Items.xml

What is Items.xml

The **items.xml** is a file which specifies types(like Customer,Product etc) of an extension.
By editing the **items.xml** file, you can define new types or extend existing types.
In addition, you can define, override, and extend attributes in the same way.

Location

The **items.xml** is located in the **resources** directory of an extension. The **items.xml** files are prefixed with the name of their respective extension in the form of **extension name-items.xml**. For example:

- For the **core** extension, the file is called **core-items.xml**.
- For the **catalog** extension, the file is called **catalog-items.xml**.

Structure

The **items.xml** defines the types for an extension in XML format.

Basic Structure

The basic structure of an **items.xml** file is as follows:

items.xml

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

<atomictypes>
Defines the list of AtomicType's for your extension.
</atomictypes>

<collectiontypes>
Defines the list of CollectionType's for your extension.
</collectiontypes>

<enumtypes>
Defines the list of EnumerationType's for your extension.
</enumtypes>

<maptypes>
Defines the list of MapType's for your extension.
```

</maptypes>

<relations>
Defines the list of RelationType's for your extension.
</relations>

<itemtypes>
Defines the list of ComposedType's for your extension.
</itemtypes>

</items>

As the items.xml file is validated against an XSD file (items.xsd), the order of type definitions must conform to the order given above.

Explanation of each items.xml Element

Atomic Types

<atomictypes>
Defines the list of AtomicType's for your extension.

<atomictype
An AtomicType represents a simple java object. (The name 'atomic' just means 'non-composed' objects.)

class=*"class type"*
(Corresponding Java class in the hybris Suite; will also be used as the code of the atomic type.)

autocreate=*"true"*
(If 'true', the AtomicType will be created during initialization.)

generate=*"false"*
Deprecated. Has no effect for atomic types. Default is 'true'.

extends =”class type”
(Defines the class which will be extended. Default is 'java.lang.Object'.)/>

```
</atomictypes>
```

Collection Types

```
<collectiontypes>
  <collectiontype
code="codeType"
(The code (that is, qualifier) of the CollectionType.)

elementtype="codeType"
The type of elements of this CollectionType.

autocreate="boolean"
If 'true', the CollectionType will be created during initialization.

generate="boolean"
Deprecated. Has no effect for collection types. Default is 'true'.

type="list"
Configures the type of this collection: 'set', 'list', 'collection'.
/>
```

enumTypes Type :

```
<enumtypes>

<enumtype code="codeType"

autocreate="true" generate="true" dynamic="true"/>
```

Whether it is possible to add new values by runtime. Also results in different types of enums: 'true' results in 'classic' hybris enums, 'false' results in Java enums. Default is false. Both kinds of enums are API compatible, and switching between enum types is possible by running a system update.
</enumtypes>

enumtypes
An EnumerationType defines fixed value types. (The typesystem provides item enumeration only)

enumtype The unique code of this Enumeration.

MapType Type :

```
<maptypes>
<maptype code="ExampleMap" argumenttype="Language"

    returntype="java.math.BigInteger" autocreate="true" generate="false"  />

<maptypes>
```

maptypes
Specifies a list of map types.

maptype
  (Like the java collection framework, a type, which defines map objects. Attention: When used as type for an attribute, the attribute will not be searchable and the access performance is not effective. Consider to use a relation.)

relationType Type :

it defines the relation between two types(types means tables)

```
<relations>
<relation

code="codeType"


localized="boolean"
A localized n-m relation can have a link between two items for each language.

generate="boolean"
autocreate="boolean"
 deployment="deploymentRefType">
```

```
it creates the physical table in a database.if we don't give deployment table ,it does
not store physically


<sourceElement type="AbstractOrder"

   qualifier="order"
cardinality="one">
   <modifiers read="true" write="true"
search="true" optional="true"/>
 </sourceElement>

<targetElement
type="AbstractOrderEntry" qualifier="entries"
cardinality="many"
collectiontype="list" ordered="false">

<modifiers read="true" write="true" search="true" optional="true" partof="true"/>
            </targetElement>
        </relation>
```

relations: Defines a list of relation types.

Relation: A RelationType defines a n-m or 1-n relation between types.

Cardinality : this is the relation between two tables like 1-many,many-many.

SourceElement :
sourece element means the attribute which are going to store in the other table

ItemType Type : it creates new ItemType in your extension.

Structure

