Cookbook - bulk import basics

Overview

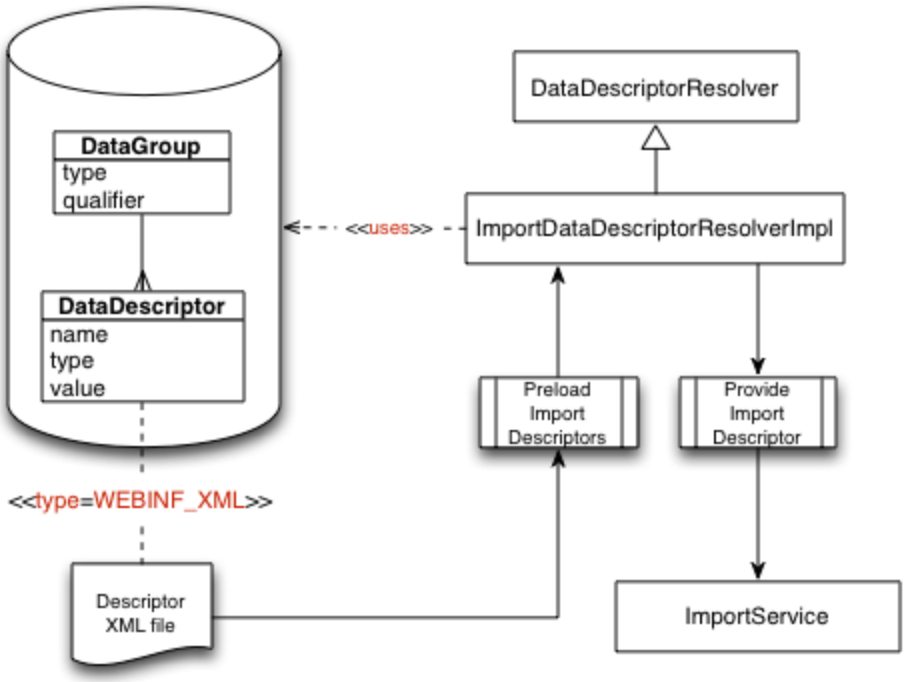
> This guide relies on basic knowledge of Bulk import

Bulk import framework uses Data group and data descriptor objects. Import groups are DataGroup objects of type "IMPORT". Each DataGroup has one or more DataDescriptor objects associated to it. Relationship is many-to-many via DataGroup.descriptors field that specifies a comma separated list of descriptor names. By design the relationship is not enforced to provide flexibility between DataGroup and DataDescriptor. E.g. it is possible to have DataGroup implementation where descriptors are specified within DataGroup.descriptors as XML.

How DataGroup and DataDescriptor are processed is up to DataDescriptorResolver implementors.

Bulk import uses ImportDataDescriptorResolverImpl implementation that uses DataGroup objects of type "IMPORT" and resolves ImportDescriptor objects within given group which are later used by the ImportService.

Figure 1: Import descriptor resolution



Out of the box version specific configuration for import groups is described in detail Manual ImpEx.

It is possible to add custom groups and data descriptors from the Admin to create import and export configurations required. Later these groups can be used in the import and export wizards in the operations section of Admin.

Note that data federation is fully integrated into ImpEx API and therefore all imported and exported records will be validated against the authorisations given to the user that requested import/export operation.

Anatomy to CSV import descriptor

XSD for import descriptor XML file can be found YC_HOME/domain-api/src/main/resources/META-INF/schema/import-descriptor.xsd

Basic import descriptor defines the following sections:

- *Optional* Mode - MERGE (default) or DELETE
- *Optional* Context
- Entity type - to give hints to import service as to what type of import entity we are working with and what data federation rules to apply.
- Import file descriptor - specifies how CSV data is to be processed (i.e. encoding, delimiters, text qualifier)
- Select/Insert/Delete statements - used by import service to manipulate CSV row tuples and look up data in database
- Import columns - describes what columns in CSV file match to what properties in target entity

