Hot Folder

Summary

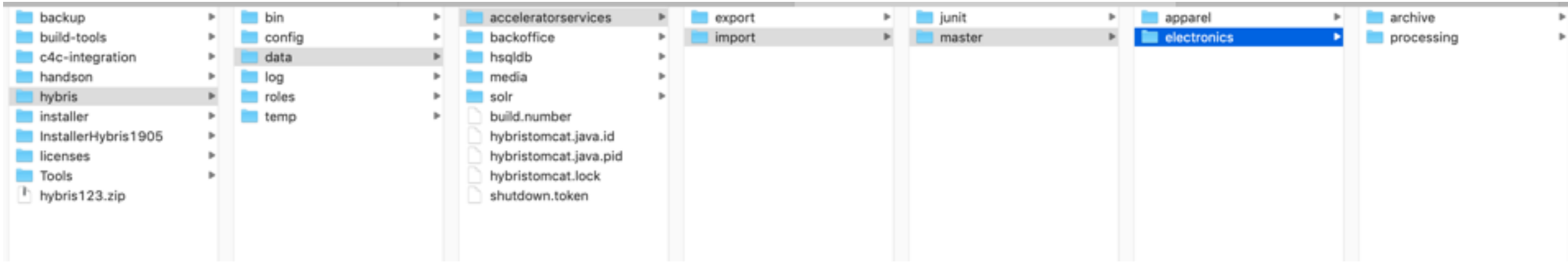
- What is Hot folder
- Hot folder in other integration frameworks
- Advantages of Hot folder
- Diagram Of Components
- Hot folder mechanism
- Hot folder configuration
- Add new import
- Advanced customisation
- Demo - Sample imports
- Exercise
- Thank You

What is Hot folder

A folder that serves as a staging area for some purpose. The hot folder is continuously monitored, and when files are copied or dropped into it, they are automatically processed.

✓ Following Hybris:

✓ Hot Folder = Scanned Folder periodically + processed



Hot folder in other integration frameworks

The hot folder mechanism is implemented by many frameworks:

- Camel
- Spring integration
- Mule ESB

Hybris case

- Hybris support hot folder by using Spring integration which:
- pluggable,
- highly configurable service-based design
- extendable.
- Easy integration with Hybris

Why using Hot folder

✓ Automatise data import: product, stock, vendors…

✓ No need of Impex knowledge

✓ It’s very fast than the regular impex import (uses multi-threading imports)

✓ The use of .csv files instead of impex files (other formats: xml need customisation)

✓ Files can be generated by other systems: SAP ERP, Talend for example.

