PTC® the product development company

WC10 Customizing-2

Eric Kim(yckim@ptc.com)

Info Page customization

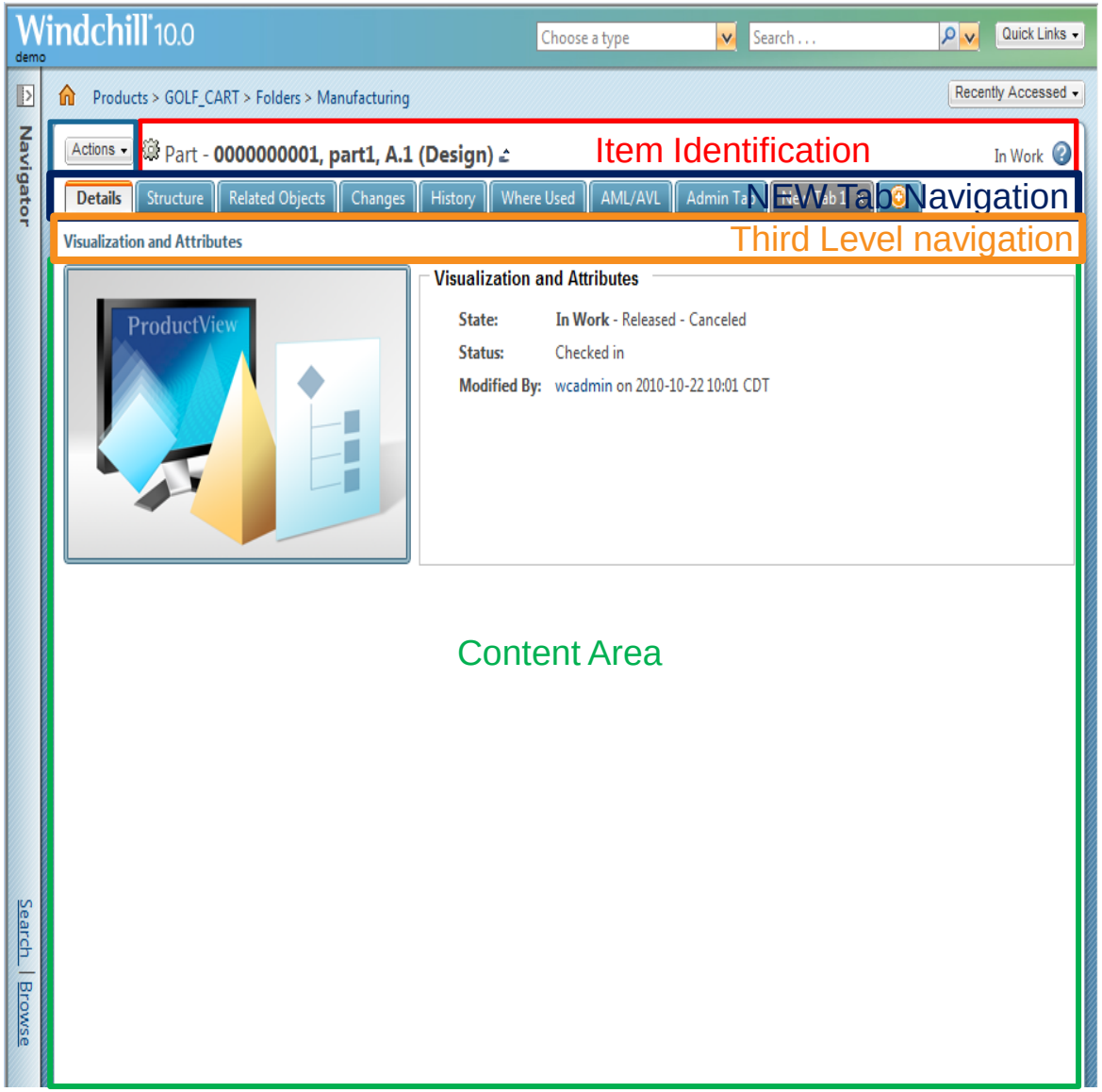
Agenda

1. Introduction Info Page
2. Architecture of custom information page
3. Add custom third navigation on Info Page
4. Excise

1.1 Understanding Info Page

When implementing info pages, following sections must be designed for describing on UI.

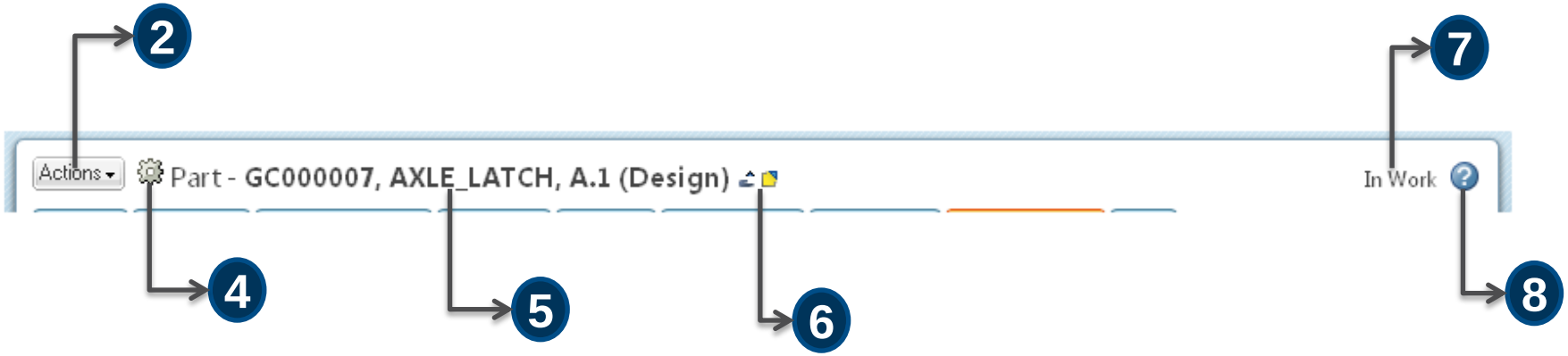
- Action menu
- Item Identification
- New Tab navigation
  - New tab navigation includes the third level navigation menu.
- Third level navigation
  - Third level navigation includes contents for displaying on content area.
- Content Area
  - Attributes and Visualization are now just content like any other content and are not always visible on the page, they are generally available on the details tab.



1.2 Title Bar Area

Title Bar Area is consisted of several configurations like following

1. Go to latest link (only appears on info pages for non-latest versions)
2. Actions Menu
3. Commonspace/Workspace toggle
4. Object Type icon
5. Item Identification
6. Status Glyphs (checkout, share, pending changes, etc)
7. State of item (only appears for LifeCycleManaged items)
8. Help icon



1.3 Tab Navigation

- There are three types of tabs
- Out of the Box.
  - End User can't edit content of the tabs
  - End User can't remove the tabs
  - End User can't rename the tabs
  - Defined as an action model in XML
  - Content is also defined in action XML
- Admin Created
  - End User can't add/remove content in tabs
  - End User can't remove the tabs
  - End User can't rename the tabs
  - End User can move content within the tab
  - Available to the entire site or org
  - Defined through the UI
  - Stored in the DB
- User Created
  - End User can edit content freely add/remove/move
  - End User can add and remove tabs freely
  - Defined through the UI
  - Stored in the DB
- Action to Add Tabs
- The user can reorder all three types
- Command line tool for transferring tab configures from test to production

1.4 Ready for Info page design

How to control for each part of Information page

Before creating Info page, you must ready some configuration because many parts of UI design for info page are separated by setting up action and action model.

- Action & Action Model
  - Action Menu (If it didn't set for itself, it will automatically set parent object's action menu)
  - New tab navigation
  - Third level navigation
- MVC component or JSP
  - Content area
- Information Page component
  - Item Identification

1.5 JCA Debug

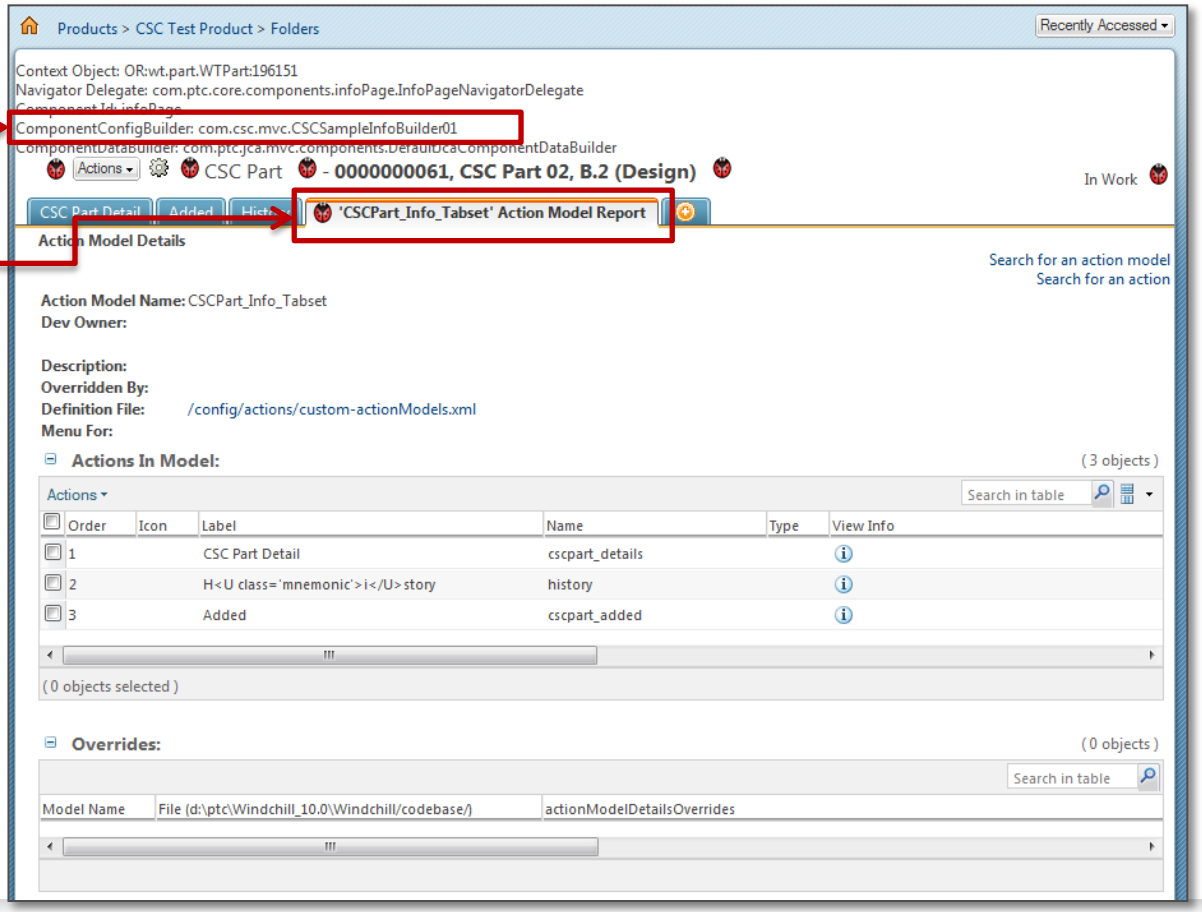
Look action and action model report

JCA Debug in info page gives us more useful action and action model reports.

- Can know Info page builder class
- Can know action model report for Tab and Third level content configuration

Info page builder class

Action model report



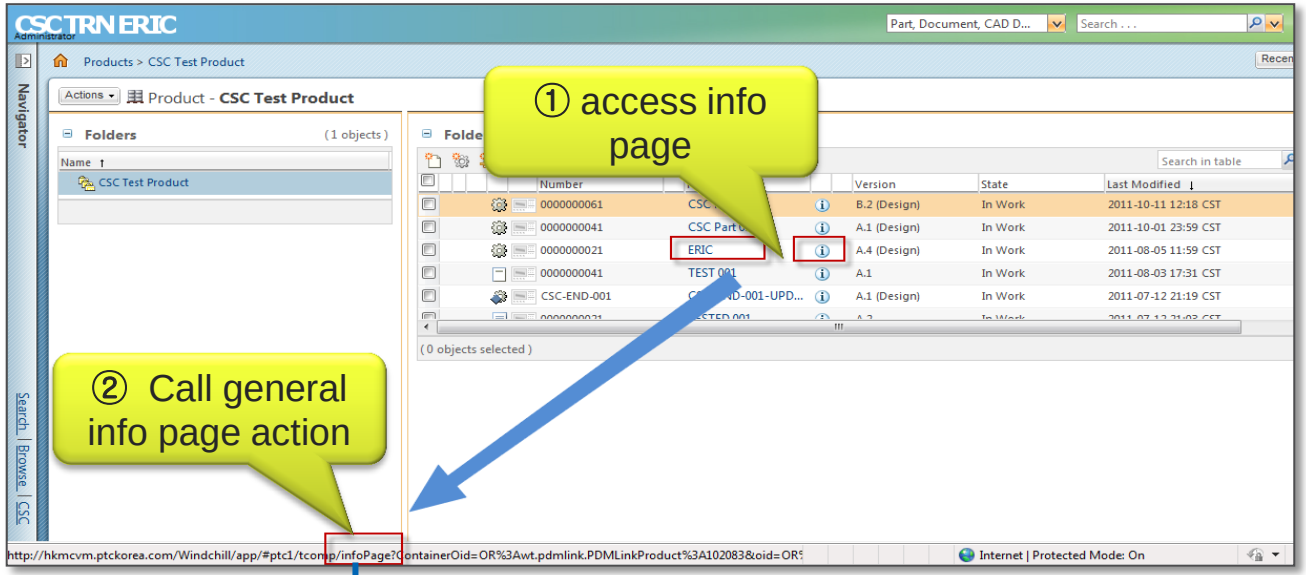
Agenda

1. Introduction Info Page
2. Architecture of custom information page
3. Add custom third navigation on Info Page
4. Excise

Architecture of custom information page

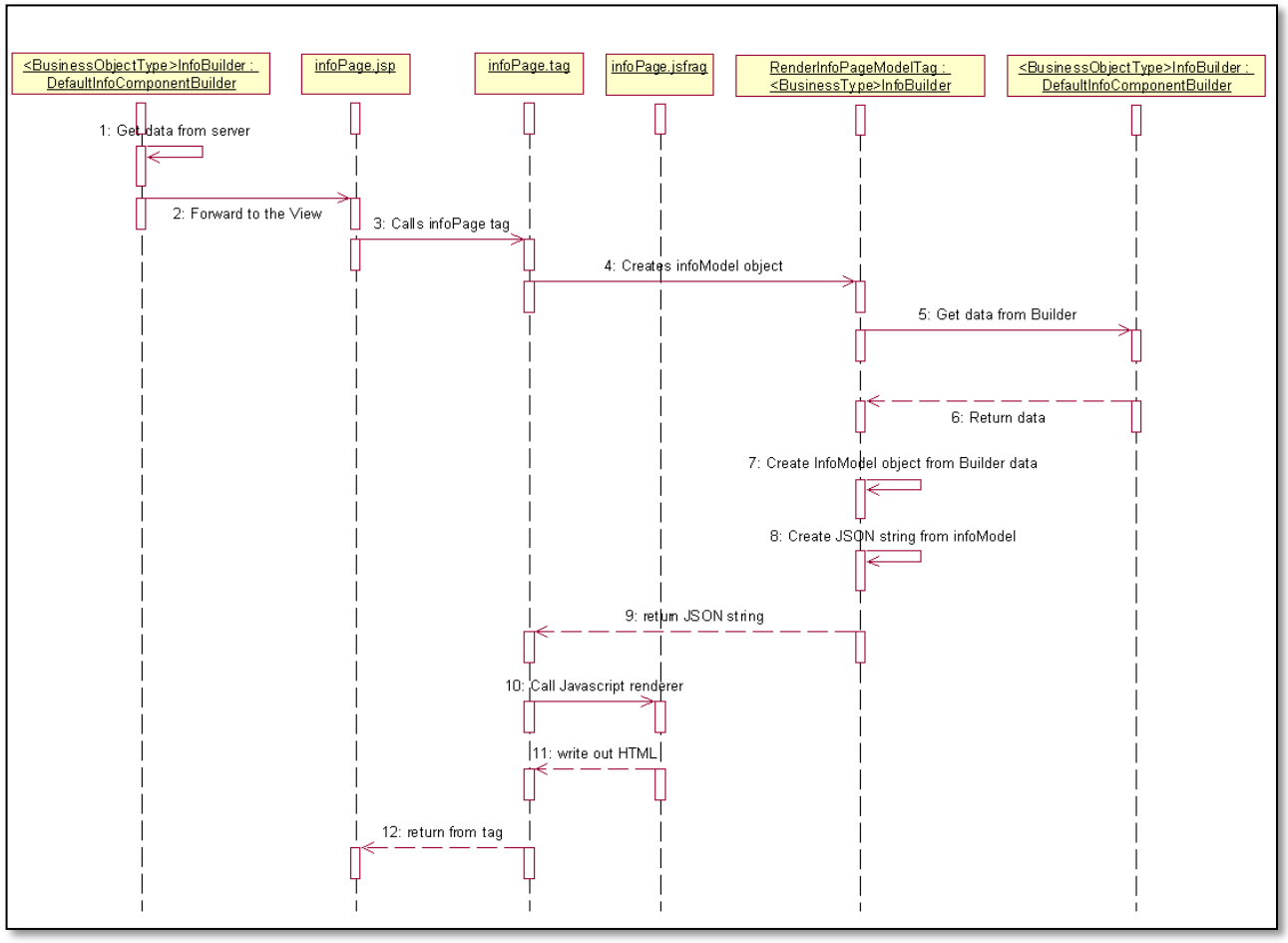
2.1 Progress of calling Info page

Info page will be called from info page action in UI (OOTB) generally. In this time, System has been progressing to find correct info page action. Finally Windchill 10 architecture start to find registered MVC class for targeted object which is displaying info page.