skuprices.xml import descriptor

```
<import-descriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.yes-cart.org/schema/import-descr
iptor.xsd">
```

```
   <entity-type>org.yes.cart.domain.entity.CarrierSla</entity-type>

   <import-file-descriptor>
       <file-encoding>UTF-8</file-encoding>
       <file-name-mask>carrierslanames(.*).csv(.*)</file-name-mask>
       <ignore-first-line>true</ignore-first-line>
       <column-delimiter>;</column-delimiter>
       <text-qualifier>&amp;quot;</text-qualifier>
   </import-file-descriptor>

   <select-sql>select c from CarrierSlaEntity c where c.name = {name} and
c.carrier.name = {carrier}</select-sql>

   <import-columns>

       <column-descriptor>
           <column-index>0</column-index>
           <field-type>FIELD</field-type>
           <name>name</name>
           <value-regex>(.{0,255})(.*)</value-regex>
           <value-regex-group>1</value-regex-group>
       </column-descriptor>

       <column-descriptor>
           <column-index>1</column-index>
           <field-type>FIELD</field-type>
           <name>displayName</name>
           <language>en</language>
           <value-regex>(.{0,4000})(.*)</value-regex>
           <value-regex-group>1</value-regex-group>
       </column-descriptor>

...

       <column-descriptor>
           <column-index>5</column-index>
           <field-type>FIELD</field-type>
           <name>displayDescription</name>
           <language>en</language>
           <value-regex>(.{0,4000})(.*)</value-regex>
           <value-regex-group>1</value-regex-group>
       </column-descriptor>

...

       <column-descriptor>
           <column-index>10</column-index>
           <field-type>FK_FIELD</field-type>
           <entity-type>org.yes.cart.domain.entity.Carrier</entity-type>
           <name>carrier</name>
           <value-regex>(.{0,255})(.*)</value-regex>
           <value-regex-group>1</value-regex-group>
           <lookup-query>select c from CarrierEntity c where c.name =
```

```
{carrier}</lookup-query>
       </column-descriptor>

       <column-descriptor>
           <column-index>11</column-index>
           <field-type>FIELD</field-type>
           <name>slaType</name>
           <value-regex>(.{1,1})(.*)</value-regex>
           <value-regex-group>1</value-regex-group>
       </column-descriptor>

       <column-descriptor>
           <column-index>12</column-index>
           <field-type>FIELD</field-type>
           <data-type>INT</data-type>
           <name>maxDays</name>
       </column-descriptor>

       <column-descriptor>
           <column-index>13</column-index>
           <field-type>FIELD</field-type>
           <data-type>BOOLEAN</data-type>
           <name>billingAddressNotRequired</name>
       </column-descriptor>

...
```

```
    </import-columns>
</import-descriptor>
```

Queries

Query types

Query section allows to look up existing objects in the system to be updated with data from CSV tuple. In MERGE mode the data is either inserted (if look up query does not return existing object) or updated. For deletions "DELETE" mode must be set in the descriptor.

Query section can contain following queries:

Queries	Syntax	Purpose	Notes
select-sql	HSQL	Look up existing objects to be updated/deleted	
insert-sql	SQL	Native insert to speed up import	Bypasses hibernate processes and thus not eligible for cache evictions or audit tracing
delete-sql	HSQL	Native delete to speed up import	Bypasses hibernate processes and thus not eligible for cache evictions or audit tracing

"insert-sql" and "delete-sql" are optional. When these configurations are omitted standard Hibernate save() and delete() methods are invoked on objects retrieved by the "select-sql".

Query templating

All queries support templating mechanism where values from CSV tuples can be used as part of the query. All that is necessary is to specify placeholders in "select-sql", "insert-sql" or "delete-sql" which is represented by column name enclosed by curly brackets.

SQL template

```
...
   <select-sql>select c from CarrierSlaEntity c where c.name = {name} and
c.carrier.name = {carrier}</select-sql>
...
       <column-descriptor>
           <column-index>0</column-index>
           <field-type>FIELD</field-type>
           <name>name</name>
           <value-regex>(.{0,255})(.*)</value-regex>
           <value-regex-group>1</value-regex-group>
       </column-descriptor>
...
       <column-descriptor>
           <column-index>10</column-index>
           <field-type>FK_FIELD</field-type>
           <entity-type>org.yes.cart.domain.entity.Carrier</entity-type>
           <name>carrier</name>
           <value-regex>(.{0,255})(.*)</value-regex>
           <value-regex-group>1</value-regex-group>
           <lookup-query>select c from CarrierEntity c where c.name =
{carrier}</lookup-query>
       </column-descriptor>
```

In the example above there are two placeholders "name" and "carrier" that will be populated using values in columns 0 and 10 respectively. Note that even though carrier is FK_FIELD the template value is raw value from CSV column and not the CarrierEntity object.

