|  |
| --- |
| LoanPath Naming and Coding Standards |
| Owner : Trantor Software Doc ID : Trantor/ADD/1.1 |
| This document describes the Coding Standards and naming conventions. It is written for a highly technical audience, and specifically for people who are involved in the coding, architecture, design, and implementation of our product. Currently, it covers standards, and not applications (e.g., order management). |

**Document Control**

|  |  |
| --- | --- |
| Title | Description |
| Document Description | LoanPath Naming and Coding Standards |
| Document Location | http://subversion/repos/loanpath-piedmont-app/trunk/docs/LoanPath%20Naming%20and%20Coding%20Standards.docx |
| Review Period |  |

**Document Creation Details**

|  |  |  |
| --- | --- | --- |
|  | **Created by** | **Reviewed by** |
| Name | Tejinder Singh | ThirdPillar |
| Date | 28/08/2013 | 30/09/2013 |

**Distribution List**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Name** | **Designation** |
| 1 | All PMs of ThirdPillar. |  |

**Confidentiality Agreement**

*This document is copyrighted and all rights are reserved. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from an authorized representative of Company.*

*This document is for internal use only and may not, in whole or in part, be provided to anyone outside of Company, including customer, clients, or prospects.*

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Date | Revision  No. | Summary of Changes | Author |
| 28/08/2011 | 1.0 | Initial version | Tejinder Singh |
| 24/10/2011 | 1.1 | Incorporated more Standards | Tejinder Singh |
| 12/12/2014 | 1.2 | Add the way to call JavaMethodBizop from command button and added the way to manage attribute choice. | Manik Goyal |
| 20/01/2015 | 1.3 | Added new feature to use custom sequence of drop down values. | Sajan Monga |
| 20/01/2015 | 1.4 | How to hide Task tab for specific role category | Ankush Bhardwaj |
| 29/01/2015 | 1.5 | Confirmation message on command button. | Sajan Monga |
| 06/02/2015 | 1.6 | Call integration service. | Manik Goyal |
| 09/02/2015 | 1.7 | Validation | Manik Goyal |

Contents

[Revision History 3](#_Toc411253038)

[Universal Practices 6](#_Toc411253039)

[Meta-Model Entity Level 7](#_Toc411253040)

[UI Meta-Model Level 9](#_Toc411253041)

[Security Level 9](#_Toc411253042)

[Integration Level 11](#_Toc411253043)

[BPMN Level 14](#_Toc411253044)

[Drools Rules Level 14](#_Toc411253045)

[Seed Data Level 15](#_Toc411253046)

[Validations 16](#_Toc411253047)

# Universal Practices

1. For any project. We should create a single XLS sheet having all Questions and Review comments going back n forth. It should house Questions for BA, Codify and UI in separate sub-sheets. This sheet should be attached to Jira. A sample XLS sheet is attached below.
2. There should be a local server where builds will be deployed for review before commit to SVN. The builds will be pushed through Hudson.
3. After the code is committed. One should check out the code in separate folder other than the one in which development was done. A new build should be created out of new folder and sanity tested. After sanity test is complete, only then build should be refreshed on JBOSS server.
4. All commits to SVN will be first reviewed by Lead first.
5. Proper indentation in code should be there. Like below.:

If(){

//some code

}else{

//some code

}

for(){

//some code

}

1. If mvel is used then utilize power of Mvel rather than coding in Java.
2. Never delete someone’s work. If you are going to add, then also you should add only the things which are asked to be added. Like recently Engagement workflow was added although not needed.
3. If there is requirement to show/hide fields in page then always give preference to do it through BusinessPermissions XML. If it cannot be done through BusinessPermission only then one should use UiPermissions XML.
4. Whenever something is committed to SVN. Always mention JIRA ID and description and comments in below fashion:

[JER-XX] – JIRA Description

-- Comment

1. For all projects which are in production(Do not apply to LP4-LP5 and codify team) we need to try to make individual commit per JIRA. This will help us in branching and merging and allow only those JIRA’s in production which client asks to be deployed. So we can selectively allow or skip JIRAs we want.
2. S

