModel.setCode(BAND_CODE);
           bandModel.setName(BAND_NAME);
           bandModel.setAlbumSales(1000L);
           bandModel.setHistory(BAND_HISTORY);
       }
       /**
        * This test tests and demonstrates that the Service's getAllBands method calls the DAOs' getBands method and returns
        * the data it receives from it.
        */
       @Test
       public void testGetAllBands()
       {
           // We construct the data we would like the mocked out DAO to return when called
           final List<BandModel> bandModels = Arrays.asList(bandModel);
           //Use Mockito and compare results
           when(bandDAO.findBands()).thenReturn(bandModels);
           // Now we make the call to BandService's getBands() which we expect to call the DAOs' findBands() method
           final List<BandModel> result = bandService.getBands();
           // We then verify that the results returned from the service match those returned by the mocked-out DAO
           assertEquals("We should find one", 1, result.size());
           assertEquals("And should equals what the mock returned", bandModel, result.get(0));
       }
       @Test
       public void testGetBand()
       {
           // Tell Mockito we expect a call to the DAO's getBand(), and, if it occurs, Mockito should return BandModel instance
           when(bandDAO.findBandsByCode(BAND_CODE)).thenReturn(Collections.singletonList(bandModel));
           // We make the call to the Service's getBandForCode() which we expect to call the DAO's findBandsByCode()
           final BandModel result = bandService.getBandForCode(BAND_CODE);
           // We then verify that the result returned from the Service is the same as that returned from the DAO
   ```

```
            assertEquals("Band should equals() what the mock returned", bandModel, result);
        }
    }
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the `DefaultBandServiceUnitTest.java` unit test class or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceUnitTest.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceUnitTest.java`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceUnitTest.java $HYBRIS_
```

```
copy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\service\impl\DefaultBandServiceUnitTest.java %HYBR
```

3. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

4. Run the unit test.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant unittests -Dtestclasses.packages="concerttours.*"
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant unittests -Dtestclasses.packages=concerttours.*
```

5. View the test results at *<HYBRIS_HOME_DIR>*`/hybris/log/junit/index.html`
6. Run the `testServiceLayerUnitTest` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerUnitTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testServiceLayerUnitTest test
```

7. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Test Your New Service with a Unit Test"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Test Your New Service with a Unit Test"
```

The Facade Layer

A facade is an abstraction layer that provides a simplified interface to the underlying implementation.

In previous sections, you saw that SAP Commerce generates Java model classes that represent the different types of item that are stored in the database. These model classes are what you pass as arguments to the different Java services in the SAP Commerce service layer. Nevertheless, there are occasions when the model classes become unwieldy:

- When you need a simpler or more convenient format for some of the data to display in JSPs
- When you need a serializable set of objects to send to another system
- When you want to prevent client code from modifying attributes in a model class object directly

In these cases, you need a simpler representation of the data in the model classes. This representation is the purpose of the Data Transfer Object. In addition, if there is a common sequence of method calls that a client must make against a service object, it makes sense to combine the sequence into one call. You make these simplified calls with a facade object.

Facade classes help simplify the calls made to your service classes. They use simpler plain old java objects (POJOs) as argument and result objects, instead of SAP Commerce model classes. In this step, you create a new `BandFacade` class.

Parent topic: Explore SAP Commerce Tour

Previous: Unit Tests

Next: The Front End

Extend The Facade Layer

Create facade classes that provide a simplified interface to your extension logic.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testFacadeLayerOk() {
assertTrue( checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"DefaultBandFacadeIntegrationTest\" package=\"concerttours
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testFacadeLayerOk test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testFacadeLayerOk test
```

Procedure

1. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

2. Declare the data transfer objects in `concerttours-beans.xml`.

   Add the following to the `concerttours-beans.xml` file in the concerttours extension's resources folder.

```
<bean class = "concerttours.data.TourSummaryData">
        <description >Data object for a tour summary which has no equivalent on the type system</description>
       <property name = "id" type = "String" />
        <property name = "tourName" type = "String" />
        <property name = "numberOfConcerts" type = "String" />
 </bean>
 <bean class = "concerttours.data.BandData" >
        <description>Data object representing a Band</description>
        <property name = "id" type = "String" />
        <property name = "name" type = "String" />
        <property name = "description" type = "String" />
        <property name = "albumsSold" type = "Long" />
        <property name = "genres" type="java.util.List&lt;String&gt;"/>
        <property name = "tours" type="java.util.List&lt;concerttours.data.TourSummaryData&gt;"/>
 </bean>
<bean class = "concerttours.data.ConcertSummaryData">
        <description >Data object for a concert summary</description>
        <property name = "id" type = "String" />
        <property name = "date" type = "java.util.Date" />
        <property name = "venue" type = "String" />
        <property name = "type" type = "String" />
 </bean>
 <bean class = "concerttours.data.TourData" >
        <description>Data object representing a tour</description>
        <property name = "id" type = "String" />
        <property name = "tourName" type = "String" />
        <property name = "description" type = "String" />
        <property name = "concerts" type="java.util.List&lt;concerttours.data.ConcertSummaryData&gt;"/>
   </bean>
```

   You are defining beans and enumerations in an XML file that a code generator takes as input. The main advantage of this method is that you can merge attributes over several extensions into the same DTO in the same way that item type definitions are combined from across multiple `items.xml` files. This makes your facade layer more extensible.

   Note

   - You create a summary DTO for tours that simply comprises an id, name and number of concerts. You can use this in pages where you need to list tours, or otherwise display brief details of a tour.
   - In the `BandData` beans, you use Java generics to constrain the types of objects returned within the lists of related genres and tours.
   - In the `TourData` beans, you are referencing the `ConcertSummaryData` beans you defined just before it in the file.

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the `concerttours-beans.xml` file yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-beans.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-beansWithFacadeLayer.xml`.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-beansWithFacadeLayer.xml $HYBRIS_HOME_DIR/hybris
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-beansWithFacadeLayer.xml %HYBRIS_HOME_DIR%\hy
```

3. Create the new DTOs from this XML definition by calling *ant clean all* from the *<HYBRIS_HOME_DIR>*`/hybris/bin/platform` directory.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

   The ant command generates model and DTO classes in the `platform/bootstrap/gensrc` directory. When complete, take a look at the newly-created data transfer object classes: `BandData`, `TourData`, `ConcertSummaryData` and `TourSummaryData`.

4. Create the Band and Tour facade interfaces in *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/facades/`.

```
package concerttours.facades;
import java.util.List;
import concerttours.data.BandData;

public interface BandFacade
{
    BandData getBand(String name);
    List<BandData> getBands();
}
```

```
package concerttours.facades;
import concerttours.data.TourData;

public interface TourFacade
{
    TourData getTourDetails(final String tourId);
}
```

   Just like SAP Commerce services, facades are created as Spring beans that implement a particular Java interface. The purpose of these facades is to provide a business-level API of use to the calling clients. In this case they retrieve details on Bands and Tours.

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the facade interfaces yourself or you want to skip this step, copy *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/*Facade.java` to *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/facades/`.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/TourFacade.java $HYBRIS_HOME_DIR/hybris/bin/cu

echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\facades\*Facade.java %HYBRIS_HOME_DIR%\hybri
```

5. Create the Band and Tour facade implementations in *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/facades/impl/.

```
package concerttours.facades.impl;
import de.hybris.platform.core.model.product.ProductModel;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Required;
import concerttours.data.BandData;
import concerttours.data.TourSummaryData;
import concerttours.enums.MusicType;
import concerttours.facades.BandFacade;
import concerttours.model.BandModel;
import concerttours.service.BandService;
import java.util.Locale;

public class DefaultBandFacade implements BandFacade
{
    private BandService bandService;
    @Override
    public List<BandData> getBands()
    {
        final List<BandModel> bandModels = bandService.getBands();
        final List<BandData> bandFacadeData = new ArrayList<>();
        for (final BandModel sm : bandModels)
        {
            final BandData sfd = new BandData();
            sfd.setId(sm.getCode());
            sfd.setName(sm.getName());
            sfd.setDescription(sm.getHistory());
            sfd.setAlbumsSold(sm.getAlbumSales());
            bandFacadeData.add(sfd);
        }
        return bandFacadeData;
    }
    @Override
    public BandData getBand(final String name)
    {
        if (name == null)
        {
            throw new IllegalArgumentException("Band name cannot be null");
        }
        final BandModel band = bandService.getBandForCode(name);
        if (band == null)
        {
            return null;
        }

        // Create a list of genres
        final List<String> genres = new ArrayList<>();
        if (band.getTypes() != null)
        {
            for (final MusicType musicType : band.getTypes())
            {
                genres.add(musicType.getCode());
            }
        }
        // Create a list of TourSummaryData from the matches
        final List<TourSummaryData> tourHistory = new ArrayList<>();
        if (band.getTours() != null)
        {
            for (final ProductModel tour : band.getTours())
            {
                final TourSummaryData summary = new TourSummaryData();
                summary.setId(tour.getCode());
                summary.setTourName(tour.getName(Locale.ENGLISH));
                // making the big assumption that all variants are concerts and ignore product catalogs
                summary.setNumberOfConcerts(Integer.toString(tour.getVariants().size()));
                tourHistory.add(summary);
            }
        }
        // Now we can create the BandData transfer object
        final BandData bandData = new BandData();
        bandData.setId(band.getCode());
        bandData.setName(band.getName());
        bandData.setAlbumsSold(band.getAlbumSales());
        bandData.setDescription(band.getHistory());
        bandData.setGenres(genres);
        bandData.setTours(tourHistory);
        return bandData;
    }
    @Required
    public void setBandService(final BandService bandService)
    {
        this.bandService = bandService;
    }
}
```

```
package concerttours.facades.impl;
import de.hybris.platform.core.model.product.ProductModel;
import de.hybris.platform.product.ProductService;
import de.hybris.platform.variants.model.VariantProductModel;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Required;
import concerttours.data.ConcertSummaryData;
import concerttours.data.TourData;
import concerttours.enums.ConcertType;
import concerttours.facades.TourFacade;
import concerttours.model.ConcertModel;

public class DefaultTourFacade implements TourFacade
{
    private ProductService productService;
    @Override
    public TourData getTourDetails(final String tourId)
```

```
        {
            if (tourId == null)
            {
                throw new IllegalArgumentException("Tour id cannot be null");
            }
            final ProductModel product = productService.getProductForCode(tourId);
            if (product == null)
            {
                return null;
            }
            // Create a list of ConcertSummaryData from the matches
            final List<ConcertSummaryData> concerts = new ArrayList<>();
            if (product.getVariants() != null)
            {
                for (final VariantProductModel variant : product.getVariants())
                {
                    if (variant instanceof ConcertModel)
                    {
                        final ConcertModel concert = (ConcertModel) variant;
                        final ConcertSummaryData summary = new ConcertSummaryData();
                        summary.setId(concert.getCode());
                        summary.setDate(concert.getDate());
                        summary.setVenue(concert.getVenue());
                        summary.setType(concert.getConcertType() == ConcertType.OPENAIR ? "Outdoors" : "Indoors");
                        concerts.add(summary);
                    }
                }
            }
            // Now we can create the TourData transfer object
            final TourData tourData = new TourData();
            tourData.setId(product.getCode());
            tourData.setTourName(product.getName());
            tourData.setDescription(product.getDescription());
            tourData.setConcerts(concerts);
            return tourData;
        }
        @Required
        public void setProductService(final ProductService productService)
        {
            this.productService = productService;
        }
    }
```

The facade implementions make calls to services in the ServiceLayer, specifically `BandService` and `ProductService` to provide the required functionality.

ℹ Note

Note that you hardcode the population of the data transfer objects from model objects. For simple scenarios, this approach is perfectly fine. Nevertheless, for more complex scenarios, the SAP Commerce platform has the notion of convertors. Converters are objects that delegate this task to a chain of populator objects. For more details, see Converters and Populators.

SAP Commerce 123 Interactive Shortcut: If you are unable to create the facade implementations yourself or you want to skip this step, copy *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultBandFacade.java` and *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultTourFacade.java` to the *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/facades/impl/` directory.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultBandFacade.java $HYBRIS_HOME_DIR/h
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\facades\impl\*Facade.java %HYBRIS_HOME_DIR%\
```

6. Declare the facades in `concerttours-spring.xml` so that they are wired in to your extension.

```
<alias name = "defaultBandFacade" alias = "bandFacade" />
<bean id = "defaultBandFacade" class ="concerttours.facades.impl.DefaultBandFacade" >
    <property name = "bandService" ref = "bandService" />
</bean>
```

```
<alias name = "defaultTourFacade" alias = "tourFacade" />
<bean id = "defaultTourFacade" class ="concerttours.facades.impl.DefaultTourFacade" >
    <property name = "productService" ref = "productService" />
</bean>
```

SAP Commerce 123 Interactive Shortcut: If you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-spring.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withFacade.xml`.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withFacade.xml $HYBRIS_HOME_DIR/hybris/bi
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring-withFacade.xml %HYBRIS_HOME_DIR%\hybri
```

7. Create the facades integration test in *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/testsrc/concerttours/facades/impl/`.

It is good practice to write a test that can verify the correct behavior of your facade implementations.

```
package concerttours.facades.impl;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertNotNull;
import de.hybris.bootstrap.annotations.IntegrationTest;
import de.hybris.platform.servicelayer.ServicelayerTransactionalTest;
import de.hybris.platform.servicelayer.exceptions.UnknownIdentifierException;
import de.hybris.platform.servicelayer.model.ModelService;
import java.lang.InterruptedException;
import java.util.List;
import java.util.concurrent.TimeUnit;
import de.hybris.platform.core.Registry;
import org.springframework.jdbc.core.JdbcTemplate;
import javax.annotation.Resource;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import concerttours.data.BandData;
import concerttours.facades.BandFacade;
```

```
import concerttours.model.BandModel;

/**
 * This test file tests and demonstrates the behavior of the BandFacade's methods getAllBands and getBand.
 */
@IntegrationTest
public class DefaultBandFacadeIntegrationTest extends ServicelayerTransactionalTest
{
    @Resource
    private BandFacade bandFacade;
    @Resource
    private ModelService modelService;
    /** Test band */
    private BandModel bandModel;
    /** Name of test band. */
    private static final String BAND_CODE = "101-JAZ";
    /** Name of test band. */
    private static final String BAND_NAME = "Tight Notes";
    /** History of test band. */
    private static final String BAND_HISTORY = "New contemporary, 7-piece Jaz unit from London, formed in 2015";
    /** Albums sold by test band. */
    private static final Long ALBUMS_SOLD = Long.valueOf(10L);
    @Before
    public void setUp()
    {
        try {
          Thread.sleep(TimeUnit.SECONDS.toMillis(1));
          new JdbcTemplate(Registry.getCurrentTenant().getDataSource()).execute("CHECKPOINT");
          Thread.sleep(TimeUnit.SECONDS.toMillis(1));
        } catch (InterruptedException exc) {}
        // This instance of a BandModel will be used by the tests
        bandModel = modelService.create(BandModel.class);
        bandModel.setCode(BAND_CODE);
        bandModel.setName(BAND_NAME);

        bandModel.setHistory(BAND_HISTORY);
        bandModel.setAlbumSales(ALBUMS_SOLD);
    }
    /**
     * Tests exception behavior by getting a band which doesn't exist
     */
    @Test(expected = UnknownIdentifierException.class)
    public void testInvalidParameter()
    {
        bandFacade.getBand(BAND_NAME);
    }
    /**
     * Tests exception behavior by passing in a null parameter
     */
    @Test(expected = IllegalArgumentException.class)
    public void testNullParameter()
    {
        bandFacade.getBand(null);
    }
    /**
     * Tests and demonstrates the Facade's methods
     */
    @Test
    public void testBandFacade()
    {
        List<BandData> bandListData = bandFacade.getBands();
        assertNotNull(bandListData);
        final int size = bandListData.size();
        modelService.save(bandModel);
        bandListData = bandFacade.getBands();
        assertNotNull(bandListData);
        assertEquals(size + 1, bandListData.size());
        assertEquals(BAND_CODE, bandListData.get(size).getId());
        assertEquals(BAND_NAME, bandListData.get(size).getName());
        assertEquals(ALBUMS_SOLD, bandListData.get(size).getAlbumsSold());
        assertEquals(BAND_HISTORY, bandListData.get(size).getDescription());
        final BandData persistedBandData = bandFacade.getBand(BAND_CODE);
        assertNotNull(persistedBandData);
        assertEquals(BAND_CODE, persistedBandData.getId());
        assertEquals(BAND_NAME, persistedBandData.getName());
        assertEquals(ALBUMS_SOLD, persistedBandData.getAlbumsSold());
        assertEquals(BAND_HISTORY, persistedBandData.getDescription());
    }
    @After public void teardown() {

    }
}
```

Note

- Annotate the class with `@IntegrationTest` so that it can be picked up by the built-in ant testing targets, and extend `ServicelayerTransactionalTest` so that the SAP Commerce environment is started up and available during the running of the test.
- Use the `ModelService` to create new instances of model classes so that the new objects are already attached to the persistence context. If you used the standard Java new operator, you would have to explicitly attach the object to the persistence context before saving it.

SAP Commerce 123 Interactive Shortcut: If you are unable to create the integration test yourself or you want to skip this step, copy *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeIntegrationTest.j` to the *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/testsrc/concerttours/facades/impl/` directory.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeIntegrationTest.java
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\facades\impl\DefaultBandFacadeIntegratio
```

8. Create a unit test in *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/testsrc/concerttours/facades/impl/` for testing `DefaultBandFacade` in isolation.

   Name the test class `DefaultBandFacadeUnitTest`, and save it in the `concerttours.facades.impl` package under the `testsrc` folder of the `concerttours` extension.

```
package concerttours.facades.impl;

import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;
```

```
import de.hybris.bootstrap.annotations.UnitTest;
import de.hybris.platform.servicelayer.model.ModelService;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import concerttours.data.BandData;
import concerttours.model.BandModel;
import concerttours.service.BandService;

@UnitTest
public class DefaultBandFacadeUnitTest
{
   private DefaultBandFacade bandFacade;
    private ModelService modelService;
   private BandService bandService;
   private static final String BAND_CODE = "ROCK-11";
   private static final String BAND_NAME = "Ladies of Rock";
   private static final Long ALBUMS_SOLD = Long.valueOf(42000l);
   private static final String BAND_HISTORY = "All female rock band formed in Munich in the late 1990s";
   // Convenience method for returning a list of Band
   private List<BandModel> dummyDataBandList()
   {
       final List<BandModel> bands = new ArrayList<BandModel>();
       final BandModel band = configTestBand();
       bands.add(band);
       return bands;
   }
   // Convenience method for returning the configured test band
   private BandModel configTestBand()
   {
       final BandModel band = new BandModel();
       band.setCode(BAND_CODE);
       modelService.attach(band);
       band.setName(BAND_NAME);
       band.setAlbumSales(ALBUMS_SOLD);
       band.setHistory(BAND_HISTORY);
       return band;
   }
   @Before
   public void setUp()
   {
       // We will be testing the POJO DefaultBandFacade - the implementation of the BandFacade interface.
       bandFacade = new DefaultBandFacade();
       modelService = mock(ModelService.class);
       bandService = mock(BandService.class);
       // We then wire this service into the BandFacade implementation.
       bandFacade.setBandService(bandService);
   }
   /**
    * The aim of this test is to test that:
    *
    * 1) The facade's method getBands makes a call to the BandService's method getBands
    *
    * 2) The facade then correctly wraps BandModels that are returned to it from the BandService's getBands into Data
    * Transfer Objects of type BandData.
    */
   @Test
   public void testGetAllBands()
   {
       /**
        * We instantiate an object that we would like to be returned to BandFacade when the mocked out BandService's
        * method getBands is called. This will be a list of two BandModels.
        */
       final List<BandModel> bands = dummyDataBandList();
       // create test band for the assert comparison
       final BandModel band = configTestBand();
       // We tell Mockito we expect BandService's method getBands to be called, and that when it is, bands should be returned
       when(bandService.getBands()).thenReturn(bands);
       /**
        * We now make the call to BandFacade's getBands. If within this method a call is made to BandService's getBands,
        * Mockito will return the bands instance to it. Mockito will also remember that the call was made.
        */
       final List<BandData> dto = bandFacade.getBands();
       // We now check that dto is a DTO version of bands..
       Assert.assertNotNull(dto);
       Assert.assertEquals(bands.size(), dto.size());
       Assert.assertEquals(band.getCode(), dto.get(0).getId());
       Assert.assertEquals(band.getName(), dto.get(0).getName());
       Assert.assertEquals(band.getAlbumSales(), dto.get(0).getAlbumsSold());
       Assert.assertEquals(band.getHistory(), dto.get(0).getDescription());
   }
   @Test
   public void testGetBand()
   {
       // create test band
       final BandModel band = configTestBand();
       // We tell Mockito we expect BandService's method getBandForCode to be called, and that when it is, the test band should be returned
       when(bandService.getBandForCode(BAND_CODE)).thenReturn(band);
       final BandData dto = bandFacade.getBand(BAND_CODE);
       // We now check that band is a correct DTO representation of the test band model
       Assert.assertNotNull(dto);
       Assert.assertEquals(band.getCode(), dto.getId());
       Assert.assertEquals(band.getName(), dto.getName());
       Assert.assertEquals(band.getAlbumSales(), dto.getAlbumsSold());
       Assert.assertEquals(band.getHistory(), dto.getDescription());
   }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the unit test yourself or you want to skip this step, copy *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeUnitTest.java to the *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/testsrc/concerttours/facades/impl/ directory.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeUnitTest.java $HYBRI
```

Windows

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\facades\impl\DefaultBandFacadeUnitTest.j
```

9. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

10. Run the tests and confirm that they pass.

```
ant alltests -Dtestclasses.packages="concerttours.*"
```

```
ant alltests -Dtestclasses.packages=concerttours.*
```

11. View the test results at *<HYBRIS_HOME_DIR>*`/hybris/log/junit/index.html` and confirm you see `DefaultBandFacadeIntegrationTest`.
12. Run the `testFacadeLayerOk` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testFacadeLayerOk test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testFacadeLayerOk test
```

13. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Extend The Facade Layer"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Extend The Facade Layer"
```

The Front End

Once you have a model and business logic in place, you can develop a suitable front-end web application. When building your front end, use the Spring MVC framework to separate the model, the view, and the controller parts.

SAP Commerce provides a range of Accelerator storefronts. Accelerator gives you basic building blocks that you can use for developing sophisticated and responsive storefronts within specific commerce domains. But to illustrate the basics of how stores are constructed, you focus here on two aspects.

The Controller Class

The interface between the user action and the underlying model.

JSP Pages

Dynamic web pages for presenting data to the end user.

You have previously created the **model** aspect of your MVC architecture, now you create the **view**, the JSP pages, and the **controller, the controller class.**

Parent topic: Explore SAP Commerce Tour

Previous: The Facade Layer

Next: Dynamic Attributes

Build a Web App Component Using Spring MVC

Develop a simple front-end store and get the basics for understanding how stores are constructed.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testWebAppComponent() throws Exception {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/concerttours/bands");
waitFor("a","The Quiet");

navigateTo("https://localhost:9002/concerttours/bands/A007");
assertTrue( waitFor("p","English choral society specialising in beautifully arranged, soothing melodies and songs"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testWebAppComponent test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testWebAppComponent test
```

Procedure

1. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

2. Create two new controller classes called `concerttours.controller.BandController` and `concerttours.controller.TourController` under the `web/src` folder in the `concerttours` extension.

```
package concerttours.controller;
import de.hybris.platform.catalog.CatalogVersionService;
import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import concerttours.data.BandData;
import concerttours.facades.BandFacade;

@Controller
public class BandController
{
    private static final String CATALOG_ID = "concertToursProductCatalog";
    private static final String CATALOG_VERSION_NAME = "Online";
    private CatalogVersionService catalogVersionService;
    private BandFacade bandFacade;
    @RequestMapping(value = "/bands")
    public String showBands(final Model model)
    {
        final List<BandData> bands = bandFacade.getBands();
        model.addAttribute("bands", bands);
        return "BandList";
    }
    @RequestMapping(value = "/bands/{bandId}")
    public String showBandDetails(@PathVariable final String bandId, final Model model) throws UnsupportedEncodingException
    {
        catalogVersionService.setSessionCatalogVersion(CATALOG_ID, CATALOG_VERSION_NAME);
        final String decodedBandId = URLDecoder.decode(bandId, "UTF-8");
        final BandData band = bandFacade.getBand(decodedBandId);
        model.addAttribute("band", band);
        return "BandDetails";
    }
    @Autowired
    public void setCatalogVersionService(final CatalogVersionService catalogVersionServiceService)
    {
        this.catalogVersionService = catalogVersionServiceService;
    }
    @Autowired
    public void setFacade(final BandFacade facade)
    {
        this.bandFacade = facade;
    }
}
```

```
package concerttours.controller;
import de.hybris.platform.catalog.CatalogVersionService;
import java.io.UnsupportedEncodingException;
import java.net.URLDecoder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import concerttours.data.TourData;
import concerttours.facades.TourFacade;

@Controller
public class TourController
{
    private static final String CATALOG_ID = "concertToursProductCatalog";
    private static final String CATALOG_VERSION_NAME = "Online";
    private CatalogVersionService catalogVersionService;
    private TourFacade tourFacade;
    @RequestMapping(value = "/tours/{tourId}")
    public String showTourDetails(@PathVariable final String tourId, final Model model) throws UnsupportedEncodingException
    {
        catalogVersionService.setSessionCatalogVersion(CATALOG_ID, CATALOG_VERSION_NAME);
        final String decodedTourId = URLDecoder.decode(tourId, "UTF-8");
        final TourData tour = tourFacade.getTourDetails(decodedTourId);
        model.addAttribute("tour", tour);
        return "TourDetails";
    }
    @Autowired
    public void setCatalogVersionService(final CatalogVersionService catalogVersionServiceService)
    {
        this.catalogVersionService = catalogVersionServiceService;
    }
    @Autowired
    public void setFacade(final TourFacade facade)
    {
        this.tourFacade = facade;
    }
}
```

You use several Annotations in these classes. The `@Controller` annotation tells Spring to handle the created Class as a Spring Controller component.