Diagram Of Components

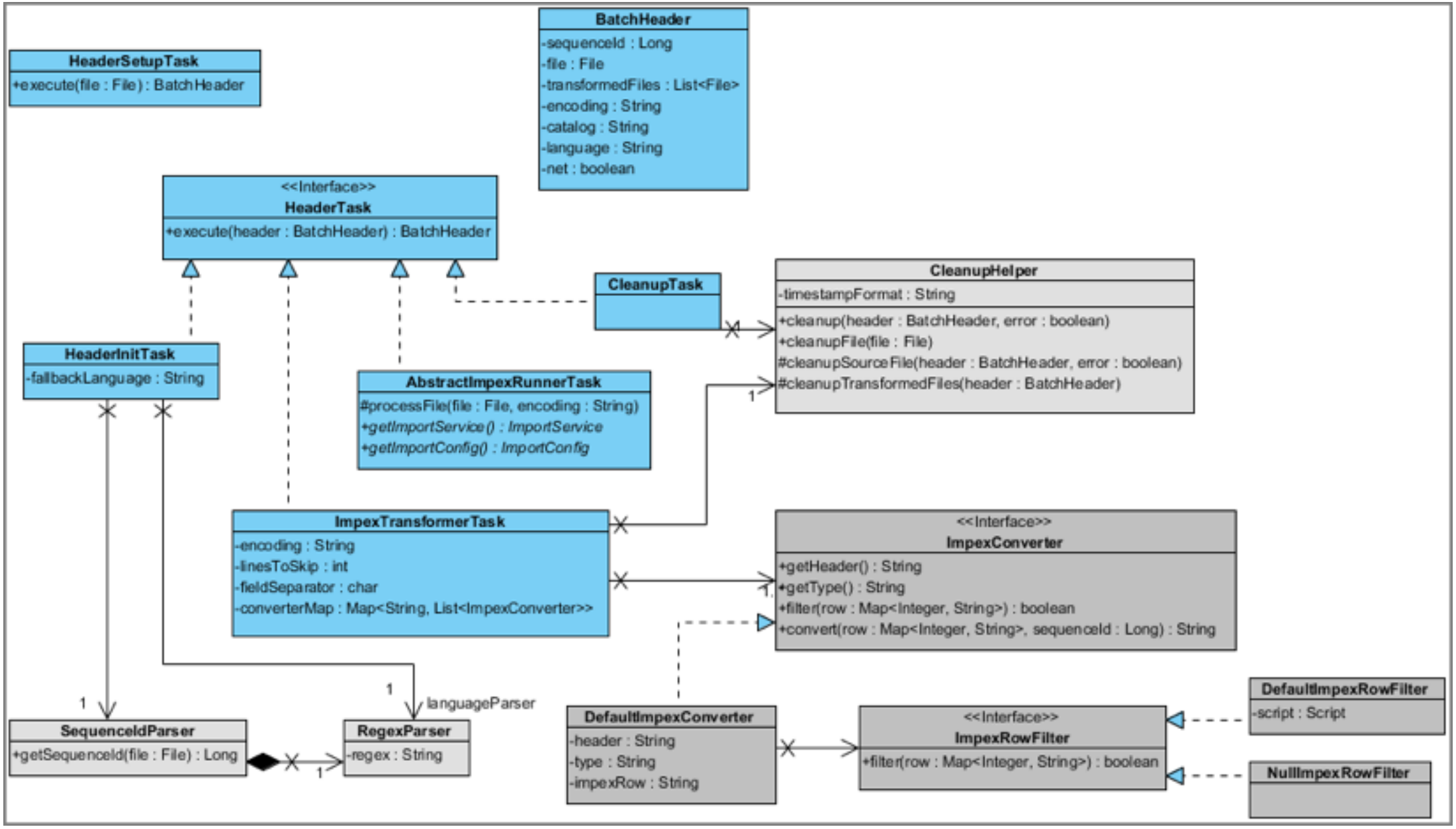
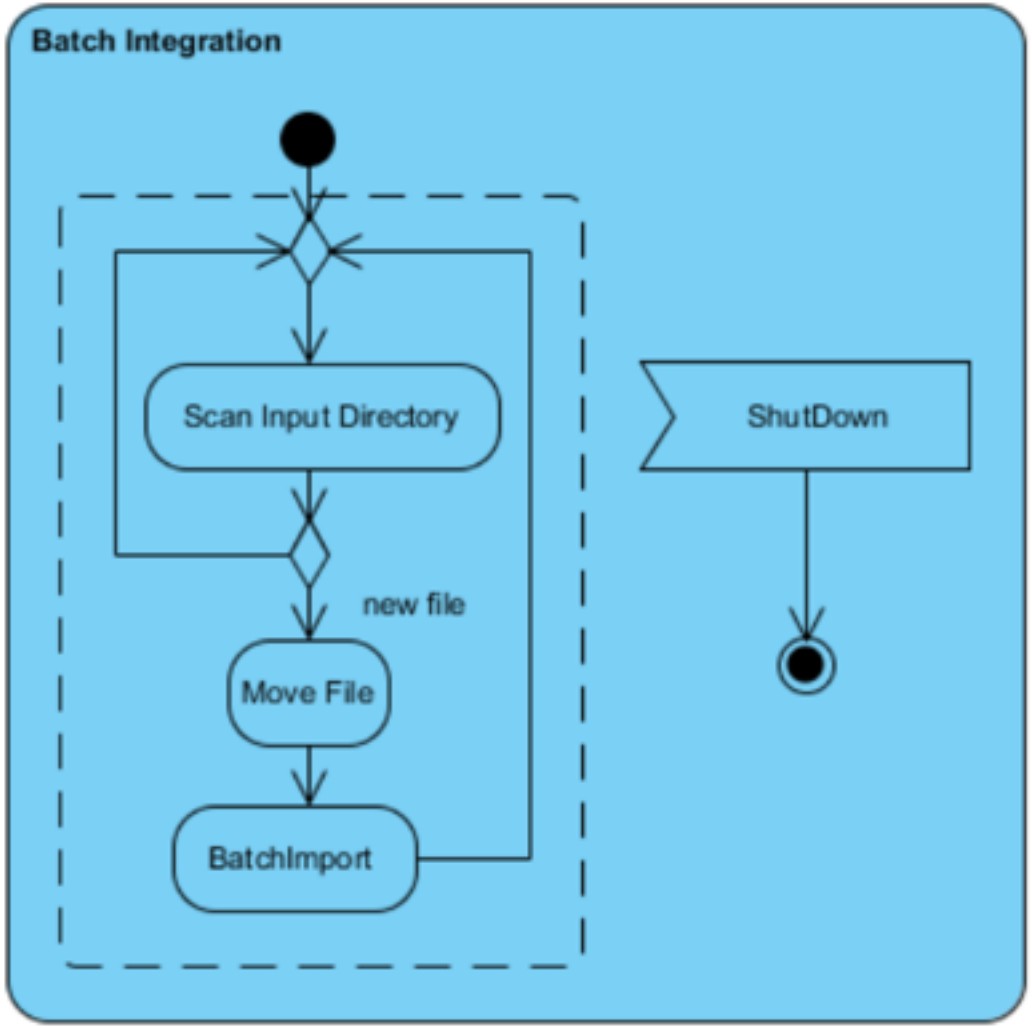