# Meta-Model Entity Level

## 

1. When we create derived field then name it as [derived+”OriginalEntity”]. Like suppose you are creating derived entity for “partnerName” field. Then name field as “derivedPartnerName”.
2. Outline Fields shared by BA should match Wireframe. If not, raise issue with BA.
3. If there are fields like CreatedBy, CreatedDate, lastUpdatedDate, lastUpdatedBy. These fields are usually needed by every table for Audit and are present out of box for every table with Metafied feature. Use it. Don’t create new fields for it.
4. Take due care when Copy Pasting. Suppose you are copying something from Entity1 to Entity2. Then fields names having Entity1 should all be changed to Entity2.
5. Filters at field level are defined to show what values in dropdown or lookup for that field need to be shown. Like below filter will only show Sales Dov and Credit Doc in dropdown whenever new generated document is created.:

<reference name=*"documentClass"* entity-type=*"AttributeChoice"* relation=*"refers-to"* label=*"Document Group"* required=*"false"* >

<lookup>

<param name=*"key"*>DOCUMENT\_CLASS</param>

</lookup>

<filters>

<filter triggerExpr=*"isdef root\_documentGroup\_generatedDocuments"* expr=*"if(isdef root\_documentGroup\_generatedDocuments){ entity.key == ('DOCUMENT\_CLASS\_CREDIT')||entity.key == ('DOCUMENT\_CLASS\_SALES')} "*/>

</filters>

</reference>

1. If you want to filter one dropdown depending upon another dropdown then use groupMvel like below. But keep in mind that you are creating it at entity level and so it will be done at all places. You cant modify it at UI meta-model afterwards.:

<reference name=*"documentType"* entity-type=*"AttributeChoice"* relation=*"refers-to"* label=*"Document Type"* required=*"false"* ruleTriggerOnEvent=*"true"*>

<lookup>

<param name=*"key"*>DOCUMENT\_TYPE</param>

<param name=*"groupMvel"*><![CDATA[if(thiz.documentCategory != null) { return thiz.documentCategory.key; } else { return null; }]]></param>

</lookup>

</reference>

1. Whenever we define one-to-one relationship between two entities. The relation from Parent to child gets created. But back-reference relationship from child to parent does not get created at database level. One should always create that relation through Drools Rules. One example is given below where Parent entity is Engagement and child is

**rule** "SetEngagementToDueDil"

**agenda-group** "SAVE"

**when**

eng:Engagement()

**then**

**if**(eng!=**null** && eng.getDueDiligenceDetails()!=**null** ){ **if**(eng.getDueDiligenceDetails().getEngagement()==**null**){

eng.getDueDiligenceDetails().setEngagement(eng);

}

}**else**{

DueDiligence duedil =**new** DueDiligence();

duedil.setEngagement(eng);

eng.setDueDiligenceDetails(duedil);

}

**end**

1. While associating any collection to entity, use referredBy and referTo. Like for relation between “Attributes” and “AttributeChoice”. In Attribute it is: <collection name=*"attributeChoices"* collection-type=*"list"* entity-type=*"AttributeChoice"* referred-by=*"attribute"* relation=*"owner"*/>. And in AttributeChoice entity it is: <reference name=*"attribute"* entity-type=*"Attribute"* relation=*"refers-to"*/>
2. Always try not to use required=”true” at entity meta-model level. Try to make fields required through ui-meta-model. Because the same field can then will not be able to be used at not required at any other place.
3. If you find that there are some fields that are repeating in many entities then make a common embeddable entity and use it as feature like:

<feature operation=*"implements"*>

<data>com.thirdpillar.foundation.model.Metafied</data>

</feature>

1. If you are trying to put custom sequence in drop down values then you have to do only 2 things:

First change is in metamodel - put “Ordered” feature in “AttributeChoice” entity like

<feature operation=*"implements"*>

<data>com.thirdpillar.foundation.model.Ordered</data>

</feature>

Second is add <sequence>seq\_no</sequence> tag in <attributeChoice>

seq\_no=1 would be first element in drop down and so on.

**<attributeChoice>**

**<value>Other</value>**

**<key>BUSINESS\_TYPE\_OTHER</key>**

**<sequence>101</sequence>**

**<attribute reference="../../.."/>**

**</attributeChoice>**

1. To show confirmation message on command button two changes has to be done.

1. Add new attribute confirmation-key in command button like

<command name="Submit to Credit" action="SubmitToCredit" type="businessOp" confirmation-key=”eApplication.submitToCredit.confirm” />

2. Make entry in help\_message.propertiesfile for key “eApplication.submitToCredit.confirm” like

eAApplication.submitToCredit.confirm=STOP.Are you sure you want to submit?