In `BandController`, you can see two methods that are annotated using the `@RequestMapping` annotation. This maps the paths `/bands` and `/band/{bandcode}` to the `Controller` methods. While the first method, the `showBands` method, is straightforward, the `showBandDetails` method also declares `@PathVariable`. The `bandId` parameter is automatically resolved by Spring. Both methods return simple Strings which will be picked up by the `ViewResolver` you declared. A returned String `hello` would therefore be mapped to `WEB-INF/views/hello.jsp` according to the setup.

To clearly separate the View from the Model part, populate the passed Model object with the data you would like to retrieve in the view. Either the List of `BandData` or a single `BandData` instance will be part of the model passed to the View.

You need to set the session catalog because flexible search constrains any search for catalog-aware items to catalogs listed in the session object.

The `TourController` class is coded similarly.

SAP Commerce 123 Interactive Shortcut: If you are unable to create the classes or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/controller` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/controller`.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/controller/BandController.java $HYBRIS_HOME_DIR/hybris
```

```
echo d | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\controller %HYBRIS_HOME_DIR%\hybris\bin\cust
```

3. Create the `BandList.jsp`, `BandDetails.jsp`, and `TourDetails.jsp` files in `web/webroot/WEB-INF/views`

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!doctype html>
<html>
    <title>Band List</title>
    <body>
        <h1>Band List</h1>
     <ul>
     <c:forEach var="band" items="${bands}">
        <li><a href="./${band.id}">${band.name}</a></li>
      </c:forEach>
      </ul>
    </body>
</html>
```

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!doctype html>
<html>
<title>Band Details</title>
<body>
    <h1>Band Details</h1>
    Band Details for ${band.name}
    <p>${band.description}</p>
    <p>Music type:</p>
    <ul>
        <c:forEach var="genre" items="${band.genres}">
            <li>${genre}</li>
        </c:forEach>
    </ul>
    <p>Tour History:</p>
    <ul>
        <c:forEach var="tour" items="${band.tours}">
            <li><a href="../tours/${tour.id}">${tour.tourName}</a>(number of concerts: ${tour.numberOfConcerts})</li>
        </c:forEach>
    </ul>
    <a href="../bands">Back to Band List</a>
</body>
</html>
```

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!doctype html>
<html>
<title>Tour Details</title>
<body>
    <h1>Tour Details</h1>
    Tour Details for ${tour.tourName}
    <p>${tour.description}</p>
    <p>Schedule:</p>
    <table>
        <tr><th>Venue</th><th></th><th>Date</th></tr>
        <c:forEach var="concert" items="${tour.concerts}">
            <tr><td>${concert.venue}</td><td>${concert.type}</td><td><fmt:formatDate pattern="dd MMM yyyy" value="${concert.date}" /></td></tr>
        </c:forEach>
    </table>
    <a href="../bands">Back to Band List</a>
</body>
</html>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the List and JSP pages or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/web/webroot/WEB-INF/views` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/web/webroot/WEB-INF/views`.

```
mkdir -p $HYBRIS_HOME_DIR/hybris/bin/custom/concerttours/web/webroot/WEB-INF/views; cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/con
```

```
echo d | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\web\webroot\WEB-INF\views %HYBRIS_HOME_DIR%\hybris\bin\custom
```

4. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

5. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

6. Navigate to `https://localhost:9002/concerttours/bands` and view the list of bands. You are now able to view the list of Bands as well as details on individual Bands, and details of their tours. Note that you don't have to restart the server if you create new or modify existing jsp pages. This can be very helpful in speeding up the development of your front end.

7. Run the `testWebAppComponent` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testWebAppComponent test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testWebAppComponent test
```

8. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Build a Web App Component Using Spring MVC"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Build a Web App Component Using Spring MVC"
```

Dynamic Attributes

Dynamic attributes enable you to add attributes to your model, and to create custom logic for them, without touching the model class itself. They provide a way to generate new data, and access it without calling a separate service to do so. Dynamic attributes are transient data that is not persisted to the database.

All the attributes you have defined so far for your `Band` items are simply stored and retrieved without any extra business logic involved. You use the getter and setter methods of the corresponding Java model class to access and modify them. However, you can also dynamically compute values for an item and expose those values through the getter and setter methods. There are many reasons you may wish to do so:

- You may want to expose a value in a different set of units. For example, you may need to represent a point on a map as both polar and cartesian coordinates, a time in both 12 hour and 24 hour form, or a relative and absolute version of a directory path or URL.
- You may want to calculate a value from several attributes of an item. For example, you may want to calculate the distance between two attributes that hold a location, add the values of a number of attributes together such as price, tax, and discount in an order item to provide a subtotal, or determine the age of a person from a date of birth attribute and the current date.

To provide the custom logic for a dynamic attribute of an item type, you need to provide read and write logic depending upon the requirements for that attribute. You cannot add that logic directly to the model classes because they are already generated, and anything you add would be overwritten and lost the next time you rebuilt the system. Instead, write a plain Java class that implements the `DynamicAttributeHandler` interface. This interface makes use of Java generics so that the actual class reflects both the attribute value type (in this case `java.lang.Long`) and the model class it works upon (in this case `Concert`). You then declare this handler class as a Spring bean.

For this trail you create a new attribute called `Concerts.daysLeft` that exposes the dynamically calculated number of days left before a concert is due to be held. This value can then be used for displaying a countdown to the concert date on a web page, for example.

The implementation of this new dynamic attribute has some key features:

- The persistence type is set to dynamic
- The persistence `attributeHandler` points to a bean that must handle the `DynamicAttributeHandler` interface
- The `write` attribute is set to false, and therefore the attribute is read-only

Parent topic: Explore SAP Commerce Tour

Previous: The Front End

Next: Dynamic Attributes Integration

Introduce a Dynamic Attribute

Create the `Concerts.daysLeft` dynamic attribute.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testDynamicAttributeUnitTest() {
assertTrue( checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"ConcertDaysUntilAttributeHandlerUnitTest\" package=\"con
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeUnitTest test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeUnitTest test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop

   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Define a new dynamic attribute handler within the Concert itemtype in the *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-items.xml` file.

   ```
   <attribute qualifier="daysUntil" type="java.lang.Long">
                   <persistence type="dynamic" attributeHandler="concertDaysUntilAttributeHandler" />
                 <modifiers read="true" write="false" />
           </attribute>
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to define a new dynamic attribute yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-items.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithDynamicAttributes.xml`

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithDynamicAttributes.xml $HYBRIS_HOME_DIR/hyb
   ```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-itemsWithDynamicAttributes.xml %HYBRIS_HOME_D
```

3. Define an attribute handler in
<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/attributehandlers/ConcertDaysUntilAttributeHandler.java

```
package concerttours.attributehandlers;
import de.hybris.platform.servicelayer.model.attribute.AbstractDynamicAttributeHandler;
import java.time.Duration;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import org.springframework.stereotype.Component;
import concerttours.model.ConcertModel;

@Component
public class ConcertDaysUntilAttributeHandler extends AbstractDynamicAttributeHandler<Long, ConcertModel>
{
    @Override
    public Long get(final ConcertModel model)
    {
        if (model.getDate() == null)
        {
            return null;
        }
        final ZonedDateTime concertDate = model.getDate().toInstant().atZone(ZoneId.systemDefault());
        final ZonedDateTime now = ZonedDateTime.now();
        if (concertDate.isBefore(now))
        {
            return Long.valueOf(0L);
        }
        final Duration duration = Duration.between(now, concertDate);
        return Long.valueOf(duration.toDays());
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to define the attribute handler yourself or you want to skip this step, replace
<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/attributehandlers/ConcertDaysUntilAttributeHandler.java with the contents of
<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/attributehandlers/ConcertDaysUntilAttributeHandler.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/attributehandlers/ConcertDaysUntilAttributeHandler.jav
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\attributehandlers\ConcertDaysUntilAttributeH
```

4. Register the attribute handler in Spring in *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/resources/concerttours-spring.xml

```
<bean id="concertDaysUntilAttributeHandler" class="concerttours.attributehandlers.ConcertDaysUntilAttributeHandler"/>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to define the attribute handler yourself or you want to skip this step, replace
<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/concerttours-spring.xml with the contents of
<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withAttributeHandler.xml

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withAttributeHandler.xml $HYBRIS_HOME_DIR/hy
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring-withAttributeHandler.xml %HYBRIS_HOME_
```

5. Create a unit test to confirm correct behavior in
<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/testsrc/concerttours/attributehandlers/ConcertDaysUntilAttributeHandlerUnitTest.java

```
package concerttours.attributehandlers;
import java.util.Date;
import org.junit.Assert;
import org.junit.Test;
import concerttours.model.ConcertModel;
import de.hybris.bootstrap.annotations.UnitTest;

@UnitTest
public class ConcertDaysUntilAttributeHandlerUnitTest
{
    @Test
    public void testGetFutureConcertDate() throws Exception
    {
        final ConcertModel concert = new ConcertModel();
        final ConcertDaysUntilAttributeHandler handler = new ConcertDaysUntilAttributeHandler();
        final Date futureDate = new Date(new Date().getTime() + 49 * 60 * 60 * 1000);
        concert.setDate(futureDate);
        Assert.assertEquals("Wrong value for concert in the future", 2L, handler.get(concert).longValue());
    }
    @Test
    public void testGetNullConcertDate()
    {
        final ConcertModel concert = new ConcertModel();
        final ConcertDaysUntilAttributeHandler handler = new ConcertDaysUntilAttributeHandler();
        Assert.assertNull(handler.get(concert));
    }
    @Test
    public void testGetPastConcertDate() throws Exception
    {
        final ConcertModel concert = new ConcertModel();
        final ConcertDaysUntilAttributeHandler handler = new ConcertDaysUntilAttributeHandler();
        final Date futureDate = new Date(new Date().getTime() - 24 * 60 * 60 * 1000);
        concert.setDate(futureDate);
        Assert.assertEquals("Wrong value for concert in the past", 0L, handler.get(concert).longValue());
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the unit test yourself or you want to skip this step, replace
<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/testsrc/concerttours/attributehandlers/ConcertDaysUntilAttributeHandlerUnitTest.java
with the contents of
<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/attributehandlers/ConcertDaysUntilAttributeHanc

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/attributehandlers/ConcertDaysUntilAttributeHandler

echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\attributehandlers\ConcertDaysUntilAttrib
```

6. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

7. Update SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant updatesystem

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant updatesystem
```

8. Run the unit test.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant unittests -Dtestclasses.packages="concerttours.*"

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant unittests -Dtestclasses.packages=concerttours.*
```

9. View the test results at *<HYBRIS_HOME_DIR>*`/hybris/log/junit/index.html` and confirm that this unit test has passed.
10. Run the `testDynamicAttributeUnitTest` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeUnitTest test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeUnitTest test
```

11. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Introduce a Dynamic Attribute"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Introduce a Dynamic Attribute"
```

Dynamic Attributes Integration

It is good practice to always run an integration test when introducing new features, to test your new classes in context.

You previously ran a unit test to test the basic logic of your new dynamic attribute class. Now write an integration test to put it through its paces. You want to provide the class with some real time data, and also test the robustness of the class with future, past, and empty date values. Use the `modelService`, to ensure that the result is in the form of an extended item model, as expected.

Parent topic: Explore SAP Commerce Tour

Previous: Dynamic Attributes

Next: Web Page Update

Test Dynamic Attributes Integration

Create and run an integration test to ensure the handler is picked up properly as a Spring bean.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testDynamicAttributeIntegrationTest() {
assertTrue( checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"ConcertDaysUntilAttributeHandlerIntegrationTest\" packag
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeIntegrationTest test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeIntegrationTest test
```

Procedure

1. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

2. Create the Integration Test:

```
package concerttours.attributehandlers;
import de.hybris.bootstrap.annotations.IntegrationTest;
```

```
import de.hybris.platform.servicelayer.ServicelayerTransactionalTest;
import de.hybris.platform.servicelayer.model.ModelService;
import java.util.Date;
import javax.annotation.Resource;
import org.junit.Assert;
import org.junit.Test;
import concerttours.model.ConcertModel;

@IntegrationTest
public class ConcertDaysUntilAttributeHandlerIntegrationTest extends ServicelayerTransactionalTest
{
    @Resource
    private ModelService modelService;
    @Test
    public void testGetFutureConcertDate() throws Exception
    {
        final ConcertModel concert = modelService.create(ConcertModel.class);
        final Date futureDate = new Date(new Date().getTime() + 49 * 60 * 60 * 1000);
        concert.setDate(futureDate);
        Assert.assertEquals("Wrong value for concert in the future: " + concert.getDaysUntil().longValue(), 2L, concert.getDaysUntil().longValu
    }
    @Test
    public void testGetNullConcertDate()
    {
        final ConcertModel concert = modelService.create(ConcertModel.class);
        Assert.assertNull("No concert date does not return null: "+ concert.getDaysUntil(), concert.getDaysUntil());
    }
    @Test
    public void testGetPastConcertDate() throws Exception
    {
        final ConcertModel concert = modelService.create(ConcertModel.class);
        final Date pastDate = new Date(new Date().getTime() - 24 * 60 * 60 * 1000);
        concert.setDate(pastDate);
        Assert.assertEquals("Wrong value for concert in the past: "+concert.getDaysUntil().longValue(), 0L, concert.getDaysUntil().longValue())
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the test class yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/testsrc/concerttours/attributehandlers/ConcertDaysUntilAttributeHandlerIntegrationTest` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/attributehandlers/ConcertDaysUntilAttributeHandlerIntegrationTest`.

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/attributehandlers/ConcertDaysUntilAttributeHandlerInt
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\attributehandlers\ConcertDaysUntilAttributeHand
```

3. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

4. Initialize your test tenant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant yunitinit
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant yunitinit
```

5. Run the integration tests and confirm that both new integration tests pass.

```
ant integrationtests -Dtestclasses.packages="concerttours.*"
```

```
ant integrationtests -Dtestclasses.packages=concerttours.*
```

6. View the test results at `<HYBRIS_HOME_DIR>/hybris/log/junit/index.html` and confirm `ConcertDaysUntilAttributeHandlerIntegrationTest` has passed.
7. Run the `testDynamicAttributeIntegrationTest` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeIntegrationTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeIntegrationTest test
```

8. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Test Dynamic Attributes Integration"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Test Dynamic Attributes Integration"
```

Web Page Update

Update the relevant parts of your extension to use the new dynamic attribute.

After introducing new item attributes, you want to apply them. In this case, you want to display the days left until a concert value in the tour details page. To do so, you need to update your `TourFacade`, `TourDetails.jsp` page, and the `ConcertSummaryData` class so that the calculated value of the dynamic attribute `Concerts.daysLeft` appears in the front end.

Parent topic: Explore SAP Commerce Tour

Previous: Dynamic Attributes Integration

Next: Events and Listeners

Update Web Page to Include New Dynamic Attribute

Add handling for the `Concerts.daysLeft` attribute, and display it on the **Tour Details** page of your web application.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testDynamicAttributeView() throws Exception {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/concerttours/bands/A001");
waitFor("a","The Grand Little x Tour");
navigateTo("https://localhost:9002/concerttours/tours/201701");
waitFor("th","Days Until");
assertTrue( waitFor("td","0"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeView test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeView test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Update the bean definition to include the new dynamic attribute in `concerttours-beans.xml`.

   ```
   <bean class = "concerttours.data.ConcertSummaryData">
          <description >Data object for a concert summary</description>
          <property name = "id" type = "String" />
          <property name = "date" type = "java.util.Date" />
          <property name = "venue" type = "String" />
          <property name = "type" type = "String" />
          <property name = "countDown" type="Long"/>
   </bean>
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to update the bean definition yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-beans.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-beansWithDynamicAttributes.xml`

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-beansWithDynamicAttributes.xml $HYBRIS_HOME_DIR/hyb
   ```

   ```
   copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-beansWithDynamicAttributes.xml %HYBRIS_HOME_D
   ```

3. Update the default tour facade implementation to include setting the new dynamic attribute.

   ```
   package concerttours.facades.impl;
   import de.hybris.platform.core.model.product.ProductModel;
   import de.hybris.platform.product.ProductService;
   import de.hybris.platform.variants.model.VariantProductModel;
   import java.util.ArrayList;
   import java.util.List;
   import org.springframework.beans.factory.annotation.Required;
   import concerttours.data.ConcertSummaryData;
   import concerttours.data.TourData;
   import concerttours.enums.ConcertType;
   import concerttours.facades.TourFacade;
   import concerttours.model.ConcertModel;

   public class DefaultTourFacade implements TourFacade
   {
       private ProductService productService;
       @Override
       public TourData getTourDetails(final String tourId)
       {
           if (tourId == null)
           {
               throw new IllegalArgumentException("Tour id cannot be null");
           }
           final ProductModel product = productService.getProductForCode(tourId);
           if (product == null)
           {
               return null;
           }

      // Create a list of ConcertSummaryData from the matches
           final List<ConcertSummaryData> concerts = new ArrayList<>();
           if (product.getVariants() != null)
           {
               for (final VariantProductModel variant : product.getVariants())
               {
                   if (variant instanceof ConcertModel)
                   {
                       final ConcertModel concert = (ConcertModel) variant;
                       final ConcertSummaryData summary = new ConcertSummaryData();
                       summary.setId(concert.getCode());
   ```

```
                    summary.setDate(concert.getDate());
                    summary.setVenue(concert.getVenue());
                    summary.setType(concert.getConcertType() == ConcertType.OPENAIR ? "Outdoors" : "Indoors");
                    summary.setCountDown(concert.getDaysUntil());
                    concerts.add(summary);
                }
            }
        }

        // Now we can create the TourData transfer object
        final TourData tourData = new TourData();
        tourData.setId(product.getCode());
        tourData.setTourName(product.getName());
        tourData.setDescription(product.getDescription());
        tourData.setConcerts(concerts);
        return tourData;
    }
    @Required
    public void setProductService(final ProductService productService)
    {
        this.productService = productService;
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to update the default tour facade implementation yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/facades/impl/DefaultTourFacade.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultTourFacadeWithDynamicAttributes`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultTourFacadeWithDynamicAttributes.java
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\facades\impl\DefaultTourFacadeWithDynamicAttributes
```

4. Update the view page.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<!doctype html>
<html>
<title>Tour Details</title>
<body>
    <h1>Tour Details</h1>
    Tour Details for ${tour.tourName}
    <p>${tour.description}</p>
    <p>Schedule:</p>
    <table>
        <tr><th>Venue</th><th></th><th>Date</th><th>Days Until</th></tr>
        <c:forEach var="concert" items="${tour.concerts}">
            <tr><td>${concert.venue}</td><td>${concert.type}</td><td><fmt:formatDate pattern="dd MMM yyyy" value="${concert.date}" /></td><td>$
        </c:forEach>
    </table>
    <a href="../bands">Back to Band List</a>
</body>
</html>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to update the view page yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/web/webroot/WEB-INF/views/TourDetails.jsp` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/web/webroot/WEB-INF/views/TourDetailsWithDynamicAttributes.jsp`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/web/webroot/WEB-INF/views/TourDetailsWithDynamicAttributes.jsp $HYBRIS_HOM
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\web\webroot\WEB-INF\views\TourDetailsWithDynamicAttributes.jsp %HYBR
```

5. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

6. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

7. Navigate to `https://localhost:9002/concerttours/bands` and view the list of bands.

   Each tour now shows a **Days Until** value, which gives the number of days until that tour takes place.

8. Run the `testDynamicAttributeView` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeView test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testDynamicAttributeView test
```

9. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Update Web Page to Include New Dynamic Attribute"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Update Web Page to Include New Dynamic Attribute"
```

Events and Listeners

The Event System is a framework provided by the `ServiceLayer` that allows you to send and receive events within SAP Commerce.

You can set up components of your extension to publish events that are then received by registered listeners. Listeners are objects that are notified of events and perform business logic depending on the event that occured. Events can be published either locally or across a cluster of nodes. You can register new listeners as Spring beans in your Spring configuration XML file.

The platform defines and publishes events for a number of predefined types of event. These include the `AfterItemCreationEvent` type, items of which are published after any new data item is saved to the database. To process these `AfterItemCreationEvent` events, you provide a listener class and register it with the event framework.

For your `concerttours` extension, you want to generate items of news that you can potentially send out to registered subscribers through various channels. You want to create a new `News` item whenever a new band is signed. In your listener class, you override the `onEvent` method to specify the code you want to execute when this event occurs. In this case, you are checking to see if the new item is a band. If it is, you ask the platform model service to create and save a new news item.

Parent topic: Explore SAP Commerce Tour

Previous: Web Page Update

Next: Interceptors and Custom Events

Related Information

Event System

Create a Listener

Write a listener class that creates a new `News` item when a new `Band` item is created and saved to the database.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testNewsEvents() {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/platform/init");
waitForThenClickButtonWithText("Initialize");
waitForThenClickOkInAlertWindow();
waitForInitToComplete();
closeBrowser();

canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/console/flexsearch");
waitForFlexQueryFieldThenSubmit("SELECT {headline} FROM {News}");
assertTrue( waitFor("td","New band, Banned"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testNewsEvents test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testNewsEvents test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Add a new itemtype called News to `concerttours-items.xml`

   ```
   <itemtype generate="true" code="News" autocreate="true">
    <deployment table="News" typecode="30270" />
    <attributes>
        <attribute qualifier="date" type="java.util.Date">
            <description>date of news item</description>
            <persistence type="property" />
        </attribute>
        <attribute qualifier="headline" type="java.lang.String">
            <description>short headline for the news item</description>
            <persistence type="property" />
        </attribute>
        <attribute qualifier="content" type="java.lang.String">
            <description>fuller description of the news item</description>
            <persistence type="property" />
        </attribute>
    </attributes>
   </itemtype>
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the `News` itemtype or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/concerttours-items.xml` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithNews.xml`

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithNews.xml $HYBRIS_HOME_DIR/hybris/bin/custo
   ```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-itemsWithNews.xml %HYBRIS_HOME_DIR%\hybris\bi
```