Columns

Field types

Each column under specified index is bound to property on the target entity specified by "name". Binding type is defined by the "field-type" configuration, which will set strategy for processing data in a given column.

field-type	Description
FIELD	Single Value field (Also used as PK if has lookup query to check for update entities).
FK_FIELD	Foreign key field. (Uses look up queries to look up parent objects).
SLAVE_INLINE_FIELD	Defines sub tuple which uses various columns to populate its object. This kind of field uses sub descriptor to define sub tuple columns.
SLAVE_TUPLE_FIELD	Defines sub tuple which is encoded fully inside current field and has no access to other fields. This kind of field uses sub descriptor to define sub tuple columns.

Data types

All CSV data is assumed to be text and to give import descriptor hint for other data types "data-type" property can be specified.

data-type	Description
STRING	java.lang.String

BOOLEAN	java.lang.Boolean
LONG	java.lang.Long
INT	java.lang.Integer
DECIMAL	java.math.BigDecimal
DATETIME	Date value. Default date format is: "yyyy-MM-dd hh:mm:ss". For example: 2014-01-24 16:54:00 is 24th January 2014 16:54

Note that all decimals must be of type java.math.BigDecimal as rounding errors in Double and Float may alter original value. Hence those low precision java types are not supported by the system.

Regular expressions

The values themselves also can be preprocessed in various ways.

The simplest one is to enforce character limit on the column thus avoiding database data size violation errors. This is accomplished by regular expression.

For example to select only up to 255 characters from CSV column a regular expression could be used:

```
<value-regex>(.{0,255})(.*)</value-regex>
<value-regex-group>1</value-regex-group>
```

Group specifies which part to use. So it is possible to create any regular expression and select the required group from the column data to use as import value.

Constant values

To set a constant value "value-constant" tag could be used that will treat value inside the tag as if there was a column in CSV with this value.

```
<value-constant>1</value-constant>
```

Master objects in slave columns

"use-master-object" tag that accepts boolean flag allows to specify that value for given property is current master object. Master objects are available for sub import descriptors from SLAVE_INLINE_FIELD and SLAVE_TUPLE_FIELD.

```
<use-master-object>true</use-master-object>
```

Localisable fields

For display values which can be translated in different language additional "language" hint can be supplied that contains two letter language code.

```
...
        <column-descriptor>
            <column-index>1</column-index>
            <field-type>FIELD</field-type>
            <name>displayName</name>
            <language>en</language>
            <value-regex>(.{0,4000})(.*)</value-regex>
            <value-regex-group>1</value-regex-group>
        </column-descriptor>

        <column-descriptor>
            <column-index>2</column-index>
            <field-type>FIELD</field-type>
            <name>displayName</name>
            <language>ru</language>
            <value-regex>(.{0,4000})(.*)</value-regex>
            <value-regex-group>1</value-regex-group>
        </column-descriptor>

        <column-descriptor>
            <column-index>3</column-index>
            <field-type>FIELD</field-type>
            <name>displayName</name>
            <language>uk</language>
            <value-regex>(.{0,4000})(.*)</value-regex>
            <value-regex-group>1</value-regex-group>
        </column-descriptor>
...
```

Columns 1, 2 and 3 are all mapped to displayName property on the target entity. The "language" tag allows to diversify which column represents which language value.

