Hot Folder

Introduction:

You have already seen how you can use ImpEx files to import data in the system. hybris supports hot folders which are folders from which data can be automatically imported into hybris by simply placing the data inside of the folder. The hybris **acceleratorservices extension template** comes with a batch package that enables automated importing of data from hot folders.

- The infrastructure enables using simple CSV files (internally translated into hybris ImpEx scripts) to import content directly into the product catalogs.
- The Spring integration provides a pluggable, highly configurable service-based design that can be extended as required.
- In this trail we're going to enable this functionality and use it for merchandise store.

Enable the standard hot folder for merchandise:

1. Creae a" hot-folder-store-hybris-spring.xml" in specified path and write this code

merchandisecore/resources/merchandisecore/integration/hot-folder-store-hybris-spring.xml

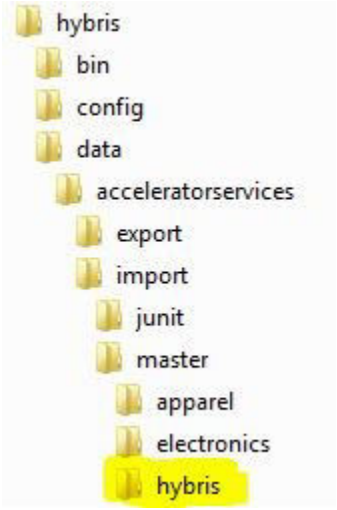
```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 [y] hybris Platform

 Copyright (c) 2000-2014 hybris AG
 All rights reserved.

 This software is the confidential and proprietary information of 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 hybris.
-->

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

```
      http://www.springframework.org/schema/integration
      http://www.springframework.org/schema/integration/spring-integration.xsd
      http://www.springframework.org/schema/integration/file
      http://www.springframework.org/schema/integration/file/spring-integration-
file.xsd">

   <bean id="baseDirectoryHybris" class="java.lang.String">
      <constructor-arg value="#{baseDirectory}/${tenantId}/hybris" />
   </bean>

   <!-- 1) Scan for files -->
   <file:inbound-channel-adapter id="batchFilesHybris"
directory="#{baseDirectoryHybris}"
      filename-regex="^(.*)-(\d+)\.csv" comparator="fileOrderComparator">
      <int:poller fixed-rate="1000" />
   </file:inbound-channel-adapter>

   <!-- 2) move the file to processing and setup header -->
   <file:outbound-gateway request-channel="batchFilesHybris" reply-
channel="batchFilesHybrisProc"
      directory="#{baseDirectoryHybris}/processing" delete-source-files="true" />

   <int:service-activator input-channel="batchFilesHybrisProc" output-
channel="batchFilesHeaderInit"
      ref="hybrisHeaderSetupTask" method="execute" />

   <bean id="hybrisHeaderSetupTask"
class="de.hybris.platform.acceleratorservices.dataimport.batch.task.HeaderSetupTask"
>
      <property name="catalog" value="hybrisProductCatalog" />
      <property name="net" value="false" />
      <property name="storeBaseDirectory" ref="baseDirectoryHybris" />
   </bean>
</beans>
```

Enable in Spring:

```
...
   <!-- Spring Integration -->
   <import resource="classpath:/merchandisecore/integration/hot-folder-store-hybris-spring.xml"/>
   ...
```

1.Restart the Server to apply the changes:
you will find the changes in this documentry struchure
hybris/data/acceleratorservices/import/master/hybris



Test the new hot folder:

1.Create price-001.csv file :

```
"0007","75.00","GBP","FALSE"
"0007","50.00","EUR","FALSE"
```

We'll do a small test by changing the price of the **Wiesnherz** to 75,00 GBP resp. 50,00 EUR.

2.Place price-001.csv file in Hot Folder ie:

```
hybris-commerce-suite-5.7.0.2/hybris/data/acceleratorservices/import/master/hybris
```

- If everything goes right you'll see that the file is cleared from that directory, renamed and moved to the new “archive” directory

You'll see the changed price immediately on the product details page,

i.e. the update was affecting both the staged and online version.

Trail:If you do only want to change the staged prices you must override the Converter which can be found in

merchandisecore/resources/merchandisecore/integration/hot-folder-common-spring.xml