3. Create a listener class that listens for new Band events under the `src` folder of the `concerttours` extension.

```
package concerttours.events;
import de.hybris.platform.servicelayer.event.events.AfterItemCreationEvent;
import de.hybris.platform.servicelayer.event.impl.AbstractEventListener;
import de.hybris.platform.servicelayer.model.ModelService;
import java.util.Date;
import concerttours.model.BandModel;
import concerttours.model.NewsModel;

public class NewBandEventListener extends AbstractEventListener<AfterItemCreationEvent>
{
    private static final String NEW_BAND_HEADLINE = "New band, %s";
    private static final String NEW_BAND_CONTENT = "There is a new band in town called, %s. Tour news to be announced soon.";
    private ModelService modelService;
    public ModelService getModelService()
    {
        return modelService;
    }
    public void setModelService(final ModelService modelService)
    {
        this.modelService = modelService;
    }
    @Override
    protected void onEvent(final AfterItemCreationEvent event)
    {
        if (event != null && event.getSource() != null)
        {
            final Object object = modelService.get(event.getSource());
            if (object instanceof BandModel)
            {
                final BandModel band = (BandModel) object;
                final String headline = String.format(NEW_BAND_HEADLINE, band.getName());
                final String content = String.format(NEW_BAND_CONTENT, band.getName());
                final NewsModel news = modelService.create(NewsModel.class);
                news.setDate(new Date());
                news.setHeadline(headline);
                news.setContent(content);
                modelService.save(news);
            }
        }
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the `NewBandEventListener.java` listener class or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/events/NewBandEventListener.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/NewBandEventListener.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/NewBandEventListener.java $HYBRIS_HOME_DIR/hybr
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\events\NewBandEventListener.java %HYBRIS_HOM
```

4. Register the new listener in Spring by adding the `concerttourEventListener` Spring bean definition to the `concerttours-spring.xml` file in the `resources` folder of the `concerttours` extension.

```
<bean id="concerttourEventListener" class="concerttours.events.NewBandEventListener" parent="abstractEventListener">
  <property name="modelService" ref="modelService" />
</bean>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to add the `concerttourEventListener` Spring bean or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-spring.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/NewBandEventListener.java`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withListener.xml $HYBRIS_HOME_DIR/hybris/bin
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring-withListener.xml %HYBRIS_HOME_DIR%\hyb
```

5. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

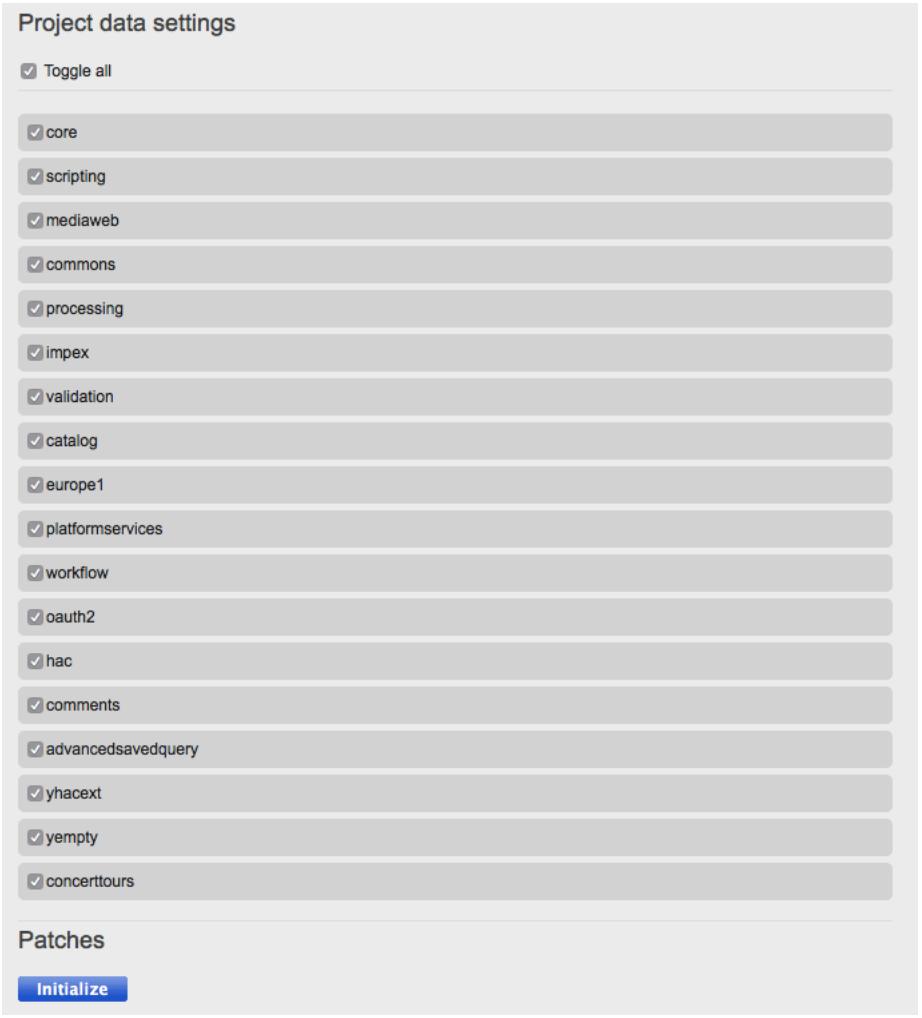
```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

6. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

7. Reinitialize in the SAP Commerce Administration Console (Administration Console)
   a. Log into the SAP Commerce Administration Console (Administration Console) at `https://localhost:9002` using the login, *admin*, and the password that you defined as an environment variable.
   b. Navigate to **Platform** > **Initialization** or go directly to `https://localhost:9002/platform/init`.
   c. In the **Project data settings** area ensure all the check boxes are selected.
   d. Click the **Initialize** button.



8. Check that the news items were successfully created.
    a. In the SAP Commerce Administration Console, go the **Console** tab and select the **FlexibleSearch** option.

       The **FlexibleSearch** page appears.

    b. Retrieve all the News items from the database by entering the query `select {pk},{date}, {headline}, {content} from {News}` into the **FlexibleSearch** query text box and click the **Execute** button.
9. Run the `testNewsEvents` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testNewsEvents test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testNewsEvents test
```

10. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Create a Listener"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Create a Listener"
```

Interceptors and Custom Events

Interceptors intercept model object lifecycle transitions and, depending on the conditions of that transition, may publish an event when they do so.

Model classes represent SAP Commerce items. Each model contains all item attributes from all extensions, which unifies access to the data for an item.

An interceptor addresses a particular step in the life cycle of a model. When the life cycle reaches a certain step, you can activate a corresponding interceptor. An interceptor can modify the model, raise an exception to interrupt the current step, or publish an event if the model matches certain criteria. For example, you could check that an attribute contains certain values before saving the model.

Parent topic: Explore SAP Commerce Tour

Previous: Events and Listeners

Next: Cluster-Aware Events

Related Information

Models
Interceptors

Create an Interceptor

Create an interceptor that throws an exception if someone tries to save a `Band` item with negative album sales, and publishes a custom event if a band's album sales exceed a certain number.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testEventInterceptorIntegrationTest() {
assertTrue( checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"BandAlbumSalesEventListenerIntegrationTest\" package=\"c
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testEventInterceptorIntegrationTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testEventInterceptorIntegrationTest test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Create a new event class.

   ```
   package concerttours.events;
   import de.hybris.platform.servicelayer.event.events.AbstractEvent;

   public class BandAlbumSalesEvent extends AbstractEvent
   {
       private final String code;
       private final String name;
       private final Long sales;
       public BandAlbumSalesEvent(final String code, final String name, final Long sales)
       {
           super();
           this.code = code;
           this.name = name;
           this.sales = sales;
       }
       public String getCode()
       {
           return code;
       }
       public String getName()
       {
           return name;
       }
       public Long getSales()
       {
           return sales;
       }
       @Override
       public String toString()
       {
           return this.name;
       }
   }
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the event class or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/events/BandAlbumSalesEvent.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/BandAlbumSalesEvent.java`

   ```
   ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/BandAlbumSalesEvent.java $HYBRIS_HOME_DIR/hybri
   ```

   ```
   echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\events\BandAlbumSalesEvent.java %HYBRIS_HOME
   ```

3. Create an interceptor for the custom event.

   ```
   package concerttours.interceptors;
   import static de.hybris.platform.servicelayer.model.ModelContextUtils.getItemModelContext;
   import de.hybris.platform.servicelayer.event.EventService;
   import de.hybris.platform.servicelayer.interceptor.InterceptorContext;
   import de.hybris.platform.servicelayer.interceptor.InterceptorException;
   import de.hybris.platform.servicelayer.interceptor.PrepareInterceptor;
   import de.hybris.platform.servicelayer.interceptor.ValidateInterceptor;
   import org.springframework.beans.factory.annotation.Autowired;
   import concerttours.events.BandAlbumSalesEvent;
   import concerttours.model.BandModel;

   public class BandAlbumSalesInterceptor implements ValidateInterceptor, PrepareInterceptor
   {
       private static final long BIG_SALES = 50000L;
       private static final long NEGATIVE_SALES = 0L;
       @Autowired
       private EventService eventService;
       @Override
       public void onValidate(final Object model, final InterceptorContext ctx) throws InterceptorException
       {
           if (model instanceof BandModel)
           {
   ```

```
                final BandModel band = (BandModel) model;
                final Long sales = band.getAlbumSales();
                if (sales != null && sales.longValue() < NEGATIVE_SALES)
                {
                    throw new InterceptorException("Album sales must be positive");
                }
            }
        }
        @Override
        public void onPrepare(final Object model, final InterceptorContext ctx) throws InterceptorException
        {
            if (model instanceof BandModel)
            {
                final BandModel band = (BandModel) model;
                if (hasBecomeBig(band, ctx))
                {
                    eventService.publishEvent(new BandAlbumSalesEvent(band.getCode(), band.getName(), band.getAlbumSales()));
                }
            }
        }
        private boolean hasBecomeBig(final BandModel band, final InterceptorContext ctx)
        {
            final Long sales = band.getAlbumSales();
            if (sales != null && sales.longValue() >= BIG_SALES)
            {
                if (ctx.isNew(band))
                {
                    return true;
                }
                else
                {
                    final Long oldValue = getItemModelContext(band).getOriginalValue(BandModel.ALBUMSALES);
                    if (oldValue == null || oldValue.intValue() < BIG_SALES)
                    {
                        return true;
                    }
                }
            }
            return false;
        }
    }
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the interceptor or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/interceptors/BandAlbumSalesInterceptor.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/interceptors/BandAlbumSalesInterceptor.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/interceptors/BandAlbumSalesInterceptor.java $HYBRIS_HO
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\interceptors\BandAlbumSalesInterceptor.java
```

4. Create an event listener.

```
package concerttours.events;
import de.hybris.platform.servicelayer.event.impl.AbstractEventListener;
import de.hybris.platform.servicelayer.model.ModelService;
import java.util.Date;
import concerttours.model.NewsModel;

public class BandAlbumSalesEventListener extends AbstractEventListener<BandAlbumSalesEvent>
{
    private static final String BAND_SALES_HEADLINE = "%s album sales exceed 50000";
    private static final String BAND_SALES_CONTENT = "%s album sales reported as %d";
    private ModelService modelService;
    public ModelService getModelService()
    {
        return modelService;
    }
    public void setModelService(final ModelService modelService)
    {
        this.modelService = modelService;
    }
    @Override
    protected void onEvent(final BandAlbumSalesEvent event)
    {
        if (event != null)
        {
            final String headline = String.format(BAND_SALES_HEADLINE, event.getName());
            final String content = String.format(BAND_SALES_CONTENT, event.getName(), event.getSales());
            final NewsModel news = modelService.create(NewsModel.class);
            news.setDate(new Date());
            news.setHeadline(headline);
            news.setContent(content);
            modelService.save(news);
        }
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the event listener or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/events/BandAlbumSalesEventListener.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/BandAlbumSalesEventListener.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/BandAlbumSalesEventListener.java $HYBRIS_HOME_D
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\events\BandAlbumSalesEventListener.java %HYB
```

5. Register the interceptor and the event listener.

```
<bean id="bandAlbumSalesInterceptor" class="concerttours.interceptors.BandAlbumSalesInterceptor" />
<bean id="BandInterceptorMapping" class="de.hybris.platform.servicelayer.interceptor.impl.InterceptorMapping">
  <property name="interceptor" ref="bandAlbumSalesInterceptor" />
  <property name="typeCode" value="Band" />
  </bean>
  <bean id="bandAlbumSalesEventListener" class="concerttours.events.BandAlbumSalesEventListener" parent="abstractEventListener" >
```

```
    <property name="modelService" ref="modelService" />
    </bean>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to register the interceptor and the event listener or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/concerttours-spring.xml` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withInterceptor.xml`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withInterceptor.xml  $HYBRIS_HOME_DIR/hybris
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring-withInterceptor.xml %HYBRIS_HOME_DIR%\
```

6. Write an integration test by creating a new test class, `BandAlbumSalesEventListenerIntegrationTest`, under the `testsrc` folder of the `concerttours` extension.

```
package concerttours.events;
import de.hybris.bootstrap.annotations.IntegrationTest;
import de.hybris.platform.servicelayer.ServicelayerTest;
import de.hybris.platform.servicelayer.exceptions.ModelSavingException;
import de.hybris.platform.servicelayer.model.ModelService;
import de.hybris.platform.servicelayer.search.FlexibleSearchService;
import java.lang.InterruptedException;
import java.util.List;
import java.util.concurrent.TimeUnit;
import javax.annotation.Resource;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import concerttours.model.BandModel;
import concerttours.model.NewsModel;
import de.hybris.platform.core.Registry;
import org.springframework.jdbc.core.JdbcTemplate;

@IntegrationTest
public class BandAlbumSalesEventListenerIntegrationTest extends ServicelayerTest
{
    @Resource
    private FlexibleSearchService flexibleSearchService;
    @Resource
    private ModelService modelService;
    private static final String BAND_CODE = "101-JAZ";
    private static final String BAND_NAME = "Tight Notes";
    private static final String BAND_HISTORY = "New contemporary, 7-piece Jaz unit from London, formed in 2015";
    private static final Long MANY_ALBUMS_SOLD = Long.valueOf(1000000L);
    @Before
    public void setUp() throws Exception
    {
        try {
          Thread.sleep(TimeUnit.SECONDS.toMillis(1));
          new JdbcTemplate(Registry.getCurrentTenant().getDataSource()).execute("CHECKPOINT");
          Thread.sleep(TimeUnit.SECONDS.toMillis(1));
        }
        catch (InterruptedException exc) {}
        createCoreData();
        createDefaultCatalog();
    }
    @Test(expected = ModelSavingException.class)
    public void testValidationInterceptor()
    {
   final BandModel band = modelService.create(BandModel.class);
   band.setCode(BAND_CODE);
   band.setAlbumSales(Long.valueOf(-10L));
   modelService.save(band);
    }

    @Test
    public void testEventSending() throws Exception
    {
   final BandModel band = modelService.create(BandModel.class);
   band.setCode(BAND_CODE);
   band.setName(BAND_NAME);
   band.setHistory(BAND_HISTORY);
   band.setAlbumSales(MANY_ALBUMS_SOLD);
   modelService.save(band);
   final NewsModel news = findLastNews();
   Assert.assertTrue("Unexpected news: " + news.getHeadline(), news.getHeadline().contains(BAND_NAME));
    }
    private NewsModel findLastNews()
    {
   final StringBuilder builder = new StringBuilder();
   builder.append("SELECT {n:").append(NewsModel.PK).append("} ");
   builder.append("FROM {").append(NewsModel._TYPECODE).append(" AS n} ");
   builder.append("ORDER BY ").append("{n:").append(NewsModel.CREATIONTIME).append("} DESC");
   final List<NewsModel> list = flexibleSearchService.<NewsModel> search(builder.toString()).getResult();
   if (list.isEmpty())
   {
       return null;
   }
   else
   {
       return list.get(0);
   }
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to write the integration test or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/testsrc/concerttours/events/BandAlbumSalesEventListenerIntegrationTest.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/events/BandAlbumSalesEventListenerIntegrationTe`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/events/BandAlbumSalesEventListenerIntegrationTest.
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\events\BandAlbumSalesEventListenerIntegr
```

7. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

8. Initialize your test tenant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant yunitinit
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant yunitinit
```

9. Run the `BandAlbumSalesEventListenerIntegrationTest` test and confirm that you see *found 7 testclasse(s)* in the log output.

```
ant integrationtests -Dtestclasses.packages="concerttours.*"
```

```
ant integrationtests -Dtestclasses.packages=concerttours.*
```

10. View the test results at *<HYBRIS_HOME_DIR>*`/hybris/log/junit/index.html` and confirm that `BandAlbumSalesEventListenerIntegrationTest` test has passed.
11. Run the `testEventInterceptorIntegrationTest` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testEventInterceptorIntegrationTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testEventInterceptorIntegrationTest test
```

12. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Create a Custom Event"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Create a Custom Event"
```

Cluster-Aware Events

SAP Commerce supports cluster-aware events. With cluster-aware events, SAP Commerce can process events in separate threads, or on particular nodes of a cluster.

By default, SAP Commerce processes all events synchronously. But synchronous messaging has some disadvantages:

- The main thread waits until all events are processed. For example, a band is not saved until a corresponding news item is created.
- A slow listener can cause the system to throw a timeout exception.

Parent topic: Explore SAP Commerce Tour

Previous: Interceptors and Custom Events

Next: Cron Jobs

Set Up Asynchronous Messaging

Implement asynchronous messaging and cluster-aware processing to avoid potential wait time issues.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testAsyncEventInterceptorIntegrationTest() {
assertTrue(
checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"BandAlbumSalesEventListenerIntegrationTest\" package=\"concerttours.
checkTestSuiteXMLMatches("(.*)testcase classname=\"concerttours.events.BandAlbumSalesEventListenerIntegrationTest\" name=\"testEventSendingAsync\"(.
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testAsyncEventInterceptorIntegrationTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testAsyncEventInterceptorIntegrationTest test
```

Procedure

1. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

2. Modify the `BandAlbumSalesEvent` to become cluster-aware.

   Add `implements ClusterAwareEvent`, then define a new publish method that ensures events are published and processed only by the node they come from.

```
package concerttours.events;
import de.hybris.platform.servicelayer.event.ClusterAwareEvent;
```

```
import de.hybris.platform.servicelayer.event.events.AbstractEvent;

public class BandAlbumSalesEvent extends AbstractEvent implements ClusterAwareEvent
{
    private final String code;
    private final String name;
    private final Long sales;
    public BandAlbumSalesEvent(final String code, final String name, final Long sales)
    {
        super();
        this.code = code;
        this.name = name;
        this.sales = sales;
    }
    public String getCode()
    {
        return code;
    }
    public String getName()
    {
        return name;
    }
    public Long getSales()
    {
        return sales;
    }
    @Override
    public String toString()
    {
        return this.name;
    }
    @Override
    public boolean publish(final int sourceNodeId, final int targetNodeId)
    {
        return (sourceNodeId == targetNodeId);
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the cluster-aware event yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/events/BandAlbumSalesEvent.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/BandAlbumSalesEventAsync.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/BandAlbumSalesEventAsync.java $HYBRIS_HOME_DIR/
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\events\BandAlbumSalesEventAsync.java %HYBRIS
```

3. Add a new test named `testEventSendingAsync` to account for a possible delay and comment out the `@Test` annotation to disable `testEventSending()`.

```
@Test
public void testEventSendingAsync() throws Exception
{
final BandModel band = modelService.create(BandModel.class);
band.setCode(BAND_CODE);
band.setName(BAND_NAME);
band.setHistory(BAND_HISTORY);
band.setAlbumSales(MANY_ALBUMS_SOLD);
modelService.save(band);
// add sleep here to wait for event processing thread to complete
Thread.sleep(2000L);
final NewsModel news = findLastNews();
Assert.assertTrue("Unexpected news: " + news.getHeadline(), news.getHeadline().contains(BAND_NAME));
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to update the test method yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/testsrc/concerttours/events/BandAlbumSalesEventListenerIntegrationTest.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/events/BandAlbumSalesEventListenerIntegrationTestAs`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/events/BandAlbumSalesEventListenerIntegrationTestA
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\events\BandAlbumSalesEventListenerIntegr
```

Note

The test extends ServicelayerTest but not ServicelayerTransactionalTest. This is intentional. If you run this test in a transaction, you might run into database isolation configuration issues. For example, if you run this test in a transaction and the database isolation level is set to `repeatable read`, the news item would be outside your test transaction since you create news in a separate thread. In other words, whatever gets committed outside the transaction is not visible within the transaction (repeatable read isolation) causing the test to fail. Changing the isolation level to less strict (`read committed`) would cause the test to pass.

4. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

5. Initialize your test tenant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant yunitinit
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant yunitinit
```

6. Run the `BandAlbumSalesEventListenerIntegrationTest` test and confirm that you see *found 7 testclasse(s)* in the log output.

```
ant integrationtests -Dtestclasses.packages="concerttours.*"
```

```
ant integrationtests -Dtestclasses.packages=concerttours.*
```

7. View the test results at *<HYBRIS_HOME_DIR>*/hybris/log/junit/index.html and confirm that BandAlbumSalesEventListenerIntegrationTest test has passed.
8. Run the testAsyncEventInterceptorIntegrationTest acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testAsyncEventInterceptorIntegrationTest test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testAsyncEventInterceptorIntegrationTest test
```

9. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Set Up Asynchronous Messaging"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Set Up Asynchronous Messaging"
```

Cron Jobs

SAP Commerce provides a means to set up regular tasks. With these tasks, or cron jobs, you can repeatedly perform complex business logic at particular times and intervals.

You may want to perform an inventory every Sunday at midnight, for example, or notify the web administrator of the peak loads on the servers every hour. You can achieve this through a combination of dedicated classes for the business logic, and the embedded cron job management functionality of SAP Commerce.

The first step in implementing a cron job is to place your business logic in a class that extends AbstractJobPerformable. You then develop a new, simple service to provide access to the news items, and encapsulate that business logic in a job class. The steps are similar to those you took to create the band service, and serve to reinforce this pattern.

The SAP Commerce platform ships with a useful email utility class, MailUtils, that simplifies the sending of e-mails with commons mail API. You make use of the MailUtils class to create an e-mail message object populated with values from the local.properties file in your config directory.

Parent topic: Explore SAP Commerce Tour

Previous: Cluster-Aware Events

Next: Triggers

Related Information

The Cronjob Service

Write the Business Logic for a Cron Job

Write a cron job to send daily news summaries.

Prerequisites

The following acceptance test is included in your Hybris123Tests.java file.

