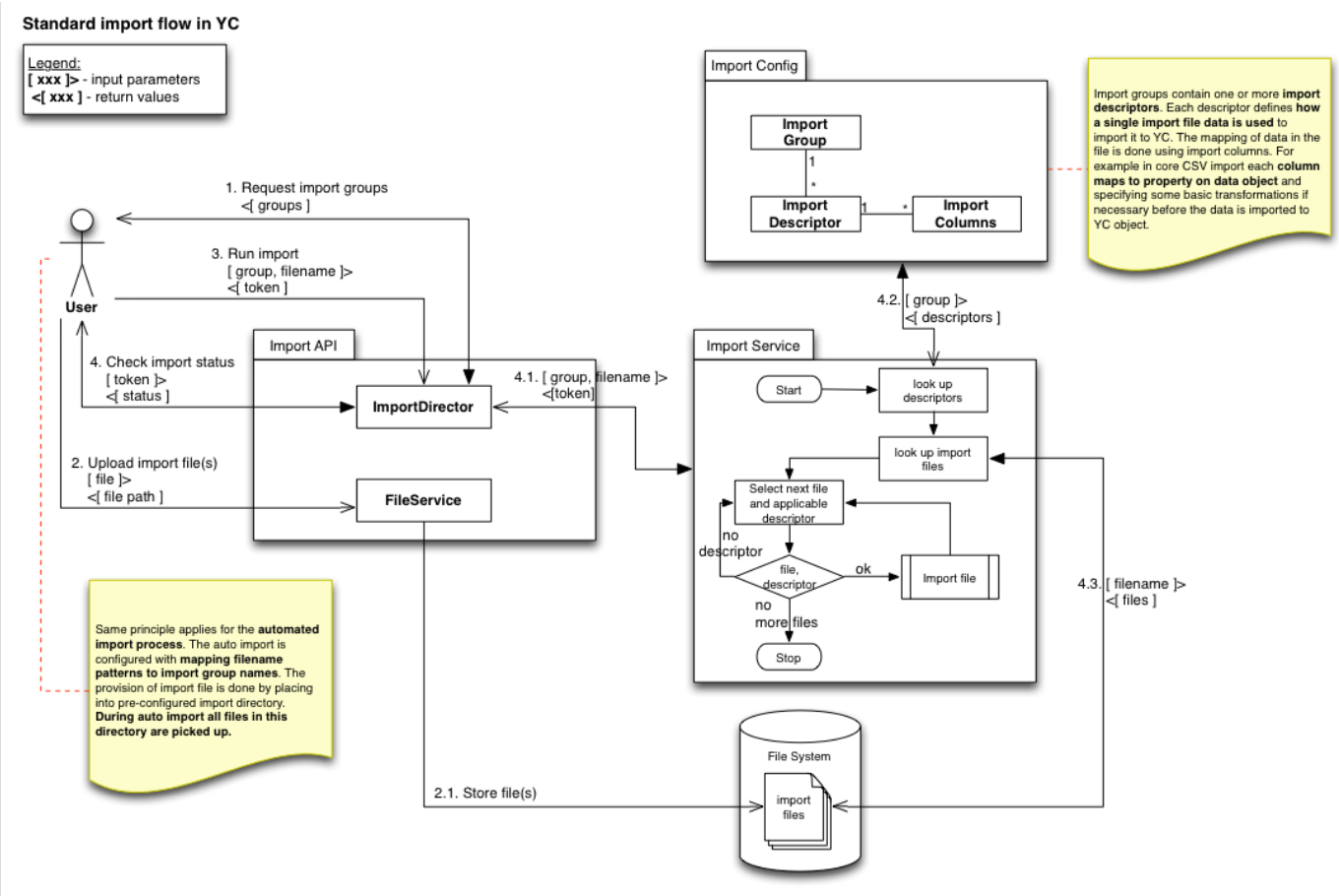
Automatic ImpEx

Overview

Main components of the impex module are depicted in the diagram below. In both cases import and export the module relies on the local file system (or a shared drive) in order to consume or produce the files respectively.

This provides a simple self contained channel for all data. All additional connectivity such as FTP connections or web services integration that download/upload files to the external systems are not part of this module and should be encapsulated in other modules or by using the infrastructure capabilities.

For example in a typical flow where external PIM system expects to push data import file to an FTP location it is recommended to install and FTP server and then symlink the location to the incoming directory for auto import listener thereby reducing the complexity and improving the security as we do not need to store any credentials concerning FTP connection.



Import

Auto import job

Import is triggered via an **autoImportJob** which runs on a schedule defined in cron config.

> Using task scheduler SaaS this can be adjusted or triggered manually

The task expects a specific directory structure where data files are placed and a configuration file that allows it to trigger import using a specific import group.

The root of the directory structure is defined by a **JOB_LOCAL_FILE_IMPORT_FS_ROOT** system preference.

Expected directory structure within root:

```
ROOT
     |- SHOP10
     |     |- archived
     |     |- config
     |     |    |- config.properties
     |     |- incoming
     |     |- processed
     |     |- processing
     |
     |- SHOP20
     |     | ...
     | ...
```

The configuration file must contain:

- import group name, which should be used
- regular expression to match the filename in the incoming directory
- reindex flag to indicate whether to run full product re-index after successful completion
- credentials of the manager on which behalf the task should run

Example config:

```
config.0.group=YC DEMO: Initial Data
  config.0.regex=import\\.zip
  config.0.reindex=false
  config.0.user=admin@yes-cart.com
  config.0.pass=1234567
  config.1.group=YC DEMO: IceCat Catalog
  config.1.regex=import\\-EN,DE,UK,RU\\.zip
  config.1.reindex=false
  config.1.user=admin@yes-cart.com
  config.1.pass=1234567
  config.2.group=YC DEMO: Product images (IceCat)
  config.2.regex=import\\-EN,DE,UK,RU\\-img\\.zip
  config.2.reindex=true
  config.2.user=admin@yes-cart.com
  config.2.pass=1234567
```

Above config defines three import groups to import the demo data using admin@yes-cart.com account.