```
   <bean id="batchPriceConverter"
class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.impl.DefaultI
mpexConverter">
      <property name="header">
         <value># ImpEx for Importing Prices into $CATALOG$
            $catalog=$CATALOG$
            #% impex.setLocale(Locale.ENGLISH);
            INSERT_UPDATE
PriceRow;product(code,catalogversion(catalog(id),version))
[unique=true];price[translator=de.hybris.platform.acceleratorservices.dataimport.batch.
converter.PriceTranslator];currency(isocode)
[unique=true];net[default=$NET$];unit(code)
[default=pieces];unitFactor[default=1];minqtd[default=1];catalogversion(catalog(id),ve
rsion);sequenceId[translator=de.hybris.platform.acceleratorservices.dataimport.batch.co
nverter.SequenceIdTranslator]
         </value>
      </property>
      <property name="impexRow">
         <value>;{+0}:$catalog:Staged;{+1};{+2};{3};;;;$catalog:Staged;{S}</value>
      </property>
   </bean>
```

1.Restart the hybris server

2.create price-001.csv file

```
"0007","12.00","GBP","FALSE"
"0007","15.00","EUR","FALSE"
```

3.Place price-001.csv file in Hot Folder

Verify that the price has only changed on the staged product catalog version

Synchronize the hybris product catalog

Task3:Allow import of regular and internal products

Create thefolder:

```
hybris/data/acceleratorservices/import/master/hybris/internal
```

Now apply the following changes to

hot-folder-store-hybris-spring.xml

1. **Define a new inbound-channel-adapter for polling the new folder**

```
<!-- 1) Scan for files (used for internal products)-->
<file:inbound-channel-adapter id="batchFilesHybrisInternal"
   directory="#{baseDirectoryHybris}/internal" filename-regex="^(.*)-(\d+)\.csv"
   comparator="fileOrderComparator">
   <int:poller fixed-rate="1000" />
</file:inbound-channel-adapter>
```

Create an outbound-gateway which copies the files to the usual *processing* folder with the difference that they are prefixed by internal

```
<!-- 2) move the file to processing and setup header -->
<file:outbound-gateway request-channel="batchFilesHybrisInternal"
    directory="#{baseDirectoryHybris}/processing"
    reply-channel="batchFilesHybrisProc"
   delete-source-files="true" filename-generator-expression="'internal_' +
payload.name" />
```

Actually there two imaginable solutions from this point.

1. Extend BatchHeader with internal attribute and set this through a service-activator.
   Pro: batchProductConverter has to be defined only once (see $CATALOG$)
   Con: BatchHeader comes from acceleratorservices and could not be overridden easily
2. Create a custom converter mapping and converter.
   Pro: easy and it does not affect to the other steps
   Con: not very generic

We go here for the second approach what means that we have to define a specific converter for internal products and according to this a converter mapping.

The mapping isn't difficult as we can identify internal products at this stage via the filename prefix

```
<bean id="batchInternalProductConverterMapping"

class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.
DefaultConverterMapping"
     p:mapping="internal_base_product"
     p:converter-ref="batchInternalProductConverter"/>