```
public void testSendNewsJobIntegrationTest() {
assertTrue( checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"SendNewsJobIntegrationTest\" package=\"concerttours.jobs'
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSendNewsJobIntegrationTest test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSendNewsJobIntegrationTest test
```

Procedure

1. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

2. Create a news data access interface in *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/daos/NewsDAO.java

```
package concerttours.daos;
import java.util.Date;
import java.util.List;
import concerttours.model.NewsModel;

public interface NewsDAO
{
    List<NewsModel> getNewsOfTheDay(Date date);
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the news data access interface yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/daos/NewsDAO.java with the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/src/concerttours/daos/NewsDAO.java

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/daos/NewsDAO.java $HYBRIS_HOME_DIR/hybris/bin/custom/c
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\daos\NewsDAO.java %HYBRIS_HOME_DIR%\hybris\b
```

3. Create a news data access implementation in *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/daos/impl/DefaultNewsDAO.java

```
package concerttours.daos.impl;

import de.hybris.platform.servicelayer.search.FlexibleSearchQuery;
import de.hybris.platform.servicelayer.search.FlexibleSearchService;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Collections;
import java.util.Date;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import concerttours.daos.NewsDAO;
import concerttours.model.NewsModel;

@Component(value = "newsDAO")
public class DefaultNewsDAO implements NewsDAO
{
   private static final String SQL_DATE_FORMAT = "yyyy-MM-dd";
   @Autowired
   private FlexibleSearchService flexibleSearchService;
   @Override
   public List<NewsModel> getNewsOfTheDay(final Date date)
   {
       if (date == null)
       {
           return Collections.emptyList();
       }
       final String theDay = new SimpleDateFormat(SQL_DATE_FORMAT).format(date);
       final String theNextDay = new SimpleDateFormat(SQL_DATE_FORMAT).format(oneDayAfter(date));
       final String queryString = //
               "SELECT {p:" + NewsModel.PK + "} "//
                       + "FROM {" + NewsModel._TYPECODE + " AS p} " //
                       + "WHERE {date} >= DATE \'" + theDay + "\' "//
                       + "AND {date} <= DATE \'" + theNextDay + "\'";
       final FlexibleSearchQuery query = new FlexibleSearchQuery(queryString);
       return flexibleSearchService.<NewsModel> search(query).getResult();
   }
   private Date oneDayAfter(final Date date)
   {
       final Calendar cal = Calendar.getInstance();
       cal.setTime(date);
       cal.add(Calendar.DATE, 1);
       return cal.getTime();
   }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the implementation yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/daos/impl/DefaultNewsDAO.java with the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/src/concerttours/daos/impl/DefaultNewsDAO.java

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/daos/impl/DefaultNewsDAO.java $HYBRIS_HOME_DIR/hybris/
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\daos\impl\DefaultNewsDAO.java %HYBRIS_HOME_D
```

4. Create a news service interface called NewsService.java in *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/service

```
package concerttours.service;
import java.util.Date;
import java.util.List;
import concerttours.model.NewsModel;

public interface NewsService
{
    List<NewsModel> getNewsOfTheDay(Date date);
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the news service interface yourself or you want to skip this step, copy *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/src/concerttours/service/NewsService.java into *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/service

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/service/NewsService.java $HYBRIS_HOME_DIR/hybris/bin/c
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\service\NewsService.java %HYBRIS_HOME_DIR%\h
```

5. Create a news service implementation called DefaultNewsService.java in *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/service/impl

```
package concerttours.service.impl;
import java.util.Date;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import concerttours.daos.NewsDAO;
import concerttours.model.NewsModel;
import concerttours.service.NewsService;

public class DefaultNewsService implements NewsService
{
    @Autowired
    private NewsDAO newsDAO;


    public void setNewsDAO(final NewsDAO newsDAO)
    {
        this.newsDAO = newsDAO;
    }

    @Override
    public List<NewsModel> getNewsOfTheDay(final Date date)
    {
```

```
        return newsDAO.getNewsOfTheDay(date);
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the implementation yourself or you want to skip this step, copy `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/service/impl/DefaultNewsService.java` into `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/service/impl`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/service/impl/DefaultNewsService.java $HYBRIS_HOME_DIR/
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\service\impl\DefaultNewsService.java %HYBRIS
```

6. Configure the email server by adding email server properties to the `local.properties` file.

   The example shows how you would do this with a generic email account. Update the SMTP server, username, and password to match yours.

   a. Add the following properties to the `local.properties` file in the config directory, adjusting them making sure to replace *<smtp.email.com>*, *<your username>*, *<your password>*, and *<your email address>* with your username, password, and email address. If port 465 is not available, change the port.

   ```
   mail.from=concerttours-no-reply@hybris.de
   mail.replyto=concerttours-no-reply@hybris.de
   mail.smtp.server=smtp.email.com
   mail.smtp.port=465

   mail.smtp.user=<your username>
   mail.smtp.password=<your password>
   news_summary_mailing_address=<your email address>
   ```

   b. Update your email account to (temporarily) allow less secure applications to access it by selecting the Allow less secure apps option.

   SAP Commerce 123 Interactive Shortcut: If you are unable to add the properties yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/config/local.properties` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/resources/config/local.properties`

   ```
   cat $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/config/local.properties >> $HYBRIS_HOME_DIR/hybris/config/local
   ```

   ```
   type %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\config\local.properties >> %HYBRIS_HOME_DIR%\hybris\config\lo
   ```

7. Create the job class.

```
package concerttours.jobs;
import de.hybris.platform.cronjob.enums.CronJobResult;
import de.hybris.platform.cronjob.enums.CronJobStatus;
import de.hybris.platform.cronjob.model.CronJobModel;
import de.hybris.platform.servicelayer.config.ConfigurationService;
import de.hybris.platform.servicelayer.cronjob.AbstractJobPerformable;
import de.hybris.platform.servicelayer.cronjob.PerformResult;
import de.hybris.platform.util.mail.MailUtils;
import java.util.Date;
import java.util.List;
import org.apache.commons.mail.Email;
import org.apache.commons.mail.EmailException;
import org.apache.commons.configuration.Configuration;
import org.apache.log4j.Logger;
import org.springframework.beans.factory.annotation.Required;
import concerttours.model.NewsModel;
import concerttours.service.NewsService;

public class SendNewsJob extends AbstractJobPerformable<CronJobModel>
{
    private static final Logger LOG = Logger.getLogger(SendNewsJob.class);
    private NewsService newsService;
    private ConfigurationService configurationService;

    @Required
    public NewsService getNewsService()
    {
        return newsService;
    }

    @Required
    public ConfigurationService getConfigurationService()
    {
        return configurationService;
    }

    @Required
    public void setNewsService(final NewsService newsService)
    {
        this.newsService = newsService;
    }

    @Required
    public void setConfigurationService(final ConfigurationService configurationService)
    {
        this.configurationService = configurationService;
    }
    @Override
    public PerformResult perform(final CronJobModel cronJob)
    {
        LOG.info("Sending news mails. Note that org.apache.commons.mail.send() can block if behind a firewall/proxy.");
        final List<NewsModel> newsItems = getNewsService().getNewsOfTheDay(new Date());
        if (newsItems.isEmpty())
        {
            LOG.info("No news items for today, skipping send of ranking mails");
            return new PerformResult(CronJobResult.SUCCESS, CronJobStatus.FINISHED);
        }
        final StringBuilder mailContentBuilder = new StringBuilder(2000);
        int index = 1;
        mailContentBuilder.append("Todays news summary:\n\n");
        for (final NewsModel news : newsItems)
        {
            mailContentBuilder.append(index++);
            mailContentBuilder.append(". ");
            mailContentBuilder.append(news.getHeadline());
```

```
                mailContentBuilder.append("\n");
                mailContentBuilder.append(news.getContent());
                mailContentBuilder.append("\n\n");
            }
            try
            {
                sendEmail(mailContentBuilder.toString());
            }
            catch (final EmailException e)
            {
                LOG.error("Problem sending new email. Note that org.apache.commons.mail.send() can block if behind a firewall/proxy.)");
                LOG.error("Problem sending new email.", e);
                return new PerformResult(CronJobResult.FAILURE, CronJobStatus.FINISHED);
            }
            return new PerformResult(CronJobResult.SUCCESS, CronJobStatus.FINISHED);
        }
        private void sendEmail(final String message) throws EmailException
        {
            final String subject = "Daily News Summary";
            // get mail service configuration
            final Email email = MailUtils.getPreConfiguredEmail();
            //send message
            Configuration config = getConfigurationService().getConfiguration();
            String recipient = config.getString("news_summary_mailing_address", null);
            email.addTo(recipient);
            email.setSubject(subject);
            email.setMsg(message);
            email.setTLS(true);
            email.send();
            LOG.info(subject);
            LOG.info(message);
        }
    }
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the job class yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/jobs/SendNewsJob.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/jobs/SendNewsJob.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/jobs/SendNewsJob.java $HYBRIS_HOME_DIR/hybris/bin/cust
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\jobs\SendNewsJob.java %HYBRIS_HOME_DIR%\hybr
```

8. Register the service and job in Spring.

```
<alias name="defaultNewsService" alias="newsService" />
<bean id="defaultNewsService" class="concerttours.service.impl.DefaultNewsService">
    <property name="newsDAO" ref="newsDAO" />
</bean>
<bean id="sendNewsJob" class="concerttours.jobs.SendNewsJob" parent="abstractJobPerformable">
    <property name="newsService" ref="newsService" />
    <property name="configurationService" ref="configurationService" />
</bean>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to register the service and job yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/concerttours-spring.xml` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withNewsService.xml`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withNewsService.xml $HYBRIS_HOME_DIR/hybris/
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring-withNewsService.xml %HYBRIS_HOME_DIR%\
```

9. Write an integration test to test successful job creation.

```
package concerttours.jobs;
import de.hybris.bootstrap.annotations.IntegrationTest;
import de.hybris.platform.cronjob.enums.CronJobResult;
import de.hybris.platform.servicelayer.ServicelayerTransactionalTest;
import de.hybris.platform.servicelayer.cronjob.PerformResult;
import de.hybris.platform.servicelayer.model.ModelService;
import java.lang.InterruptedException;
import java.util.Date;
import java.util.concurrent.TimeUnit;
import de.hybris.platform.core.Registry;
import org.springframework.jdbc.core.JdbcTemplate;
import javax.annotation.Resource;
import org.junit.Assert;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import concerttours.model.NewsModel;

@IntegrationTest
public class SendNewsJobIntegrationTest extends ServicelayerTransactionalTest
{
    @Resource
    private ModelService modelService;
    @Resource
    private SendNewsJob sendNewsJob;

    @Before
    public void setUp() throws Exception
    {
        try {
                Thread.sleep(TimeUnit.SECONDS.toMillis(1));
                new JdbcTemplate(Registry.getCurrentTenant().getDataSource()).execute("CHECKPOINT");
                Thread.sleep(TimeUnit.SECONDS.toMillis(1));
        }
        catch (InterruptedException exc) {}
    }

    @Test
    public void testNoNewsItems()
    {
        final PerformResult result = sendNewsJob.perform(null);
        Assert.assertEquals("Job did not perform correctly", CronJobResult.SUCCESS, result.getResult());
```

```
        }

        @Test
        public void testSendingNews() throws Exception
        {
            final NewsModel news1 = modelService.create(NewsModel.class);
            news1.setHeadline("test headline 1");
            news1.setContent("test content 1");
            news1.setDate(new Date());
            final NewsModel news2 = modelService.create(NewsModel.class);
            news2.setHeadline("test headline 2");
            news2.setContent("test content 2");
            news2.setDate(new Date());
            modelService.saveAll();
            final PerformResult result = sendNewsJob.perform(null);
            Assert.assertEquals("Job did not perform correctly", CronJobResult.SUCCESS, result.getResult());
        }

        @After
        public void tearDown() {

        }
    }
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the integration test yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/testsrc/concerttours/jobs/SendNewsJobIntegrationTest.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/jobs/SendNewsJobIntegrationTest.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/jobs/SendNewsJobIntegrationTest.java $HYBRIS_HOME_
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\jobs\SendNewsJobIntegrationTest.java %HY
```

10. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

11. Initialize your test tenant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant yunitinit
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant yunitinit
```

12. Run the integration test and confirm that `SendNewsJobIntegrationTest` passes.

```
ant integrationtests -Dtestclasses.packages="concerttours.*"
```

```
ant integrationtests -Dtestclasses.packages=concerttours.*
```

13. View the test results at `<HYBRIS_HOME_DIR>/hybris/log/junit/index.html`.
14. Run the `testSendNewsJobIntegrationTest` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSendNewsJobIntegrationTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSendNewsJobIntegrationTest test
```

15. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Write the Business Logic for a Cron Job"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Write the Business Logic for a Cron Job"
```

Triggers

With your business logic successfully factored into a job class, you can trigger its execution with the use of a cron job.

A cron job consists of the `job` class containing the business logic, and a trigger to start the job at regular intervals. The first step in setting up a new cron job is to notify SAP Commerce of your new class by creating your essential data. During the creation of essential data, a `ServicelayerJob` item is created for every Spring definition that has a class implementing the `JobPerformable` interface. The code attribute of each of the new job item is set to the name of the relevant Spring bean.

Once a `ServicelayerJob` item is created, you can create a cron job to wrap the `ServicelayerJob`, and define a trigger to that starts it.

A cron expression is a string comprised of 6 or 7 fields separated by white space. Fields can contain any of the allowed values, along with various combinations of allowed special characters for that field.

Field Name	Mandatory	Allowed Values	Allowed Special Characters
Seconds	YES	0-59	, - * /
Minutes	YES	0-59	, - * /
Hours	YES	0-23	, - * /
Day of month	YES	31-Jan	, - * ? / L W
Month	YES	1-12 or JAN-DEC	, - * /
Day of week	YES	1-7 or SUN-SAT	, - * ? / L #

Field Name	Mandatory	Allowed Values	Allowed Special Characters
Year	NO	empty, 1970-2099	, - * /

The first cron job you create for your extension runs on a daily basis, and sends out summaries of new items by email to a specific email address or distribution list. Then you create a second cron job using a scripting language instead of a Java class. Using a scripting language, you can add cron jobs to a system without having to rebuild and redeploy it.

Parent topic: Explore SAP Commerce Tour

Previous: Cron Jobs

Next: Groovy

Related Information

Cron Trigger Tutorial

Set Up Your Cron Job with a Trigger Using Impex

Set up your cron job to get triggered at regular intervals.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testSendingNewsMails() throws Exception {
long timeSinceLastMailWasSentMS = LogHelper.getMSSinceLastNewsMailsLogged();
assertTrue("A log of the last mail sent should have been timestamped recently " + timeSinceLastMailWasSentMS,
timeSinceLastMailWasSentMS < 5*60*1000);
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSendingNewsMails test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSendingNewsMails test
```

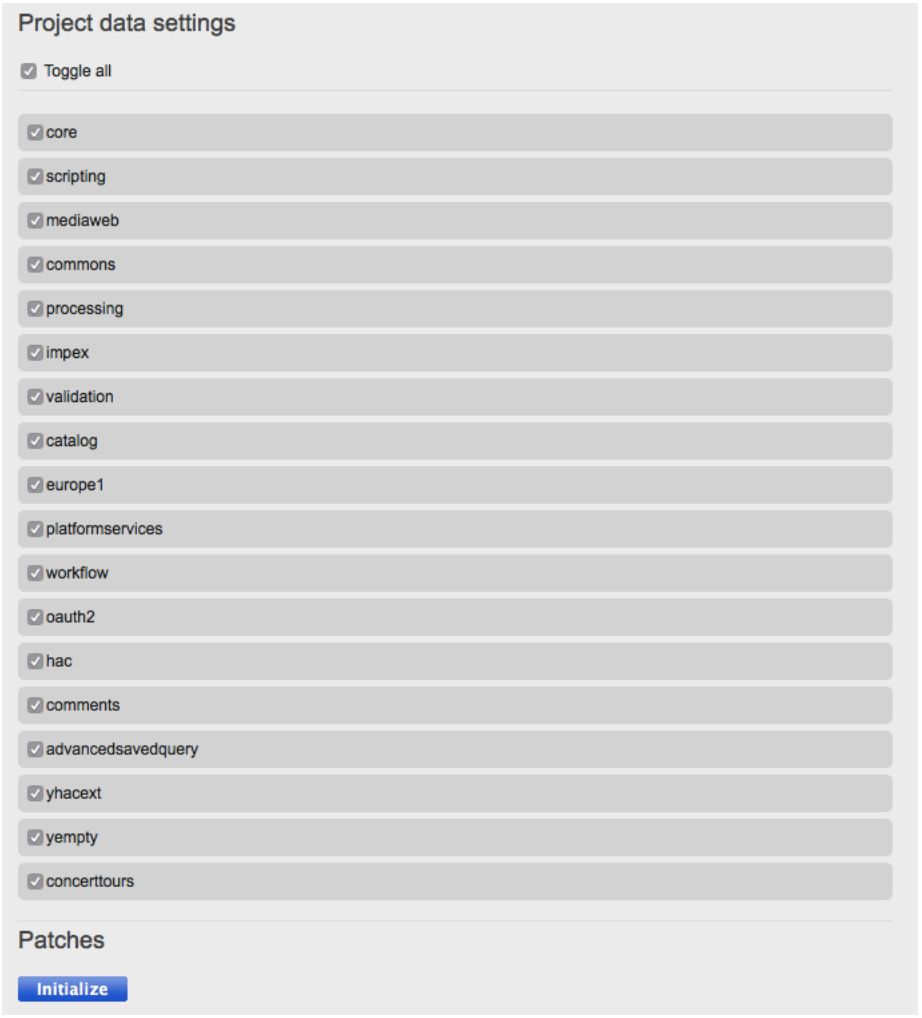
Procedure

1. Start SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
   ```

2. Reinitialize in the SAP Commerce Administration Console (Administration Console).
   a. Log into the SAP Commerce Administration Console (Administration Console) at `https://localhost:9002` using the login, *admin*, and the password that you defined as an environment variable.
   b. Navigate to **Platform** > **Initialization** or go directly to `https://localhost:9002/platform/init`
   c. In the **Project data settings** area ensure all the checkboxes are selected.
   d. Click the **Initialize** button.



SAP Commerce 123 Interactive Shortcut: If you are unable to initialize the system yourself or you want to skip this step, run the following maven command.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#simulateInitialization test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#simulateInitialization test
```

3. Check that the ServicelayerJob was successfully created.

   a. In the SAP Commerce Administration Console, go to the Console > FlexibleSearch option.

      The FlexibleSearch page appears.

   b. To check that a new ServicelayerJob item has been created, enter the query `select {code} from {servicelayerjob} where {code} = 'sendNewsJob'` into the FlexibleSearch query text box and click the Execute button.

4. Create the cron job and the trigger.

   a. In the SAP Commerce Administration Console, go the Console tab and select the Impex Import option.

   b. Expand the settings area, select the Enable code execution checkbox, and import the impex script.

```
INSERT_UPDATE ServicelayerJob;code[unique=true];SpringId;
;sendNewsJob;sendNewsJob

# Define the cron job and the job that it wraps
INSERT_UPDATE CronJob; code[unique=true];job(code);singleExecutable;sessionLanguage(isocode)
;sendNewsCronJob;sendNewsJob;false;de

# Define the trigger that periodically invokes the cron job
INSERT_UPDATE Trigger;cronjob(code)[unique=true];cronExpression
#% afterEach: impex.getLastImportedItem().setActivationTime(new Date());
; sendNewsCronJob; 0/10 * * * * ?
```

      The console output shows the job being triggered every 10 seconds. In a production environment, you would change the trigger to run once at the end of the day.

SAP Commerce 123 Interactive Shortcut: If you are unable to trigger the cron job yourself or you want to skip this step, run the following command.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#simulateLoadingJobImpex test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#simulateLoadingJobImpex test
```

5. So that the ImpEx script is loaded by CoC, create the `essentialdataJobs.impex` file in `resources/impex/` with this same impex content so that these items are recreated during the essential data loading phase of any initialization or update operation.

SAP Commerce 123 Interactive Shortcut: If you are unable to create the `essentialdataJobs.impex` file or you want to skip this step, copy *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/script/essentialdataJobs.impex` to *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/impex`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/script/essentialdataJobs.impex $HYBRIS_HOME_DIR/hybris/bin/cu
```

```
xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\script\essentialdataJobs.impex %HYBRIS_HOME_DIR%\hybris\bin\
```

6. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

7. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

8. Run the `testSendingNewsMails` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSendingNewsMails test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testSendingNewsMails test
```

9. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Set Up Your Cron Job with a Trigger Using Impex"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Set Up Your Cron Job with a Trigger Using Impex"
```

Groovy

You can use the Groovy scripting language to write jobs for execution.

Parent topic: Explore SAP Commerce Tour

Previous: Triggers

Next: Backoffice Administration Cockpit

Create a Cronjob Script Using Groovy

Write a job using Groovy to modify the approval status of the concerts.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testGroovyScript() throws Exception {
        canLoginToHybrisCommerce();
        navigateTo("https://localhost:9002/console/flexsearch");
        waitForFlexQueryFieldThenSubmit("SELECT {p.pk}, {p.code}, {p.name}, {q.code} FROM {Concert AS p}, {ArticleApprovalStatus AS q} WHERE {p.appr
        assertTrue( waitForText("check") );
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testGroovyScript test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testGroovyScript test
```

Procedure

1. Retrieve the list of concerts with their approval status.
    a. Log into the SAP Commerce Administration Console (Administration Console) at `https://localhost:9002` using the login, *admin*, and the password that you defined as an environment variable.
    b. In the SAP Commerce Administration Console, go to the **Console** tab and select the **FlexibleSearch** option.

    The **FlexibleSearch** page appears.

    c. To check the approval status of all concerts, enter the query `SELECT {p.pk}, {p.code}, {p.name}, {q.code} FROM {Concert AS p}, {ArticleApprovalStatus AS q} WHERE {p.approvalstatus} = {q.pk}` into the **FlexibleSearch** query text box and click the **Execute** button.
2. Create a job using Groovy script.
    a. In the SAP Commerce Administration Console, go the **Console** tab, select the **Scripting Languages** option, and select **Groovy** in the **Script type** drop-down menu.
    b. Enter the Groovy script.

```
import de.hybris.platform.cronjob.enums.*
import de.hybris.platform.servicelayer.cronjob.PerformResult
import de.hybris.platform.servicelayer.search.*
import de.hybris.platform.servicelayer.model.*
import de.hybris.platform.catalog.enums.ArticleApprovalStatus
import concerttours.model.ConcertModel
```

```
searchService = spring.getBean("flexibleSearchService")
modelService = spring.getBean("modelService")
query = new FlexibleSearchQuery("Select {pk} from {Concert}");
searchService.search(query).getResult().each {
  if (it.daysUntil < 1)
  {
    it.approvalStatus = ArticleApprovalStatus.CHECK
  }
  modelService.saveAll()
}
```

c. Save the script in the database by entering the name `clearoldconcerts` in the **code** field above the script and click the **Save** button.

d. Run the script by clicking on the **Execute** button.

The script should run without errors.

e. Click on the **RollBack** button so that it shows Commit and execute the script again.

This time the results are committed in the database, and you should be able to see them in other consoles.

SAP Commerce 123 Interactive Shortcut: If you are unable to create the Groovy script yourself or you want to skip this step, run the following command.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#simulateGroovyScript test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#simulateGroovyScript test
```

3. Retrieve the list of concerts with their approval status again.

a. In the SAP Commerce Administration Console, go to the **Console** tab and select the **FlexibleSearch** option.

The **FlexibleSearch** page appears.

b. To check that a new ServicelayerJob item has been created, enter the query `SELECT {p.pk}, {p.code}, {p.name}, {q.code} FROM {Concert AS p}, {ArticleApprovalStatus AS q} WHERE {p.approvalstatus} = {q.pk}` into the **FlexibleSearch** query text box and click the **Execute** button.

The approvalStatus values have changed.

4. Wrap the job in a cron job and invoke with a trigger by running the ImpEx script in the **Impex Import** console.

```
# Define the ScriptingJob
INSERT_UPDATE ScriptingJob; code[unique= true ];scriptURI
;clearoldconcertsJob;model://clearoldconcerts

# Define the CronJob
INSERT_UPDATE CronJob; code[unique= true ];job(code);sessionLanguage(isocode)
;clearoldconcertsCronJob;clearoldconcertsJob;en

# Define the trigger
INSERT_UPDATE Trigger;cronjob(code)[unique=true];cronExpression
#% afterEach: impex.getLastImportedItem().setActivationTime(new Date());
; clearoldconcertsCronJob; 0/10 * * * * ?
```

The console output shows the job being triggered every 10 seconds. In a production environment, you would change the trigger to run once at the end of the day.

5. Run the `testGroovyScript` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testGroovyScript test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testGroovyScript test
```

6. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Create a Cronjob Script Using Groovy"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Create a Cronjob Script Using Groovy"
```

Backoffice Administration Cockpit

Backoffice Administration Cockpit is a user-friendly, browser-based, user interface for viewing, creating, and manipulating data within SAP Commerce.

Backoffice Administration Cockpit consists of widgets, each with a specific function. This allows you to manage a range of different data types. The primary data used by SAP Commerce is available in the Administration perspective. You can select individual categories from the explorer tree, a menu listing in the left hand side of the main window. From here, you can view, list, enter, and edit catalog and product data, user data including roles and permissions, price settings, internationalization, or basic system configuration data.

For the `concerttours` extension, Backoffice Administration Cockpit is useful for viewing and editing band and tour data in a user-friendly way.

Parent topic: Explore SAP Commerce Tour

Previous: Groovy

Next: Localization

Related Information

Backoffice Administration Cockpit

Install Backoffice Administration Cockpit

Add the `platformbackoffice` extension to your installation, and use it to list your items.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testBackofficeProductListingContainsTheBands() {
    loginToBackOffice();
    accessBackofficeProducts();
    assertTrue(waitFor("span", "Grand Tour - Montreal"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBackofficeProductListingContainsTheBands test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBackofficeProductListingContainsTheBands test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Add the `platformbackoffice` extension to your `localextensions.xml` file located in the `config` folder.

   ```
   <hybrisconfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="resources/schemas/extensions.xsd">
       <extensions>
           <!--
                   All extensions located in ${HYBRIS_BIN_DIR}/platform/ext are automatically loaded.
                   More information about how to configure available extensions can be found here : https://help.sap.com/viewer/DRAFT/b490bb4e85bc
           -->
           <path dir="${HYBRIS_BIN_DIR}" autoload="false"/>
           <extension name="concerttours"/>
           <extension name="platformbackoffice"/>

           <!-- ext-template -->
           <extension name="yempty" />
           <extension name="yhacext" />

       </extensions>
   </hybrisconfig>
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to add the extension or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/config/localextensions.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/withbackoffice/localextensions.xml`.