2.2 Information Page Process Flow

Following is the process sequence for rendering the info page.



The builder is called first to collect the necessary data from the server. The builder forwards the request to the view (i.e. infoPage.jsp).

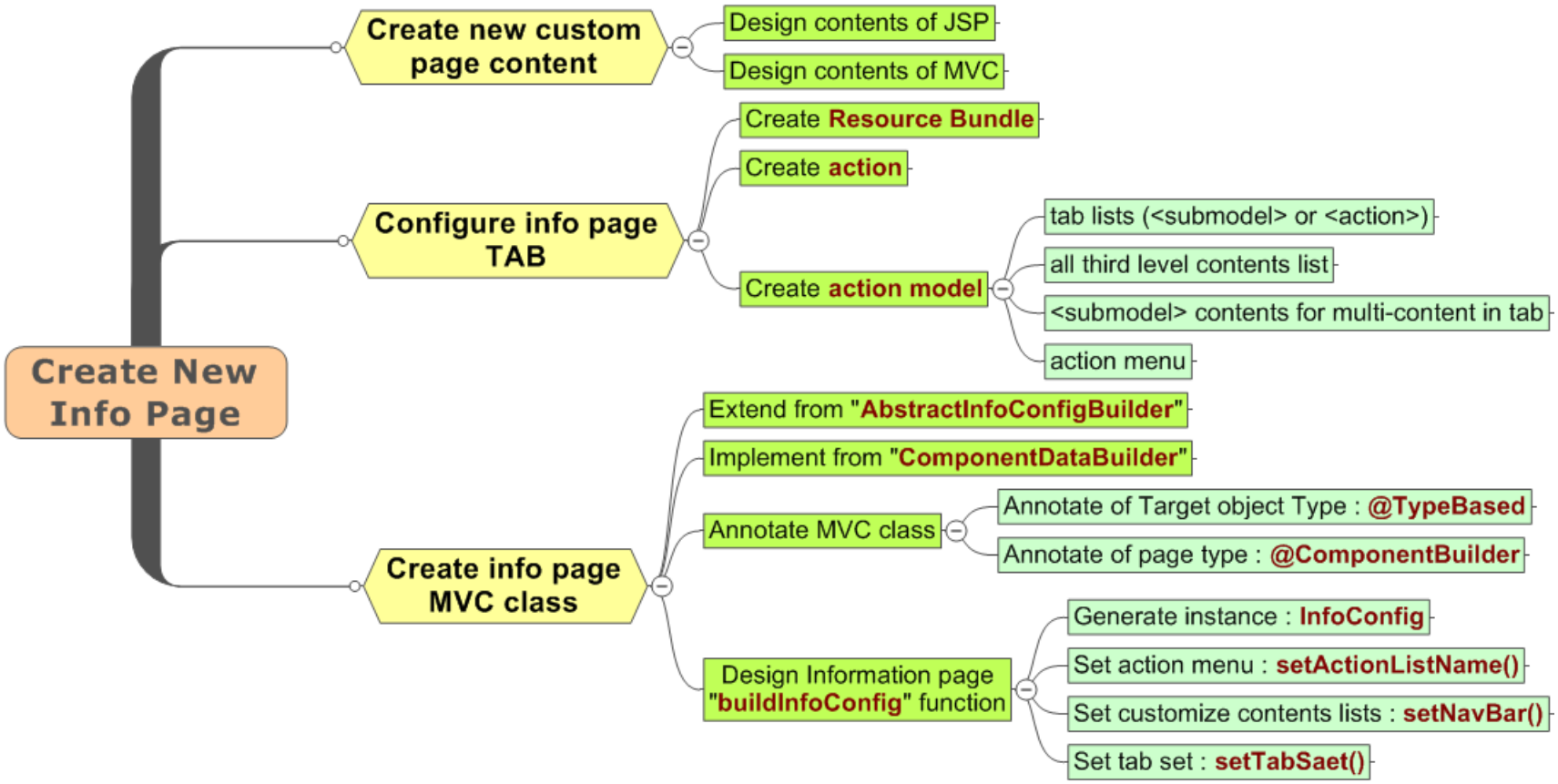
The default view, infoPage.jsp, includes the infoPage tag.

The tag creates a JSON object, known as the infoModel object, from the data it collected from the builder.

The infoModel is serialized (by writing it out as JSON string to the JSP writer) and passed to the JavaScript render which ultimately generates the HTML.

2.3 Contents of configuration lists

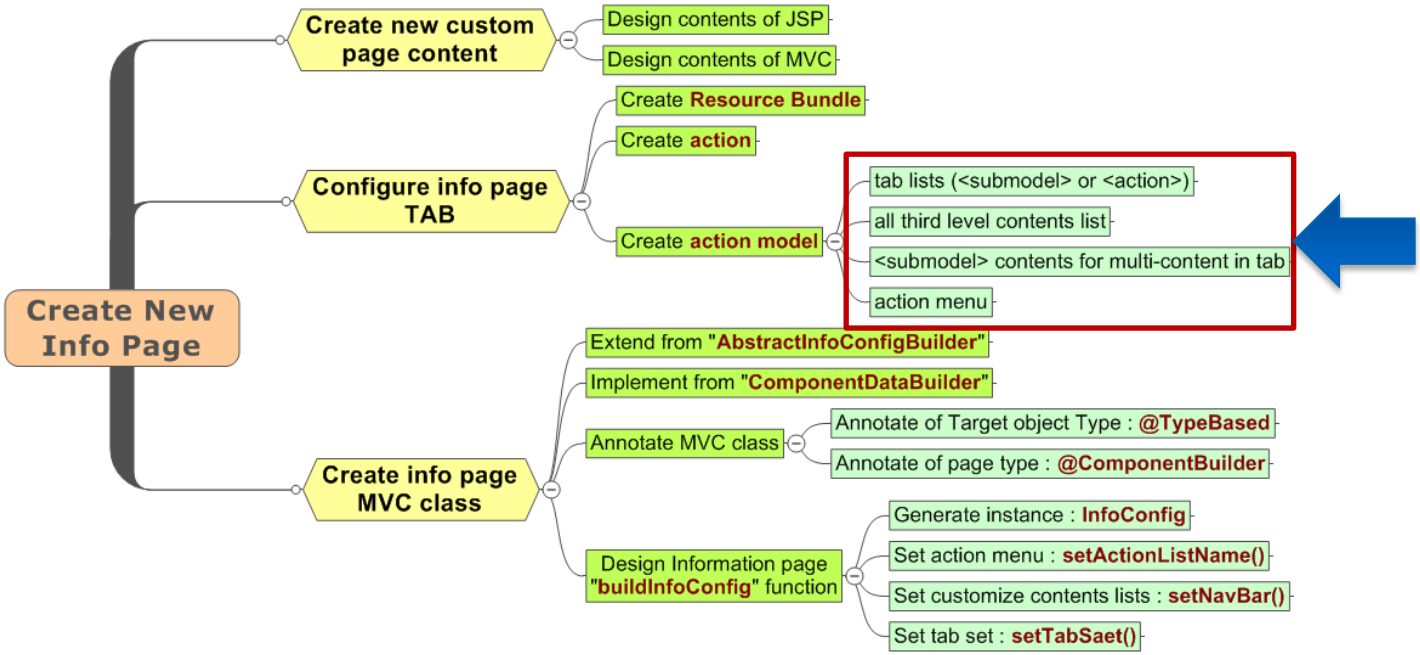
Information page is needed to be ready for many kinds of configuration for creating new pages. First of all, you must understand all related configuration.



2.4 Introduction of action model for information page

You have to set three group of action model for information page.

- TAB list
- Third level content lists (the lists of customized button)
- Action menu (optional. If you do not define, information page will use parent object's action menu)
- Submodel group (If you use <submodel> in TAB list or Third level contents)



Architecture of custom information page

2.5 Description of Action model

```
<!-------- CUSTOM MENU LIST -------->
<model name="third_level_nav_part">
      <submodel name="general"/>                    <!-- General -->
      <submodel name="relatedItems"/>               <!-- Related Objects -->
      <submodel name="changes"/>                    <!-- Change -->
      <submodel name="history"/>                    <!-- History -->
      <submodel name="collaboration"/>              <!-- Collaboration -->
      <submodel name="prodAnalytics"/>              <!-- Product Analytics -->
</model>

<!-------- ACTION MODEL FOR TAB -------->
<model name="partInfoPageTabSet" resource="com.ptc.core.ui.navigationRB">
     <submodel name="partInfoDetailsTab"/>
     <action name="productStructureGWT" type="psb"/>
     <submodel name="partInfoRelatedItemsTab"/>
     <submodel name="changesTab"/>
     <submodel name="partInfoHistoryTab"/>
     <submodel name="partInfoWhereUsedTab"/>
     <submodel name="requirementTraceabilityTab" />
     <submodel name="amlAvlTab" />
     <submodel name="prodAnalyticsTab" />
</model>

<!-------- submodel Design if Tab set and third nav lists have submodel -------->
<model name="partInfoDetailsTab" resourceBundle="com.ptc.core.ui.navigationRB">
        <action name="visualizationAndAttributes" type="object"/>
        <action name="attributes" type="object"/>
</model>
```

The content of tab can be used one <action>. If you want to show several content in tab you should use <submodel>.

Existed in

Some action must be included in custom menu list. If the action isn't included in custom menu list, the action will not be show in a body of info page.

Custom Menu List – This list will be shown in the customization menu.

Action model for Tab – This is the design of TAB.

Submodel – if "Custom menu list" and "Tab design" have sub model you must design for each of sub model

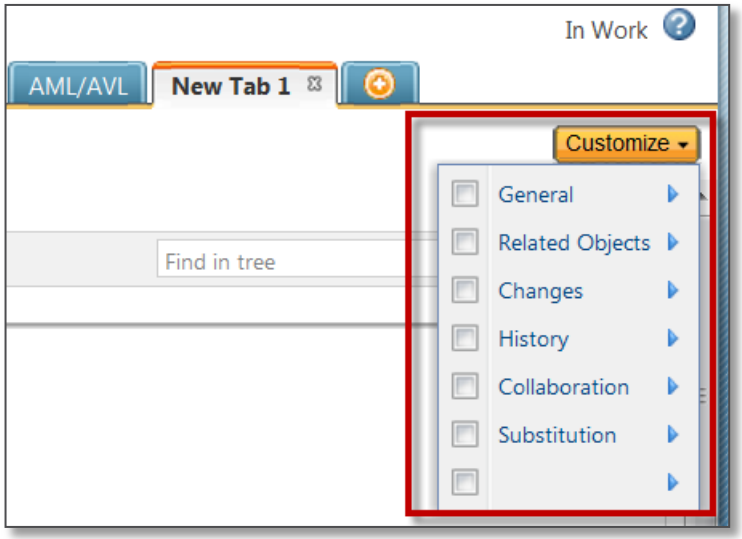
Architecture of custom information page

2.5 Description of Action model (Custom menu list)

```
<!-------- CUSTOM MENU LIST -------->
<model name="third_level_nav_part">
      <submodel name="general"/>                    <!-- General -->
      <submodel name="relatedItems"/>               <!-- Related Objects -->
      <submodel name="changes"/>                    <!-- Change -->
      <submodel name="history"/>                    <!-- History -->
      <submodel name="collaboration"/>              <!-- Collaboration -->
      <submodel name="prodAnalytics"/>              <!-- Product Analytics -->
</model>
```

1. **The Design of Custom Menu List** is for showing customize menu.
2. The action model can use <action> and <submodel> tag.
3. If you are using <submodel> tag, you must design submodel block.
4. All of action must be defined in the action.xml

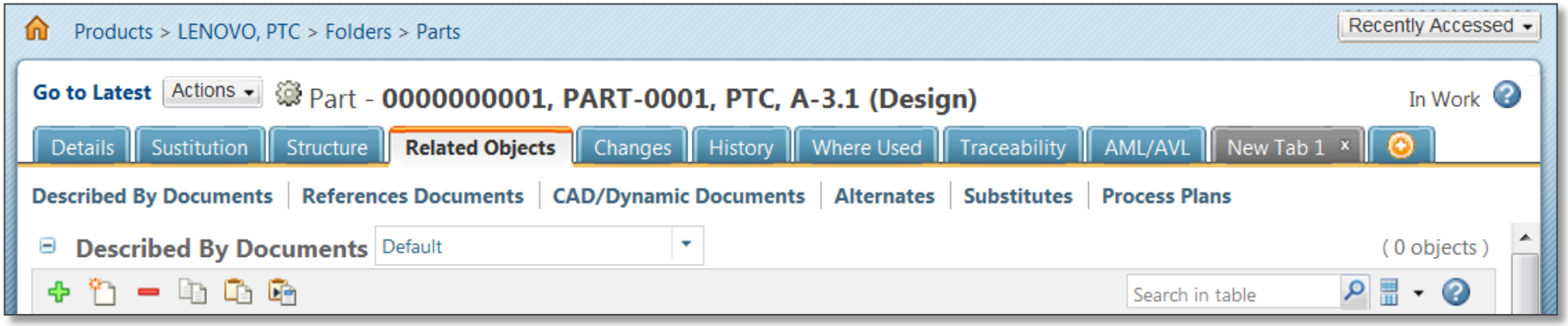
Some tab content which is <action> will be not displayed although it was included for tab design. In this case, the action will be existed in custom menu list. (Issue of tab design)



Architecture of custom information page

2.5 Description of Action model (Tab design)

```
<!-------- ACTION MODEL FOR TAB -------->
<model name="partInfoPageTabSet" resource="com.ptc.core.ui.navigationRB">
     <submodel name="partInfoDetailsTab"/>
     <action name="productStructureGWT" type="psb"/>
     <submodel name="partInfoRelatedItemsTab"/>
     <submodel name="changesTab"/>
     <submodel name="partInfoHistoryTab"/>
     <submodel name="partInfoWhereUsedTab"/>
     <submodel name="requirementTraceabilityTab" />
     <submodel name="amlAvlTab" />
     <submodel name="prodAnalyticsTab" />
</model>
```



1. **Tab design** – This list will be shown for tab list on info page.
2. The action model can use <action> and <submodel> tag.
3. If you are using <submodel> tag, you must design submodel block.
4. The content action of <submodel> will be show third level content list below of target tab.
5. All of action must be defined in the action.xml

Architecture of custom information page

2.6 Review custom MVC class of info page

Building info page is needed to use other Abstract Class component comparing table and tree.

1) Extend from "**AbstractInfoConfigBuilder**" abstract class
2) Implements from "**ComponentDataBuilder**" interface class
3) Register target object to system using annotation – **@TypeBased(value = "$[CLASS_TYPE_NAME]")**
4) Register info page id using annotation - **@ComponentBuilder(value = ComponentId.*INFOPAGE_ID)***
5) Override "**buildComponentData**" function for getting instance of target object
6) Override "**buildInfoConfig**" function for building info pages. (Set **InfoConfig** instance for UI)

```
1  package com.csc.mvc;
2
3  import wt.part.build.WTPartBuildHelper;
18
19 @TypeBased(value = "wt.part.WTPart|com.ptckorea.hkmcvm.CSCPart")
20 @ComponentBuilder(value = ComponentId.INFOPAGE_ID)
21 public class CSCSampleInfoBuilder01 extends AbstractInfoConfigBuilder implements ComponentDataBuilder {
22
23     @Override
24     public Object buildComponentData(ComponentConfig arg0, ComponentParams arg1) throws Exception {
25         return arg1.getContextObject();
26     }
27
28     @Override
29     protected InfoConfig buildInfoConfig(ComponentParams arg0) throws WTException {
30         InfoComponentConfigFactory factory = getComponentConfigFactory();
31         InfoConfig result = factory.newInfoConfig();
32
33         result.setNavBarName("cacpart_third_level_nav");
34         result.setTabSet("CSCPart_Info_Tabset");
35
36         return result;
37
38     }
39 }
```

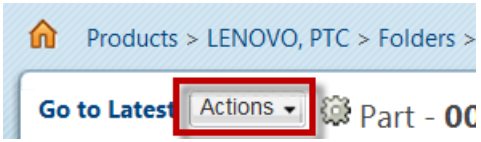
Architecture of custom information page

2.7 MVC function

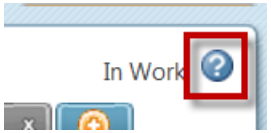
Following is a general function list on the InfoConfig for designing of info page.

1) setNavBarName() : required
2) setTabSet() : required
3) setActionList()
4) setHelpContext()
5) setView()
6) addComponent()

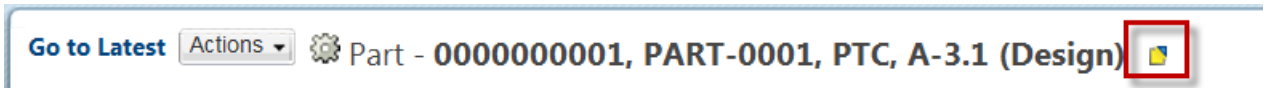
- **setNavBarName()** – Set the action model configuration of customize menu list.
- **setTabSet()** – Set the action model configuration of tab list design.
- **setActionList()** – Set action menu set for context object of information page.