1. H
2. I
3. J
4. K
5. L

# UI Meta-Model Level

There should not be any Special Characters on Lookup Keys when you are defining Lookups under attributes.xml.

1. Do all your developments in attributes.xml. attributes\_unsorted.xml is for storing OOB application specific attributes, never add or delete anything into it.
2. Always try to put labels in crud. Usually codify does generate labels, but just in some scenarios it might not work well.
3. The consistency of fields should be maintained. Like recently panel Id was mentioned as individualInformation. Although rest were named like vendor.IndividualInformation. So never do that. If you have question on it then ask. Like we can ask to create a standard if two panels have same name.
4. If some fields don’t make sense to you then raise question with BA.
5. Don’t specify Type in CRUD where not needed. Like <field name="vendor.venderParent" type="SearchNSelect" /> is wrong.
6. Dsds
7. xasda

# Security Level

Article: How to hide Task tab for specific role category.

Step 1: Get the *ui\_labels.properties* files from the following location:

*\web\target\loanpath-barcelona-web\WEB-INF\classes\ui\_labels.properties*

(*This file is auto-generated by codify)*

In this files you will see the all labels & their mapping. For task tab we have

*TaskView=Tasks*

Where TaskView is the ID & Tasks is the label.

Step2: Create \service\src\main\resources\permissions\task-uiPermissions.xml file with content :

------------------------------------------------------------------------------------------------------------

<tns:ui-permissions xmlns:tns=*"http://www.thirdpillar.com/codify/ui-permissions"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.thirdpillar.com/codify/ui-permissions ui-permissions.xsd"*>

<tns:component id=*"taskView"*>

<tns:read>ctx.hasAnyRole('ROLE\_CATEGORY\_AUDITOR','ROLE\_CATEGORY\_OPERATIONS','ROLE\_CATEGORY\_OPERATIONS\_TASKS','ROLE\_CATEGORY\_SUPER\_ADMINISTRATOR')</tns:read>

</tns:component>

</tns:ui-permissions>

You can put the condition as per your project requirement.

Step3 : Put the entry of task-uiPermissions.xml under :

*\service\src\main\resources\applicationContext.xml*

Step 4: How to set the default page for particular role category:

Go to \*web\src\main\resources\codify-app-web.properties* file

In this file you can align the default pages either by direct seam or by variable:

-----------------------------------------------------------------

# Default homeUrl

default.homeUrl=/search/request/main/results.seam

default.homeUrl.ROLE\_CATEGORY\_CREDIT\_ADMINISTRATOR=${default.homeUrl}

default.homeUrl.ROLE\_CATEGORY\_USER\_MANAGEMENT=/search/user/main/results.seam

In this way you can hide/show tabs even panels/fields as well.

# Integration Level

1. If you want to call any integration service. Let’s take Generated Document example. You first have to define Command for trigger button at crud level like:

<command id="Generate\_Generated\_Document" name="Generate" action="GenGeneratedDocument" type="businessOp"/>

Then define which BPMN event to call for the command like :

<bean name=*"com.thirdpillar.codify.loanpath.model.Document.GenGeneratedDocument"* class=*"com.thirdpillar.foundation.service.WorkflowEventBizOp"*>

<property name=*"eventName"* value=*"GenGeneratedDocument"*/>

</bean>

Then create new BPMN like DocumentFlow.bpmn to handle signal event “GenGeneratedDocument” and register it at rule-set.xml. You may also like to create DucumentRules.drl for handling some rules associated and resister it at default-rule-set.xml. The Bpmn will call “Integration Task” component of BPMN and in “Service Name” you will input integration service name defined in integrationServiceConfig.xml. For our purpose it will be “docGenClient”.