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/withbackoffice/localextensions.xml $HYBRIS_HOME_DIR/hybris/config/localextensions.xml
   ```

   ```
   copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\withbackoffice\localextensions.xml %HYBRIS_HOME_DIR%\hybris\config\localextension
   ```

3. Rebuild the system.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant build
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant build
   ```

4. Reinitialize the system.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant initialize
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant initialize
   ```

5. Start SAP Commerce.

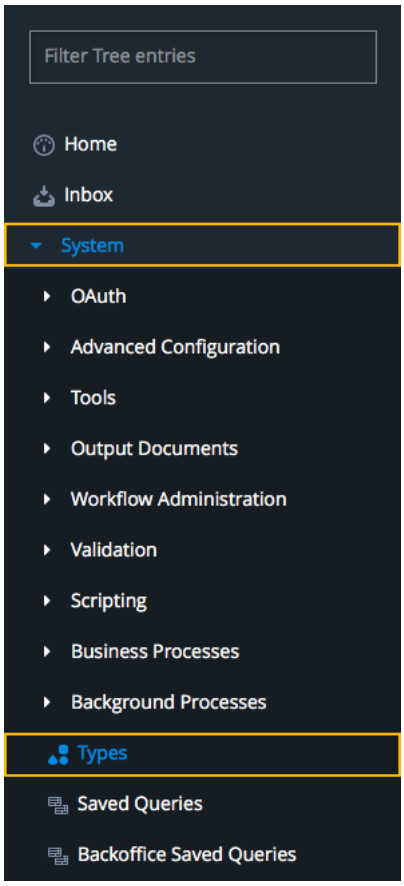
   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
   ```

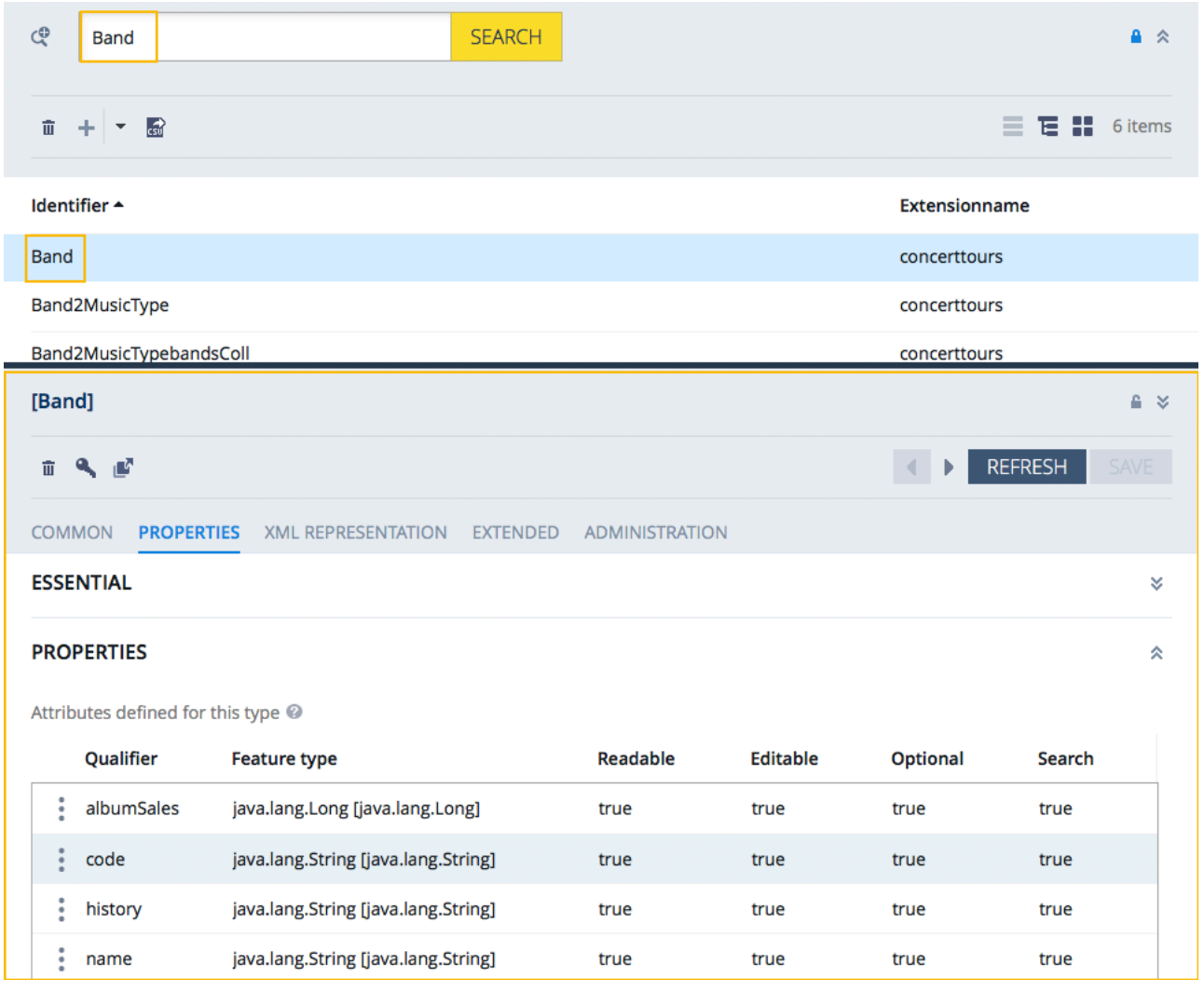
6. Explore Backoffice Administration Cockpit.

   a. Open Backoffice at `https://localhost:9002/backoffice` and log in with user `admin` and the password that you defined as an environment variable.

   b. In the Explorer Tree of Backoffice Administration Cockpit, navigate to **System** > **Types**.

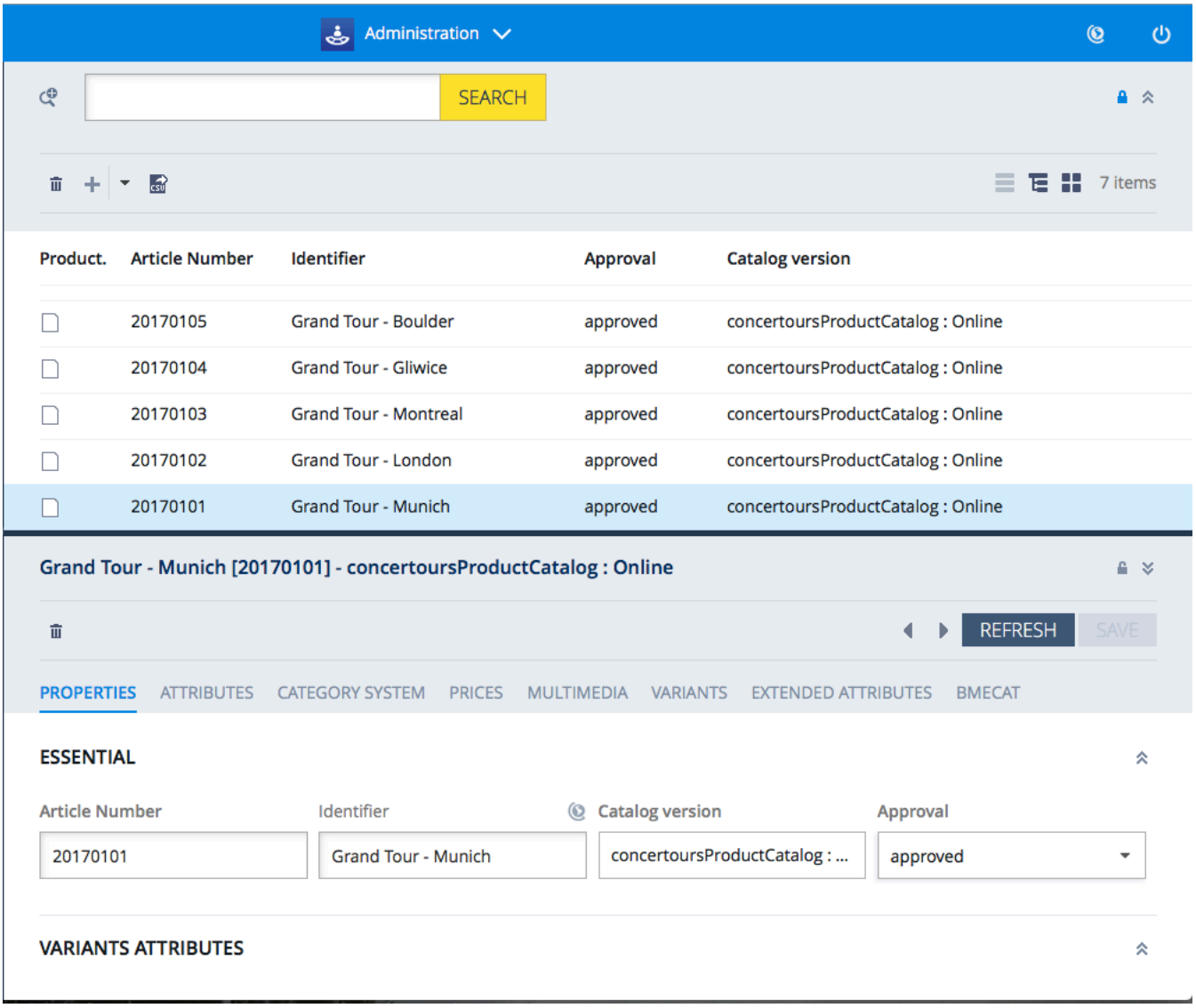


c. In the Search field, enter Band. Click the Band item in the search results to view the details of the item type.



7. In the Explorer Tree of Backoffice Administration Cockpit, navigate to Catalog > Products to locate the concert tours.

   You should see the listing of concerts.



8. Run the `testBackofficeProductListingContainsTheBands` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBackofficeProductListingContainsTheBands test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBackofficeProductListingContainsTheBands test
```

9. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Install Backoffice Administration Cockpit"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Install Backoffice Administration Cockpit"
```

Localization

Localization is intended to adapt SAP Commerce to multiple languages.

Localization is built into SAP Commerce Platform from the ground up, not bolted-on as an after thought. As a result, you can declare attributes and relations of item types as `localized` in an extension's `*-items.xml` file, and the system automatically provides for multiple values for different locales. Even the type system names themselves can be localized so that when item type and attribute names appear in user interfaces, they can be in the user's chosen language.

When the SAP build system encounters the `localized` keyword, instead of a single valued attribute, it creates a map of values keyed by language for that attribute, and generates an equivalent construct in the database.

Impex has a built-in syntax for specifying values for localized values, and it is often more manageable to separate content across impex files by language. While it is not necessary to use separate files for language-specific content, it usually makes it much easier to understand and manage. Particularly, it makes adding and removing the content for a specific language easier.

In the hybris123 integration tests, you used the model service to create new instances of bands, concerts, and others. The model service handles the provision of a locale to use when working with localized attributes. Unfortunately, when it comes to unit tests you don't have the context provided by the model service. As a result, when you call the usual getter or setter of a localized attribute, the model object doesn't know which language to map the value to. If you are only accessing localized attributes in your test code, you can fix this problem by using a getter and setter that specifies the locale that you want used as an extra parameter.

If the simple accessors are being called in the code that you are testing, there is no way to provide a `LocaleProvider` for them to use. As a result, you can no longer test that code in a unit test. You have to fall back to using integration tests.

If for example you replace the calls to `getHistory()` and `setHistory()` in the test `DefaultBandFacadeUnitTest` methods with `getHistory(Locale.ENGLISH)` and `setHistory(BAND_HISTORY, Locale.ENGLISH)`, that test will still fail because the code that you are testing in the actual `DefaultBandFacade` class calls the simple `getHistory()`

accessor.

The only way you can make this test pass is to have the code you are testing check if you are running outside the SAP Commerce Server context and modify its behaviour to cope with that. That makes no sense, so unfortunately you have to stop using this unit test and rely on the slower integration test instead.

Parent topic: Explore SAP Commerce Tour

Previous: Backoffice Administration Cockpit

Next: Localization in Backoffice Administration Cockpit

Related Information

Internationalization and Localization
Character Encoding Requirements for Localization Files
Languages and Localization

Localize Attribute Values Using ImpEx

Localize attribute values and type system names.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testLocalizedServiceLayerTest() {
assertTrue( checkTestSuiteXMLMatches("(.*)testsuite errors=\"2\" (.*) name=\"DefaultBandFacadeUnitTest\" package=\"concerttours.facades.impl\" tests
checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"DefaultBandServiceUnitTest\" package=\"concerttours.service.impl\" t
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testLocalizedServiceLayerTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testLocalizedServiceLayerTest test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Modify the `concerttours-items.xml` file to enable localization of the `hashtag` and `history` attributes.

   ```
   <attribute qualifier="hashtag" type="localized:java.lang.String">
     <description>hashtag of concert tour for social media</description>
     <persistence type="property" />
   </attribute>
   ```

   ```
   <attribute qualifier="history" type="localized:java.lang.String">
                 <description>history of band</description>
                 <persistence type="property" />
            </attribute>
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to modify the `concerttours-items.xml` file or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-items.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithLocalization.xml`

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithLocalization.xml $HYBRIS_HOME_DIR/hybris/b
   ```

   ```
   copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-itemsWithLocalization.xml %HYBRIS_HOME_DIR%\h
   ```

3. Split the `concerttours-bands.impex` file into three files to localize the related impex files: one file for the language-independent content (`/impex/concerttours-bands.impex`), one for the English content (`/impex/concerttours-bands-en.impex`), and one for the German content (`/impex/concerttours-bands-de.impex`).

   ```
   # ImpEx for Importing Bands into Little Concert Tours Store

   INSERT_UPDATE Band;code[unique=true];name;albumSales
   ;A001;yRock;1000000
   ;A006;yBand;
   ;A003;yJazz;7
   ;A004;Banned;427
   ;A002;Sirken;2000
   ;A005;The Choir;49000
   ;A007;The Quiet;1200
   ```

   ```
   # ImpEx for Importing Bands' German attributes into Little Concert Tours Store
   $lang=de

   INSERT_UPDATE Band;code[unique=true];history[lang=$lang]
   ;A001;Gelegentliche Tribut-Rock-Band bestehend aus Führungskräften eines führenden Commerce-Software-Anbieters
   ;A006;Die niederländische Tribut-Rock-Band, die im Jahr 2013 mit klassischen Rock-Melodien aus den sechziger, siebziger und achtziger Jahren ge
   ```

```
;A003;Experimental Jazz Gruppe aus London, die viele musikalische Notizen zusammen in unerwarteten Kombinationen und Sequenzen spielt
;A004;Verjüngte polnische Jungenband aus den 1990er Jahren - dieses Genre der Popmusik am zweifelhaftesten am besten
;A002;A cappella singing group mit Sitz in München; Eine erhebende Mischung aus traditionellen und zeitgenössischen Liedern
;A005;Enthusiastischer, lärmender Gospelchor singt traditionelle Gospel-Songs aus dem tiefen Süden
;A007;Englisch Choralgesellschaft, spezialisiert auf wunderschön arrangierte, beruhigende Melodien und Lieder


# ImpEx for Importing Bands' English content into Little Concert Tours Store
$lang=en

INSERT_UPDATE Band;code[unique=true];history[lang=$lang]
;A001;Occasional tribute rock band comprising senior managers from a leading commerce software vendor
;A006;Dutch tribute rock band formed in 2013 playing classic rock tunes from the sixties, seventies and eighties
;A003;Experimental Jazz group from London playing many musical notes together in unexpected combinations and sequences
;A004;Rejuvenated Polish boy band from the 1990s - this genre of pop music at its most dubious best
;A002;A cappella singing group based in Munich; an uplifting blend of traditional and contemporaneous songs
;A005;Enthusiastic, noisy gospel choir singing traditional gospel songs from the deep south
;A007;English choral society specialising in beautifully arranged, soothing melodies and songs
```

SAP Commerce 123 Interactive Shortcut: If you are unable to split the `concerttours-bands.impex` file into three files or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/impex/` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/impex/withlocalization/*`

```
mkdir -p $HYBRIS_HOME_DIR/hybris/bin/custom/concerttours/resources/impex; cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/impex/withloc
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\impex\withlocalization\concerttours-bands* %HYBRIS_HOME_DIR%\hybris\bin\cu
```

4. Modify `concerttours-yBandTour.impex` to include both English and German hashtags.

```
# Insert Products
INSERT_UPDATE Product;code[unique=true];name;band(code);hashtag[lang=en];hashtag[lang=de];$supercategories;manufacturerName;manufacturerAID;uni
;201701;The Grand Little Tour;A001;GrandLittle;GrossWenig;;x;y;pieces;;Concert
```

SAP Commerce 123 Interactive Shortcut: If you are unable to modify the `concerttours-yBandTour.impex` file or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/impex/` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/impex/withlocalization/*`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/impex/withlocalization/concerttours-yBandTour.impex $HYBRIS_HOME_DIR/hybris/bin/cust
```

```
xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\impex\withlocalization\concerttours-yBandTour.impex %HYBRIS_HOME_DIR%\hybris\bin\cu
```

5. Update the `ConcerttoursCustomSetup` class to load the three localized files instead of the default, which is one non-localized file.

```
@SystemSetup(type = SystemSetup.Type.PROJECT)
   public boolean addMyProjectData()
   {
      LOG.info("Starting custom project data loading for the Concerttours extension");
      impexImport("/impex/concerttours-bands.impex");
      impexImport("/impex/concerttours-bands-en.impex");
      impexImport("/impex/concerttours-bands-de.impex");
      impexImport("/impex/concerttours-yBandTour.impex");
      LOG.info("Custom project data loading for the Concerttours extension completed.");
      return true;
   }
```

SAP Commerce 123 Interactive Shortcut: If you are unable to update the `ConcerttoursCustomSetup` class or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/setup/ConcerttoursCustomSetup.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/setup/LocalizedConcerttoursCustomSetup.java`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/setup/LocalizedConcerttoursCustomSetup.java $HYBRIS_HO
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\setup\LocalizedConcerttoursCustomSetup.java
```

6. In the `DefaultBandServiceUnitTest` class, modify the call to the `setHistory()` method to include a reference to the required locale so that the unit test is locale-aware.

```
bandModel.setHistory(BAND_HISTORY, Locale.ENGLISH);
```

7. Add `import java.util.Locale;` to the list of imports in the `DefaultBandServiceUnitTest.java` test.

```
package concerttours.service.impl;
import static org.junit.Assert.assertEquals;
import static org.mockito.Mockito.mock;
import static org.mockito.Mockito.when;
import de.hybris.bootstrap.annotations.UnitTest;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import org.junit.Before;
import org.junit.Test;
import concerttours.daos.BandDAO;
import concerttours.model.BandModel;
import java.util.Locale;
```

SAP Commerce 123 Interactive Shortcut: If you are unable to modify the test or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceUnitTest.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/service/impl/Localized_DefaultBandServiceUnitTe`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/service/impl/Localized_DefaultBandServiceUnitTest.jav
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\service\impl\Localized_DefaultBandServiceUnitTe
```

8. Rebuild and re-initialize the database.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all initialize yunitinit
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all initialize yunitinit
```

9. Run the tests and note that `DefaultBandFacadeUnitTest` fails but `DefaultBandServiceUnitTest` passes for reasons described in the discussion on Localization.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant alltests -Dtestclasses.packages="concerttours.*"
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant alltests -Dtestclasses.packages=concerttours.*
```

10. Run the `testLocalizedServiceLayerTest` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testLocalizedServiceLayerTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testLocalizedServiceLayerTest test
```

11. Remove the file `DefaultBandFacadeUnitTest.java`.

    The `DefaultBandFacadeUnitTest` is no longer compatible and will fail, for reasons described in the Localization overview.

```
rm $HYBRIS_HOME_DIR/hybris/bin/custom/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeUnitTest.java
```

```
del %HYBRIS_HOME_DIR%\hybris\bin\custom\concerttours\testsrc\concerttours\facades\impl\DefaultBandFacadeUnitTest.java
```

12. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Localize Attribute Values Using ImpEx"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Localize Attribute Values Using ImpEx"
```

13. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

Localization in Backoffice Administration Cockpit

You can define localized values for item type attributes directly in the Backoffice Administration Cockpit.

Parent topic: Explore SAP Commerce Tour

Previous: Localization

Next: Validation

Localize Attribute Values Using Backoffice

Provide localized string entries for the **history** field of items of type `Band`.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testBackofficeLocalization() throws Exception {
        canLoginToHybrisCommerce();
        navigateTo("https://localhost:9002/platform/init");
        waitForThenClickButtonWithText("Initialize");
        waitForThenClickOkInAlertWindow();
        waitForInitToComplete();
        closeBrowser();

        loginToBackOffice("Deutsch");
        waitForThenClickMenuItem("System");

        if (VersionHelper.getVersion().equals(Version.V2211))
                waitForthenScrollToThenClick("//span[text()='Typen']");
        else
                waitForThenClickMenuItem("Typen");

        searchForConcertInBackoffice();
        waitForThenAndClickSpan("Concert");
        waitForThenAndClickSpan("Eigenschaften");
        waitForThenClickDotsBySpan("daysUntil");
        waitForThenAndClickSpan("Details bearbeiten", "Edit Details");
        assertTrue( waitForValue("input", "Tage bis es stattfindet") );
}
```

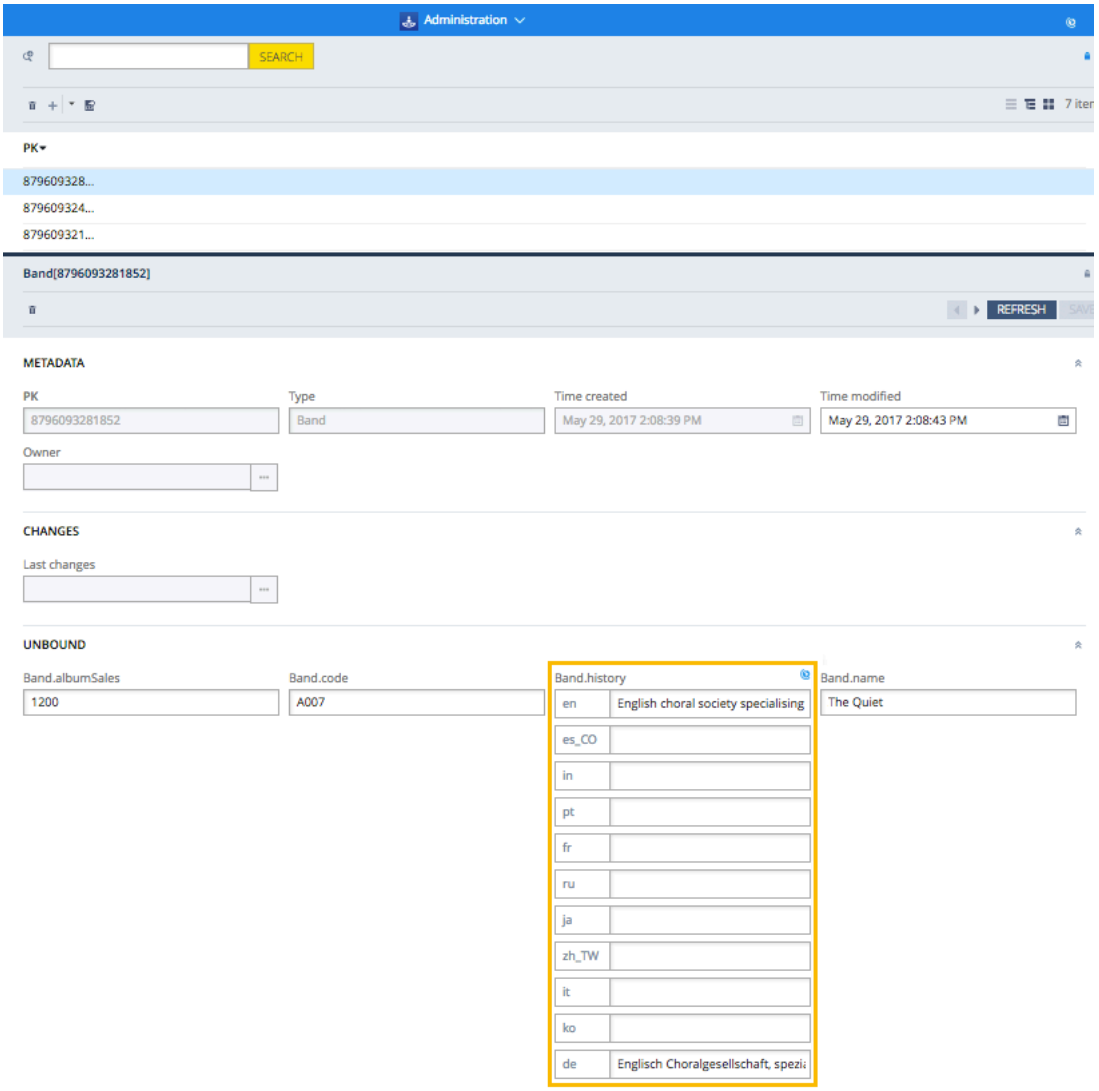
Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBackofficeLocalization test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBackofficeLocalization test
```

Procedure

1. Open Backoffice at `https://localhost:9002/backoffice` and log in with user `admin` and the password that you defined as an environment variable.
2. In the Explorer Tree of Backoffice Administration Cockpit, navigate to System > Types.
3. Search for the Band item type and, in the details panel toolbar, click the Search by type button  .
4. Select one of the Band items, and explore its localized history values by clicking the Band.history world icon.



5. Add localized string entries in the `concerttours/resources/localization/concerttours-locales_en.properties` type system localization file, and the `concerttours/resources/localization/concerttours-locales_de.properties` type system localization file, to provide the localized version of the Type System.

   Each entry is in the form `key=localized value`, as follows:

```
type.Product.hashtag.name=hashtag
type.Product.hashtag.description=social media tag

type.Band.name=Band
type.Band.description=a group or band of musicians of singers
type.Band.code.name=code
type.Band.code.description=unique short id for the band
type.Band.name.name=name
type.Band.name.description=name of the band
type.Band.history.name=history
type.Band.history.description=short story about the band
type.Band.albumSales.name=album sales
type.Band.albumSales.description=number of albums sold

type.Concert.name=Concert
type.Concert.description=a performance by a band
type.Concert.venue.name=venue
type.Concert.venue.description=where it is being held
type.Concert.date.name=date
type.Concert.date.description=when it is being held
type.Concert.concertType.name=concert type
type.Concert.concertType.description=type of the concert
type.Concert.daysUntil.name=days until
type.Concert.daysUntil.description=days until it is being held

type.News.name=News
type.News.description=news about a band
type.News.date.name=date
type.News.date.description=when the news broke
type.News.headline.name=headline
type.News.headline.description=headline for the news item
```

```
type.News.content.name=content
type.News.content.description=content for the news item


type.Product.hashtag.name=hashtag
type.Product.hashtag.description=Social Media Tag

type.Band.name=Bande
type.Band.description=Eine Gruppe oder eine Band von Musikern oder Sängern
type.Band.code.name=code
type.Band.code.description=Einzigartige kurze ID für die Bande
type.Band.name.name=Name
type.Band.name.description=Name der Bande
type.Band.history.name=Geschichte
type.Band.history.description=Kurzgeschichte über die Bande
type.Band.albumSales.name=Albumverkauf
type.Band.albumSales.description=Anzahl der verkauften Alben

type.Concert.name=Konzert
type.Concert.description=Eine Aufführung von einer Band
type.Concert.venue.name=Schauplatz
type.Concert.venue.description=Wo es stattfindet
type.Concert.date.name=Datum
type.Concert.date.description=Wenn es gehalten wird
type.Concert.concertType.name=Konzertart
type.Concert.concertType.description=Art des Konzertes
type.Concert.daysUntil.name=Tage bis
type.Concert.daysUntil.description=Tage bis es stattfindet

type.News.name=Nachrichten
type.News.description=Nachrichten über eine Bande
type.News.date.name=Datum
type.News.date.description=Als die Nachricht angekündigt wurde
type.News.headline.name=Überschrift
type.News.headline.description=Schlagzeile für die Nachricht
type.News.content.name=Inhalt
type.News.content.description=Inhalt für die Nachricht
```

If you define new item types, attributes and relations, and others, you should provide localized versions of their names that can be used by Backoffice applications such as Backoffice Administration Cockpit.

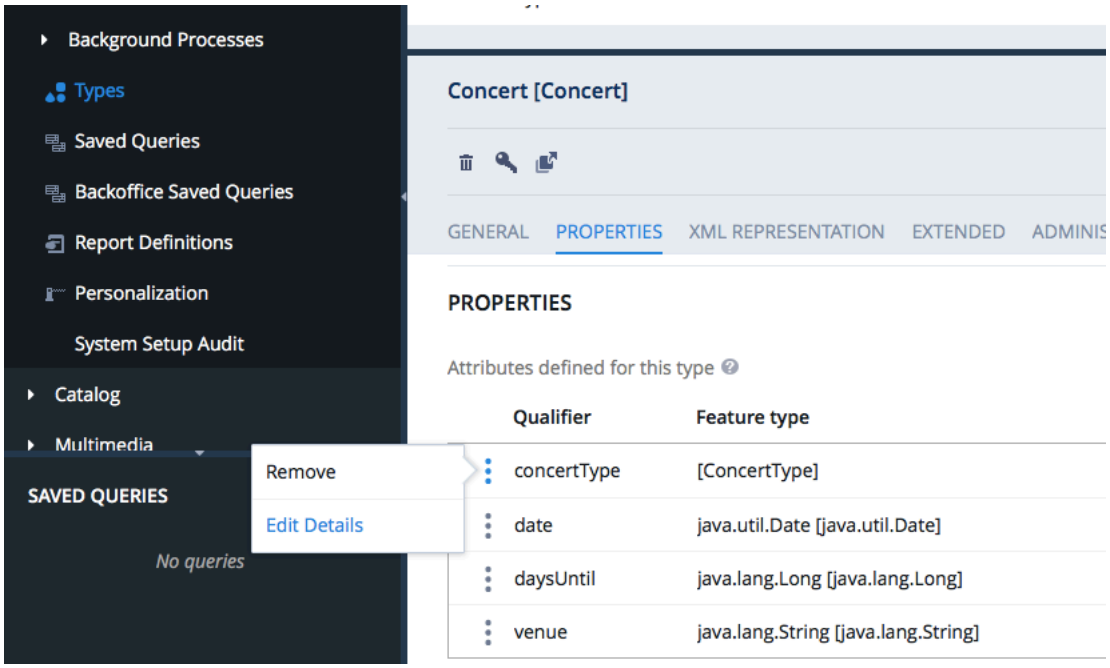
SAP Commerce 123 Interactive Shortcut: If you are unable to add the entries or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/localization/concerttours-locales_en.properties` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/resources/localization/concerttours-locales_en.localizedproperties`, and `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/localization/concerttours-locales_de.properties` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/resources/localization/concerttours-locales_de.localizedproperties`.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/localization/concerttours-locales_en.localizedproperties $HYB
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/localization/concerttours-locales_de.localizedproperties $HYB
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\localization\concerttours-locales_en.localizedprope
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\localization\concerttours-locales_de.localizedprope
```

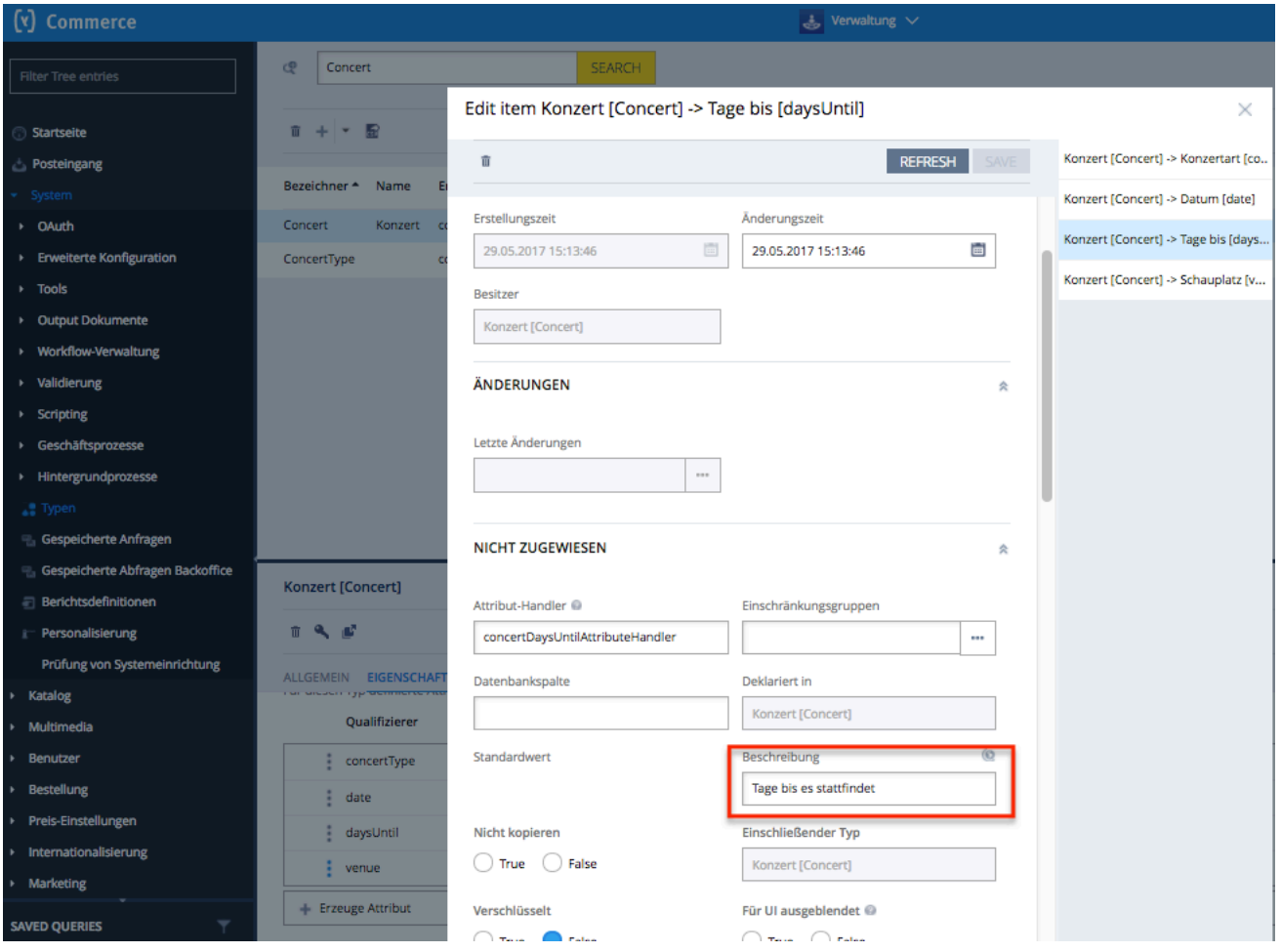
6. Log in to SAP Commerce Administration Cockpit, and update the system with the new localizations from the Platform > Update screen. To save time select only the Localize types option. Deselect everything else.

7. Explore the localized types in Backoffice Administration Cockpit.
   a. Open Backoffice and in the Explorer Tree navigate to System > Types.
   b. Search for `Concert`, then click on the Properties tab.
   c. Click the three dots next to an item, and select Edit Details to explore the property details. Notice the localized name and description.



d. Repeat the last step, this time searching for `Band`, then again for `News`.

8. Log out of Backoffice Administration Cockpit and log in again selecting Deutsch as the language. Notice that the supplied German values are now used for the names and description.



9. Run the `testBackofficeLocalization` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBackofficeLocalization test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBackofficeLocalization test
```

10. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Localize Attribute Values Using Backoffice"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Localize Attribute Values Using Backoffice"
```

Validation

The SAP Commerce data validation framework ensures clean, correct, and useful data. The validation framework is based on the Java validation specification, JSR 303. It offers an easy and extensible way to validate data before it is passed on to the persistence layer.

The data validation framework has several goals:

- To offer a framework for defining data validation constraints in easy and intuitive way
- To validate data before it is saved
- To notify about any validation violations should they occur

Validation logic may be triggered in the following ways:

- Implicitly with the `ValidationInterceptor` that hooks into calls to the `save` method in a model
- Explicitly by manually calling the `validate` method of the `ValidationService`, and passing in a SAP Commerce model or POJO to be validated

When validation violations are found, they are presented to the caller for a resolution. The validation framework does not extend to performing client side validation. Validation happens on the server side only.

The main components of data validation constraints are:

- `ValidationService`: Manages validation constraints, and validates data
- Backoffice Administration Cockpit: Provides a front-end for managing instantiated validation constraint types
- Cockpit integration: Provides users with validation feedback in cockpits

For validation to work, you define constraints. In the procedure that follows you define validation constraints in the `Items.xml` file, but there are more ways to define constraints. You can use an ImpEx file, which allows you to automate the process and quickly add more constraints. ImpEx is an ideal way to work between multiple platforms. You can also define constraints in Backoffice Administration Cockpit. In this way, you can update validation constraints manually and at runtime. To know more about these procedures, see Validation Constraints in Backoffice.

Parent topic: Explore SAP Commerce Tour

Previous: Localization in Backoffice Administration Cockpit

Next: Validation Constraints in Backoffice

Related Information

Data Validation Framework
Validation Service in Backoffice Framework

Define Validation Constraints in the Items.xml File

Define a validation constraint in the `Items.xml` file.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testValidationConstraintViaItemsXml() {
        loginToBackOffice();
        waitForThenClickMenuItem("System");
        if (VersionHelper.getVersion().equals(Version.V2211))
                waitForthenScrollToThenClick("//span[text()='Types']");
        else
                waitForThenClickMenuItem("Types");
        waitForThenDoBackofficeSearch("Band");
        waitForThenClick("span","Band");
        waitForImageWithTitleThenClick("Search by type");
        waitForThenDoBackofficeSearch("");
        waitForThenClick("span","The Quiet");

        waitForThenUpdateInputField("The Quiet", "The Choir");
        assertTrue( waitForThenClick("button","Save") );
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintViaItemsXml test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintViaItemsXml test
```

Procedure

1. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
   ```

   ```
   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

2. Make band names mandatory and unique by adding the following modifier tags to the `name` attribute of the Band item type definition in the `concerttours-items.xml` file of the `resources` folder of the `concerttours` extension.

```
<attribute qualifier="name" type="java.lang.String">
        <description>name of band</description>
        <persistence type="property" />
        <modifiers optional="false" unique="true" />
</attribute>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the `concerttours-items.xml` file or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-items.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithValidation.xml`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithValidation.xml $HYBRIS_HOME_DIR/hybris/bin
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-itemsWithValidation.xml %HYBRIS_HOME_DIR%\hyb
```

3. Rebuild and initialize SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant build initialize
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant build initialize
```

4. Test the validation constraints in Backoffice.
   a. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

   b. Log in to `https://localhost:9002/backoffice` using the login **admin** and the password that you defined as an environment variable.
   c. Select the System > Types nodes in the explorer tree on the left.
   d. Search for the **Band** item Type, and, in the details panel toolbar, click the Search by Type icon.
   e. Select one of the **Band** items and try to remove the value of its name attribute by deleting the contents of the name field and pressing the Save button.

      A technical error message will appear and the save operation will be rejected.

   f. Try to change the name attribute of one band to the same name as another band.

      A technical error message will appear and the save operation will be rejected.

5. Run the `testValidationConstraintViaItemsXml` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintViaItemsXml test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintViaItemsXml test
```

6. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Define Validation Constraints in the Items.xml File"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Define Validation Constraints in the Items.xml File"
```

Validation Constraints in Backoffice

You can create and define validation constraints in the SAP Commerce Backoffice Administration Cockpit.

Parent topic: Explore SAP Commerce Tour

Previous: Validation

Next: Validation Constraints in ImpEx

Define a Validation Constraint Using Backoffice

Define a validation constraint in Backoffice.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testValidationConstraintViaBackoffice() {
loginToBackOffice();
selectConstraintsPage();
tryToViolateTheNewConstraint();
assertTrue( waitFor("span","Album sales must not be negative"));
}
```

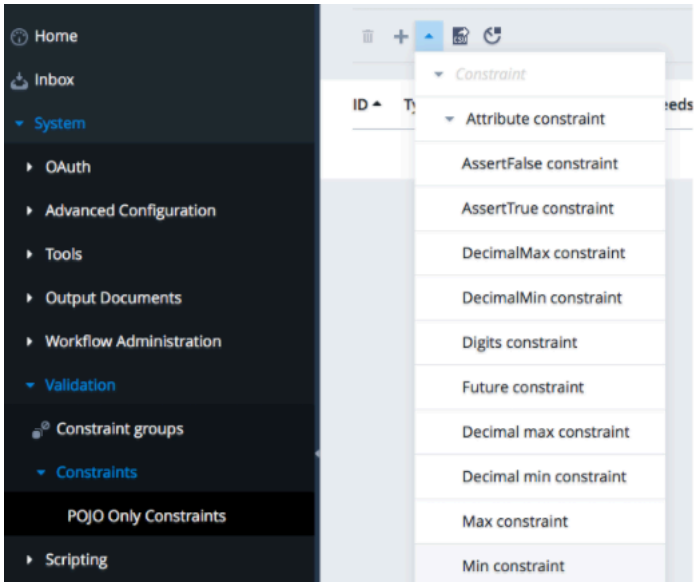
Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintViaBackoffice test
```

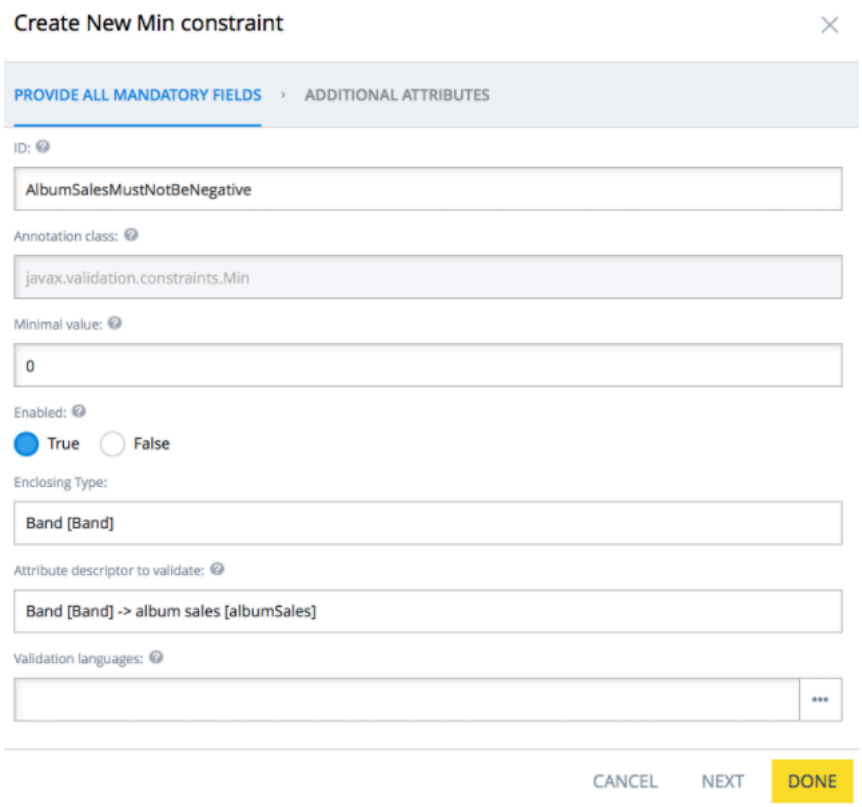
```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintViaBackoffice test
```

Procedure

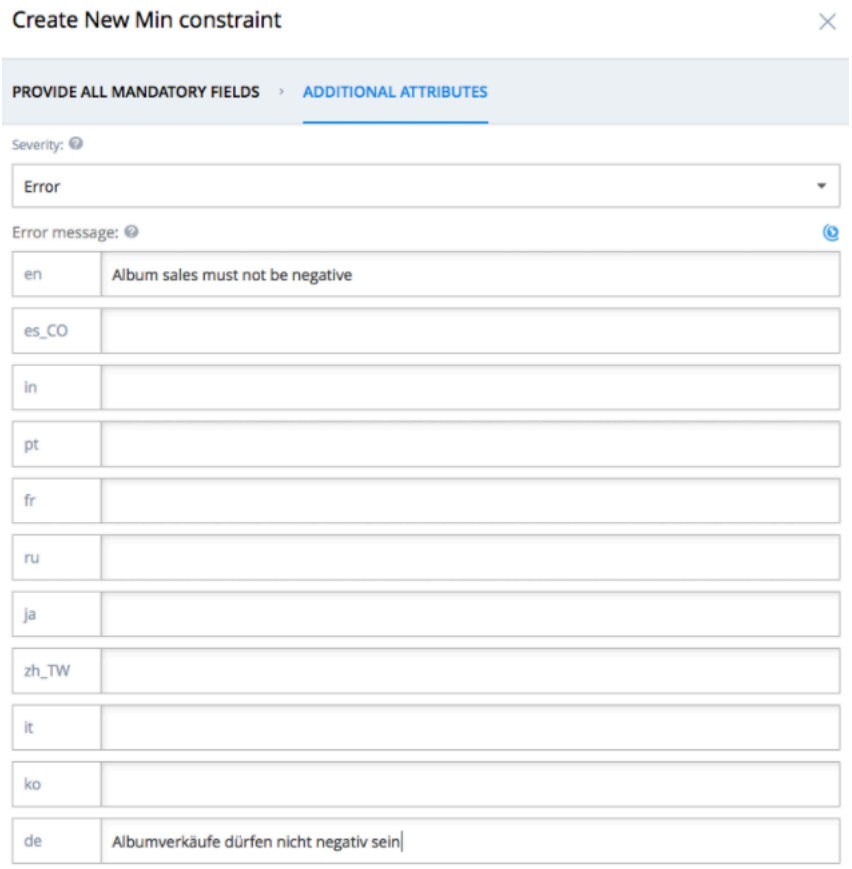
1. In the Backoffice Administration Cockpit, open the Administration perspective of the Explorer pane.
2. Navigate to System > Validation > Constraints.
3. Create a new minimum value constraint.
   a. Press the menu arrow on the right of the + toolbar button to drop down a hierarchical list of constraint types.
   b. Expand the Attribute constraint menu item by clicking on the arrow to the left of the menu item text.
   c. Click on the Min Constraint item to display the Create New Min Constraint wizard.



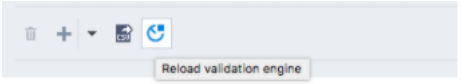
   d. Give the new constraint the ID `newConstraint`.
   e. Set the Minimal value field to 0 to forbid negative values.
   f. Since you want to validate the enclosing type, start typing Band in the Composed type to validate field and choose Band from the list of suggestions when it appears.
   g. Do the same in the Attribute descriptor to validate field to pick the albumSales attribute.
   h. Click Next.



   i. Enter the error message *Album sales must be > 0. Do pay attention.* in the Error message field.
   j. **Optional:** Click on the globe symbol to access other language options so that you can enter the error message in different languages.
   k. Click on Done.



l. Click on the right most toolbar button to reload the validation engine so that it picks up the new constraint.

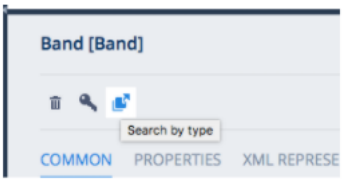


SAP Commerce 123 Interactive Shortcut: If you are unable to create the new constraint in Backoffice yourself or you want to skip this step, run the following test. This test creates the constraint.

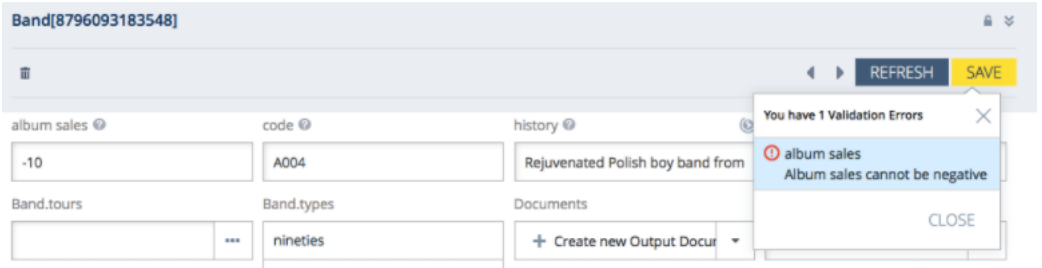
```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCreateValidationConstraintViaBackoffice test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCreateValidationConstraintViaBackoffice test
```

4. Check that the new constraint works.
    a. Navigate to System > Types in the Explorer pane.
    b. Search for the Band item type.
    c. Click the Search by type toolbar button to list the bands in the database.



    d. Pick one of the bands.
    e. Try to set the album sales attribute to a negative number in the details pane and click Save.
    f. Confirm that the correct error message is displayed in the list of validation errors.