Diagram Of Components …

The classes are structured into **three** major parts:

- Tasks executed by the Spring integration infrastructure.
- Converters providing the ImpEx header and converting CSV rows into ImpEx rows with optional filtering.
- Helper and utility classes: SequenceIdParser, RegexParser, and CleanupHelper
- Tasks are nothing but java class used to retrieve configurations and create impex
- Tasks used fancy to import impex into Hybris using import service.

Hot folder mechanism : Batch Integration

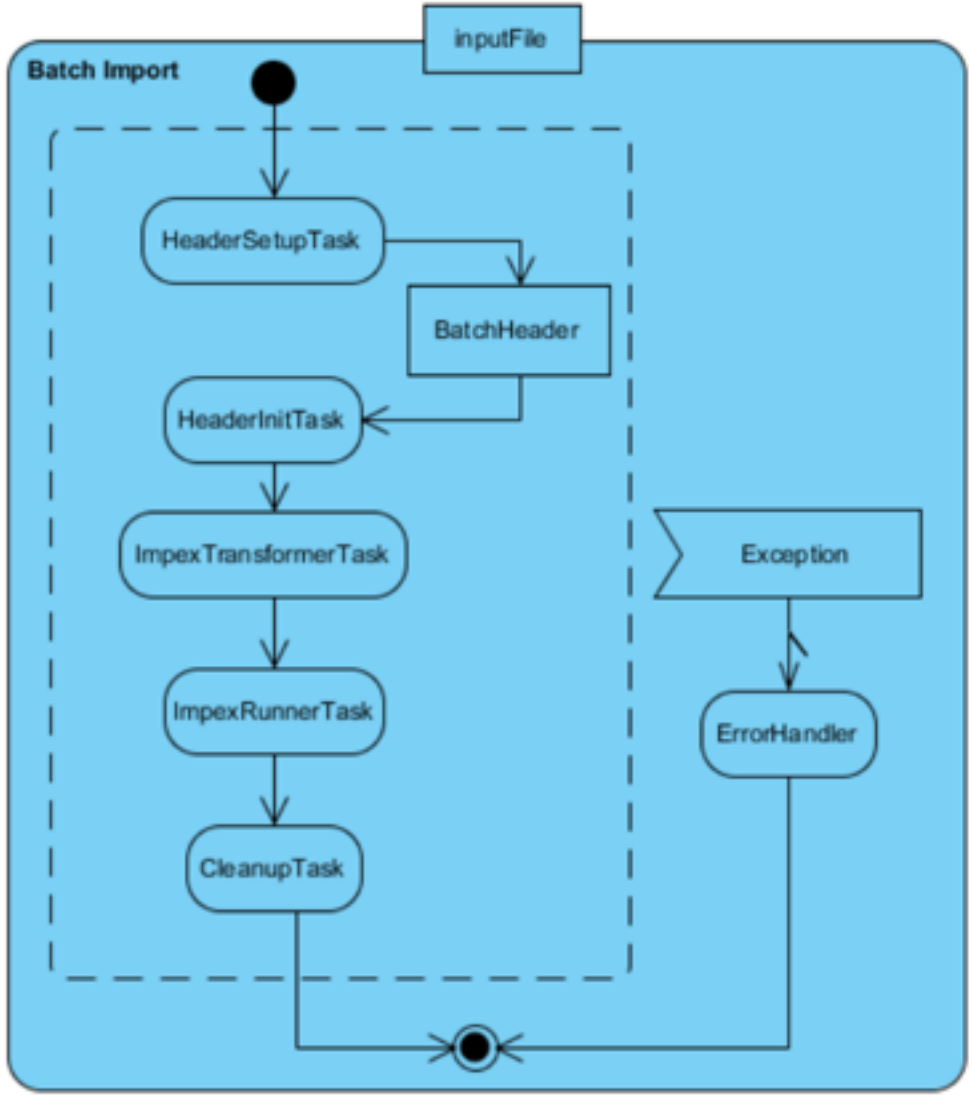


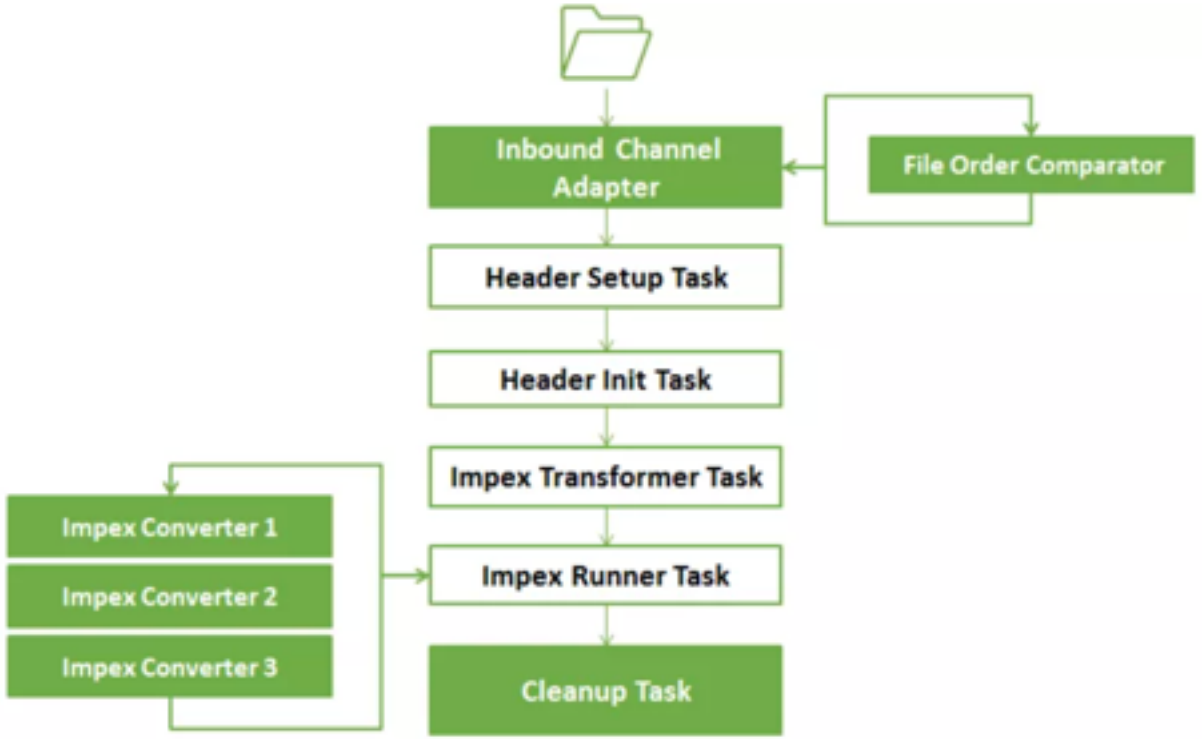
Spring integration periodically scans the configured input directory for new files.If new files are found, they are moved to the processing subdirectory and then sent to the Batch Import pipeline, which consists of the following tasks:

- HeaderSetupTask: Creates a new BatchHeader.
- HeaderInitTask: Retrieves a sequence ID and (optionally) a language from the file name.
- ImpexTransformerTask: Creates one or many ImpEx files from the CSV input and writes error lines to the error subdirectory.
- ImpexRunnerTask: Processes all ImpEx files sequentially with multiple threads.
- CleanupTask: Deletes all transformed files and moves the imported file with an optionally appended timestamp to the archive subdirectory.

- ErrorHandler: Deletes all transformed files and moves the imported file with an optionally appended timestamp to the error subdirectory.