Sample CSV

Below is an example of skuprices.csv format from the demo import (that corresponds to the above import descriptor). The header line is purely for human reference an is completely ignored by the import process. Textual data is enclosed by quotes (") thus giving opportunity to specify special characters such as new line. The separator used is semicolon (;)

```
sku code;model;shop code;currency;list price;tier
"L2708A#BEA";"Scanjet Enterprise 7000nx Document Capture
Workstation";"SHOIP1";"USD";2124.70;1
"L2708A#BEA";"Scanjet Enterprise 7000nx Document Capture
Workstation";"SHOIP1";"EUR";1730.00;1
"L2708A#BEA";"Scanjet Enterprise 7000nx Document Capture
Workstation";"SHOIP1";"UAH";17205.82;1
"IPICS2GO";"iPics2Go";"SHOIP1";"USD";1411.80;1
"IPICS2GO";"iPics2Go";"SHOIP1";"EUR";1149.53;1
"IPICS2GO";"iPics2Go";"SHOIP1";"UAH";11432.76;1
"6015ES3";"2 Year Extended On-Site Service";"SHOIP1";"USD";1179.14;1
"6015ES3";"2 Year Extended On-Site Service";"SHOIP1";"EUR";960.09;1
"6015ES3";"2 Year Extended On-Site Service";"SHOIP1";"UAH";9548.68;1
"80-106-085";"Universal Tablet Cradle";"SHOIP1";"USD";1374.64;1
"80-106-085";"Universal Tablet Cradle";"SHOIP1";"EUR";1119.28;1
"80-106-085";"Universal Tablet Cradle";"SHOIP1";"UAH";11131.83;1
"WF723A#B1K";"Virtual Rooms (up to 15 people in one meeting)
License";"SHOIP1";"USD";2677.30;1
"WF723A#B1K";"Virtual Rooms (up to 15 people in one meeting)
License";"SHOIP1";"EUR";2179.94;1
"WF723A#B1K";"Virtual Rooms (up to 15 people in one meeting)
License";"SHOIP1";"UAH";21680.78;1
"PLL50E-00V012EN";"Toshiba NB500-10L";"SHOIP1";"USD";689.64;1
```

Anatomy of image import descriptor

Image descriptors are somewhat specialised and contain far less configurations.

brandimages.xml import descriptor

```
<import-descriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="http://www.yes-cart.org/schema/import-descr
iptor.xsd">

   <entity-type>IMAGE</entity-type>

   <import-file-descriptor>
       <file-encoding>UTF-8</file-encoding>
       <file-name-mask>^(.*)_([.[Cookbook bulk import
basics^_]]*)_[a-z](_[a-z]{2})?\.(jpe?g|png)$</file-name-mask>
       <ignore-first-line>false</ignore-first-line>
       <column-delimiter>;</column-delimiter>
       <text-qualifier>&amp;quot;</text-qualifier>
   </import-file-descriptor>

   <select-sql>/imgvault/brand/</select-sql>

   <import-columns>

       <column-descriptor>
           <column-index>0</column-index>
           <field-type>FIELD</field-type>
           <data-type>STRING</data-type>
           <name>description</name>
           <value-constant>This is a brand image import</value-constant>
       </column-descriptor>

   </import-columns>

</import-descriptor>
```

The entity type for image import descriptors is always "IMAGE". Import file descriptor is fairly standard and can be reused.

"select-sql" defined the type of the image data object in this instance "/imgvault/brand/" which is Brand entity.

select-sql	Image entity type
/imgvault/brand/	brands
/imgvault/category/	master catalog categories and content
/imgvault/product/	products and SKU
/imgvault/shop/	shop instance

The column 0 definition is purely for readability purposes and is not used by the image service. All data objects' codes are inferred from the image file name as explained in the bulk import article.

Anatomy of XML import descriptor 3.6.0+

XML import and export build on the same concepts as any other import and export. Descriptor definition specifies the import descriptor resolver that will parse the configurations. Import director service will invoke the XML import or export service to perform the action.

customers.xml XML import

```
<import-descriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="file:../../../../../../../domain-api/src/ma
in/resources/META-INF/schema/import-descriptor-xml.xsd">

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

    <import-file-descriptor>
        <file-encoding>UTF-8</file-encoding>
        <file-name-mask>customers-data.xml</file-name-mask>
    </import-file-descriptor>

    <xml-handler>CUSTOMER</xml-handler>

</import-descriptor>
```

Because XML import and export have a well defined schema the only option available is specifying appropriate handler to use. For exports all handler have "pretty" format by suffixing the name of the handler with "_PRETTY".

customers.xml XML export

```
<export-descriptor xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="file:../../../../../../../domain-api/src/ma
in/resources/META-INF/schema/export-descriptor-xml.xsd">

    <context>
        <shop-code>SHOIP1</shop-code>
    </context>

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

    <export-file-descriptor>
        <file-encoding>UTF-8</file-encoding>
        <file-name>target/customers_export-{timestamp}.xml</file-name>
    </export-file-descriptor>

    <select-cmd>select t from CustomerEntity t</select-cmd>

    <xml-handler>CUSTOMER_PRETTY</xml-handler>

</export-descriptor>
```