5. Run the `testValidationConstraintViaBackoffice` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintViaBackoffice test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintViaBackoffice test
```

6. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Define a Validation Constraint Using Backoffice"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Define a Validation Constraint Using Backoffice"
```

Next Steps

In the next section, you will explore another way of creating this constraint. So before proceeding, delete the constraint you created.

Validation Constraints in ImpEx

You can define validation constraints in ImpEx files, making it easy to reload constraints after initializing the database.

Validation constraints are persisted to the database. If you frequently reinitialize the database during the development and testing phase of your project, there is a danger that you could erase any stored constraints. To ease the effort of restoring validation constraints, it is a good idea to define and load them using ImpEx files.

Parent topic: Explore SAP Commerce Tour

Previous: Validation Constraints in Backoffice

Next: Custom Validation Constraints

Define a Validation Constraint Using ImpEx

Define and load validation constraints using ImpEx files.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testValidationConstraintAfterImpex() {
        loginToBackOffice();
        modifyABandToHaveNegativeAlbumSales();
        if (VersionHelper.getVersion().equals(Version.V2205) || VersionHelper.getVersion().equals(Version.V2211))
                waitFor("div","You have 1 Validation Messages");
        else
                waitFor("div","You have 1 Validation Errors");
        assertTrue( waitFor("span","Album sales must not be negative"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintAfterImpex test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintAfterImpex test
```

Procedure

1. Create a new ImpEx file, called `essentialdata-constraints.impex`, in the `resources/impex` folder of the `concerttours` extension.

   ```
   INSERT_UPDATE MinConstraint;id[unique=true];severity(code,itemtype(code));active;annotation;descriptor(enclosingType(code),qualifier);message[l
   ;AlbumSalesMustNotBeNegative;ERROR:Severity;true;javax.validation.constraints.Min;Band:albumSales;Albumverkäufe dürfen nicht negativ sein;Album
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the ImpEx file or you want to skip this step, copy `/hybris123/src/main/webapp/resources/impex/validation/essentialdata-constraints.impex` to `/hybris/bin/custom/concerttours/resources/impex/`

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/impex/validation/essentialdata-constraints.impex $HYBRIS_HOME_DIR/hybris/bin/custom/con

   copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\impex\validation\essentialdata-constraints.impex %HYBRIS_HOME_DIR%\hybris\bin\cus
   ```

2. Stop SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop

   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
   ```

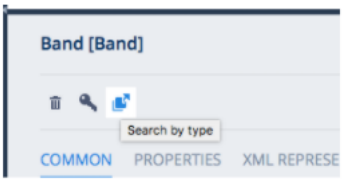
3. Initialize the database.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant initialize

   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant initialize
   ```

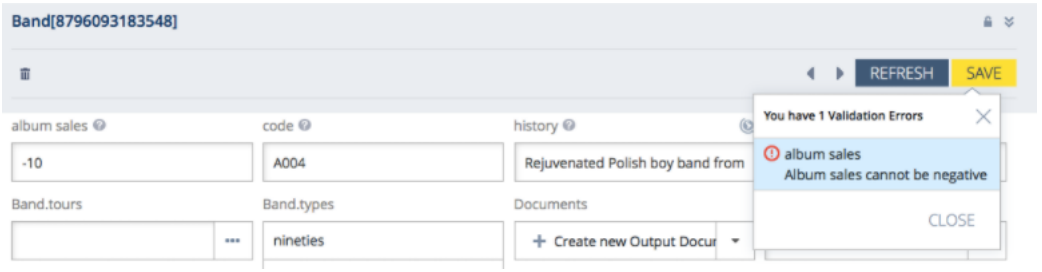
4. Start SAP Commerce.

   ```
   cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start

   cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
   ```

5. Confirm that you are unable to save a band with negative album sales.
    a. Navigate to System > Types in the Explorer pane.
    b. Search for the Band item type.
    c. Click the Search by type toolbar button to list the bands in the database.



    d. Pick one of the bands.
    e. Try to set the album sales attribute to a negative number in the details pane and click Save.
    f. Confirm that the correct error message is displayed in the list of validation errors.



6. Run the `testValidationConstraintAfterImpex` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintAfterImpex test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationConstraintAfterImpex test
```

7. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Define a Validation Constraint Using ImpEx"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Define a Validation Constraint Using ImpEx"
```

Custom Validation Constraints

Although the validation framework provides all constraints from the JSR 303 specification, sometimes you need other constraint types that are specific to your project.

A band's history is often set to a piece of placeholder text instead of the meaningful, genuine background of the band. Create a constraint that prevents this occurrence by looking for text that is missing, or starts with "lorem ipsum".

The new constraint type extends `AttributeConstraint` because you are validating the values of attributes, not of the type as a whole. If you want to create constraints that apply to the whole of an item, extend `TypeConstraint` instead. The `only` attribute holds a reference to the Java annotation class that defines our constraint as a Java annotation.

The `@Constraint` annotation specifies a `NotLoremIpsumValidator` class that provides the actual logic for checking the constraint. The constraint checks that if the value is not null or empty, and that it does not start with the phrase lorem ipsum. Of course, you can get more sophisticated but this good enough for the purposes of the exercise.

Note that you can see the other mapping properties for the built-in constraint types in the file `ext/validation/project.properties` in the platform project.

Parent topic: Explore SAP Commerce Tour

Previous: Validation Constraints in ImpEx

Next: Integration Test for the Custom Constraint

Define a Custom Validation Constraint

Although the validation framework provides all constraints from the JSR 303 specification, sometimes you need other constraint types.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testValidationCustomConstraint() {
loginToBackOffice();
selectConstraintsPage();
deleteExistingMinConstraint("NotIpsum");
addNewCustomConstraint("NotIpsum");
reloadConstraints();
tryToViolateTheNewCustomConstraint();
assertTrue( waitFor("span","No Lorem Ipsum"));
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationCustomConstraint test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationCustomConstraint test
```

Procedure

1. Create the Constraint Item Type.

   Add the following item type definition to the `concerttours-items.xml` file in the `resources` folder of the `concerttours` extension.

   ```
   <itemtype code="NotLoremIpsumConstraint" extends="AttributeConstraint"
       autocreate="true" generate="true">
       <description>Custom constraint which checks for Lorem Ipsum text.</description>
       <attributes>
           <attribute qualifier="annotation" type="java.lang.Class"
               redeclare="true">
               <modifiers write="false" initial="true" optional="false" />
               <defaultvalue>
                   concerttours.constraints.NotLoremIpsum.class
               </defaultvalue>
           </attribute>
       </attributes>
   </itemtype>
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to edit the `concerttours-items.xml` file or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-items.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithCustomValidation.xml`

   ```
   cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithCustomValidation.xml $HYBRIS_HOME_DIR/hybr
   ```

   ```
   type %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-itemsWithCustomValidation.xml > %HYBRIS_HOME_DIR
   ```

2. Create the annotation interface called `NotLoremIpsum` in a package called `concerttours.constraints` under the `src` folder of the `concerttours` extension.

   ```
   package concerttours.constraints;
   import static java.lang.annotation.ElementType.FIELD;
   import static java.lang.annotation.RetentionPolicy.RUNTIME;
   import java.lang.annotation.Documented;
   import java.lang.annotation.Retention;
   import java.lang.annotation.Target;
   import javax.validation.Constraint;
   import javax.validation.Payload;

   @Target(
   { FIELD })
   @Retention(RUNTIME)
   @Constraint(validatedBy = NotLoremIpsumValidator.class)
   @Documented
   public @interface NotLoremIpsum
   {
       String message() default "{concerttours.constraints.NotLoremIpsum.message}";
       Class<?>[] groups() default {};
       Class<? extends Payload>[] payload() default {};
   }
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the interface or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/constraints/NotLoremIpsum.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/constraints/NotLoremIpsum.java`

   ```
   ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/constraints/NotLoremIpsum.java $HYBRIS_HOME_DIR/hybris
   ```

   ```
   echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\constraints\NotLoremIpsum.java %HYBRIS_HOME_
   ```

3. Create the validator class by creating a new class called `NotLoremIpsumValidator` in the same `concerttours.constraints` package under the `src` folder of the `concerttours` extension, and copying in the following content.

   ```
   package concerttours.constraints;
   import javax.validation.ConstraintValidator;
   import javax.validation.ConstraintValidatorContext;

   public class NotLoremIpsumValidator implements ConstraintValidator<NotLoremIpsum, String>
   {
       @Override
       public void initialize(final NotLoremIpsum constraintAnnotation)
       {
           // empty
       }
       @Override
       public boolean isValid(final String value, final ConstraintValidatorContext context)
       {
           return value != null && !value.isEmpty() && !value.toLowerCase().startsWith("lorem ipsum");
       }
   }
   ```

   SAP Commerce 123 Interactive Shortcut: If you are unable to create the custom validator or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/src/concerttours/constraints/NotLoremIpsumValidator.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/src/concerttours/constraints/NotLoremIpsumValidator.java`

   ```
   ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/constraints/NotLoremIpsumValidator.java $HYBRIS_HOME_D
   ```

   ```
   echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\constraints\NotLoremIpsumValidator.java %HYB
   ```

4. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

5. Rebuild and initialize the database because you have changed an `items.xml` file.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant build initialize
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant build initialize
```

6. Add the following property to the `local.properties` file in the *<HYBRIS_CONFIG_DIR>* directory to tell the platform that your constraint is for checking character strings (and not for numbers and dates).

```
validation.constraints.attribute.mapping.concerttours.constraints.NotLoremIpsum=strings
```

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the `local.properties` file or you want to skip this step, run the following command.

```
cat $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/config/validation.properties >> $HYBRIS_HOME_DIR/hybris/config/
```

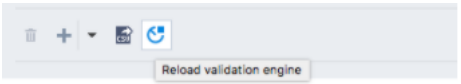
```
type %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\config\validation.properties >> %HYBRIS_HOME_DIR%\hybris\conf
```

7. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

8. Check the new constraint type.
   a. Open Backoffice at `https://localhost:9002/backoffice` and log in with user `admin` and the password that you defined as an environment variable.
   b. Create a new `NotLoremIpsum` constraint for the Band history attribute in the same way you created the new `MinConstraint` previously and give it the ID `NotIpsum`.
   c. Click on the right most toolbar button to reload the validation engine so that it picks up the new constraint.



   d. Try modifying the history attributes of one of the bands so that it starts with *lorem ipsum*, and confirm that the correct error message appears when you click Save.

9. Run the `testValidationCustomConstraint` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationCustomConstraint test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testValidationCustomConstraint test
```

10. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Define a Custom Validation Constraint"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Define a Custom Validation Constraint"
```

Integration Test for the Custom Constraint

Become familiar with how to use the SAP Commerce validation service in code.

Write a test that checks whether the constraint is working as expected. Use ImpEx to load the constraint into the database, and then the validation service to load it from the database into the validation engine.

Parent topic: Explore SAP Commerce Tour

Previous: Custom Validation Constraints

Next: Media Files

Write an Integration Test for the Custom Constraint

Check that the constraint is working as expected and introduces the Hybris Validation service in the code.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testCustomConstraintIntegrationTest() {
assertTrue( checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"NotLoremIpsumConstraintTest\" package=\"concerttours.cons
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCustomConstraintIntegrationTest test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCustomConstraintIntegrationTest test
```

Procedure

1. Load the Constraint via Impex by adding the new Lorem Ipsum constraint to the `essentialdata-constraints.impex` file.

```
INSERT_UPDATE MinConstraint;id[unique=true];severity(code,itemtype(code));active;annotation;descriptor(enclosingType(code),qualifier);message[l
;AlbumSalesMustNotBeNegative;ERROR:Severity;true;javax.validation.constraints.Min;Band:albumSales;Albumverkäufe dürfen nicht negativ sein;Album

INSERT_UPDATE NotLoremIpsumConstraint;id[unique=true];severity(code,itemtype(code));active;annotation;descriptor(enclosingType(code),qualifier)
;NotIpsum;ERROR:Severity;true;concerttours.constraints.NotLoremIpsum;Band:history;Band Geschichte sollte nicht Lorem ipsum Platzhalter Text.;Ba
```

SAP Commerce 123 Interactive Shortcut: If you are unable to add the constraint or you want to skip this step, replace *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/resources/impex/essentialdata-constraints.impex with the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/impex/validation/essentialdata-constraints-2.impex.

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/impex/validation/essentialdata-constraints-2.impex $HYBRIS_HOME_DIR/hybris/bin/custom/c
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\impex\validation\essentialdata-constraints-2.impex %HYBRIS_HOME_DIR%\hybri
```

2. Create the integration test called `NotLoremIpsumConstraintTest` in the `concerttours.constraints` package under the `testsrc` folder of the `concerttours` extension.

```
package concerttours.constraints;
import de.hybris.bootstrap.annotations.IntegrationTest;
import de.hybris.platform.servicelayer.ServicelayerTransactionalTest;
import de.hybris.platform.servicelayer.model.ModelService;
import de.hybris.platform.validation.exceptions.HybrisConstraintViolation;
import de.hybris.platform.validation.services.ValidationService;
import java.util.Set;
import javax.annotation.Resource;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;
import concerttours.model.BandModel;
import concerttours.model.NotLoremIpsumConstraintModel;

@IntegrationTest
public class NotLoremIpsumConstraintTest extends ServicelayerTransactionalTest
{
   @Resource
   private ModelService modelService;
   @Resource
   private ValidationService validationService;
   @Before
   public void setup() throws Exception
   {
       createCoreData();
       importCsv("/impex/essentialdata-constraints.impex", "UTF-8");
       validationService.reloadValidationEngine();
   }
   @Test
   public void testLoremIpsumConstraint()
   {
       final BandModel band = modelService.create(BandModel.class);
       band.setCode("LoremIpsumTest1");
       band.setName("LoremIpsumBand");
       band.setHistory("Lorem Ipsum is here");
       final Set<HybrisConstraintViolation> violations = validationService.validate(band);
       Assert.assertTrue("The violation set sould not be null or empty", violations != null && violations.size() > 0);
       Assert.assertEquals("There should be one constraint violations", 1, violations.size());
       for (final HybrisConstraintViolation hybrisConstraintViolation : violations)
       {
           Assert.assertEquals(NotLoremIpsumConstraintModel.class, hybrisConstraintViolation.getConstraintModel().getClass());
       }
   }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to add the constraint or you want to skip this step, replace *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/testsrc/concerttours/constraints/NotLoremIpsumConstraintTest.java with the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/constraints/NotLoremIpsumConstraintTest.java.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/constraints/NotLoremIpsumConstraintTest.java $HYBR
```

```
echo f | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\constraints\NotLoremIpsumConstraintTest.
```

3. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

4. Initialize the system.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all yunitinit
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all yunitinit
```

5. Run the integration tests.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant integrationtests -Dtestclasses.packages="concerttours.*"
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant integrationtests -Dtestclasses.packages=concerttours.*
```

6. View the test results at *<HYBRIS_HOME_DIR>*/hybris/log/junit/index.html
7. Run the `testCustomConstraintIntegrationTest` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCustomConstraintIntegrationTest test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testCustomConstraintIntegrationTest test
```

8. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Write an Integration Test for the Custom Constraint"

cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Write an Integration Test for the Custom Constraint"
```

Media Files

SAP Commerce supports media files. A media file can be anything that can be saved on a file system.

Media files include a Flash animation file, a JPEG image, an MPEG video file, a CSV file, a text file, an XML file, and more. The SAP Commerce platform does not store media files in its database. Instead, it defines a media item type that names and describes a particular media file, and knows where the file is physically stored. In other words, media items in SAP Commerce are not the media files themselves, but references (URLs) to those files. The actual referenced media files can be stored in various locations. You can keep them locally, or store them remotely using Amazon S3, Windows Azure Blob, or MongoDB GridFS solutions, for example.

In addition to a unique ID and a description, a media item has the following attributes:

- It is assigned to a catalog version. Media items for products can be synchronized between catalog versions at the same time as the related product details.
- It can have access to it controlled through lists of permitted and denied principals (users).
- It is associated with a media format that is used to tag different media items as being appropriate for the same uses.
- It is included in a media container, typically used to group media files of the same content that are in different formats: for example, the same image but in different sizes or encodings (png, jpg, bmp).

To make the band pages a little more interesting, you can include pictures of each band. Use a smaller version of the image for the list page, and a larger one for the details page.

Parent topic: Explore SAP Commerce Tour

Previous: Integration Test for the Custom Constraint

Next: Properties Files

Related Information

Digital Asset Management

Add Media Files

Add images for the yRock band. You can of course add images for others, too.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testBandImages() throws Exception {
canLoginToHybrisCommerce();
navigateTo("https://localhost:9002/concerttours/bands");
waitForValidImage();
navigateTo("https://localhost:9002/concerttours/bands/A006");
assertTrue( waitForValidImage());
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBandImages test

cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBandImages test
```

Procedure

1. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop

cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

2. Modify the `Band` business logic to include images.
   a. Add a media container attribute to the `Band` item type.

   Add the following attribute definition to the Band item type in the `concerttours-items.xml` file in the resources folder of the concerttours extension:

```
<attribute qualifier="image" type="MediaContainer">
    <description>picture of band in different formats</description>
```

```
    <persistence type="property" />
</attribute>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the `concerttours-items.xml` file, or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-items.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithMedia.xml` .

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-itemsWithMedia.xml $HYBRIS_HOME_DIR/hybris
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-itemsWithMedia.xml %HYBRIS_HOME_
```

b. Add a URL attribute to the `BandData` bean.

Add the following attribute definition to the `BandData` definition in the `concerttours-beans.xml` file in the `resources` folder of the `concerttours` extension:

```
<bean class = "concerttours.data.BandData" >
       <description>Data object representing a Band</description>
       <property name = "id" type = "String" />
       <property name = "name" type = "String" />
       <property name = "description" type = "String" />
       <property name = "albumsSold" type = "Long" />
       <property name = "genres" type="java.util.List&lt;String&gt;"/>
       <property name = "tours" type="java.util.List&lt;concerttours.data.TourSummaryData&gt;"/>
       <property name="imageURL" type="String" />
</bean>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the `concerttours-beans.xml` file, or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours-beans.xml` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-beansWithMedia.xml` .

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-beansWithMedia.xml $HYBRIS_HOME_DIR/hybris
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-beansWithMedia.xml %HYBRIS_HOME_
```

c. Update the facades to retrieve the image in the relevant format.

Modify the contents of the `DefaultBandFacade` class to add the handling of the different formats of images:

```
package concerttours.facades.impl;
import de.hybris.platform.core.model.media.MediaContainerModel;
import de.hybris.platform.core.model.media.MediaFormatModel;
import de.hybris.platform.core.model.product.ProductModel;
import de.hybris.platform.servicelayer.media.MediaService;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Required;
import concerttours.data.BandData;
import concerttours.data.TourSummaryData;
import concerttours.enums.MusicType;
import concerttours.facades.BandFacade;
import concerttours.model.BandModel;
import concerttours.service.BandService;
import java.util.Locale;

public class DefaultBandFacade implements BandFacade
{
    private BandService bandService;
    private MediaService mediaService;
    @Override
    public List<BandData> getBands()
    {
        final List<BandModel> bandModels = bandService.getBands();
        final List<BandData> bandFacadeData = new ArrayList<>();
        final MediaFormatModel format = mediaService.getFormat("bandList");
        for (final BandModel sm : bandModels)
        {
            final BandData sfd = new BandData();
            sfd.setId(sm.getCode());
            sfd.setName(sm.getName());
            sfd.setDescription(sm.getHistory(Locale.ENGLISH));
            sfd.setAlbumsSold(sm.getAlbumSales());
            sfd.setImageURL(getImageURL(sm, format));
            bandFacadeData.add(sfd);
        }
        return bandFacadeData;
    }
    @Override
    public BandData getBand(final String name)
    {
        if (name == null)
        {
            throw new IllegalArgumentException("Band name cannot be null");
        }
        final BandModel band = bandService.getBandForCode(name);
        if (band == null)
        {
            return null;
        }
        // Create a list of genres
        final List<String> genres = new ArrayList<>();
        if (band.getTypes() != null)
        {
            for (final MusicType musicType : band.getTypes())
            {
                genres.add(musicType.getCode());
            }
        }
        // Create a list of TourSummaryData from the matches
        final List<TourSummaryData> tourHistory = new ArrayList<>();
        if (band.getTours() != null)
        {
            for (final ProductModel tour : band.getTours())
            {
                final TourSummaryData summary = new TourSummaryData();
```

```
                summary.setId(tour.getCode());
                summary.setTourName(tour.getName(Locale.ENGLISH));
                // making the big assumption that all variants are concerts and ignore product catalogs
                summary.setNumberOfConcerts(Integer.toString(tour.getVariants().size()));
                tourHistory.add(summary);
            }
        }
        // Now we can create the BandData transfer object
        final MediaFormatModel format = mediaService.getFormat("bandDetail");
        final BandData bandData = new BandData();
        bandData.setId(band.getCode());
        bandData.setName(band.getName());
        bandData.setAlbumsSold(band.getAlbumSales());
        bandData.setImageURL(getImageURL(band, format));
        bandData.setDescription(band.getHistory(Locale.ENGLISH));
        bandData.setGenres(genres);
        bandData.setTours(tourHistory);
        return bandData;
    }
    protected String getImageURL(final BandModel sm, final MediaFormatModel format)
    {
        final MediaContainerModel container = sm.getImage();
        if (container != null)
        {
            return mediaService.getMediaByFormat(container, format).getDownloadURL();
        }
        return null;
    }
    @Required
    public void setBandService(final BandService bandService)
    {
        this.bandService = bandService;
    }
    @Required
    public void setMediaService(final MediaService mediaService)
    {
        this.mediaService = mediaService;
    }
}
```

i Note

With this modification you do the following:

- You use the `mediaService` to get specific media format models.
- In the `getBands()` method, you retrieve the `bandList` format of the image, the smaller version of the image.
- In the `getBand()` method, we retrieve the `bandDetail` format of the image, the larger version of the image.
- You return the URL of the image that needs to be displayed.
- You have hardcoded a dependency on the existence of two specific media formats. If they are not present, an exception is thrown. The two media formats have become essential data items.

An optional enhancement that you could make is to return a default image when a Band has not had an image provided yet. Another enhancement would also return any alt-text specified for the image within the data transfer object (DTO).

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the `DefaultBandFacade`, or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/src/concerttours/facades/impl/DefaultBandFacade.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultBandFacadeWithMedia.java`.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultBandFacadeWithMedia.java $HY
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\facades\impl\DefaultBandFacadeWithMedi
```

d. Update `concerttours-spring.xml` to inject the `DefaultBandFacade` with the `MediaService`.

```
<alias name = "defaultBandFacade" alias = "bandFacade" />
<bean id = "defaultBandFacade" class ="concerttours.facades.impl.DefaultBandFacade" >
    <property name = "bandService" ref = "bandService" />
    <property name="mediaService" ref="mediaService"/>
</bean>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the `concerttours-beans.xml` file, or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/resources/concerttours-spring.xml` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withMedia.xml` .