- **setHelpContext()** – Set help context of context object.



- **setView()** – If you have some special display or configuration for info page, you can make JSP for view and set the JSP. The JSP file must be located in "$WT_HOME/codebase/WEB-INF/jsp"
- **setComponent()** – If you want to add some especial component in identification line, set this function.



Agenda

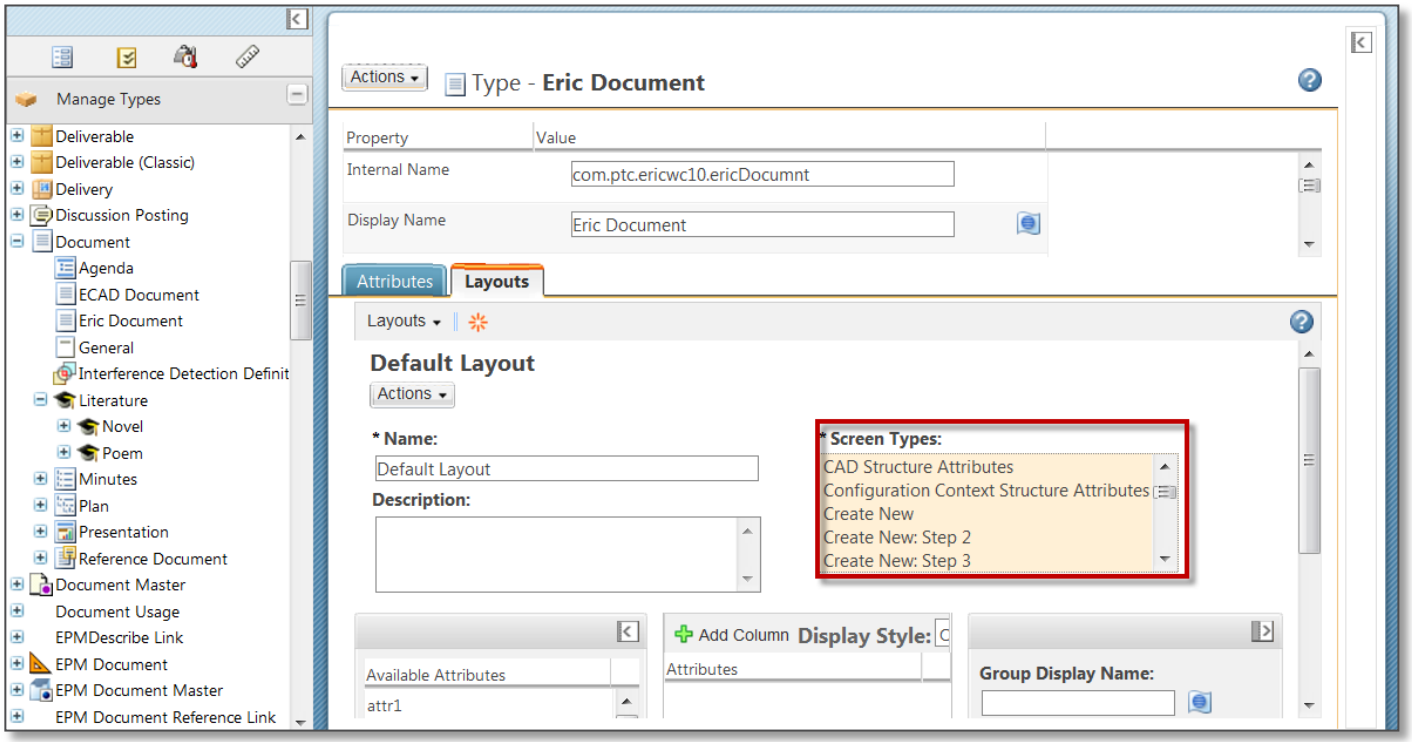
1. Introduction Info Page
2. Architecture of custom information page
3. Add custom third navigation on Info Page
4. Excise

Architecture of custom information page

3.1 Understanding of layout

Windchill 10 has new design architecture using layout in type management. By that architecture, you don't need especial page design for several complex design sets. Windchill OOTB action had been prepared action set for each of screen type template like following.

- Show Primary/More attributes (Detail tab of info page)
- Input attributes set (Create/Edit wizard page)
- Etc (Several UI pages are using layout)

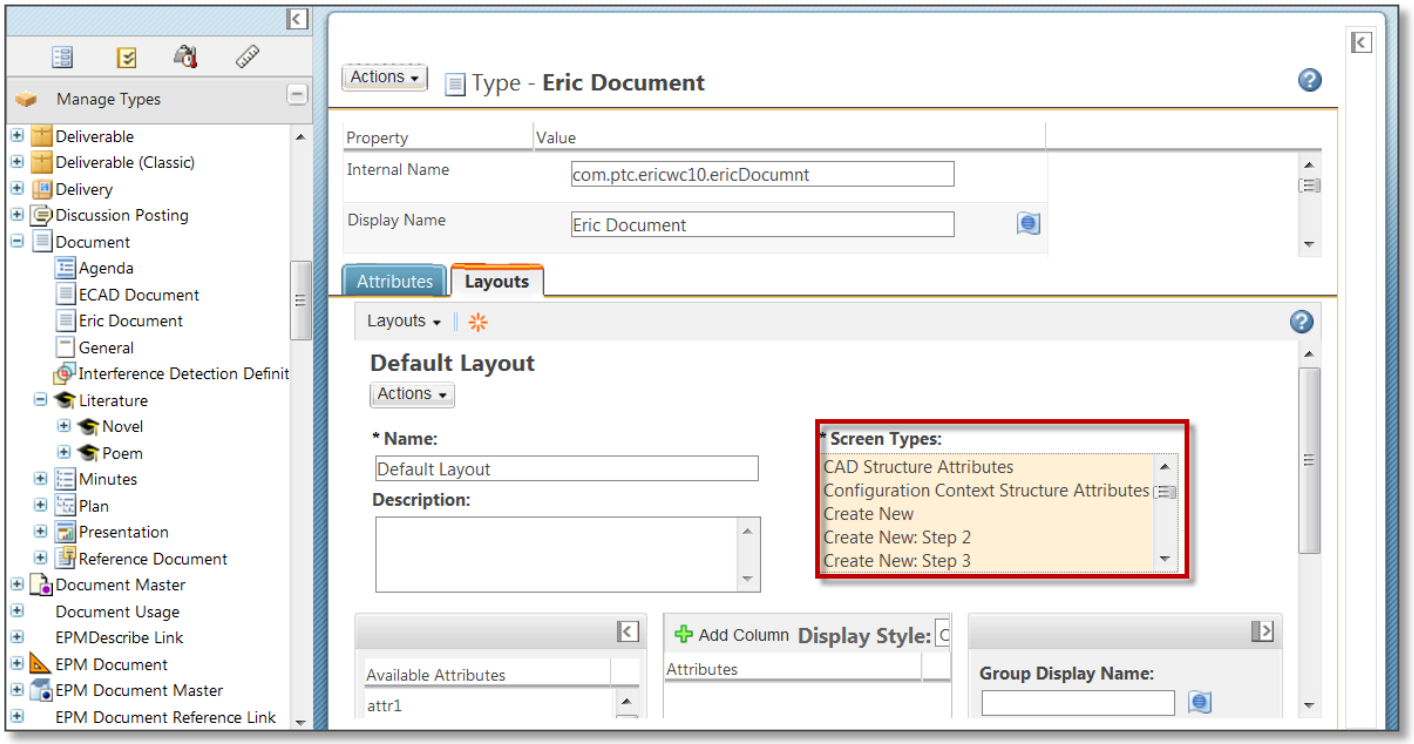


Architecture of custom information page

3.1 Understanding of layout

Screen Type of Layout

- Screen type was already fixed by Windchill core, so layout is just supported on the fixed screen type.
- The screen type can be selected just one time by layout. (if one layout select a screen type, the type can not display on creating other layout) – except default layout
- If you want to use attribute lists on other screen, you have to use attribute panel component.

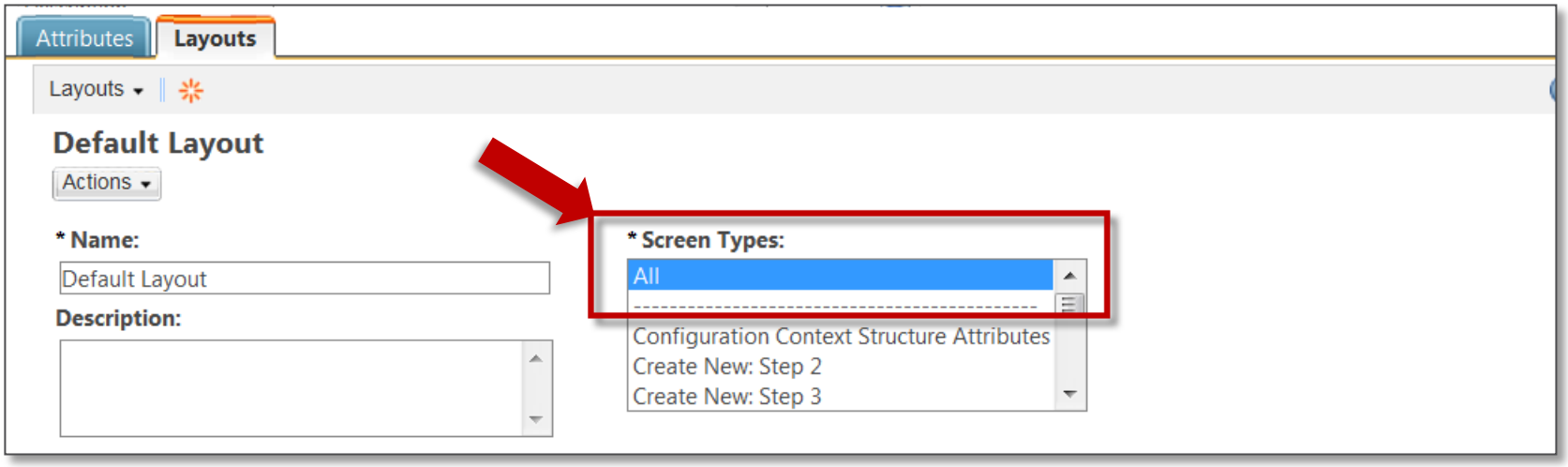


Architecture of custom information page

3.1 Understanding of layout

Default layout

- Default layout is assigned for all screen type, so if the screen type didn't have layout system will show default layout
- By the caution, each of types must have one default layout which are using all screen type.
- The layout name is not any effect on the system. Screen type is a real effect on the system.

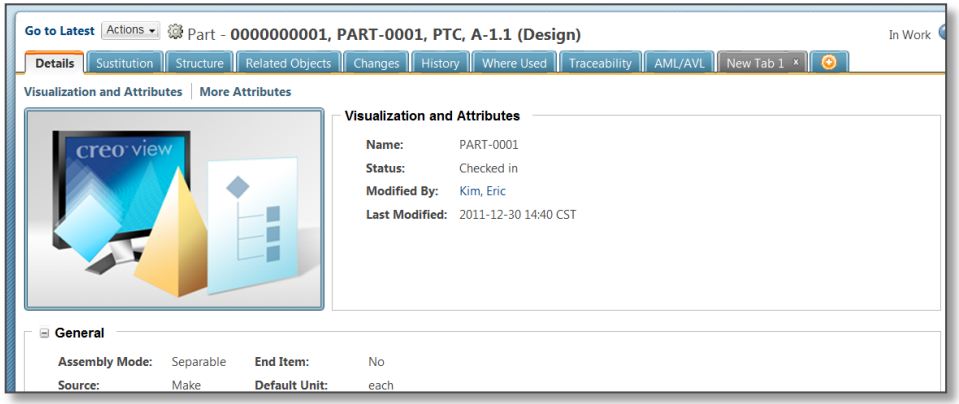


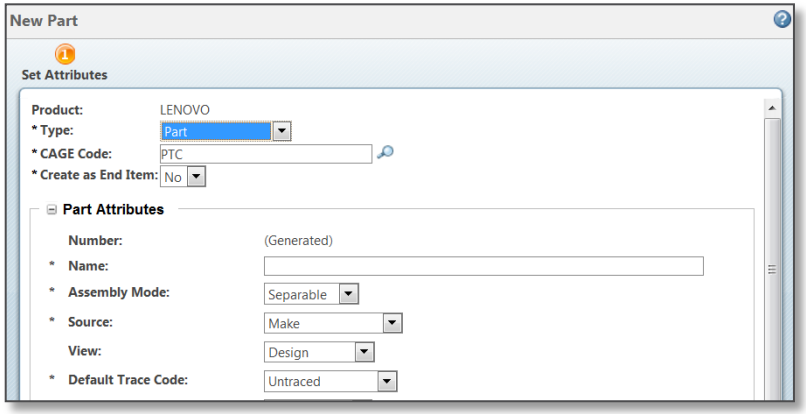
Architecture of custom information page

3.2 OOTB Action set for using Layout

Following is an sample of OOTB action set lists for using layout. (It is not everything)

1) <action name="visualizationAndAttributes" type="object"/> : **Show visualization and primary layout**
2) <action name=" primaryAttributes" type="object"/> : **Show primary attribute layout**
3) <action name="attributes" type="object"/> : **Show more attribute layout**
4) <jca:wizardStep action="defineItemAttributesWizStep" type="object"/> : **Show create attribute layout on wizard**
5) <jca:wizardStep action="editAttributesWizStep" type="object"/> : **Show edit attribute layout on wizard**
6) <jca:wizardStep action="createDocumentSetTypeAndAttributesWizStep" type="document" /> : **Show create attributes layout for document on wizard**





The described action is for general action. If you want to use especial layout which is EPMDocument, you have to use the layout action.

3.3 Example of layout in info page

Especially, info page are using layout for showing primary and more attributes.

```
<!-------- ACTION MODEL FOR TAB -------->
<model name="partInfoPageTabSet" resource="com.ptc.core.ui.navigationRB">
     <submodel name="partInfoDetailsTab"/>
     <action name="productStructureGWT" type="psb"/>
     <submodel name="partInfoRelatedItemsTab"/>
     <submodel name="changesTab"/>
     <submodel name="partInfoHistoryTab"/>
     <submodel name="partInfoWhereUsedTab"/>
     <submodel name="requirementTraceabilityTab" />
     <submodel name="amlAvlTab" />
     <submodel name="prodAnalyticsTab" />
</model>

<model name="partInfoDetailsTab" resourceBundle="com.ptc.core.ui.navigationRB">
        <action name="visualizationAndAttributes" type="object"/>
        <action name="attributes" type="object"/>
</model>
```

Show attributes on “Detail” tab

Show visualization and primary attributes. And also show more attributes(IBA)

This document will not describe how to use and create layout. You can find several document on PTC support sites.

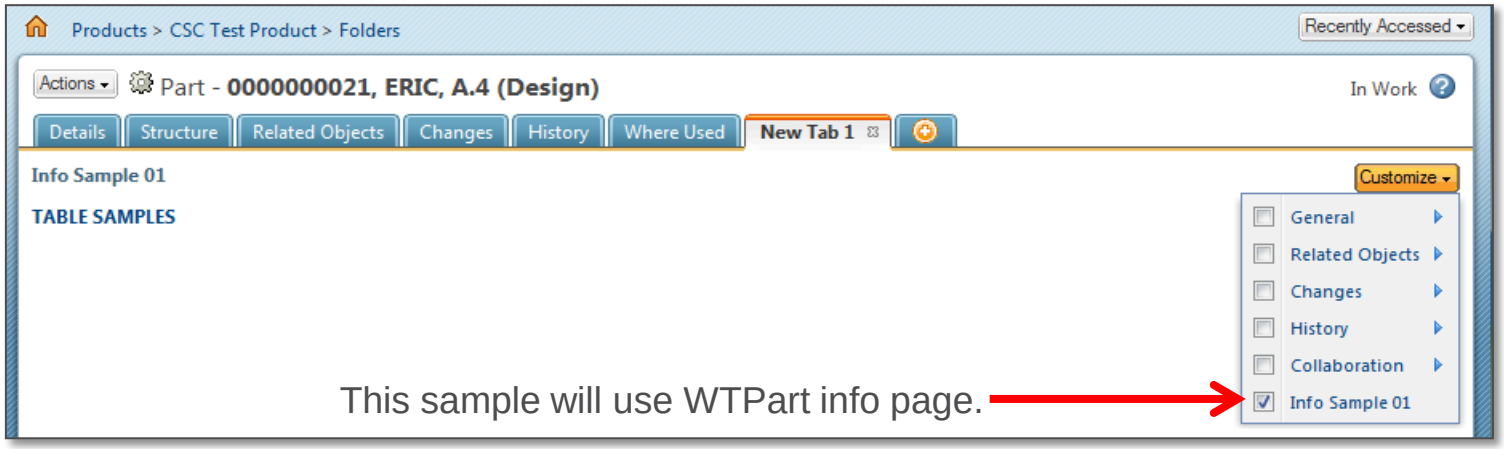
Agenda

1. Introduction Info Page
2. Architecture of custom information page
3. Add custom third navigation on Info Page
4. Excise

3.1 Excise 1 - Add custom action on customize menu

Purpose of adding custom action on customize menu

1. It can help you to understand how to add action on new information page.
2. How to add customized contents (MVC or JSP) without non-modify info page source code. (Just change configuration)
3. Understanding to create “custom New Tab”
4. Understanding how or which configuration file need to modify for info page
5. Design rule in customize button
   - **TAB design for using <submodel>** : Tab will show several contents in the area.
     if page contents included in <submodel> is not existed on customize menu, the contents will not be shown on the info page although you forcefully set the content to some tab.
   - **TAB design for using <action>** - Tab just show one contents. Although the content isn’t existed in customized menu lists, it can be shown on info page.