Configurable auto import job SaaS

Import is triggered via an **autoImport2Job** which runs on a schedule defined in cron config.

The difference compared to **autoImportJob** is that this task instead of scanning the configuration files and triggering corresponding imports simply invokes all active RawDataImporter services.

The importers are defined in java "properties" configuration style with importer ID pointing to system preference CODE that contains the properties for specific importer.

For example **JOB_RAW_DATA_IMPORT_CONFIG** may look like:

```
RAWIMPORTER1=JOB_RAW_DATA_RAWIMPORTER1
RAWIMPORTER2=JOB_RAW_DATA_RAWIMPORTER2
...
RAWIMPORTERN=JOB_RAW_DATA_RAWIMPORTERN
```

where RAWIMPORTER1, RAWIMPORTER2 ... RAWIMPORTERN corresponds to a RawDataImporter service ID as it is declared in the platform and JOB_RAW_DATA_RAWIMPORTERX is the system attribute value containing importer specific configurations.

Each raw data importer should be a self contained thread safe unit that is able to perform all its functions based on the configurations in the JOB_RAW_DATA_RAWIMPORTERX attribute that is mapped to it.

RawDataImporter can be defined in any custom module via configuration extension point.

Export

In order to define an export an additional task can be registered in custom module, which would contribute to the schedules extension point.

Generic export group processor SaaS

Generic export group processor is a generic implementation of a scheduled task that uses predefined export group to perform recurring export.

The task can be declared as following:

```
   <!-- 1. Declare task that uses config-xxxxxx.properties -->

   <bean id="xxxExport"
class="org.yes.cart.bulkjob.cron.ConfiguredPausableProcessorWrapperImpl">
       <property name="processor">
           <bean
class="org.yes.cart.bulkjob.impl.GenericExportGroupProcessorImpl">
               <constructor-arg index="0" ref="bulkExportService"/>
               <constructor-arg index="1" ref="ioProviderFactory"/>
               <constructor-arg index="2" ref="authenticationManager"/>
               <property name="configFile"
value="config-xxxxxx.properties"/>
           </bean>
       </property>
```

```
        <property name="systemService" ref="systemService"/>
        <property name="pausePreferenceKey" value="JOB_XXXXX_PAUSE"/>
        <property name="pausePreferenceDefault" value="false"/>
        <property name="cfgContext">
            <bean class="org.yes.cart.config.impl.ConfigurationContextImpl">
                <property name="functionalArea" value="marketing"/>
                <property name="name" value="xxxExport"/>
                <property name="cfgInterface" value="Runnable"/>
                <property name="cfgDefault" value="false"/>
                <property name="properties">
                    <props>
                        <prop key="extension">Pausable cron job</prop>
                        <prop key="description">XXXX export job, configured
in property file under export root (config-xxxxxx.properties)</prop>
                        <prop key="SYSTEM[JOB_XXXXX_PAUSE]">Pause</prop>
                    </props>
                </property>
            </bean>
        </property>
    </bean>

    <!-- 2. Declare trigger -->

    <bean name="xxxExportJob"
class="org.springframework.scheduling.quartz.JobDetailFactoryBean">
        <property name="jobClass"
value="org.yes.cart.bulkjob.cron.YcCronJob" />
        <property name="jobDataAsMap">
            <map>
                <entry key="jobName" value="xxx Auto Export" />
                <entry key="job" value-ref="xxxExport" />
                <entry key="nodeService" value-ref="nodeService" />
            </map>
        </property>
    </bean>

    <bean id="xxxExportJobCronTrigger"
class="org.springframework.scheduling.quartz.CronTriggerFactoryBean">
        <property name="jobDetail" ref="xxxExportJob" />
        <property name="cronExpression" value="${admin.cron.xxxExportJob}"
/>
    </bean>

    <!-- 3. Register with extension -->

    <bean id="xxxExportExtension"
class="org.yes.cart.utils.spring.ArrayListBean">
        <constructor-arg ref="managerCronScheduleTriggers"/>
        <property name="extension">
            <list>
                <ref bean="xxxExportJobCronTrigger" />
```

```
            </list>
        </property>
    </bean>
```

The config file should include:

- export group to trigger
- credentials for user to trigger it on behalf of

```
export.group=XXXX Group
config.user=XXXX
config.pass=XXXX
```

The export descriptor will specify the location where the file to be exported to:

```
<export-descriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:noNamespaceSchemaLocation="http://www.yes-cart.org/s
chema/export-descriptor.xsd">

   <context>
       <shop-code>XXX</shop-code>
   </context>

   <entity-type>org.yes.cart.domain.entity.Product</entity-type>

   <export-file-descriptor>
       <file-encoding>UTF-8</file-encoding>
       <file-name>/home/yc/server/share/resources/2l8YSRX1tTAQq8Ax0YbQIwqo
uYv3mdpq4I81m6vDMWtKED6jwY3orv5q5lY8vvfb_xxx_export.csv</file-name>
       <print-header>true</print-header>
       <column-delimiter>;</column-delimiter>
       <text-qualifier>&amp;quot;</text-qualifier>
   </export-file-descriptor>

...
```

The outcome of this configuration would be product export CSV file generated at the specified location on a cron schedule.