XSD Notes:

The order of not optional elements (XSD <–>XML) is important when XSD evaluates XML.

**Implementation steps:**

1. XML to cover the use cases.
2. **XSD** for created XML (to add the new “standalone” processes like Export)
3. Check if the XML match XSD by applying the parser in **ProcessService**
4. **ModelObject** for XSD (DataforwartT, exportT etc.)

These \*T auto generated classes are just getter/setter classes. To represent and work with all these Dataforwarding objects a ModelObject XML should be created in:

**de.kisters.belvis.model.xml 🡪 modeldefinition\_processfwk.xml**

All the properties from XSD elements should be referenced here

1. After the XML file is created, then modify

**de.kisters.belvis.model.xml 🡪 generate-sources.xml**

for procfwk section

run pom.xml as “maven generate sources” 🡪 the generated classes a created in

**de.kisters.belvis.model** 🡪target/generated-sources 🡪

de.kisters.belvis.model.gen.procfwk

**To make t these generated Classes available for the server and client:**

Import bundle packages to use these generated model objects (like Dataforward etc.):

\*.**common** means the Objects are needed on the Client AND the Server side!

The generated ModelObjects from **de.kisters.belvis.model** are packed by maven to

the repository and downloaded into **de.kisters.bundle.belvis.common\javalibs\kisters\belvismodel.jar**

**🡪**

**de.kisters.bundle.belvis.common** exports **de.kisters.belvis.model.gen.procfwk**

**de.kisters.bundle.belvis.server.base** imports **de.kisters.belvis.model.gen.procfwk**

1. The next step is to implement converters (two way):

**XML 🡪 XML \*T Classes 🡪 Internal Model Object**

**processDefinition.xml 🡪 DataforwardT 🡪 Dataforward**

The Dataforward etc. are used to represent and evaluate the DF objects in GUI

With help of UIAdatpter and are the network objects for Client <-> Server transfer

After Editing (Editor/Wizard) the DF objects should be saved in the DB as XML

So the converter **Dataforward 🡪DataforwardT 🡪 processDefinition.xml**

Is also needed.

**ProcFWKConverterService** (interface IConverterService) is needed to use the

according converter

(s. BDIConverterService)

This service has to be written to **BelvisBaseService.xml** and is

Needed as reference section by **ProcessService**

**ProcessService** calls the **ProcFWKConverterService** to use the Converters

**DataforwardConverter Dataforward 🡪 DataforwardT**

**XMLDataforwardConverter: DataforwardT 🡪 Dataforward**

7. **ProcessService**: parseXML, importProcess() modify to handle different types of processes

8. To open editor : register the application type for process type (e.g. Export) in

**de.kisters.plugin.belvis.common.plugin.xml** 🡪 Extensions 🡪 **de.kisters.plugin.common.extension.cluster.clusterConfiguration**

**9**: To save the changes 🡪 register save support/load support for process :

**de.kisters.plugin.belvis.common.plugin.xml** 🡪 Extensions 🡪 **de.kisters.plugin.common.accessConfiguration (s. Dataforward,**

1. message: “nothing to save” 🡪 **the vEdit & vOriginal in ModelObjectEditor are identical** 🡪 the working copy was not created 🡪

Check if the Process type includes the InstanceFilter and implement the createWorkingCopy() method (s. Dataforward/Export UI Adaptors)