Hot folder mechanism : Batch import





Hot folder : How it works

1. **Inbound Channel Adapter** : is a Spring integration component that allows us to create a watcher over csv files on a specific directory.
2. **File Order Comparator** : the Inbound Channel Adapter uses a file order comparator to treat files by order of priority, this comparator compares files based on their file names.
3. **Header Setup Task** : in this step of the flow a BatchHeader is created/initialized with the file, catalog, and some other information, this BatchHeader will be used throughout the flow as a reference for file and some other information.
4. **Header Init Task** : this is just another initialization step, in this step we extract the sequenceId and the language then we add them to the BatchHeader for later use.For example for this file name product-fr-2313213672186.csv the sequenceId is 2313213672186 and the language is fr.
5. **Impex Transformer Task** : this is one of the important steps in the flow, basically here where the original file (csv) is converted to an impex with the help of a pre-configured ImpexConverter.
6. **Impex Runner Task** : imports the transformed file (impex) using ImportService.importData() method.
7. **Cleanup Task** : deletes the transformed file (impex) and archive the original file (csv).

The majority of the components that constitute the hot folder flow can be found at:

hybris1905/hybris/bin/modules/core-accelerator/acceleratorservices/resources/acceleratorservices/integration/hot-folder-spring.xml

Configuration

File	Location	Configuration Items
hot-folder-spring.xml	<HYBRIS_BIN_DIR>/modules/core-accelerator/acceleratorservices/resources/acceleratorservices/integration	• Spring integration infrastructure including AOP setup. • Common ImpEx converters, filters, and converter mappings.
hot-folder-common-spring.xml **hot-folder-store-electronics-spring.xml**	<HYBRIS_BIN_DIR>/modules/base-accelerator/yacceleratorcore/resources/yacceleratorcore/integration	• Extension-related Spring integration infrastructure including hot folder setup. • Extension-related ImpEx converters, filters, and converter mappings.
project.properties	<HYBRIS_BIN_DIR>/modules/core-accelerator/acceleratorservices	• acceleratorservices.batch.impex.max-threads: number of threads used for ImpEx. • acceleratorservices.batch.impex.basefolder: base folder of the import infrastructure containing catalog specific subfolders.

Configuration

+ Task configuration

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

Configuration …

+ Converter configuration

Configuration Option	Parameters
header	NET: the net setting *$CATALOG$*: the catalog prefix *$LANGUAGE$*: the language setting *$TYPE$*: an optional type attribute that can be applied if filtering is configured
impexRow	**{('+')? (<columnId> | 'S')}**.: {+0} to make check of mandatory attribute, {+S} to generate a sequence ID. **rowFilter:** an optional row filter. Allow use of expression which must be a valid Groovy expression For example, querying if the first column is not empty can be written as **!row**. **type**: an optional type that can be retrieved in the header using the header substitution *$TYPE$*.

Configuration …

+ Spring integration configuration

Configuration Option	Parameters
int:channel	Setup of a channel
int:service-activator	activates a referenced bean when receiving a message on a configured channel to act as service
file:inbound-channel-adapter	Scan a directory in a configurable interval and send files to a configured channel following order defined by **FileOrderComparator**
file:outbound-gateway	Move a file to the **processing** subdirectory.

Adding a New Import

To add a new import:

1. Define the file name schema for the new import, for example: customer-<sequenceId>.csv.

2. Set up a new ImpexConverter configuration describing the ImpEx header and the row expression.

3. Add the file prefix and the converter reference to the converterMap property of the ImpexConverter.

Converter Definition

```
<bean id="batchTaxConverter" class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.impl.DefaultImpexConverter">
                        <property name="header">
                                <value>...</value>
                        </property>
                        <property name="impexRow">
                                <value>...</value>
                        </property>
        </bean>
```

Adding a New Import …

Mapping Definition

```
<bean id="batchTaxConverterMapping"


class="de.hybris.platform.acceleratorservices.dataimport.batch.converter.mapping.impl.DefaultConverterMapping"

                    p:mapping="tax"

                    p:converter-ref=“batchTaxConverter”/>
```

- The batch Transformer Task collects every mapping (i.e. every bean that implements the ConverterMapping interface) and uses it for mapping the right converter.
- There is no need for further configuration; just define a mapping bean in the Spring context.