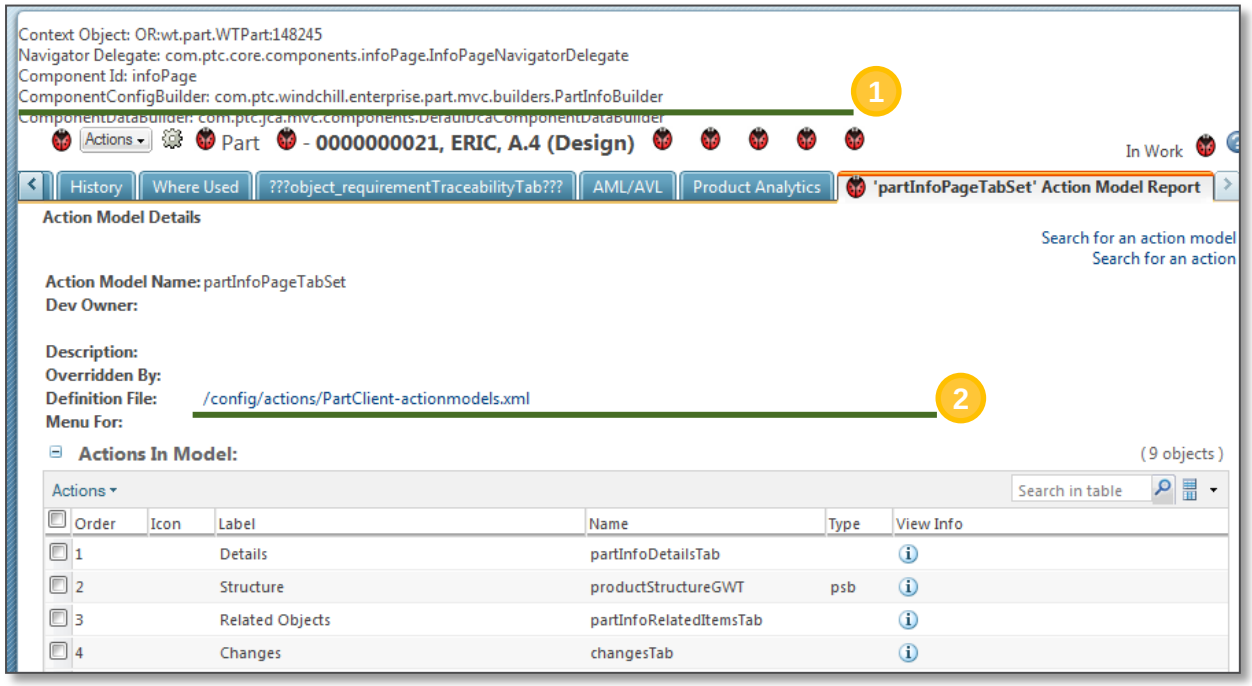


3.1 Excise 1 - Add custom action on customize menu

Find OOTB Action model and Builder source code

1. Open OOTB info page and look jcaDebug

Open one info page of WTPart and set jcaDebug on the WTPart info page for finding info builder class and action model file.



1. That is **info builder class** for WTPart OOTB. Remember this class name.
2. The **action model file name** for designing of WTPart info page

3.1 Excise 1 - Add custom action on customize menu

PTC

Find correct action model name

2. Find action model name using PartInfoBuilder class on OpenGrok site of PTC

OpenGrok has native source codes of Windchill, you can find OOTB info builder class and check which kinds of configure name are using for info page. Look http://ah-grok.ptcnet.ptc.com/xref/x-20-M010/wcEnterprise/PartClient/src/com/ptc/windchill/enterprise/part/mvc/builders/PartInfoBuilder.java link for part info builder.

```
31      public InfoConfig buildInfoConfig(ComponentParams params) throws WTException {
32          log.debug("Info page config begin");
33
34          InfoComponentConfigFactory factory = getComponentConfigFactory();
35          InfoConfig infoConfig = factory.newInfoConfig();
36
37          List<ComponentConfig> standardConfigList = factory.getStandardStatusConfigs();
38
39          for(ComponentConfig componentConfig : standardConfigList){
40              infoConfig.addComponent(componentConfig);
41          }
42
43          infoConfig.setNavBarName("third_level_nav_part");  1
44          infoConfig.setHelpContext("part.view");
45          infoConfig.setTabSet("partInfoPageTabSet");  2
```

1. **The configuration name is configuration of customize menu lists (remember this name)**
   → If customized menu contents will be used on <submodel>, there must be registered in this area although it didn't set on the tab.
2. **The configuration name for tab set**
   → This is configure for tab design.

PartInfoBuilder.java

3.1 Excise 1 - Add custom action on customize menu

Update action model and action

tree.jsp

3. Open “PartClient-actionmodels.xml” and copy “third_level_nav_part” block to custom action model. And the add custom action

csc-actionmodels.xml

```
<model name="third_level_nav_part" resourceBundle="com.ptc.windchill.enterprise.part.partResource">
      <submodel name="general"/>                          <!-- General -->
      <submodel name="relatedItems"/>                     <!-- Related Objects -->
      <submodel name="changes"/>                          <!-- Change -->
      <submodel name="history"/>                          <!-- History -->
      <submodel name="collaboration"/>                    <!-- Collaboration -->
      <submodel name="prodAnalytics"/>                    <!-- Product Analytics -->
      <action name="infoSample01" type="csc"/>
</model>
```

4. Create new action (if you do not have the action)

csc-actions.xml

```
<objecttype name="csc" resourceBundle="com.ptc.core.ui.navigationRB">
       <action name="infoSample01">
            <command windowType="page" url="wtcore/jsp/csc/jca/tree/tree.jsp"/>
       </action>
</objecttype>
```

5. If no have resource bundle, you must create resource bundle for action

action.properties, action_[locale].properties

```
csc.infoSample01.description=Info Sample 01
```

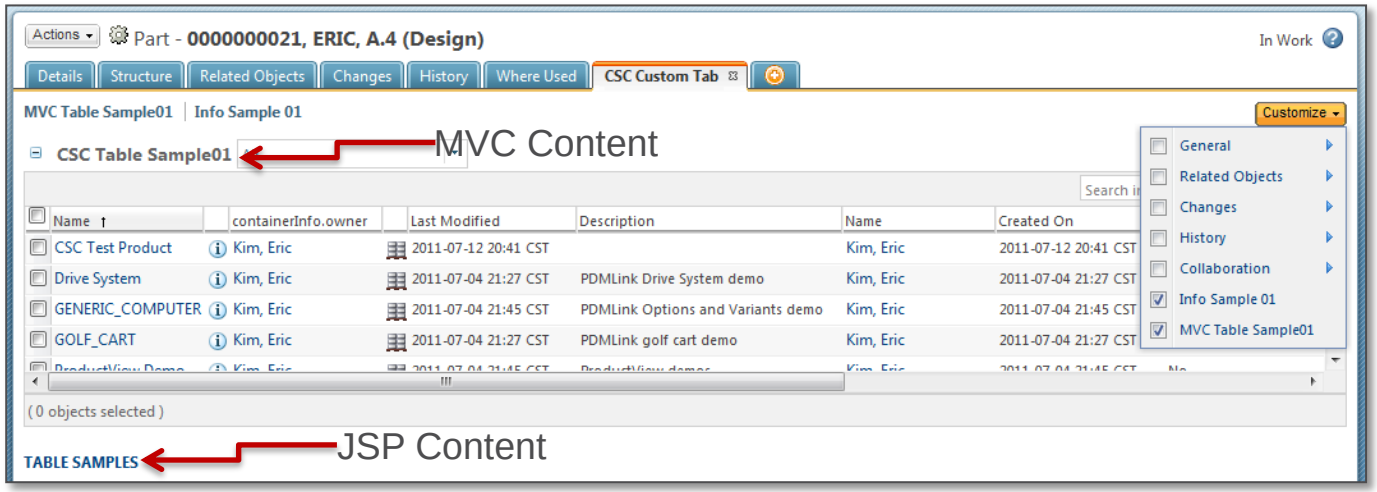
6. Restart MethodServer service or Reload actions

3.1 Excise 1 - Add custom action on customize menu

Result

- **Check customize button (add more action for testing MVC)**

- **Add more content and add the custom component and rename tab**



3.1 Excise 1 - Add custom action on customize menu

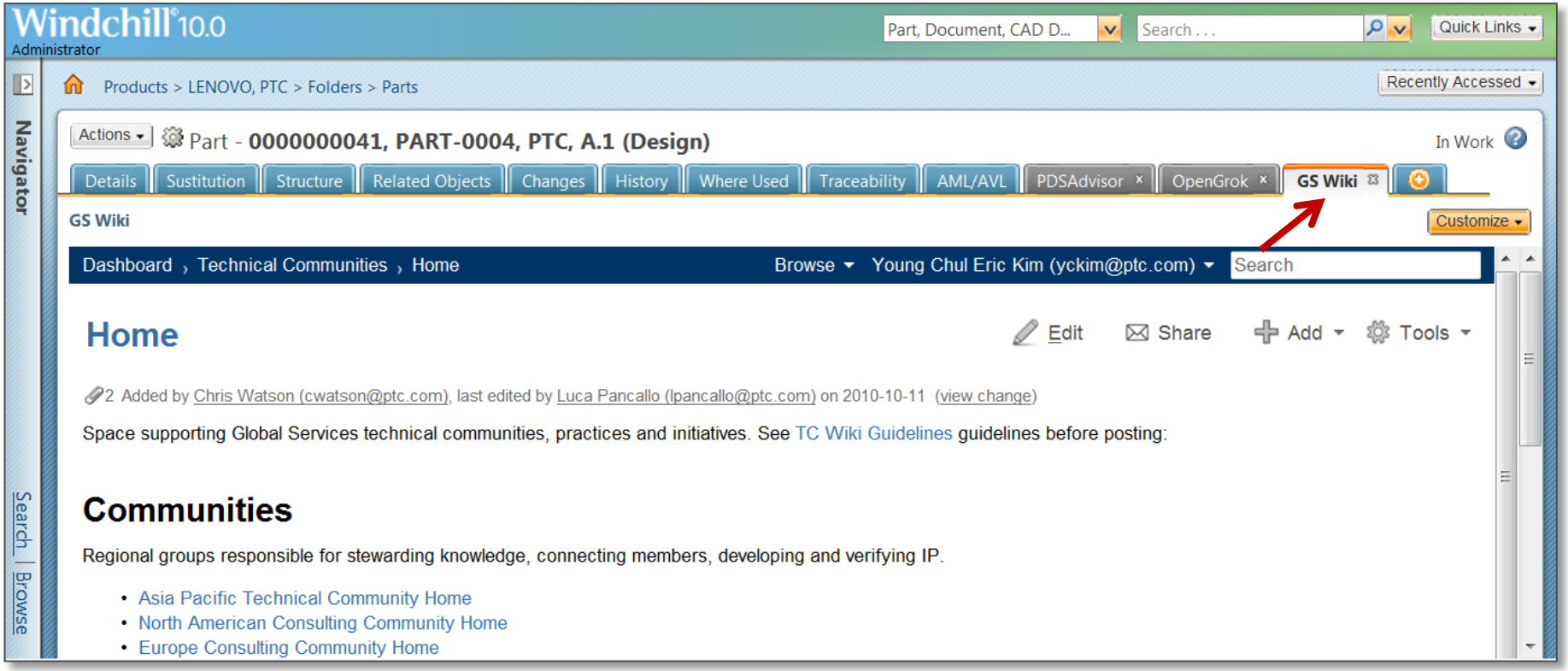
Summary note

Please enter the exercise summary

3.2 Excise 2 – Show other web page in info page

Purpose

Some customer want to see additional web page on info page. For example, they should be checked cost of part when showing part information.



3.2 Excise 2 – Show other web page in info page

tree.jsp

Update action model and action

1. Open “PartClient-actionmodels.xml” and copy “third_level_nav_part” block to custom action model. And the add custom action

csc-actionmodels.xml

```
<model name="third_level_nav_part" resourceBundle="com.ptc.windchill.enterprise.part.partResource">
      <submodel name="general"/>                          <!-- General -->
      <submodel name="relatedItems"/>                     <!-- Related Objects -->
      <submodel name="changes"/>                          <!-- Change -->
      <submodel name="history"/>                          <!-- History -->
      <submodel name="collaboration"/>                    <!-- Collaboration -->
      <submodel name="prodAnalytics"/>                    <!-- Product Analytics -->
      <action name="wikiPageShow" type="csc"/>
</model>
```

2. Create new action (if you do not have the action)

csc-actions.xml

```
<objecttype name="csc" resourceBundle="com.ptc.core.ui.navigationRB">
       <action name="wikiPageShow" ajax="row">
            <command windowType="page" url="wtcore/jsp/csc/wikiPageShow.jsp"/>
       </action>
</objecttype>
```

3. If no have resource bundle, you must create resource bundle for action

action.properties, action_[locale].properties

```
csc.wikiPageShow.description=GS Wiki
```

3.2 Excise 2 – Show other web page in info page

Create JSP and check result

wikiPageShow.jsp

4. Create JSP page in "$WT_HOME/codebase/wtcore/jsp/csc"

wikiPageShow.jspl

```
<%@ taglib uri="http://www.ptc.com/windchill/taglib/components" prefix="jca"%>
<jca:tabToHighlight actionName="tableSamples" objectType="csc" />
<%@ include file="/netmarkets/jsp/util/begin.jspf"%>

<p>
<iframe src="https://ssp.ptc.com/wiki/login.action?os_destination=%2Fhomepage.action" name="iframe" width="100%"
height="600" frameborder="0"></iframe>

<%@ include file="/netmarkets/jsp/util/end.jspf"%>
```

5. Restart MethodServer service or Reload actions

3.2 Excise 2 – Show other web page in info page

Summary note

Please enter the exercise summary

3.3 Excise3 - Create new info page

Scenario

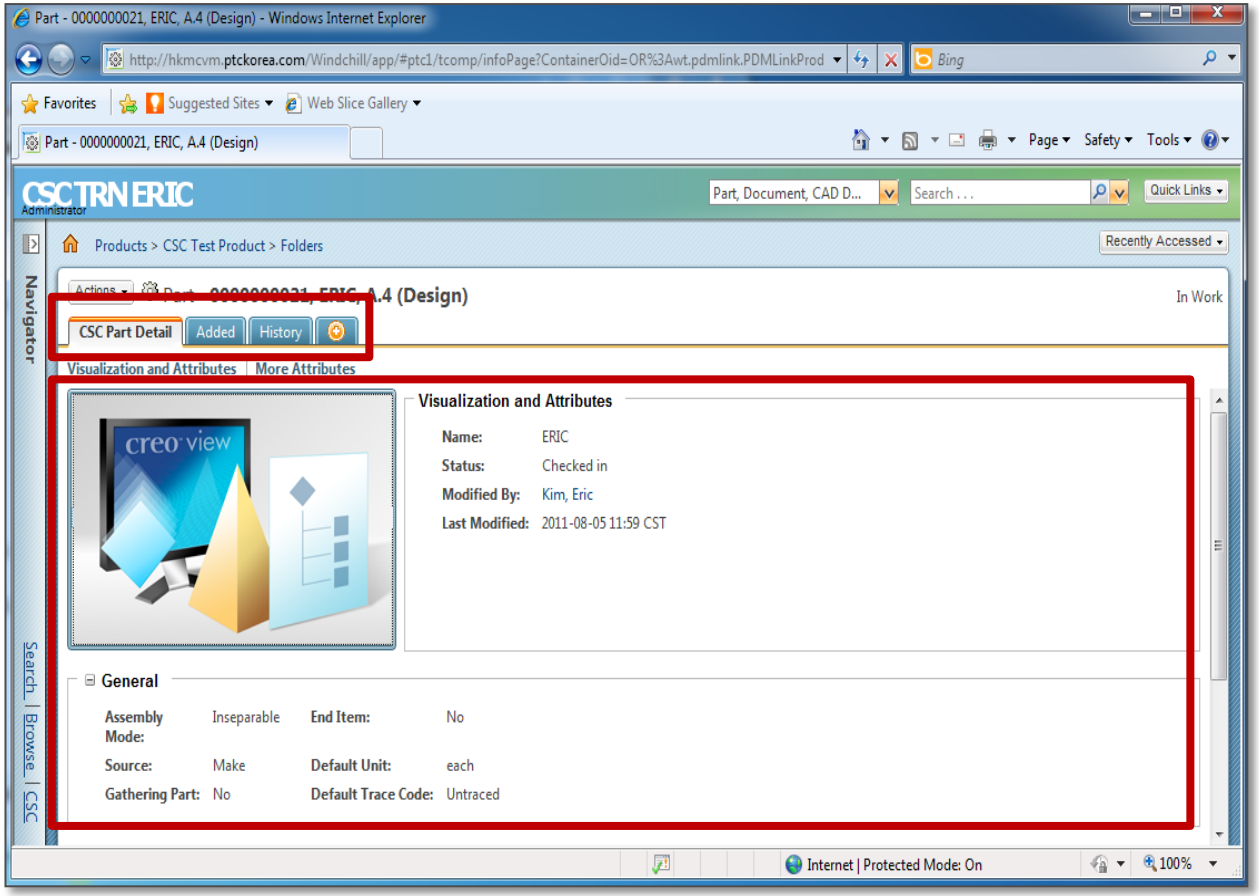
Customer created one new soft type object which is extended from WTPart. But the soft type info page show like part, so customer want to change tab list and content.