```

<typeGroup>

<itemtype code="codetype"
                  extends="any codeType"
                  jaloclass=""
                  deployment="package namme to generate"
                  autocreate="boolean"
```

```
                generate="boolean" abstract="boolean">

 <attributes>
     <attribute autocreate="boolean" qualifier="attribute name"
type="java.lang.String">
                  <modifiers read="true" write="true" search="boolean"
optional="boolean"/>
                  <persistence type="property"/>
              </attribute>
          </attributes>
</itemtype></typeGroup>
```

typeGroup : Defines the name of this group. Only for structural purpose, will have no effect on runtime. Default is empty.

Itemtype :Specifies a specific ComposedType. It creates new item type.

Attribute : it creates attribute in a itemType.

Persistence : it has 4 types. Jalo,property,dynamic.

Jalo : Jalo is depricated.

Property :means it creates persistant type in db.

Dynamic : if attribute is dynamic we can do appened operation on dynamic at run time

Finally Items.xml is used to create types of a extension i,e it creates a model class and a physical table on the database. By using that model class we can perform db operations.

Task 1:
Aim:Create an attribute to an existing item Type.

Here i am creating a defult address attribute to an existing itemType.

Step 1:

Copy the Customer itemType from the core-items.xml and paste it in your Extension name-items.xml.

Step 2:

Create an attribute in your items.xml

Note :In your items.xml ‘autocrate’ should be false in <typecode> tag because if it is true,it will create a new table in the database instead of creating extra attribute to the existing table.

Step 3:

Then type the command “ant build” in the conole and start the hybris server.

Step 4:

Now go to hac->platform,then update the running system.then the created attribute in items.xml will be available to the database.

Code Level

```
<itemtype code="Customer"
                 extends="User"
                 jaloclass="de.hybris.platform.jalo.user.Customer"
                 autocreate="false"
                 generate="true">
            <attributes>
                <!--  auto ID which is generated by NumberSeries -->
                <attribute autocreate="true" qualifier="defaultAddress"
type="java.lang.String">
                    <modifiers read="true" write="true" search="true"
optional="true"/>
                    <persistence type="property"/>
                </attribute>
            </attributes>
        </itemtype>
```

- goto command prompt
- ant build
- Start the server
- open hac
- goto platform
- update the running system only.

Task 2 :

Aim : Create a new itemType

Here i am creating new itemType(i,e new table in the db)AnotherAddress.

```
<itemtype code="AnotherAddress"
              extends="GenericItem"
              jaloclass="com.lycamobile.jalo.address.AnotherAddress"
              autocreate="true"
              generate="true">
          <attributes>

        <attribute autocreate="true" qualifier="street" type="java.lang.String">
                  <modifiers read="true" write="true" search="true"
optional="true"/>
                  <persistence type="property"/>
              </attribute>


<attribute autocreate="true" qualifier="lane" type="java.lang.String">
                  <modifiers read="true" write="true" search="true"
optional="true"/>
                  <persistence type="property"/>
              </attribute>
```

```
<attribute autocreate="true" qualifier="city" type="java.lang.String">
                  <modifiers read="true" write="true" search="true"
optional="true"/>
                  <persistence type="property"/>
              </attribute>



 </attributes>
       </itemtype>
```

- ant build
- start the server
- open hac
- go to platform
- update the running system.

Task 3 :

Aim : create relation between two itemTypes(one – many or many – many)

Here i am creating one-manyb relation between customer and AnotherAddress.i,e one customer has many address.

So here the customer attribute mapped with every address in the customer table.

The below code creates new relation '*Customer2AnotherAddress*' between two itemTypes Customer and AnotherAddress.

```
<relations>
  <relation code="Customer2AnotherAddress" localized="false"
generate="true" autocreate="true">
          <sourceElement type="Customer" qualifier="order"
cardinality="one">
             <modifiers read="true" write="true" search="true"
optional="true"/>
             <custom-properties>
                 <property name="ordering.attribute">
                     <value>"entryNumber"</value>
                 </property>
             </custom-properties>
          </sourceElement>
          <targetElement type="AnotherAddress" qualifier="entries"
cardinality="many" collectiontype="list"
                        ordered="false">
             <modifiers read="true" write="true" search="true" optional="true"
partof="true"/>
          </targetElement>
      </relation>
```

- ant build
- start server
- go to platform
- update running system.

In above relation i did not give any deployement table.so it does not create any physical table in the db.