1. If you want to call java Business Operation by click button from crud.
   1. Create a command button for trigger at crud level like :

<command name="Upload Excel" action="CallExcel" type="businessOp"/>

* 1. Then which java Method bizops to call like :

<bean name=*"com.thirdpillar.codify.loanpath.model.SurveyGroup.CallExcel"* class=*"com.thirdpillar.foundation.service.JavaMethodBizOp"* >

<property name=*"hostingClassName"* value=*"com.thirdpillar.codify.loanpath.service.ExcelUtil"*/>

</bean>

* 1. Then create a class in package as (*com.thirdpillar.codify.loanpath.service) and register a method to trigger from click of button.*

**public** **class** ExcelUtil {

**public** **void** CallExcel(BaseDataObject entity,

HashMap<Long, BaseDataObject> map) {}

}

**Agenda**

1. How to call integration Service from BPMN?
2. How to Marshall and Un-Marshall Object using XStream and customize XML element using profile?

Implementation in the JBPM

1. To call integration service from BPMN needs to basic knowledge of jbpm.
   1. Drag integration service component from the service task tab.
   2. Create Map object and put necessary objects which need to Marshall into XML further during integration call.

Map<String,Object> inputMap = **new** HashMap<String,Object>();

inputMap.put("contextEntity",contextEntity);

inputMap.put("listPaperNotification",listPaperNotification);

context.setParameter(“inputMap”,inputMap);

* 1. Map parameters in the parameter mapping column.

Entity =contextEntity;

Input=inputMap;

* 1. Put the service name in the ServiceName Column.

1. Browse to integrationServiceConfig in the Web Folder.
   1. Create a service as follow

<service name=*"SubmitToCredit"* enabled=*"true"*>

<marshaler style=*"XSTREAM"* resource=*"Full"* inputKey=*"eAApplicationReq"* outputKey=*"eAppRequestXml"*/>

<transformer resource=*"/xsl/addModelNamespace.xsl"* inputKey=*"eAppRequestXml"* outputKey=*"eAppRequestModXml"*/>

<soaptransport resource=*"${loanpath4.url}/beacon-core/services/CreateCreditApplicationService"* responseType=*"PAYLOAD"* validateResponse=*"false"* readTimeout=*"300000"* soapAction=*"Dummy"*>

<operation localName=*"list"*>

<parameters>

<parameter paramValue=*"eAppRequestModXml"* paramType=*"Element"*/>

</parameters>

</operation>

</soaptransport>

<transformer resource=*"/integration/xsl/submitToCreditResponse.xsl"*/>

<unmarshaler style=*"XSTREAM"* resource=*"Full-No-Binary"* contextLookupKey=*"eAApplicationReq"* />

</service>

* + 1. <service name=*"SubmitToCredit"* enabled=*"true"*> : Enter Service Name.
    2. <marshaler style=*"XSTREAM"* resource=*"Full"* inputKey=*"* listPaperNotification *"* outputKey=*"* lPNotification *"*/> : Now Pass the entity from the Map which we have created in the JBPM. listPaperNotification will be converted in XML using Xtream and saved in the outputkey with profile as “full”. We can create our own customize profile as per requirement. Profile is used to filter the xml elements.
    3. <soaptransport resource=*"${loanpath4.url}/beacon-core/services/CreateCreditApplicationService"* responseType=*"PAYLOAD"* validateResponse=*"false"* readTimeout=*"300000"* soapAction=*"Dummy"*>: Soap Envelope Body where we pass url as resource to trigger the integration call. We can pass authentication parameters: username and password as per service requirements.
    4. <parameters>

<parameter paramValue=*"* lPNotification*"* paramType=*"Element"*/>

</parameters> : Now pass the outputkey object to make the XML format as per client requirement. We can add header elements as well in the parameters.

* + 1. <transformer resource=*"/integration/xsl/submitToCreditResponse.xsl"*/> : this is used to convert the marshalled and un-marshalled object as per client requirement by using XSLT.
    2. <unmarshaler style=*"XSTREAM"* resource=*"Full-No-Binary"* contextLookupKey=*"eAApplicationReq"* /> : It is to store the response object and convert the XML object java object(un-marshalling).