Requirement

1. Change tab list
2. Change detail information
3. Change tab content
4. No change action menu
5. No change Identification bar

Solution Progress

1. Create new soft type object
2. Design layout for detail page
3. Configure Design tab and customize list
4. Create a MVC class of info page.
5. Create a tab content of info page. (If OOTB contents can not meet at user requirements)

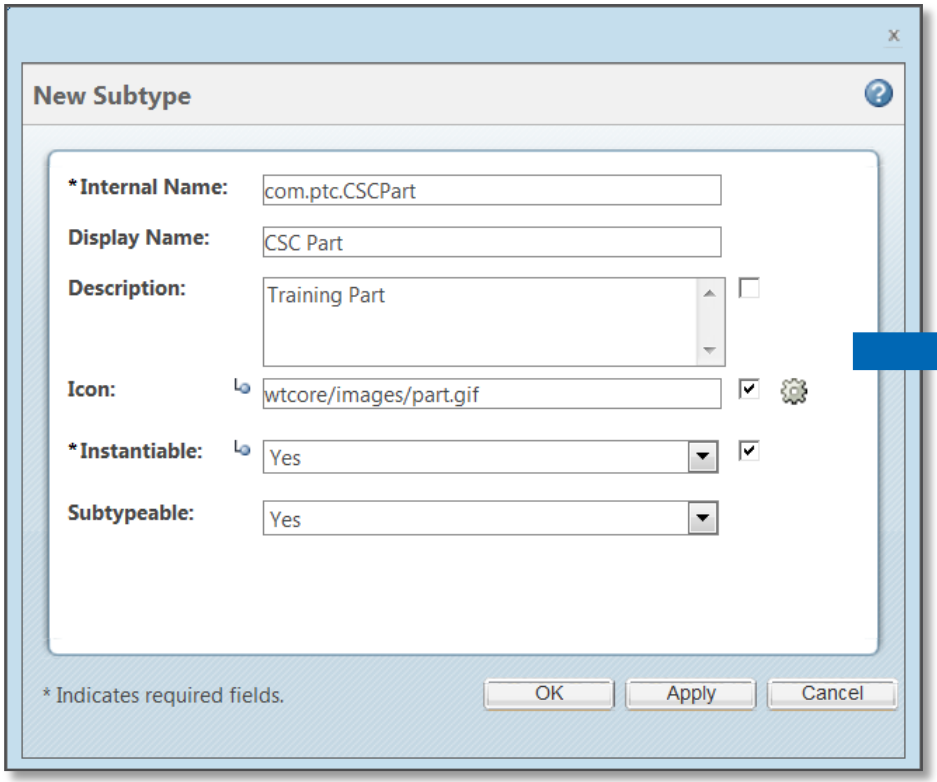


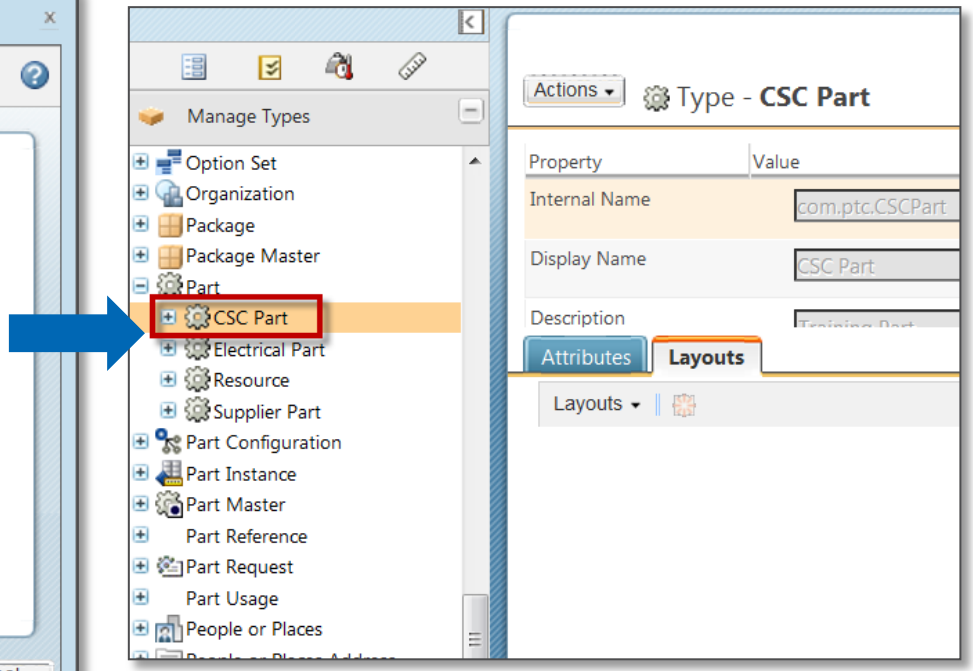
Create Soft type of part

1. Create Soft type object

Create sub type object below part like following attribute

- Internal Name : com.ptc.CSCPart
- Instantiable : Yes



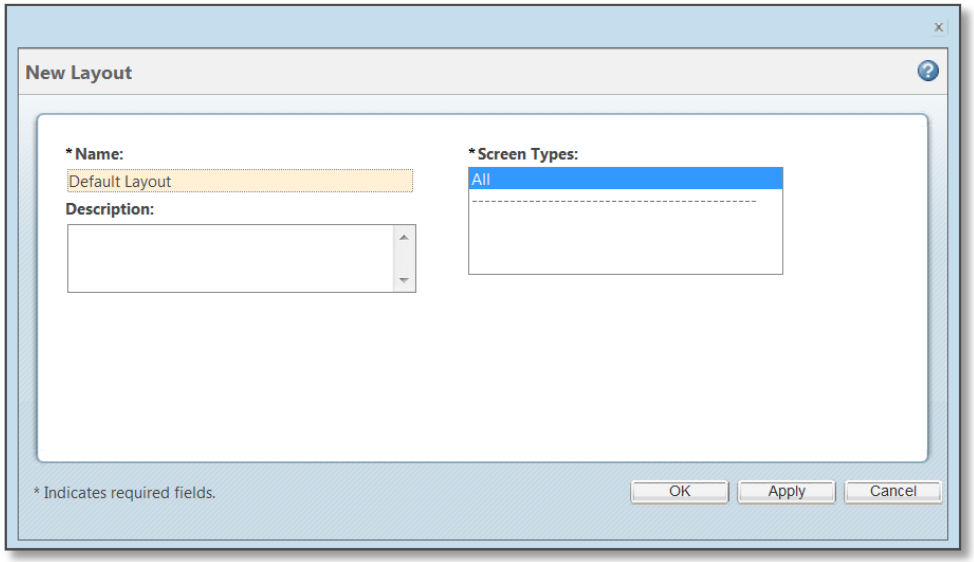


Create default layout

2. Create Default Layout

Create default layout for new sub type object. If you want to know why we need to create default layout, look "3.1 Understanding of layout". When you create new layout of sub type, the all of inherited layout will be gone. So you have to design your screen type design.

1) Select created new sub type
2) Select action menu : Edit
3) Select Layout tab and click create new layout icon
4) Enter information
   - Layout name : Default Layout
   - Screen Type : All
5) Edit default attributes on default layout (Optional)

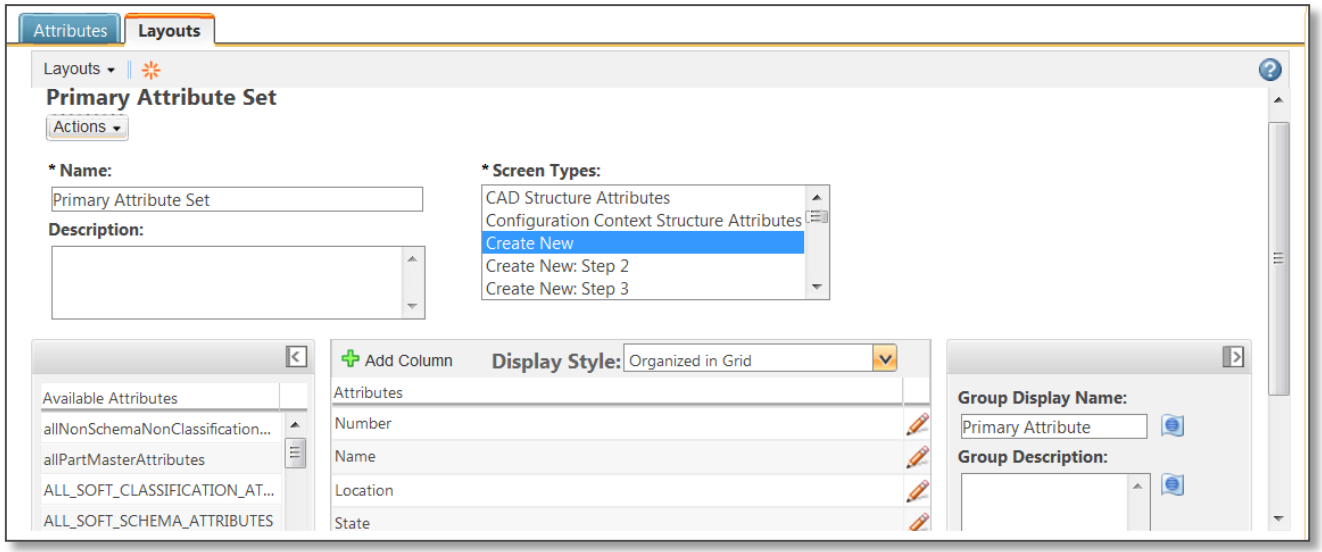


Create create/modify screen layout (Primary attributes Set)

3. Create Primary attribute set for create/modify

Create primary attribute set for create/modify. If your design has different attributes set for create/edit/info-page, please separate layout for each of screen types.

1) Click create new layout icon
2) Enter information
   - Layout name : Primary Attribute Set
   - Screen Type : Create New / Edit / Information Page – Primary Attributes
3) Edit primary attributes attributes on primary attributes
   - Group name : Primary Attribute
   - Add Attribute : Number / Name / Location / State

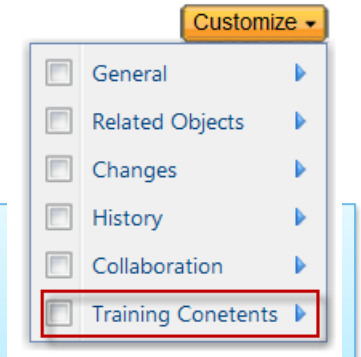


Modify custom-actionModels.xml

4. Create Customize menu

It must be configured for info page. In this case, we will use most of WTPart configuration and add more custom action in this menu

1) Open “partClient-actionmodel.xml” and copy block of "third_level_nav_part“
2) Open “custom-actionmdoel.xml” and past copied block on the file
3) Add all of custom action in copied block freely.



custom-actionmodels.xml

```
<model name="CSCPart_customize_menu">
      <submodel name="general"/>                    <!-- General -->
      <submodel name="relatedItems"/>               <!-- Related Objects -->
      <submodel name="changes"/>                    <!-- Change -->
      <submodel name="history"/>                    <!-- History -->
      <submodel name="collaboration"/>              <!-- Collaboration -->
      <submodel name="prodAnalytics"/>              <!-- Product Analytics -->
      <submodel name="CSCPartContents"/>            <!-- CUSTOM SET -->
</model>
```

Design rule in customize button

- **TAB design for using <submodel>**: Tab will show several contents in the area.
  if page contents included in <submodel> is not existed on customize menu, the contents will not be shown on the info page although you forcefully set the content to some tab.
- **TAB design for using <action>** - Tab just show one contents. Although the content isn’t existed in customized menu lists, it can be shown on info page.

Modify custom-actionModels.xml

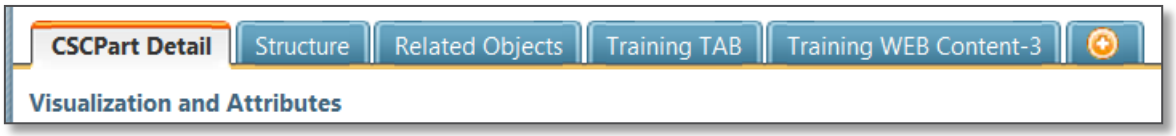
5. Create Tab list and submodel

It must also be configured for info page. It will be also referred from WTPart tab configuration.

1) Open “partClient-actionmodel.xml” and copy block of “partInfoPageTabSet “
2) Open “custom-actionmdoel.xml” and past copied block on the file
3) Edit for custom style like following

custom-actionmodels.xml

```
<model name="CSCPartInfoPageTabSet">
      <submodel            name="CSCPartInfoDetailsTab"/>            <!-- TRAINING INFO PAGE -->
      <action              name="productStructureGWT" type="psb"/>   <!-- OOTB -->
      <submodel            name="partInfoRelatedItemsTab"/>          <!-- OOTB -->
      <submodel            name="customTab1"/>                       <!-- CUSTOM TAB (SUBMODEL) -->
      <action              name="CSCPartContent3" type="info_trn"/>  <!-- CUSTOM TAB (ACTION) -->
</model>


<model name="CSCPartInfoDetailsTab" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
      <action name="visualizationAndAttributes" type="object"/>
</model>

<model name="customTab1" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
      <action name="CSCPartContent1"            type="info_trn"/>
      <action name="CSCPartContent2"           type="info_trn"/>
</model>
```



3.3 Excise3 - Create new info page

Modify custom-actions.xml

6. Create new action

If you created new custom action, you must configure and set up the custom action.

1) Open “custom-action.xml” and edit for custom action like following

custom-actionmodels.xml

```
<model name="CSCPartInfoPageTabSet">
      <submodelname="CSCPartInfoDetailsTab"/>            <!-- TRAINING INFO PAGE -->
      <action  name="productStructureGWT" type="psb"/>   <!-- OOTB -->
      <submodelname="partInfoRelatedItemsTab"/>          <!-- OOTB -->
      <submodelname="customTab1"/>                       <!-- CUSTOM TAB (SUBMODEL) -->
1     <action  name="CSCPartContent3" type="info_trn"/>  <!-- CUSTOM TAB (ACTION) -->
</model>

<model name="CSCPartInfoDetailsTab" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
      <action name="visualizationAndAttributes" type="object"/>
</model>

<model name="customTab1" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
2     <action name="CSCPartContent1"            type="info_trn"/>
3     <action name="CSCPartContent2"           type="info_trn"/>
</model>
```

custom-actions.xml

```
<objecttype name="info_trn" class="" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
1     <action name="CSCPartContent1">
          <component windowType="popup" name="netmarkets.project.list.table"/>
      </action>
2     <action name="CSCPartContent2" ajax="row">
          <command windowType="popup" class="ext.csc.jca.wizard.CSCWizardSampleProcess" method="execute"
url="wtcore/jsp/csc/info_content2.jsp"/>
      </action>
3     <action name="CSCPartContent3" ajax="row">
          <command windowType="page" url="wtcore/jsp/csc/webPageShow.jsp"/>
      </action>
</objecttype>
```

Create resource bundle file

7. Create resource bundle file

If you added new action and tab on the configuration, you should create resource bundle for displaying

1) Open eclipse and create new package "**ext.csc.training.resource**"
2) Create new Resource bundle class
   - Class name : **CSCTrainingRB**
   - Super class : **WTListResourceBundle**

```
@RBUUID("ext.csc.training.resource.CSCTrainingRB")
public final class CSCTrainingRB extends WTListResourceBundle {

    @RBEntry("Training Conetents")
    public static final String PRIVATE_CONSTANT_1 = "object.CSCPartContents.description";

    @RBEntry("Training MVC Content-1")
    public static final String PRIVATE_CONSTANT_2 = "info_trn.CSCPartContent1.description";

    @RBEntry("Training JSP Content-2")
    public static final String PRIVATE_CONSTANT_3 = "info_trn.CSCPartContent2.description";

    @RBEntry("Training WEB Content-3")
    public static final String PRIVATE_CONSTANT_4 = "info_trn.CSCPartContent3.description";

    @RBEntry("Training TAB")
    public static final String PRIVATE_CONSTANT_5 = "object.customTab1.description";

    @RBEntry("CSCPart Detail")
    public static final String PRIVATE_CONSTANT_6 = "object.CSCPartInfoDetailsTab.description";

}
```

If you have multi language environment in client, please make resource bundle of each of multi language also.

PTC®

Create MVC class of info builder

8. Create info builder class

Finally we have to make info builder for especial sub type which is “CSCPart”.