<bean id="batchInternalProductConverter"
class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.impl.DefaultI
mpexConverter">
   <property name="header">
      <value>#{defaultImpexProductHeader}
         # Insert Products
         INSERT_UPDATE
Product;code[unique=true];varianttype(code);name[lang=$lang];description[lang=$lan
g];ean;manufacturerName;manufacturerAID;unit(code)[default=pieces];
$approved;Europe1PriceFactory_PTG(code)[default=eu-vat-
full];sequenceId[translator=de.hybris.platform.acceleratorservices.dataimport.batch.con
verter.SequenceIdTranslator];$catalogVersion;internalOnly[default=true]
      </value>
   </property>
   <property name="impexRow">
      <value>;{+0};{1};{2};{3};{4};{5};{6};{7};{8};{9};{S}</value>
   </property>
   <property name="rowFilter">
      <bean
class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.impl.DefaultI
mpexRowFilter">
         <property name="expression" value="!row[1]"/>
      </bean>
   </property>
   <property name="type" value="Product"/>
</bean>
```

The complete file will look like this:

merchandisecore/resources/merchandisecore/integration/hot-folder-store-hybris-spring.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 [y] hybris Platform

 Copyright (c) 2000-2014 hybris AG
 All rights reserved.

 This software is the confidential and proprietary information of 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 hybris.
-->

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:int="http://www.springframework.org/schema/integration"
    xmlns:file="http://www.springframework.org/schema/integration/file"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/integration
        http://www.springframework.org/schema/integration/spring-integration.xsd
        http://www.springframework.org/schema/integration/file
        http://www.springframework.org/schema/integration/file/spring-integration-
file.xsd">
```

```
   <bean id="baseDirectoryHybris" class="java.lang.String">
      <constructor-arg value="#{baseDirectory}/${tenantId}/hybris" />
   </bean>

   <!-- 1) Scan for files -->
   <file:inbound-channel-adapter id="batchFilesHybris"
directory="#{baseDirectoryHybris}"
      filename-regex="^(.*)-(\d+)\.csv" comparator="fileOrderComparator">
      <int:poller fixed-rate="1000" />
   </file:inbound-channel-adapter>

   <!-- 2) move the file to processing and setup header -->
   <file:outbound-gateway request-channel="batchFilesHybris" reply-
channel="batchFilesHybrisProc"
      directory="#{baseDirectoryHybris}/processing" delete-source-files="true" />

   <int:service-activator input-channel="batchFilesHybrisProc" output-
channel="batchFilesHeaderInit"
      ref="hybrisHeaderSetupTask" method="execute" />

   <bean id="hybrisHeaderSetupTask"
class="de.hybris.platform.acceleratorservices.dataimport.batch.task.HeaderSetupTask"
>
      <property name="catalog" value="hybrisProductCatalog" />
      <property name="net" value="false" />
      <property name="storeBaseDirectory" ref="baseDirectoryHybris" />
   </bean>



   <!-- 1) Scan for files (used for internal products)-->
   <file:inbound-channel-adapter id="batchFilesHybrisInternal"
                        directory="#{baseDirectoryHybris}/internal" filename-
regex="^(.*)-(\d+)\.csv"
                        comparator="fileOrderComparator">
      <int:poller fixed-rate="1000" />
   </file:inbound-channel-adapter>

   <!-- 2) move the file to processing and setup header -->
   <file:outbound-gateway request-channel="batchFilesHybrisInternal"
                     directory="#{baseDirectoryHybris}/processing"
                     reply-channel="batchFilesHybrisProc"
```

```
                    delete-source-files="true" filename-generator-expression="'internal_' +
payload.name" />

   <bean id="batchInternalProductConverterMapping"

class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.
DefaultConverterMapping"
        p:mapping="internal_base_product"
        p:converter-ref="batchInternalProductConverter"/>

   <bean id="batchInternalProductConverter"
class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.impl.DefaultI
mpexConverter">
      <property name="header">
         <value>#{defaultImpexProductHeader}
            # Insert Products
            INSERT_UPDATE
Product;code[unique=true];varianttype(code);name[lang=$lang];description[lang=$lan
g];ean;manufacturerName;manufacturerAID;unit(code)[default=pieces];
$approved;Europe1PriceFactory_PTG(code)[default=eu-vat-
full];sequenceId[translator=de.hybris.platform.acceleratorservices.dataimport.batch.con
verter.SequenceIdTranslator];$catalogVersion;internalOnly[default=true]
         </value>
      </property>
      <property name="impexRow">
         <value>;{+0};{1};{2};{3};{4};{5};{6};{7};{8};{9};{S}</value>
      </property>
      <property name="rowFilter">
         <bean
class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.impl.DefaultI
mpexRowFilter">
            <property name="expression" value="!row[1]"/>
         </bean>
      </property>
      <property name="type" value="Product"/>
   </bean>
</beans>
```

Testing internalfolder:

Restart your server to apply your changes.:

base_product-001.csv

```
"0007","50.00","GBP","FALSE"
"0007","1.00","EUR","FALSE"
```

hybris/data/acceleratorservices/import/master/hybris/internal folder

- Now let's import some images for the new product

Download the imges folder by using ths url

```
https://wiki.hybris.com/pages/viewpage.action?pageId=294094383&preview=/252609535/251534966/images.zip
```

1. Unpack images.zip into the

   hybris

   hotfolder (NOT in the internal folder)
2. Execute media-001.csv on the

```
"0008","0008.jpg"
```

   hybris

   hotfolder

-