Advanced customisation

Filter

- Used before conversion
- Intercept format errors: date, numbers…
- Reject or import line

Cell decorator

- intercept the interpreting of a specific cell of a value line between parsing and translating
- Acts on a specific column

Translator

- Change the translate logic for some attribute.
- Ex: header attributes that have no counterpart at the type system
- Act on a specific column

Advanced customization: Filter

```
public class PriceImpexRowFilter implements ImpexRowFilter
{
   private Integer columnIndexDate;
   private Integer columnIndexPrice;

   /**
    * {@inheritDoc}
    */
   @Override
   public boolean filter(final Map<Integer, String> row)
   {

      String date = row.get(columnIndexDate);

      if (date != null && AuchanUtils.isFormattedDateValid(date))
      {
         date = date.replaceAll("/", ".");
         row.put(columnIndexDate, date);
      }
      else
      {
         row.put(columnIndexDate, "01.01.1970");

      }
      final String price = row.get(columnIndexPrice);
      if (price.contains(","))
      {
         return false;
      }
      return true;

   }
```

Hot folder Customization: Decorator

```java
/**
 * Decorator which fill name of customer in case it is blank. Name for customer is taken from
email address. It assume
 * that email address is in the column previous to customer name column
 */
public class CustomerNameDecorator extends AbstractImpExCSVCellDecorator
{
   private static final String SPLIT_SIGN = "@";

   @Override
   public String decorate(final int position, final Map<Integer, String> srcLine)
   {
      String name = srcLine.get(Integer.valueOf(position));
      if (name == null || name.trim().isEmpty())
      {
         final Integer emailPosition = Integer.valueOf(position - 1);
         final String email = srcLine.get(emailPosition);
         if (email != null)
         {
            name = email.split(SPLIT_SIGN)[0];
         }
      }

      return name;
   }

}
```

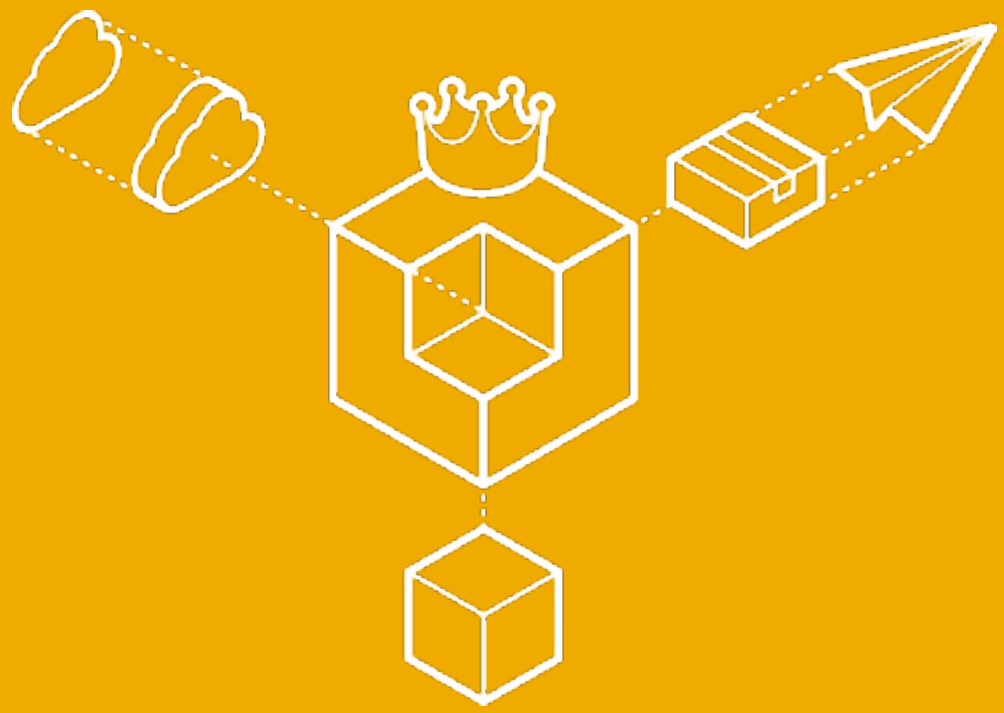
✓ Hot folder Customization: Other customizations convert XML to CSV

```
class Xml2Csv {

    public static void main(final String args[]) throws Exception {
        final File stylesheet = new File("src/main/resources/style.xsl");
        final File xmlSource = new File("src/main/resources/os.xml");

        final DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        final DocumentBuilder builder = factory.newDocumentBuilder();
        final Document document = builder.parse(xmlSource);

        final StreamSource stylesource = new StreamSource(stylesheet);
        final Transformer transformer =
TransformerFactory.newInstance().newTransformer(stylesource);
        final Source source = new DOMSource(document);
        final Result outputTarget = new StreamResult(new File("src/main/resources/os.csv"));
        transformer.transform(source, outputTarget);
    }
}
```



DEMO

Exercise

Thank you.