1) Open eclipse and create new package “**ext.csc.training.mvc**”
   - the package must be registered in MVCDispatcher
2) Create new info builder class
   - Class name : **CSCPartInfoBuilder**
   - Super class : **AbstractInfoConfigBuilder**
   - Implements class : **ComponentDataBuilder**
   - TypeBased of annotation : **wt.part.WTPart|com.ptc.CSCPart**

```
@TypeBased(value = "wt.part.WTPart|com.ptc.CSCPart")
@ComponentBuilder(value = ComponentId.INFOPAGE_ID)
public class CSCPartInfoBuilder extends AbstractInfoConfigBuilder implements ComponentDataBuilder {
    @Override
    public Object buildComponentData(ComponentConfig arg0, ComponentParams arg1) throws Exception {
        // TODO Auto-generated method stub
        return arg1.getContextObject();
    }

    @Override
    protected InfoConfig buildInfoConfig(ComponentParams arg0) throws WTException {
        // TODO Auto-generated method stub
        InfoComponentConfigFactory factory = getComponentConfigFactory();
        InfoConfig result = factory.newInfoConfig();

        result.setNavBarName("CSCPart_customize_menu");
        result.setTabSet("CSCPartInfoPageTabSet");
        return result;
    }
}
```

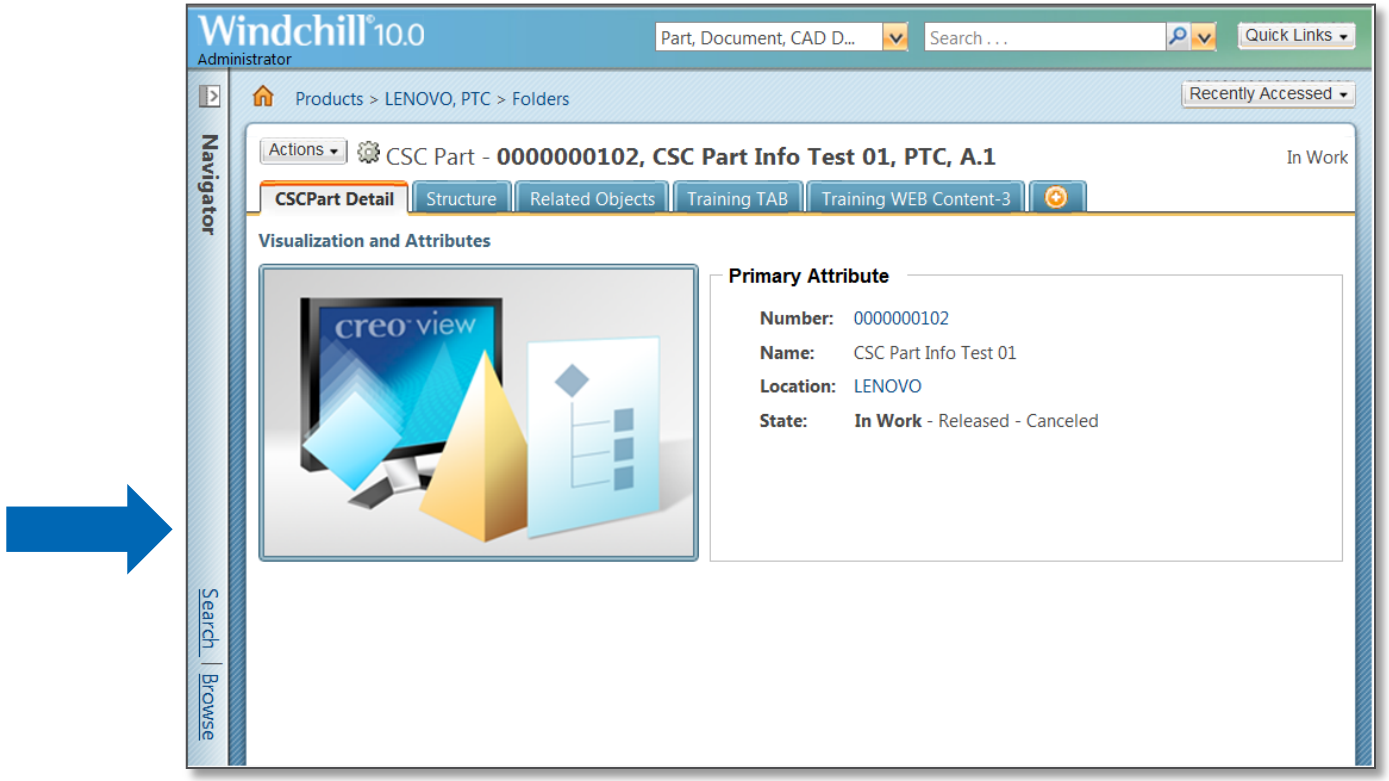
3.3 Excise3 - Create new info page

Restart Method Server and check result

9. Restart MethodServer

10. Check result

1) **Create CSCPart in any container**
2) **Open info page**



3.3 Excise3 - Create new info page

Summary note

Please enter the exercise summary

3.4 Excise4 – Validate tab & action

Scenario

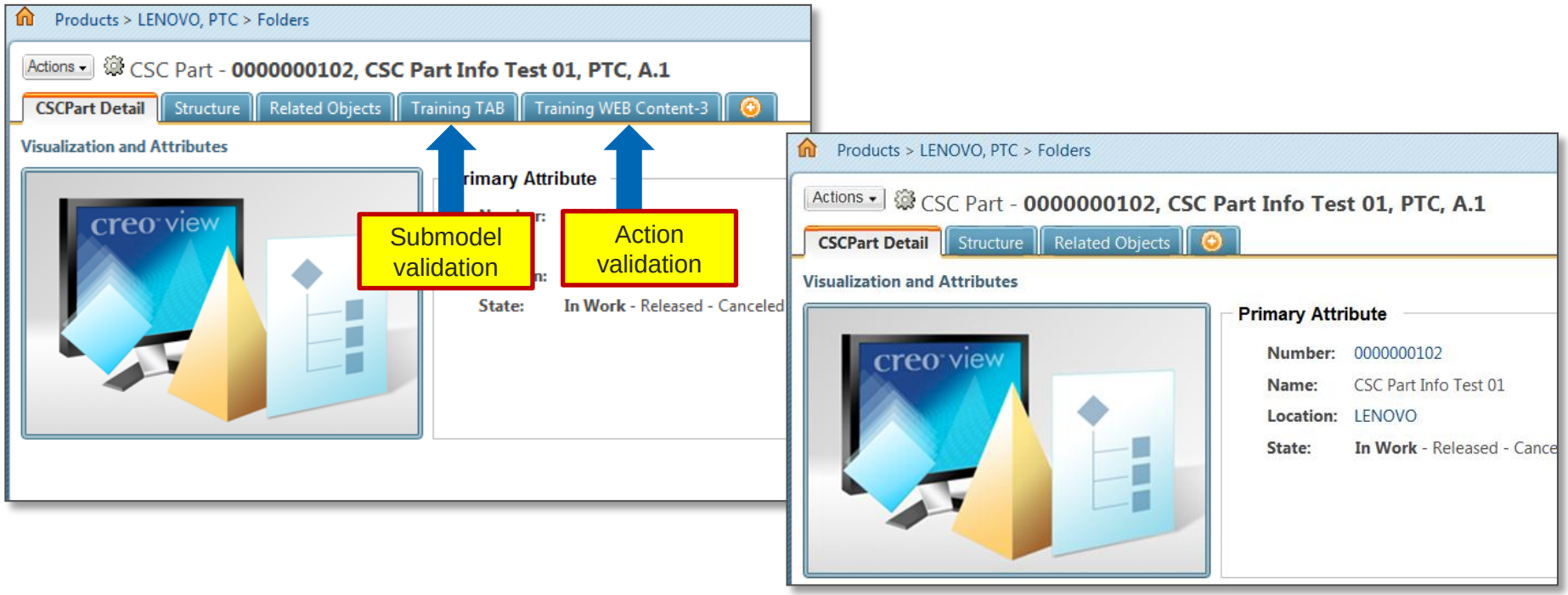
1. Scenario

User want to validate to show or hide for target tab or action (content/menu) inside of correct condition. Now we will try to make validation using excise 3 result on following condition.

- “Training TAB” and “Training WEB content 3 “ tabs want to show just “CSC” group user.

2. Objective

- To understand how to set validator for <submodel> in *-actionModels.xml
- To understand how to set validator for <action> in *-action.xml



3.4 Excise4 – Validate tab & action

Set custom service properties

1. Register custom service properties

If possible we have to try to not touch OOTB configuration file, so we will try to make new custom service property file and will set for indicating by MethodServer

- New Custom service file name : **trainingService.properties**
- Custom service xconf file name : **trainingService.properties.xconf**

Also If possible, all properties file should be controlled by X-configuration file, so we will register x-configuration file for custom properties.

Registering for X-configuration

$WT_HOME/site.xconf

```
<ConfigurationRef xlink:href="codebase/ext/csc/training/trainingService.properties.xconf"/>

<Property name="wt.services.applicationcontext.WTServiceProviderFromProperties.defaultPropertyFiles" overridable="true"
            targetFile="codebase/wt.properties"

value="service.properties,……,com/ptc/windchill/enterprise/report/reporting.dataUtilities.properties,ext/csc/training/
trainingService.properties"/>
```

Copy value of "**wt.services.applicationcontext.WTServiceProviderFromProperties.defaultPropertyFiles**" on wt.properties, and add "**trainingService.properties**"

Understand <model> tag validation

2. Update <submodel> tag for validator.

From this section, we need to understand how to set model tag.
As you know, <submodel> tag will show one tab on the information page, so <model> tag's validator will registered on the <submodel> tag configuration on action model design.

$WT_HOME/codebase/config/actions/custom-actionsModels.xml

```
<!-- IT DESIGNED ALREADY ON PREVIOUS EXCISE -->
<model name="CSCPartInfoPageTabSet">
      <submodel            name="CSCPartInfoDetailsTab"/>           <!-- TRAINING INFO PAGE -->
               <action     name="productStructureGWT" type="psb"/>  <!-- OOTB -->
      <submodel            name="partInfoRelatedItemsTab"/>         <!-- OOTB -->
      <submodel            name="customTab1"/>                      <!-- CUSTOM TAB (SUBMODEL) -->
               <action     name="CSCPartContent3" type="info_trn"/> <!-- CUSTOM TAB (ACTION) -->
</model>

<model name="customTab1" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
      <action name="CSCPartContent1"            type="info_trn"/>
      <action name="CSCPartContent2"           type="info_trn"/>
      <includeFilter name="customTab1" />   <!-- REGISTER VALIDATOR KEY -->
</model>
```

The "name" is validation key that will be registered on Windchill services.
You have to remember the key before setting services.

The tag type is <includeFilter> and <excludeFilter>.
Basically the info page has some validator which is supported from OOTB, so if you want to not use OOTB validator you can exclude the validator using <excludeFilter> tag

Understand <model> tag validation

3. Register services key for validator of <submodel> tag

We had been pre-set for validator of <submodel> tag which name is “customTab1”, so now we need to try to set the key on Windchill services like following.

$WT_HOME/codebase/ext/csc/training/trainingService.properties.xconf

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Configuration SYSTEM "xconf.dtd">
<Configuration targetFile="codebase/ext/csc/training/trainingService.properties">
        <Service context="default" name="com.ptc.core.ui.validation.UIComponentValidator">
            <Option serviceClass="ext.csc.training.validator.CSCPartSubmodelValidator"
                    selector="customTab1"
                    requestor="null"  />
        </Service>
</Configuration>
```

Following is the detail description of service parameter.

- Name : Super class for validator. Generally we use “UIComponentBalidator”
- serviceCalss: Custom validator class
- Selector: Registered key name on <submodel> tag.

```
<model name="customTab1" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
   <action name="CSCPartContent1"            type="info_trn"/>
   <action name="CSCPartContent2"            type="info_trn"/>
   <includeFilter name="customTab1" />
</model>
```

When you finished updated x-configuration file, try to generate properties file using “xconfmanager”

Understand <model> tag validation

4. Create Validator Class

Validator class must be extended from "DefaultUIComponentValidator" at least. Basically the class has several function. In this functions, you have to design correct function for validator.

$WT_HOME/codebase/ext/csc/training/trainingService.properties.xconf

```
package ext.csc.training.validator;

public class CSCPartSubmodelValidator extends DefaultUIComponentValidator {
    public UIValidationResultSet performFullPreValidation(UIValidationKey validationKey, UIValidationCriteria validationCriteria, Locale locale)
throws WTException {
        UIValidationResult result = null;
        UIValidationResultSet resultSet = UIValidationResultSet.newInstance();

        WTPrincipal principal = SessionHelper.manager.getPrincipal();
        WTGroup cscGroup = (WTGroup)OrganizationServicesHelper.manager.getGroup("CSC");
        WTGroup administratorGroup = OrganizationServicesHelper.manager.getGroup("Administrators");

        if ( OrganizationServicesHelper.manager.isMember(cscGroup, principal) ||
             OrganizationServicesHelper.manager.isMember(administratorGroup, principal) ) {
            result = UIValidationResult.newInstance(validationKey, UIValidationStatus.ENABLED);
        } else {
            result = UIValidationResult.newInstance(validationKey, UIValidationStatus.HIDDEN);
        }

        resultSet.addResult(result);

        return resultSet;
    }
}
```

Validation result set

Validation result

3.4 Excise4 – Validate tab & action

Understand <model> tag validation

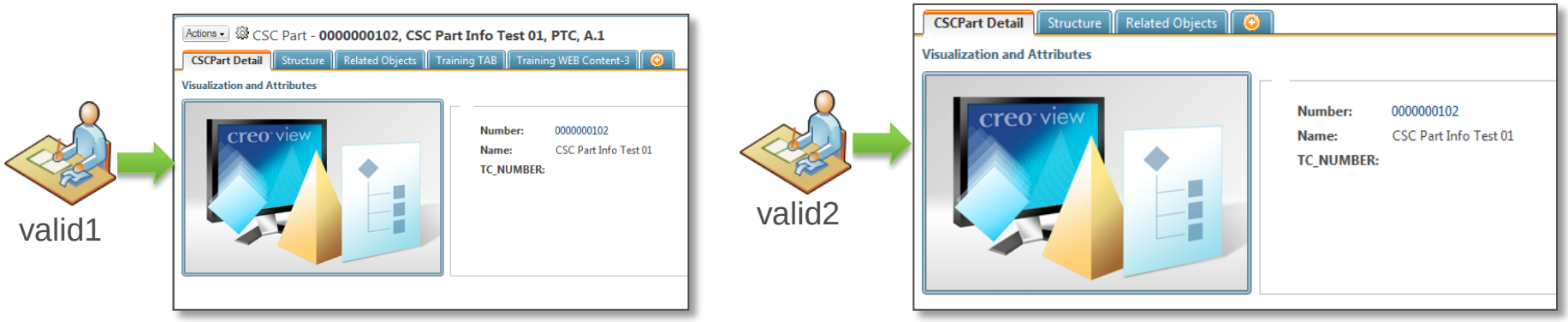
5. Restart server and check your validation for tab

Restart server, because you changed configuration files, so it needs to be restart.

6. Check <submodel> tag validation.

Following is test system configuration.

- *User: valid1, valid2*
- *Group: CSC(member=valid1)*
- *Product: CSC_TEST / members = valid1, valid2*



Understand action validation

7. Check action key name for validation.

Action validator doesn't need some configuration on the "actions.xml". You just need to remember action key and class name for set services configuration

$WT_HOME/codebase/config/actions/custom-actions.xml

```
<objecttype name="info_trn" class="wt.part.WTPart" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
      <action name="CSCPartContent1">
          <component windowType="popup" name="netmarkets...e"/>
      </action>
      <action name="CSCPartContent2" ajax="row">
          <command windowType="popup" class="ext.csc.jca.wizard.CSCWizardSampleProcess" method="execute"
          url="wtcore/jsp/csc/info_content2.jsp"/>
      </action>
      <action name="CSCPartContent3" ajax...
          <command windowType="page" url="w...PageShow.jsp"/>
      </action>
</objecttype>
```

Remember class name

Remember key name

8. Register service for validation of action

$WT_HOME/codebase/config/actions/custom-actions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Configuration SYSTEM "xconf.dtd">
<Configuration targetFile="codebase/ext/csc/training/trainingService.properties">
                <Service context="default" name="com.ptc.core.ui.validation.UIComponentValidator">
                        <Option serviceClass="ext.csc.training.validator.CSCPartSubmodelValidator"
                                selector="CSCPartContent3"
                                requestor="wt.part.WTPart"  />
                </Service>
</Configuration>
```

3.4 Excise4 – Validate tab & action

Understand action validation

9. Detail of service configuration

Following is the description for action validator on service property

1. **name** : Super class for validator. Generally we use “UIComponentBalidator”
2. **serviceClass** : Custom validator class
3. **Selector** : Action name (we remembered when we design action)
4. **Requestor** : object type class in designed action