Here is the full list of XML descriptor available out of the box:

The following handlers are supported OOTB:

Entity	Import	Export	Notes
Address	ADDRESS,CUSTOMER	ADDRESS,CUSTOMER	
Association	PRODUCT	PRODUCT	Created automatically during product import
Attribute	ATTRIBUTE	ATTRIBUTE	
AttributeGroup	ATTRIBUTEGROUP	ATTRIBUTEGROUP	
AttrValueBrand	BRAND	BRAND	
AttrValueCategory	CATEGORY	CATEGORY	
AttrValueCustomer	CUSTOMER	CUSTOMER	
AttrValueProduct	PRODUCT	PRODUCT	
AttrValueProductSku	PRODUCT,SKU	PRODUCT,SKU	
AttrValueShop	SHOP	SHOP	
AttrValueSystem	SYSTEM	SYSTEM	
Brand	PRODUCT,BRAND	BRAND	Created automatically during product import
Carrier	SHIPPINGPROVIDER	SHIPPINGPROVIDER	
CarrierShop	SHOP,SHOP_CARRIERS	SHOP	
CarrierSla	SHIPPINGMETHOD,SHIPPINGPROVIDER	SHIPPINGMETHOD,SHIPPINGPROVIDER	
Category	CATEGORY,PRODUCT,PRODUCT_CATEGORIES	CATEGORY	Created automatically during product import
Country	COUNTRY	COUNTRY	
Customer	CUSTOMER	CUSTOMER	
CustomerOrder	CUSTOMERORDER	CUSTOMERORDER	Create ONLY for now
CustomerOrderDelivery	CUSTOMERORDER	CUSTOMERORDER	Create ONLY for now
CustomerOrderDeliveryDet	CUSTOMERORDER	CUSTOMERORDER	Create ONLY for now
CustomerOrderDet	CUSTOMERORDER	CUSTOMERORDER	Create ONLY for now
CustomerShop	CUSTOMER	CUSTOMER	
CustomerWishList	CUSTOMER	CUSTOMER	
DataDescriptor	DATADESCRIPTOR	DATADESCRIPTOR	
DataGroup	DATAGROUP	DATAGROUP	
Etype	ETYPE	ETYPE	
Manager	ORGANISATIONUSER	ORGANISATIONUSER	
ManagerRole	ORGANISATIONUSER	ORGANISATIONUSER	
ManagerShop	ORGANISATIONUSER	ORGANISATIONUSER	
ProdTypeAttributeViewGroup	PRODUCTTYPE	PRODUCTTYPE	
Product	PRODUCT	PRODUCT	
ProductAssociation	PRODUCT,PRODUCT_LINKS	PRODUCT	
ProductCategory	PRODUCT,PRODUCT_CATEGORIES	PRODUCT	
ProductEnsembleOption	✗	✗	
ProductSku	PRODUCT,SKU	PRODUCT,SKU	Included in product but can run standalone as SKU
ProductType	PRODUCTTYPE	PRODUCTTYPE	
ProductTypeAttr	PRODUCTTYPE	PRODUCTTYPE	

Promotion	PROMOTION	PROMOTION	
PromotionCoupon	PROMOTION,PROMOTIONCOUPON	PROMOTION,PROMOTIONCOUPON	
PromotionCouponUsage	CUSTOMERORDER	CUSTOMERORDER	Create ONLY for now
Role	❌	❌	
SeoImage	❌	❌	
Shop	SHOP	SHOP	
ShopAlias	SHOP,SHOP_ALIASES	SHOP	
ShopCategory	SHOP,SHOP_CATEGORIES	SHOP	
ShoppingCartState	❌	❌	
ShopUrl	SHOP,SHOP_URLS	SHOP	
ShopWarehouse	SHOP,SHOP_CENTRES	SHOP	
SkuPrice	PRICE	PRICE	
SkuPriceRule	PRICERULE	PRICERULE	
SkuWarehouse	INVENTORY	INVENTORY	
State	COUNTRYSTATE	COUNTRYSTATE	
System	❌	❌	
Tax	TAX	TAX	
TaxConfig	TAXCONFIG	TAXCONFIG	
Warehouse	FULFILMENTCENTRE	FULFILMENTCENTRE	