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withMedia.xml $HYBRIS_HOME_DIR/hybr
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring-withMedia.xml %HYBRIS_HOM
```

3. Initialize and update your system.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all build initialize updatesystem
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all build initialize updatesystem
```

4. Modify the Band JSPs to include images by updating the`BandList.jsp` and `BandDetails.jsp` pages as follows.

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!doctype html>
<html>
<title>Band Details</title>
<body>
    <h1>Band Details</h1>
    Band Details for ${band.name}
    <p><img src="${band.imageURL}"/></p>
    <p>${band.description}</p>
    <p>Music type:</p>
    <ul>
```

```
            <c:forEach var="genre" items="${band.genres}">
                <li>${genre}</li>
            </c:forEach>
        </ul>
        <p>Tour History:</p>
        <ul>
            <c:forEach var="tour" items="${band.tours}">
                <li><a href="../tours/${tour.id}">${tour.tourName}</a>(number of concerts: ${tour.numberOfConcerts})</li>
            </c:forEach>
        </ul>
        <a href="../bands">Back to Band List</a>
    </body>
    </html>
```

```
    <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <!doctype html>
    <html>
        <title>Band List</title>
        <body>
            <h1>Band List</h1>
         <ul>
         <c:forEach var="band" items="${bands}">
            <li><a href="./bands/${band.id}"><img src="${band.imageURL}" />${band.name}</a></li>
          </c:forEach>
          </ul>
        </body>
    </html>
```

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the JSP files or you want to skip this step, replace button to create two new media formats. The first with qualifier *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/web/webroot/WEB-INF/views/BandList.jsp` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/web/webroot/WEB-INF/views/BandListWithMedia.jsp`, and *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/web/webroot/WEB-INF/views/BandDetails.jsp` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/web/webroot/WEB-INF/views/BandDetailsWithMedia.jsp`.

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/web/webroot/WEB-INF/views/BandListWithMedia.jsp $HYBRIS_HOME_DIR/hybris
```

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/web/webroot/WEB-INF/views/BandDetailsWithMedia.jsp $HYBRIS_HOME_DIR/hyb
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\web\webroot\WEB-INF\views\BandListWithMedia.jsp %HYBRIS_HOME_
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\web\webroot\WEB-INF\views\BandDetailsWithMedia.jsp %HYBRIS_HO
```

5. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

6. Add two new media formats.
    a. In the Backoffice Administration Cockpit, navigate to the **Multimedia** > **Media Formats** item in the Explorer Tree.
    b. Click on the **plus** button to create two new media formats. The first with qualifier `bandDetail` and name `Band Detail Format`, and the second with qualifier `bandList` and name `Band List Format`.
7. Find appropriate images for each of the six bands.
8. Use an image editing tool to create two JPEG versions of each image: one 600 pixels wide for the details page and one 100 pixels wide for the list page.

   Give the images the following names:

    bigbandBig.jpg
    bigbandSmall.jpg
    danceBig.jpg
    danceSmall.jpg
    jazzBig.jpg
    jazzSmall.jpg
    omphaBig.jpg
    omphaSmall.jpg
    orchestraBig.jpg
    orchestraSmall.jpg
    rockBig.jpg
    rockSmall.jpg

9. Add two new media items for the yRock band.
    a. In the Backoffice Administration Cockpit, navigate to the **Multimedia** > **Media** item in the Explorer Tree.
    b. Create one new media item with the identifier `yRockSmall` and catalog version `concertoursProductCatalog: Online`.
    c. Add a second new media item with the identifier `yRockLarge` and catalog version `concertoursProductCatalog: Online`.
10. Create a media container for the yRock band images.
    a. In the Backoffice Administration Cockpit, navigate to the **Multimedia** > **Media Containers** item in the Explorer Tree.
    b. Click on the **plus** button to create a new media container with the name and qualifier set to `yRockImageContainer`.
11. Add an image to each item.
    a. In the Backoffice Administration Cockpit, navigate to the **Multimedia** > **Media** item in the Explorer Tree.
    b. Click **search** to list all media items, including `rockSmall.jpg` and `rockLarge.jpg`.
    c. Click on `yRockSmall`, upload the image `rockSmall.jpg`, specify `bandList` as the Media format, and `yRockImageContainer` as the Media container.

d. Click on `yRockLarge`, upload the image `rockBig.jpg` and specify `bandDetail` as the Media format, and `yRockImageContainer` as the Media container.

12. Assign the media container to its respective band.

    a. In the Backoffice Administration Cockpit, navigate to System > Types in the Explorer pane.

    b. Search for the yRock band.

    c. Set its Band.image field to `yRockImageContainer`.

13. Verify that you can see the images on the band pages.

    You should see the images on the following pages:

    - `https://localhost:9002/concerttours/bands`
    - `https://localhost:9002/concerttours/bands/A001`

14. To define multiple media items, formats, and containers, use ImpEx and define them as essential data so that the media format items are loaded into the database before you load the project data

    a. Create a new file called `essentialdata-mediaformats.impex` in the `resources/impex` folder of the `conccerttours` extension, and setting the contents as follows.

```
INSERT_UPDATE MediaFormat;qualifier[unique=true];name
;bandList;Band List Format
;bandDetail;Band Detail Format
```

    SAP Commerce 123 Interactive Shortcut: If you are unable to create the ImpEx file or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/impex/media/essentialdata-mediaformats.impex` with the contents of *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/impex/`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/impex/media/essentialdata-mediaformats.impex $HYBRIS_HOME_DIR/hybris/bin/custom/c
```

```
copy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\impex\media\essentialdata-mediaformats.impex %HYBRIS_HOME_DIR%\hybris\bin\cust
```

    b. Add media files to the project by copying the band images into a new folder called `bandimages` under the existing `resources/concerttours` folder in the `concerttours` extension.

    SAP Commerce 123 Interactive Shortcut: If you are unable to add the media files or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/bandimages/*` with the contents of *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/concerttours/bandimages/`

```
mkdir -p $HYBRIS_HOME_DIR/hybris/bin/custom/concerttours/resources/concerttours/bandimages; cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\bandimages\* %HYBRIS_HOME_DIR%\hybris\bin\custom\concer
```

    c. Add media data to the bands ImpEx file by updating the contents of the `concerttours-bands.impex` file in the `resources/impex` folder of the `conccerttours` extension as follows.

```
# ImpEx for Importing Bands into Little Concert Tours Store

# Macros / Replacement Parameter definitions
$productCatalog=concertoursProductCatalog
$catalogVersion=catalogversion(catalog(id[default=$productCatalog]),version[default='Online'])[unique=true,default=$productCatalog:Online
$medias=medias(code, $catalogVersion)
$siteResource=jar:concerttours.constants.ConcerttoursConstants&/concerttours/bandimages

# Product catalog
INSERT_UPDATE Catalog;id[unique=true]
;$productCatalog

# Product catalog version
INSERT_UPDATE CatalogVersion;catalog(id)[unique=true];version[unique=true];active;languages(isoCode);readPrincipals(uid)
;$productCatalog;Online;true;en;employeegroup

INSERT_UPDATE Media;mediaFormat(qualifier);code[unique=true];@media[translator=de.hybris.platform.impex.jalo.media.MediaDataTranslator];m
;bandList;rockSmall.jpg;$siteResource/rockSmall.jpg;;
;bandDetail;rockBig.jpg;$siteResource/rockBig.jpg;;
;bandList;danceSmall.jpg;$siteResource/danceSmall.jpg;;
;bandDetail;danceBig.jpg;$siteResource/danceBig.jpg;;
;bandList;jazzSmall.jpg;$siteResource/jazzSmall.jpg;;
;bandDetail;jazzBig.jpg;$siteResource/jazzBig.jpg;;
;bandList;bigbandSmall.jpg;$siteResource/bigbandSmall.jpg;;
;bandDetail;bigbandBig.jpg;$siteResource/bigbandBig.jpg;;
;bandList;omphaSmall.jpg;$siteResource/omphaSmall.jpg;;
;bandDetail;omphaBig.jpg;$siteResource/omphaBig.jpg;;
;bandList;orchestraSmall.jpg;$siteResource/orchestraSmall.jpg;;
;bandDetail;orchestraBig.jpg;$siteResource/orchestraBig.jpg;;
INSERT_UPDATE MediaContainer;qualifier[unique=true];$medias;$catalogVersion
;yRockImage;rockSmall.jpg,rockBig.jpg;
;yBandImage;danceSmall.jpg,danceBig.jpg;
;yJazzImage;jazzSmall.jpg,jazzBig.jpg;
;BannedImage;bigbandSmall.jpg,bigbandBig.jpg;
;SirkenImage;omphaSmall.jpg,omphaBig.jpg;
;TheChoirImage;orchestraSmall.jpg,orchestraBig.jpg;

INSERT_UPDATE Band;code[unique=true];name;albumSales;image(qualifier)
;A001;yRock;1000000;yRockImage
;A006;yBand;;yBandImage
;A003;yJazz;7;yJazzImage
;A004;Banned;427;BannedImage
;A002;Sirken;2000;SirkenImage
;A005;The Choir;49000;TheChoirImage
;A007;The Quiet;1200;;
```

    ℹ Note

    One band is left without associated images to check that the system processes this situation robustly.

    SAP Commerce 123 Interactive Shortcut: If you are unable to create the ImpEx file or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/impex/media/concerttours-bands.impex` with the contents of *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/resources/impex/`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/impex/media/concerttours-bands.impex $HYBRIS_HOME_DIR/hybris/bin/custom/concertto
```

```
copy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\impex\media\concerttours-bands.impex %HYBRIS_HOME_DIR%\hybris\bin\custom\conce
```

d. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

e. Initialize the system.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant initialize
```

```
cd %HYBRIS_HOME_DIR%/hybris/bin/platform & ant initialize
```

15. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

16. Verify that you can see the images on the band pages.

You should see the images on the following pages:

- `https://localhost:9002/concerttours/bands`
- `https://localhost:9002/concerttours/bands/A001`

17. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

18. Fix the test cases.

a. Add a call to `importCsv` to the `setUp()` method of the `DefaultBandFacadeIntegrationTest` class.

```
@Before
public void setUp() throws Exception
{
    try {
        Thread.sleep(TimeUnit.SECONDS.toMillis(1));
        new JdbcTemplate(Registry.getCurrentTenant().getDataSource()).execute("CHECKPOINT");
        Thread.sleep(TimeUnit.SECONDS.toMillis(1));
    }
    catch (InterruptedException exc) {}
    importCsv("/impex/essentialdata-mediaformats.impex", "UTF-8");

    // This instance of a BandModel will be used by the tests
    bandModel = modelService.create(BandModel.class);
    bandModel.setCode(BAND_CODE);
    bandModel.setName(BAND_NAME);

    bandModel.setHistory(BAND_HISTORY);
    bandModel.setAlbumSales(ALBUMS_SOLD);
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the integration test or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeIntegrationTest.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeIntegration`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeIntegrationTestWi
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\facades\impl\DefaultBandFacadeIntegration
```

b. Add a call to `importCsv` to the `testBandServiceTours()` method of the `DefaultBandServiceIntegrationTest` class.

```
@Test
public void testBandServiceTours() throws Exception
{
    createCoreData();
    importCsv("/impex/essentialdata-mediaformats.impex", "UTF-8");
    importCsv("/impex/concerttours-bands.impex", "utf-8");
    importCsv("/impex/concerttours-yBandTour.impex", "utf-8");
    final BandModel band = bandService.getBandForCode("A001");
    assertNotNull("No band found", band);
    final Set<ProductModel> tours = band.getTours();
    assertNotNull("No tour found", tours);
    Assert.assertEquals("not found one tour", 1, tours.size());
    final Object[] objects = new Object[5];
    final Collection<VariantProductModel> concerts = ((ProductModel) tours.toArray(objects)[0]).getVariants();
    assertNotNull("No tour found", tours);
    Assert.assertEquals("not found one tour", 6, concerts.size());
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to edit the integration test or you want to skip this step, replace *<HYBRIS_HOME_DIR>*`/hybris/bin/custom/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceIntegrationTest.java` with the contents of *<HYBRIS_HOME_DIR>*`/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceIntegratio`

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/service/impl/DefaultBandServiceIntegrationTestW
```

```
copy /y %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\service\impl\DefaultBandServiceIntegratio
```

c. Initialize the system.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all yunitinit
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all yunitinit
```

d. Run the tests to confirm that they all pass.

```
ant alltests -Dtestclasses.packages="concerttours.*"
```

```
ant alltests -Dtestclasses.packages=concerttours.*
```

19. Start SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh start
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat start
```

20. Run the `testBandImages` acceptance test again and confirm that now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBandImages test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testBandImages test
```

21. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Add Media Files"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Add Media Files"
```

Properties Files

SAP Commerce relies on two essential configuration files: project.properties and local.properties. Project properties are the SAP Commerce defaults, while local properties is where you may define your own configuration for your extension.

You can set values in different properties files. Each file has a different priority and its values can override the values of another file with lower priority. The order of priority from hight to low is:

1. The `local.properties` file is a working copy of the `project.properties` file, located in the *<HYBRIS_CONFIG_DIR>* directory. It allows you to override default settings from the `project.properties` file. Use the `local.properties` file to set values for configuration properties that you need to configure for your project.
2. The extension-specific `project.properties` file is located in the *<HYBRIS_BIN_DIR>/<EXTENSION_DIR>*. It defines the values used for the extension.
3. The global `project.properties` file is located in the *<HYBRIS_BIN_DIR>*`/platform` directory, and provides factory default settings. It is not recommended to edit this file.

Parent topic: Explore SAP Commerce Tour

Previous: Media Files

Related Information

Configuring the Behavior of SAP Commerce

Define Default Property Values

Create a `project.properties` file that defines default configuration property values for your extension.

Prerequisites

The following acceptance test is included in your `Hybris123Tests.java` file.

```
public void testPropertiesFiles() {
assertTrue(
checkTestSuiteXMLMatches("(.*)testsuite errors=\"0\" failures=\"0\" (.*) name=\"DefaultBandFacadeIntegrationWithPropertiesTest\" package=\"concertto
}
```

Before you begin this step, run the test to confirm that it fails by executing this command:

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testPropertiesFiles test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testPropertiesFiles test
```

Procedure

1. Stop SAP Commerce.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ./hybrisserver.sh stop
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & hybrisserver.bat stop
```

2. Update the facade class to use property values.

Edit the `DefaultBandFacade` class in the `concerttours.facades.impl` package under the `src` directory of the `concerttours` extension. Replace hard-coded values with property names as follows:

```
package concerttours.facades.impl;
import de.hybris.platform.core.model.media.MediaContainerModel;
import de.hybris.platform.core.model.media.MediaFormatModel;
import de.hybris.platform.core.model.product.ProductModel;
import de.hybris.platform.servicelayer.config.ConfigurationService;
import de.hybris.platform.servicelayer.media.MediaService;
import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Required;
import concerttours.data.BandData;
import concerttours.data.TourSummaryData;
import concerttours.enums.MusicType;
import concerttours.facades.BandFacade;
import concerttours.model.BandModel;
import concerttours.service.BandService;
import java.util.Locale;

public class DefaultBandFacade implements BandFacade
{
   public  static final String BAND_LIST_FORMAT = "band.list.format.name";
   private static final String BAND_DETAIL_FORMAT = "band.detail.format.name";
   private BandService bandService;
   private MediaService mediaService;
   private ConfigurationService configService;
   @Override
   public List<BandData> getBands()
   {
      final List<BandModel> bandModels = bandService.getBands();
      final List<BandData> bandFacadeData = new ArrayList<>();
      if (bandModels!=null && !bandModels.isEmpty()) //6.2
      {
         final String mediaFormatName = configService.getConfiguration().getString(BAND_LIST_FORMAT);
        System.out.println("mediaFormatName:"+mediaFormatName);
        final MediaFormatModel format = mediaService.getFormat(mediaFormatName);
         for (final BandModel sm : bandModels)
         {
            final BandData sfd = new BandData();
            sfd.setId(sm.getCode());
            sfd.setName(sm.getName());
            sfd.setDescription(sm.getHistory(Locale.ENGLISH));
            sfd.setAlbumsSold(sm.getAlbumSales());
            sfd.setImageURL(getImageURL(sm, format));
            bandFacadeData.add(sfd);
         }
      }
      return bandFacadeData;
   }

   @Override
   public BandData getBand(final String name)
   {
      if (name == null)
      {
         throw new IllegalArgumentException("Band name cannot be null");
      }
      final BandModel band = bandService.getBandForCode(name);
      if (band == null)
      {
         return null;
      }
      // Create a list of genres
      final List<String> genres = new ArrayList<>();
      if (band.getTypes() != null)
      {
         for (final MusicType musicType : band.getTypes())
         {
            genres.add(musicType.getCode());
         }
      }
      // Create a list of TourSummaryData
      final List<TourSummaryData> tourHistory = new ArrayList<>();
      if (band.getTours() != null)
      {
         for (final ProductModel tour : band.getTours())
         {
            final TourSummaryData summary = new TourSummaryData();
            summary.setId(tour.getCode());
            summary.setTourName(tour.getName(Locale.ENGLISH));
            // making the big assumption that all variants are concerts and ignore product catalogs
            summary.setNumberOfConcerts(Integer.toString(tour.getVariants().size()));
            tourHistory.add(summary);
         }
      }
      // Now we can create the BandData transfer object
      final String mediaFormatName = configService.getConfiguration().getString(BAND_DETAIL_FORMAT);
      final MediaFormatModel format = mediaService.getFormat(mediaFormatName);
      final BandData bandData = new BandData();
      bandData.setId(band.getCode());
      bandData.setName(band.getName());
      bandData.setAlbumsSold(band.getAlbumSales());
      bandData.setImageURL(getImageURL(band, format));
      bandData.setDescription(band.getHistory(Locale.ENGLISH));
      bandData.setGenres(genres);
      bandData.setTours(tourHistory);
      return bandData;
   }

   protected String getImageURL(final BandModel sm, final MediaFormatModel format)
   {
      final MediaContainerModel container = sm.getImage();
      if (container != null)
      {
         return mediaService.getMediaByFormat(container, format).getDownloadURL();
```

```
        }
        return null;
    }

    @Required
    public void setBandService(final BandService bandService)
    {
        this.bandService = bandService;
    }

    @Required
    public void setMediaService(final MediaService mediaService)
    {
        this.mediaService = mediaService;
    }

    @Required
    public void setConfigurationService(final ConfigurationService configService)
    {
        this.configService = configService;
    }
}
```

SAP Commerce 123 Interactive Shortcut: If you are unable to update the facade yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/src/concerttours/facades/impl/DefaultBandFacade.java with the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultBandFacadeWithProperties.java

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/src/concerttours/facades/impl/DefaultBandFacadeWithProperties.java $HYB
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\src\concerttours\facades\impl\DefaultBandFacadeWithProperties
```

3. Update your Spring configuration by adding the injection of the `ConfigurationService` to the `concerttours-spring.xml` file.

   The configuration for the `defaultBandFacade` now looks as follows:

```
<alias name = "defaultBandFacade" alias = "bandFacade" />
<bean id = "defaultBandFacade" class ="concerttours.facades.impl.DefaultBandFacade" >
    <property name = "bandService" ref = "bandService" />
    <property name="mediaService" ref="mediaService"/>
    <property name="configurationService" ref="configurationService" />
</bean>
```

   SAP Commerce 123 Interactive Shortcut: If you are unable to update the Spring XML yourself or you want to skip this step, replace *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/resources/concerttours-spring.xml with the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withConfigurationService.xml

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/resources/concerttours-spring-withConfigurationService.xml $HYBRIS_HOME
```

```
echo a | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\resources\concerttours-spring-withConfigurationService.xml %H
```

4. Define default project properties.

   Add the following lines to your `concerttours` extension's `project.properties` file.

```
# media format names
band.list.format.name = bandList
band.detail.format.name = bandDetail
```

   SAP Commerce 123 Interactive Shortcut: If you are unable to update `project.properties` or you want to skip this step, add the contents of *<HYBRIS_HOME_DIR>*/hybris123/src/main/webapp/resources/concerttours/projectpropertieswithbands.txt to *<HYBRIS_HOME_DIR>*/hybris/bin/custom/concerttours/project.properties

```
cp $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/projectpropertieswithbands.txt $HYBRIS_HOME_DIR/hybris/bin/custom/concertt
```

```
copy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\projectpropertieswithbands.txt %HYBRIS_HOME_DIR%\hybris\bin\custom\conc
```

5. Create a new integration test to test the `DefaultBandFacade` facade when using these new properties by creating a test class called `concerttours.facades.impl.DefaultBandFacadeIntegrationWithPropertiesTest` under the `testsrc` folder of the `concerttours` extension.

```
package concerttours.facades.impl;
import static org.junit.Assert.assertTrue;
import java.lang.InterruptedException;
import java.util.List;
import java.util.concurrent.TimeUnit;

import de.hybris.bootstrap.annotations.IntegrationTest;
import de.hybris.platform.servicelayer.ServicelayerTransactionalTest;
import javax.annotation.Resource;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;
import concerttours.data.BandData;
import concerttours.facades.BandFacade;
import de.hybris.platform.core.Registry;
import org.springframework.jdbc.core.JdbcTemplate;

@IntegrationTest
public class DefaultBandFacadeIntegrationWithPropertiesTest extends ServicelayerTransactionalTest
{
@Resource
private BandFacade bandFacade;

@Before
public void setUp() throws Exception
{
    try {
        Thread.sleep(TimeUnit.SECONDS.toMillis(1));
        new JdbcTemplate(Registry.getCurrentTenant().getDataSource()).execute("CHECKPOINT");
        Thread.sleep(TimeUnit.SECONDS.toMillis(1));
```

```
        }
        catch (InterruptedException exc) {}
    }

    @Test
    public void testProperties() throws Exception
    {
        createCoreData();
        importCsv("/impex/essentialdata-mediaformats.impex", "UTF-8");
        importCsv("/impex/concerttours-bands.impex", "UTF-8");
        importCsv("/impex/withlocalization/concerttours-yBandTour.impex", "UTF-8");

        List<BandData> bands = bandFacade.getBands();
        assertTrue(bands.size() > 0);
        assertTrue( DefaultBandFacade.BAND_LIST_FORMAT.equals("band.list.format.name"));
    }

    @After
    public void teardown() {

    }
    }
```

SAP Commerce 123 Interactive Shortcut: If you are unable to create the test class yourself or you want to skip this step, replace `<HYBRIS_HOME_DIR>/hybris/bin/custom/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeIntegrationWithPropertiesTest.java` with the contents of `<HYBRIS_HOME_DIR>/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeIntegrationWithPr`

```
ditto $HYBRIS_HOME_DIR/hybris123/src/main/webapp/resources/concerttours/testsrc/concerttours/facades/impl/DefaultBandFacadeIntegrationWithPrope
```

```
echo F | xcopy %HYBRIS_HOME_DIR%\hybris123\src\main\webapp\resources\concerttours\testsrc\concerttours\facades\impl\DefaultBandFacadeIntegratio
```

6. Rebuild SAP Commerce with Ant.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant clean all
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant clean all
```

7. Run the integration tests.

```
cd $HYBRIS_HOME_DIR/hybris/bin/platform; ant integrationtests -Dtestclasses.packages="concerttours.*"
```

```
cd %HYBRIS_HOME_DIR%\hybris\bin\platform & ant integrationtests -Dtestclasses.packages=concerttours.*
```

8. View the test results at `<HYBRIS_HOME_DIR>/hybris/log/junit/index.html` and confirm that `DefaultBandFacadeIntegrationWithPropertiesTest` has passed.
9. Run the `testPropertiesFiles` acceptance test again and confirm that it now passes.

```
cd $HYBRIS_HOME_DIR/hybris123; mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testPropertiesFiles test
```

```
cd %HYBRIS_HOME_DIR%\hybris123 & mvn -Dtest=com.hybris.hybris123.runtime.tests.Hybris123Tests#testPropertiesFiles test
```

10. Commit the changes to your local Git repository.

```
cd $HYBRIS_HOME_DIR/hybris; git add . ; git commit -m "Define Default Property Values"
```

```
cd %HYBRIS_HOME_DIR%\hybris & git add . & git commit -m "Define Default Property Values"
```

11. Clear the *<INITIAL_ADMIN>* environment variable.
    a. Open your shell profile using a command line text editor.

    ```
    pico ~/.bash_profile
    ```

    b. Replace the content of the file with the following lines.

    ```
    export HYBRIS_HOME_DIR=/opt/CXCOMM220500P_X-XXXXXXXX
    export ANT_HOME=${HYBRIS_HOME_DIR}/hybris/bin/platform/apache-ant
    export PATH=${PATH}:${ANT_HOME}/bin
    ```

```
setx INITIAL_ADMIN "" /m
```

Results

Congratulations! You have completed the tour. You should now have a good understanding of the core concepts of SAP Commerce.

Next Steps

If you are going through multiple guided tours, delete any unzipped SAP Commerce folders and restart your computer between tours to be sure that there are no SAP Commerce threads still running. Then perform the steps in Before You Start and start your next tour.