<!-------How to create a Profile will be added later --------->

# BPMN Level

# Drools Rules Level

1. If rule is setting some value then it should be named like:

Set<Entity Name><Entities Field Name>At<AgendaGroup or action>

1. Constant
2. In BPMN, you should use a logger as follows –

RuleFunctions.logDebug(context, "Decision by Approver  " + decisionByApprover + " on approver " + app.getId());

1. In DRL, you can you use a logger as follows –

import function com.thirdpillar.foundation.rules.RuleFunctions.logDebug;

logDebug(drools, "setting the status as Saved for UCC here");

1. Always try to use less costly operations for performance. Like I you want to know if document has been uploaded try to use size for uploaded documents to check whether its uploaded or not as per below code shared by Anil. You can use below code to use size:

boolean valid= true;

if(o !=null && o.getDocumentGroup()!=null && o.getDocumentGroup().getUploadedDocuments()!=null){

foreach(document : o.getDocumentGroup().getUploadedDocuments()){

if(document.getDocumentType().getValue().equals("Engagaement Overview Pitch (EOP)")){

if(document.getUploadedStream() document.getUploadedStreamSize()==null){

valid = false;

} }}}

return valid;

1. Always use keys when determining dropdown values. Values are meant to be changed. But keys are meant to be static. So always use Keys in the workflow design.
2. For validations of Tasks. Please use component-Validation tasks for the same
3. While defining Constraints in constraints-[Entity].xml try to use field validations, not class level ones. Like:

<map:getter name="promotionType">

<map:constraint annotation="javax.validation.constraints.NotNull">

<map:message>{product.eAApplication.promotionType.required}</map:message>

</map:constraint>

</map:getter>

1. F
2. A
3. S
4. d

# Seed Data Level

1. Don’t delete neither modify the key. You can move it to the Deleted Choice attribute as follows.

<attributeChoice>

<value>Healthcare</value>

<key>BUSINESS\_TYPE\_HEALTHCARE</key>

<attribute>

<key>DELETED </key>

</attribute>

</attributeChoice>

# Validations

**Agenda**

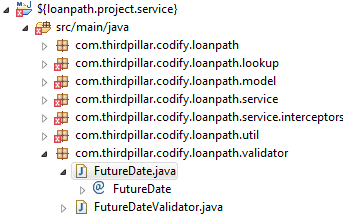
1. **How to create Custom hibernate Constraint (Annotation).**

**Requirement:** User needs to select a date not to be a future date including current date also from a calendar and hibernate validator provides only **Past** (Includes current as past), **Future** (Excludes current date) constraints.

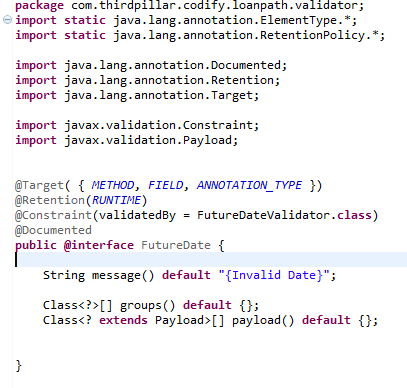
But user needs all Future Dates including current also.

**So according to requirement, we need to create a custom constraint and following major three steps are required to create custom hibernate constraint:**

1. Create a constraint annotation.
2. Create a constraint validation.
3. Define a default error message.
4. Firstly we have created a packages under service named as ”validator” and created two java files named as FutureDate.java as constraint annotation and FutureDateValidator as constraint validator ( **given in below image** ):

****

1. **Constraint annotation:** Firstly create a annotation interface as given in below code :(**FutureDate.java**)

****

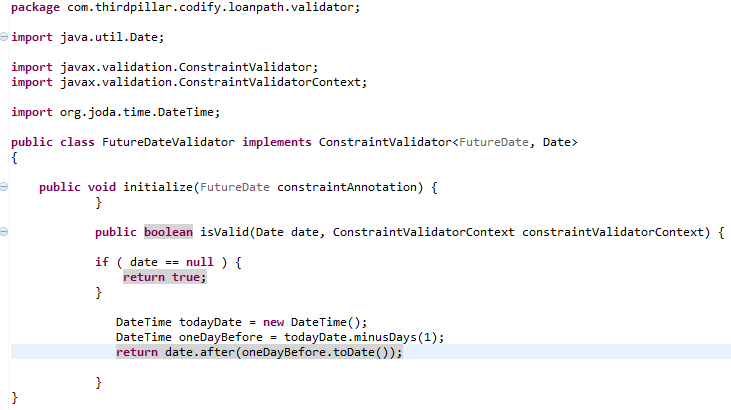
The annotation interface has three mandatory attributes: **message, groups and payload .**