$WT_HOME/codebase/config/actions/custom-actions.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE Configuration SYSTEM "xconf.dtd">
<Configuration targetFile="codebase/ext/csc/training/trainingService.properties">
        <Service context="default" name="com.ptc.core.ui.validation.UIComponentValidator">
            <Option serviceClass="ext.csc.training.validator.CSCPartSubmodelValidator"
                selector="CSCPartContent3"
                requestor="wt.part.WTPart"  />
        </Service>
</Configuration>
```

10. Restart server and check your validation for tab

The checking progress same <submodel> tag validation.

3.4 Excise4 – Validate tab & action

Summary note

Please enter the exercise summary

Wizard Customization

Wizard

- Customization of wizard was not changed basically. It is same methodology with previous version.
- However WC10 are using ExtJS, so partially some javascript function was changed calling ExtJS function. For example, button action.

2. Architecture of Wizard

Base wizard

Wizard body will be called from action (button, link, menu, navigation). By the action call, the body JSP of wizard page will be open. The body is including all of step pages which is also made by JSP. Especially Wizard is still using JCA tag for designing.

Basic of Body JSP

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ include file="/netmarkets/jsp/components/beginWizard.jspf"%>
<%@ include file="/netmarkets/jsp/components/includeWizBean.jspf"%>
.
.
.
.
<%@include file="/netmarkets/jsp/util/end.jspf"%>
```

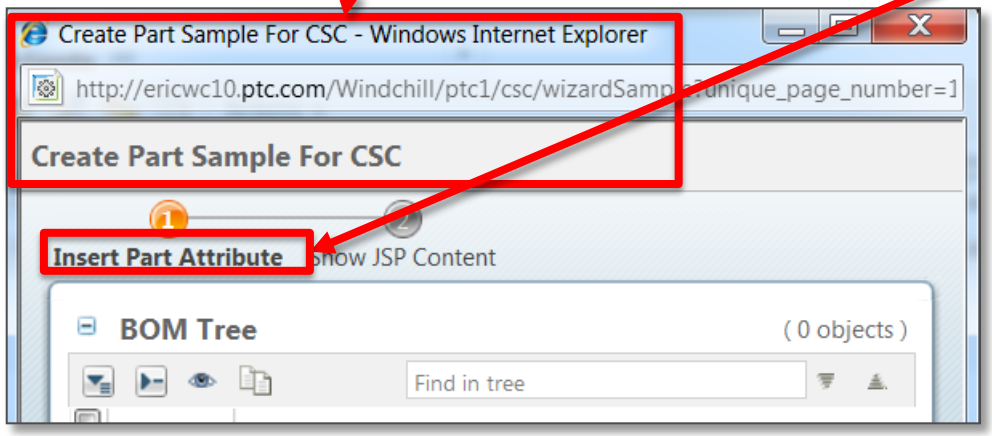
- **<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>**
  It is required tag declaration for using JCA component. (Required)
- **<%@ include file="/netmarkets/jsp/components/beginWizard.jspf"%>**
- **<%@ include file="/netmarkets/jsp/components/includeWizBean.jspf"%>**
  It is similar “begin.jspf” of general JCA page, but you have to use above JSPF for wizard.
- **<%@include file="/netmarkets/jsp/util/end.jspf"%>**
  If is the latest including file, it is same JSPF with general JCA declaration.

JCA tag for wizard

Following is the description of wizard tag.

```
<jca:wizard  title="Create Part Sample For CSC">
             <jca:wizardStep action="setProcessStepAttributes" type="cscWzard“ label=“Insert Part Attribute”/>
             <jca:wizardStep action="checkAttribute" type="cscWzard"/>
</jca:wizard>
```

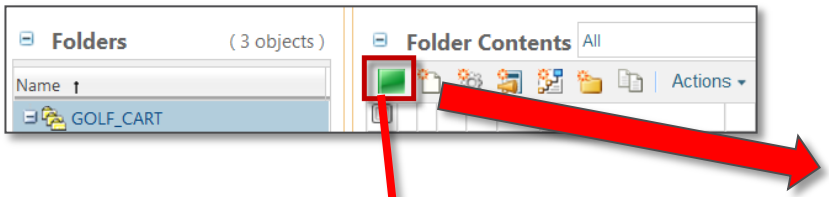
- “<jca:wizard>” is using for configuration of whole wizard page.

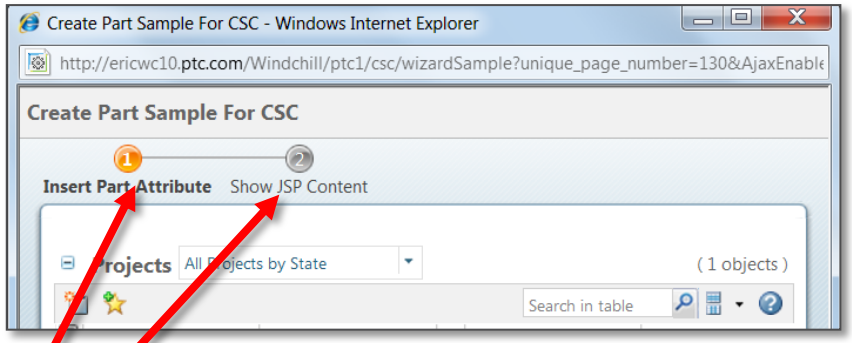


- “<jca:wizardStep>” is for each of step action. All step Action name must be existed on “action.xml”.

2. Architecture of Wizard

Definition of action.xml





Definition of Actions.xml

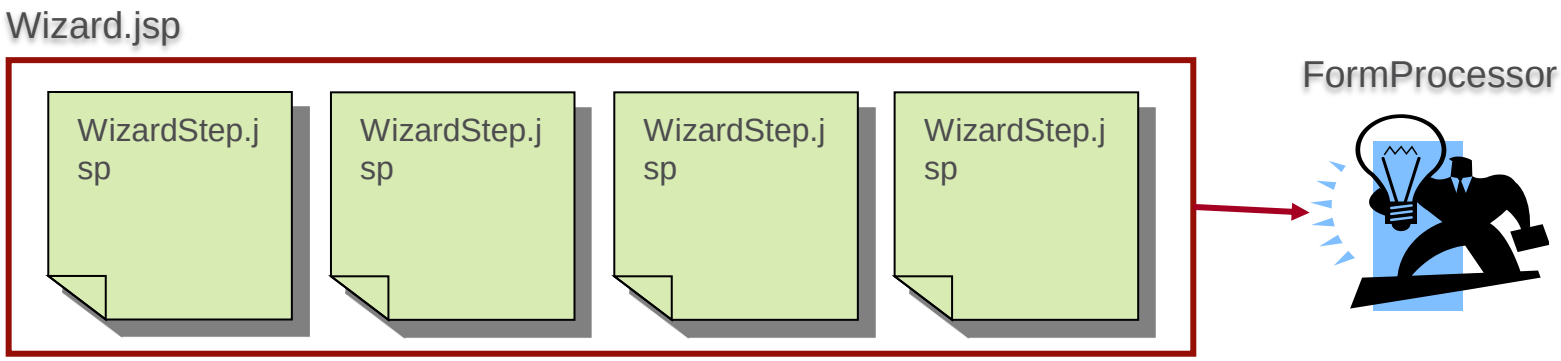
```
<objecttype name="csc" class="" resourceBundle="com.ptc.core.ui.navigationRB“>
   <action name="wizardSamples">
        <command windowType="popup" class="ext.csc.jca.wizard.CSCWizardSampleProcess“  method=“execute”
url="wtcore/jsp/csc/jca/wizard/wizardSample.jsp"/>
   </action>
</objecttype>

<objecttype name="cscWzard">
   <action name="setProcessStepAttributes">
        <command windowType="wizard_step" url="wtcore/jsp/csc/jca/wizard/setProcessStepAttributes.jsp"/>
   </action>
   <action name="checkAttribute" preloadWizardPage="false">
        <command windowType="wizard_step" url="wtcore/jsp/csc/jca/wizard/checkAttribute.jsp"/>
   </action>
</objecttype>
```

- Button action has class and method field. It is the main execution class by calling from wizard when finished. This class must be extended from FormProcessor.

2. Architecture of Wizard

Wizard architecture

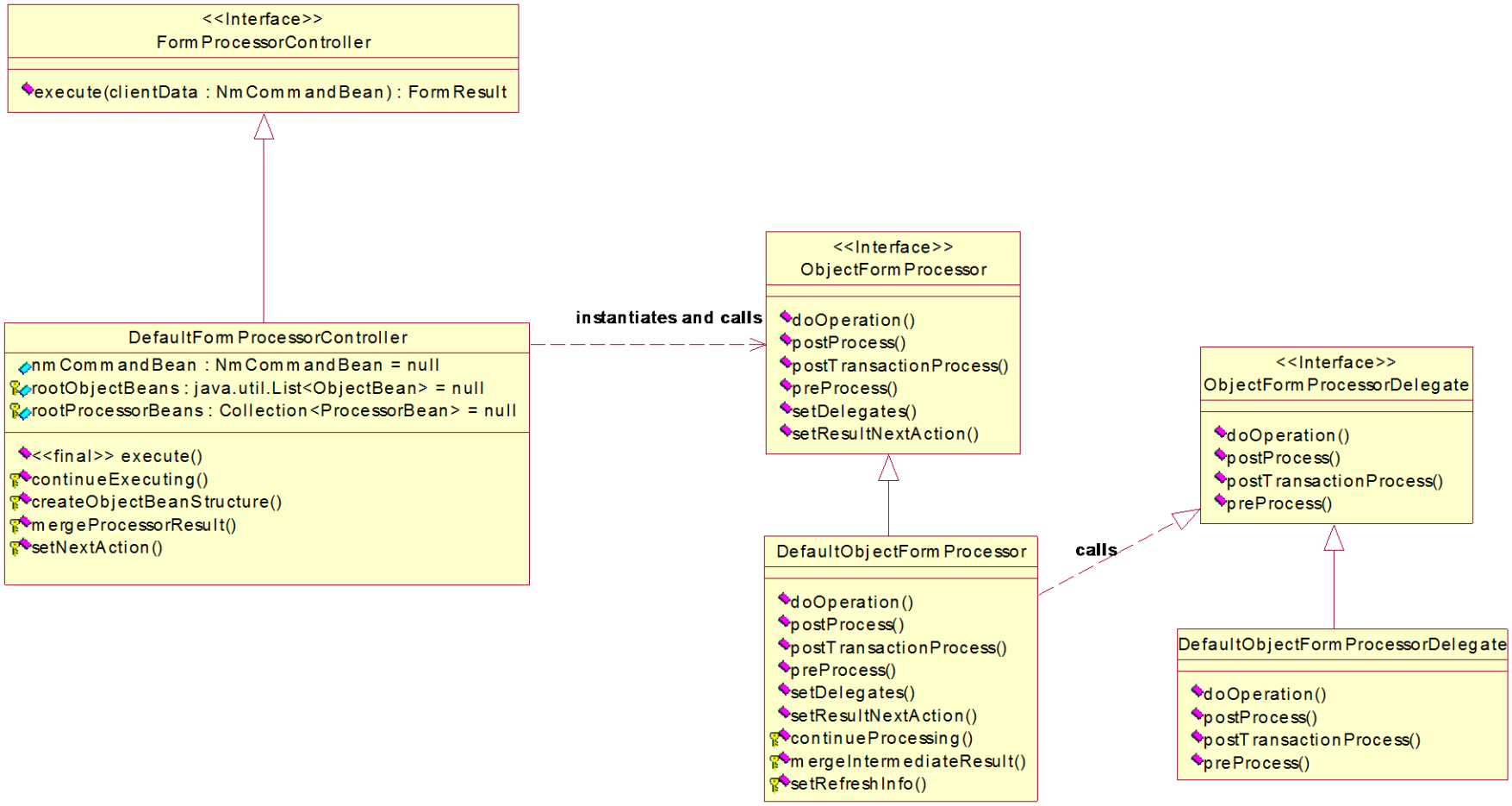
After finishing configure of basic environment configuration, you should design of detail wizard page which is including wizard page design and step design. Following is the detail of wizard basic understanding.



1. First of all, you have to decide designing of each steps on the base wizard. And decide to call which wizard step JSPs.
2. **Basically all step pages will be loaded on the session when loading basic wizard page**. It is seems like all step page is including page of basic wizard although all step page has independent JSP page. When loading wizard page, if one step page has error system can't load total wizard page.
3. If one step want to show page base on previous page's information, in this case, we can set preload flag using "preloadWizardPage" on step configuration which is existed "<action>" tag in "action.xml". However it can be used just one time load. After loaded this page, it can't be control reload although return to previous page and change information.
4. **Each of wizard step is an independent JCA page**.
5. **Each of wizard step configuration is set on "action.xml". Base wizard just will call the action name**.

Modeling of Form Process

Generally form process is including PRE/DO/POST function. By the function, you can control the form process based on time period. When finishing the wizard process, the processor class will return form result. By the result, wizard will work for complete/closing, show warning or stopping.



PTC®

Sample of Form Process

Generally wizard form process will be extended from "**DefaultObjectFormProcess**" class.

```
public class CSCWizardSampleProcess extends CreateObjectFormProcessor {

    /**
     * @param args
     */
    public FormResult doOperation(NmCommandBean nmcommandbean, List list) throws WTException {
        FormResult formresult = new FormResult();

        System.out.println("====== CSCWizardSampleProcess.doOperation() Start =======");

        ObjectBean objectbean = (ObjectBean) list.iterator().next();

        System.out.println("objectbean="+objectbean);
        formresult.setNextAction(FormResultAction.NONE);

        System.out.println("--------------------> FormProcessingStatus TEST = SUCCESS");

        formresult.setStatus(FormProcessingStatus.SUCCESS);
        return formresult;
    }

    public FormResult postProcess(NmCommandBean commandBean, List list) throws WTException {
        FormResult formresult = new FormResult();
        System.out.println("====== CSCWizardSampleProcess.postProcess() Start =======");
        formresult.setStatus(FormProcessingStatus.SUCCESS);
        return formresult;
    }


    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}
```

- Overriding doOperation(), postProcess() or preProcess() FormProcess.
- Generally wizard information will be included on NmCommandBean.
- Form processor function must return form result for the end action of wizard.

Excise

1. Simple Wizard – understanding how to customize
2. Create Wizard within layout
3. Edit Wizard

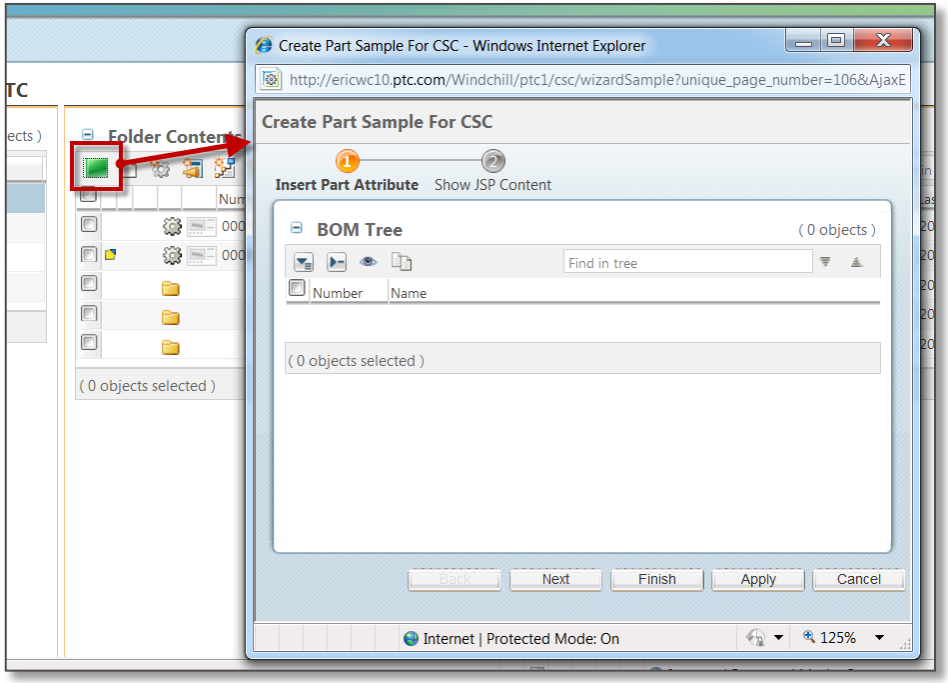
3. Excise 1- Create Simple Wizard

Design of wizard customization

- Add New one toolbar button for calling wizard. The button also include FormProcessor. (add one action and modify toolbar action)
- Create one wizard page (JSP page)
- Create two step page (JSP or MVC page)
- Add two action for step page (add actions) – because wizard page call action name for adding step page on wizard

Customization Action

1. Add action for button
2. Add action for step
3. Create Resource Bundle
4. Modify folder toolbar actionModel
5. Create wizard body JSP
6. Create wizard step MVC & JSP
7. Create Form Processor



Configure step and base wizard

1. Design actions for step and button

On the wizard configuration, you have to set the execution class. And also you have to make an actions for each of step, and configure which JSP or MVC page needs to call from step action.

$WT_HOME/codebase/config/actions/custom-actions.xml

```
<objecttype name="cscwizard" class="" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
    <action name="wizardSamples">
        <command windowType="popup" class="ext.csc.jca.wizard.CSCWizardSampleProcess" method="execute"
url="netmarkets/jsp/csc/wizardSample.jsp"/>
    </action>

    <action name="show_mvc_content">
        <component windowType="wizard_step" name="netmarkets.project.list.table"/>
    </action>
    <action name="show_jsp_content" preloadWizardPage="false">
        <command windowType="wizard_step" url="netmarkets/jsp/csc/showJspContent.jsp"/>
    </action>
</objecttype>
```

<action name="**wizardSamples**">
→ Button action for calling wizard. This action should be set calling back form processor.