Following is the attributes’ brief description:

* 1. @Target({ METHOD, FIELD, ANNOTATION\_TYPE }): means that methods, fields and annotation declarations may be annotated with **@FutureDate** (but not type declarations e.g.) .
  2. @Retention(RUNTIME): means that annotations of FutureDate type will be available at runtime by the means of reflection.
  3. @Constraint(validatedBy = FutureDateValidator.**class**): means the validator to be used to validate elements annotated with **@FutureDate .**
  4. @Documented: Says, that the use of **@FutureDate** will be contained in the JavaDoc of elements annotated with it.
  5. **message attribute** that returns the default key for creating error messages in case the constraint is violated.
  6. **groups attribute** that allows the specification of validation groups, to which this constraint belongs and This must default to an empty array of type Class<?>.
  7. **payload attribute** that can be used by clients of the Bean Validation API to asign custom payload objects to a constraint. This attribute is not used by the API itself.

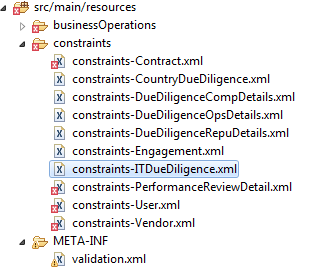
**3) Constraint Validator**

**N**ext, we created as Constraint Validator named as “FutureDateValidator“ and need to implement a constraint validator.(**given in below image code)**:



1. Create constraint xml file .According to “FutureDate” scenario and we have created below given files [As per VMS-OOB project requirement all these files are created].
2. constraints-CountryDueDiligence.xml
3. constraints-DueDiligenceCompDetails.xml
4. constraints-DueDiligenceOpsDetails.xml
5. constraints-DueDiligenceRepuDetails.xml
6. constraints-ITDueDiligence.xml

And added these resources in validation.xml file.

****

Now say in **“constraints-CountryDueDiligence.xml”** add the below code to call @*FutureDate :*

<map:constraint-mappings xsi:schemaLocation=*"http://jboss.org/xml/ns/javax/validation/mapping http://www.jboss.org/xml/ns/javax/validation/mapping/validation-mapping-1.0.xsd"* xmlns:map=*"http://jboss.org/xml/ns/javax/validation/mapping"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*>

<map:default-package>com.thirdpillar.codify.loanpath.model</map:default-package>

<map:bean class=*"CountryDueDiligence"* ignore-annotations=*"false"*>

<map:class>

</map:class>

<map:field name=*"completedDate"*>

<map:constraint annotation=*"com.thirdpillar.codify.loanpath.validator.FutureDate"*>

<map:message>{countryDueDiligence.completedDate.futureDate}

</map:message>

</map:constraint>

</map:field>

</map:bean>

</map:constraint-mappings>

In validation.xml :

<validation-config

xmlns=*"http://jboss.org/xml/ns/javax/validation/configuration"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://jboss.org/xml/ns/javax/validation/configuration"*>

<default-provider>org.hibernate.validator.HibernateValidator</default-provider>

<message-interpolator>org.hibernate.validator.messageinterpolation.ResourceBundleMessageInterpolator

</message-interpolator>

<traversable-resolver>org.hibernate.validator.engine.resolver.DefaultTraversableResolver</traversable-resolver>

<constraint-validator-factory>org.hibernate.validator.engine.ConstraintVali datorFactoryImpl

</constraint-validator-factory>

<constraint-mapping>/constraints/constraints-DueDiligenceOpsDetails.xml

</constraint-mapping>

<constraint-mapping>/constraints/constraints-DueDiligenceCompDetails.xml

</constraint-mapping>

<constraint-mapping>/constraints/constraints-ITDueDiligence.xml

</constraint-mapping>

<constraint-mapping>/constraints/constraints-CountryDueDiligence.xml

</constraint-mapping>

<constraint-mapping>/constraints/constraints-DueDiligenceRepuDetails.xml

</constraint-mapping>

</validation-config>

***Note: When we have implemented these functionality “validation.xml” was the required resource, but in updated Codify “validation.xml” is removed and we need not to add constraints resources in “validation.xml’ file***