<action name="**show_mvc_content**"> <action name="**show_jsp_content**">
→ action configuration for step page

Add button on Folder tool bar

2. Add one button on folder tool bar

For starting wizard, we have to make one button or menu link. This button or menu will include wizard executing process when finished all step of wizard.

$WT_HOME/codebase/config/actions/custom-actionModels.xml

```
<model name="folderbrowser_toolbar_actions">
    <description>Folder browser toolbar actions menu for all Folders.</description>
    <action name="wizardSamples"          type="cscwizard" shortcut="true"/>
    <action name="separator"              type="separator"/>
    <submodel name="folderbrowser_toolbar_open_submenu" />
    <action   name="separator"                    type="separator" />
    <submodel name="folderbrowser_toolbar_new_submenu" />
    <action   name="separator"                    type="separator" />
    <action   name="list_cut"                     type="object" />
     ……
</model>
```

- Copy whole block which name is "***folderbrowser_toolbar_actions***" from "***FolderManagement-actionModels.xml***"
- Add custom action on the folder browser toolbar for calling wizard.
  <action name="wizardSamples" type="cscwizard" shortcut="true"/>
  <action name="seperator" type="seperator"/>
- Don't touch other existed action.

Create Wizard Page

3. Create wizard page

Wizard page including step page configuration, and also wizard page and step page internally will be merged same one page.. So if you have constants or whole environment you can set on wizard page design. Following is just simple wizard source. If you want to add more complexity, you can make your source code. However, you should keep this format at least.

$WT_HOME/codebase/netmarkets/jsp/csc/wizardSample.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ include file="/netmarkets/jsp/components/beginWizard.jspf"%>
<%@ include file="/netmarkets/jsp/components/includeWizBean.jspf"%>

<jca:wizard  title="Create Part Sample For CSC">
      <jca:wizardStep action="show_mvc_content" type="cscwizard" label="Insert Part Attribute"/>
      <jca:wizardStep action="show_jsp_content" type="cscwizard"/>
</jca:wizard>

<%@include file="/netmarkets/jsp/util/end.jspf"%>
```

- For setting wizard, you have to use JCA tag.
  - <jca:wizard> : Configuration for body of wizard
  - <jca:wizardStep> : Configuration for wizard step
- You must use “beginWizard.jspf” instead of “begin.jspf” for wizard page, and also you have to include “includeWizBean.jspf” for sending wizard request to server.

Create Wizard Step

4. Create wizard step

This sample has two wizard step. One will call MVC. Another will call JSP page. In action configuration, we already set "netmarkets.project.list.table". It is OOTB MVC function, so it doesn't need to create new MVC. If you want to add your MVC in the step, you have to design and create for your step.

$WT_HOME/codebase/config/actions/custom-actios.xml

```
<action name="show_mvc_content">
    <component windowType="wizard_step" name="netmarkets.project.list.table"/>
</action>
<action name="show_jsp_content" preloadWizardPage="false">
    <command windowType="wizard_step" url="netmarkets/jsp/csc/showJspContent.jsp"/>
</action>
```

It is OOTB. No need new create

$WT_HOME/codebase/netmarkets/jsp/csc/showJspContent.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ include file="/netmarkets/jsp/components/beginWizard.jspf" %>
<%@ include file="/netmarkets/jsp/components/includeWizBean.jspf" %>

JSP PAGE FOR WIZARD STEP

<%@include file="/netmarkets/jsp/util/end.jspf"%>
```

Note) Although step page included in wizard body, the step also has same architecture like wizard body.

Create Resource Bundle

5. Create Resource Bundle

We set custom resource bundle for displaying action name, so we have to make resource bundle file.

$WT_HOME/codebase/config/actions/custom-actions.xml

```
<objecttype name="cscwizard" class="" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
```

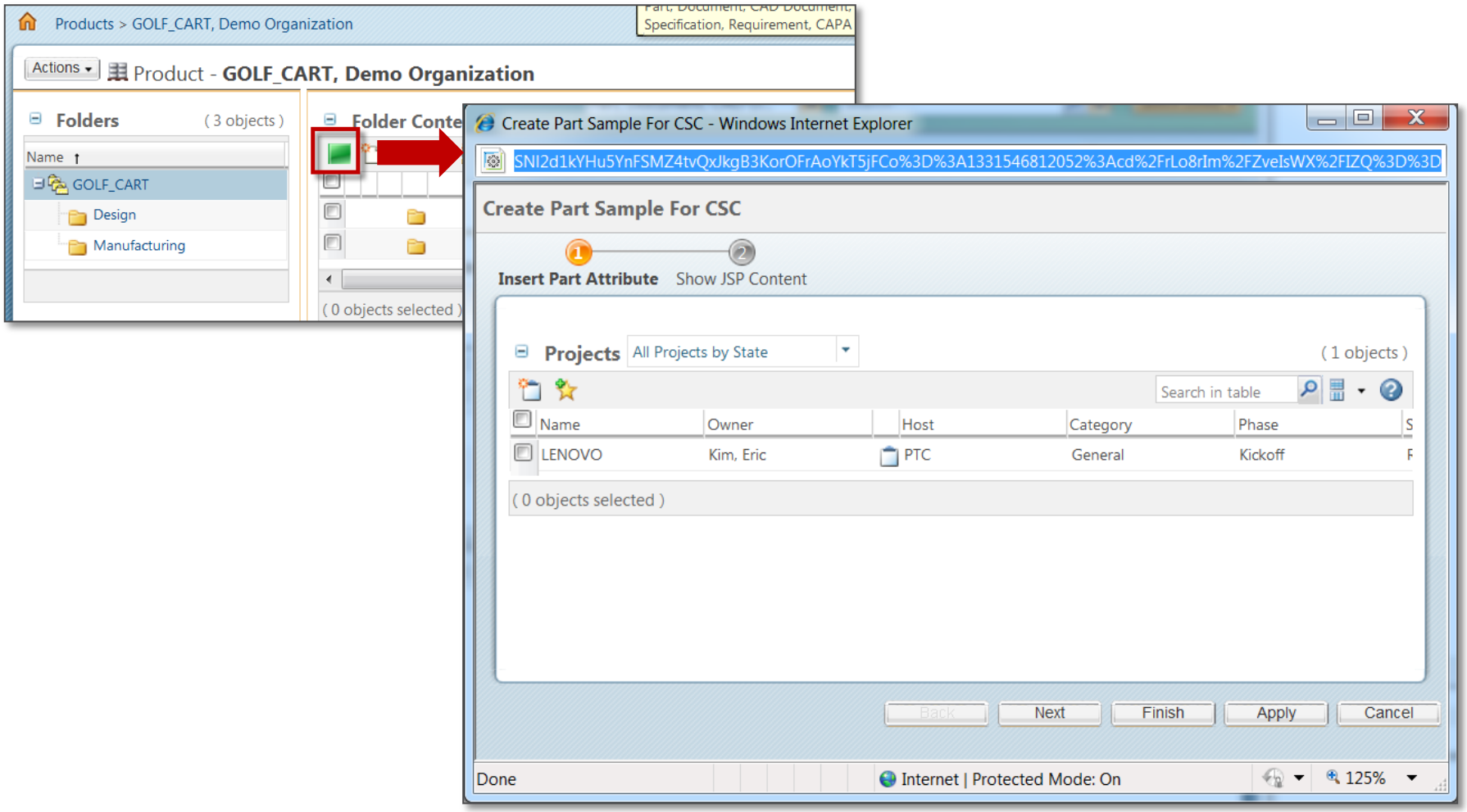
$WT_HOME/codebase/ext/csc/training/resource/CSCTrainingRB.java

```
package ext.csc.training.resource;

import wt.util.resource.RBComment;
import wt.util.resource.RBEntry;
import wt.util.resource.RBUUID;
import wt.util.resource.WTListResourceBundle;

@RBUUID("ext.csc.training.resource.CSCTrainingRB")
public final class CSCTrainingRB extends WTListResourceBundle {
   @RBEntry("Wizard Sample")
    public static final String PRIVATE_CONSTANT_7 = "cscwizard.wizardSamples.description";
    @RBEntry("Wizard-current_step.gif")
    public static final String PRIVATE_CONSTANT_8 = "cscwizard.wizardSamples.icon“;
    @RBEntry("Show MVC Content")
    public static final String PRIVATE_CONSTANT_9 = "cscwizard.show_mvc_content.description";
    @RBEntry("Show JSP Content")
    public static final String PRIVATE_CONSTANT_10 = "cscwizard.show_jsp_content.description";
}
```

3. Excise 1- Create Simple Wizard

Create Wizard Step

6. Restart Service and check result



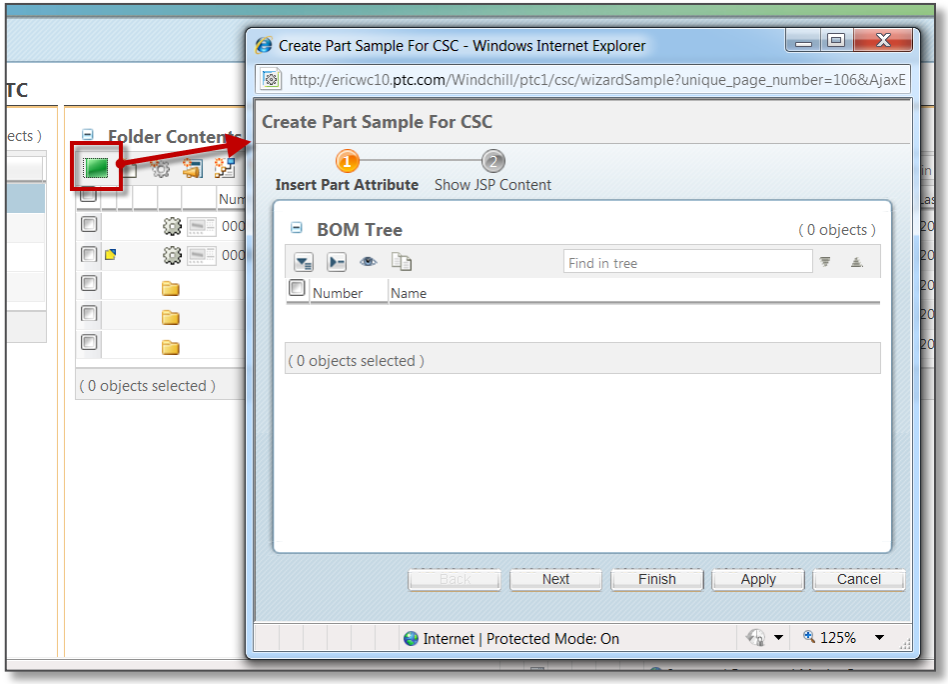
3. Excise 2 – Simple Wizard for Create part

Design of wizard customization

- When creating part, simply it just needs ***number, name and target container***.
- So we will ***add one more icon on folder*** because folder is managed by container. And also when opening wizard, the container information is transferred from folder to wizard. (Existed in NmCommandBean)
- It needs ***one creating form process***.

Customization Action

1. Add action for button
2. Add action for step
3. Create Resource Bundle
4. Modify folder toolbar actionModel
5. Create wizard body JSP
6. Create wizard step JSP
7. Create Form Processor



3. Excise 2 – Simple Wizard for Create part

Configure step and base wizard action

1. Design actions for step and button

We will add one more action button for calling create wizard. And add one step page for input attributes.

$WT_HOME/codebase/config/actions/custom-actions.xml

```
<objecttype name="cscwizard_create_sample" class="" resourceBundle="ext.csc.training.resource.CSCTrainingRB">

        <action name="createWizardSamples" ajax="component">
            <command windowType="popup" class="ext.csc.jca.wizard.CSCCreateWizardSampleProcess"
                method="execute" url="netmarkets/jsp/csc/createWizardSample.jsp"/>
        </action>

        <action name="setPartAttribute">
            <command windowType="wizard_step" url="netmarkets/jsp/csc/setPartAttribute.jsp"/>
        </action>
    </objecttype>
```

“ajax” on button action

Specifies what portion of the parent page should be refreshed when the wizard processing completes.
The value “page” refreshes the entire page (equivalent to not using ajax).
The value “component” refreshes the table from which the wizard was launched.
The value “row” refreshes one or more table rows.

Add button on Folder tool bar

2. Add one button on folder tool bar

Add button on folder toolbar.

$WT_HOME/codebase/config/actions/custom-actionModels.xml

```
<model name="folderbrowser_toolbar_actions">
    <description>Folder browser toolbar actions menu for all Folders.</description>
    <action name="wizardSamples"              type="cscwizard" shortcut="true"/>
    <action name="createWizardSamples"        type="cscwizard_create_sample" shortcut="true"/>
    <action name="separator"                  type="separator"/>
    <submodel name="folderbrowser_toolbar_open_submenu" />
    <action   name="separator"                type="separator" />
    <submodel name="folderbrowser_toolbar_new_submenu" />
    <action   name="separator"                type="separator" />
    <action   name="list_cut"                 type="object" />
     ……
</model>
```

- Copy whole block which name is "***folderbrowser_toolbar_actions***" from "***FolderManagement-actionModels.xml***"
- Add custom action on the folder browser toolbar for calling wizard.
  *<action name="*createWizardSamples*" type="*cscwizard_create_sample*" shortcut="true"/>*
- Don't touch other existed action.

Create Wizard Page

3. Create wizard page

Create wizard page for creating part.

$WT_HOME/codebase/netmarkets/jsp/csc/createWizardSample.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>

<%@ include file="/netmarkets/jsp/components/beginWizard.jspf" %>
<%@ include file="/netmarkets/jsp/components/includeWizBean.jspf" %>

<jca:wizard  title="Create Part">
        <jca:wizardStep action="setPartAttribute" type="cscwizard_create_sample"/>
</jca:wizard>

<%@include file="/netmarkets/jsp/util/end.jspf"%>
```

- Wizard page is simple. It just design for wizard.
- Add step for input attribute

Create Wizard Step

4. Create wizard step (Using HTML Tag)

Create wizard step for inputting attributes

$WT_HOME/codebase/netmarkets/jsp/csc/setPartAttribute.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ taglib prefix="w" uri="http://www.ptc.com/windchill/taglib/wrappers"%>
<%@ include file="/netmarkets/jsp/components/beginWizard.jspf" %>

<table>
        <tr>
            <td scope="row" width="50" class="tableColumnHeaderfont" align="right">*Number:</td>
            <td class="tabledatafont" align="left"> 
                <input type="text" name="null___number___textbox" id="number" maxlength="30" size="10" />
            </td>
        </tr>

        <tr>
            <td scope="row" width="50" class="tableColumnHeaderfont" align="right">Name:</td>
            <td class="tabledatafont" align="left"> 
                <input type="text" name="null___name___textbox" id="name" maxlength="100" size="20"/>
            </td>
        </tr>
</table>

<%@include file="/netmarkets/jsp/util/end.jspf"%>
```

If using HTML tag for input, the tag name must be following style for being read at NmCommandBean.
“null___[attribute name]___textbox”
If you want to use combo box, you must set name like following for NmCommandBean.
“null___[attribute name]___combobox”

3. Excise 2 – Simple Wizard for Create part

Create Wizard Page

4. Create wizard step (Using Wrapper Tag)

Wrapper tag already explained Customization-1 document. Look that document.

$WT_HOME/codebase/netmarkets/jsp/csc/setPartAttribute.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ taglib prefix="w" uri="http://www.ptc.com/windchill/taglib/wrappers"%>
<%@ include file="/netmarkets/jsp/components/beginWizard.jspf" %>

<table>
        <tr>
            <td scope="row" width="50" class="tableColumnHeaderfont" align="right">*Number:</td>
            <td class="tabledatafont" align="left"> 
                <w:textBox name="number" id="number" maxlength="30" size="10" />
            </td>
        </tr>

        <tr>
            <td scope="row" width="50" class="tableColumnHeaderfont" align="right">Name:</td>
            <td class="tabledatafont" align="left"> 
                <w:textBox name="name" id="name" maxlength="100" size="20"/>
            </td>
        </tr>
</table>

<%@include file="/netmarkets/jsp/util/end.jspf"%>
```

Create Resource Bundle

5. Create Resource Bundle

We set custom resource bundle for displaying action name, so we have to make resource bundle file.

$WT_HOME/codebase/config/actions/custom-actions.xml

```
<objecttype name="cscwizard_create_sample" class="" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
```

$WT_HOME/codebase/ext/csc/training/resource/CSCTrainingRB.java

```
package ext.csc.training.resource;

import wt.util.resource.RBComment;
import wt.util.resource.RBEntry;
import wt.util.resource.RBUUID;
import wt.util.resource.WTListResourceBundle;

@RBUUID("ext.csc.training.resource.CSCTrainingRB")
public final class CSCTrainingRB extends WTListResourceBundle {
   // Wizard Excise 2
    @RBEntry("Create Part Wizard")
    public static final String PRIVATE_CONSTANT_11 = "cscwizard_create_sample.createWizardSamples.description";
    @RBEntry("part_add.gif")
    public static final String PRIVATE_CONSTANT_12 = "cscwizard_create_sample.createWizardSamples.icon";
    @RBEntry("Set Part Attributes")
    public static final String PRIVATE_CONSTANT_13 = "cscwizard_create_sample.setPartAttribute.description";
}
```

3. Excise 2 – Simple Wizard for Create part

PTC

Create Form Process

6. Create form process for creating part

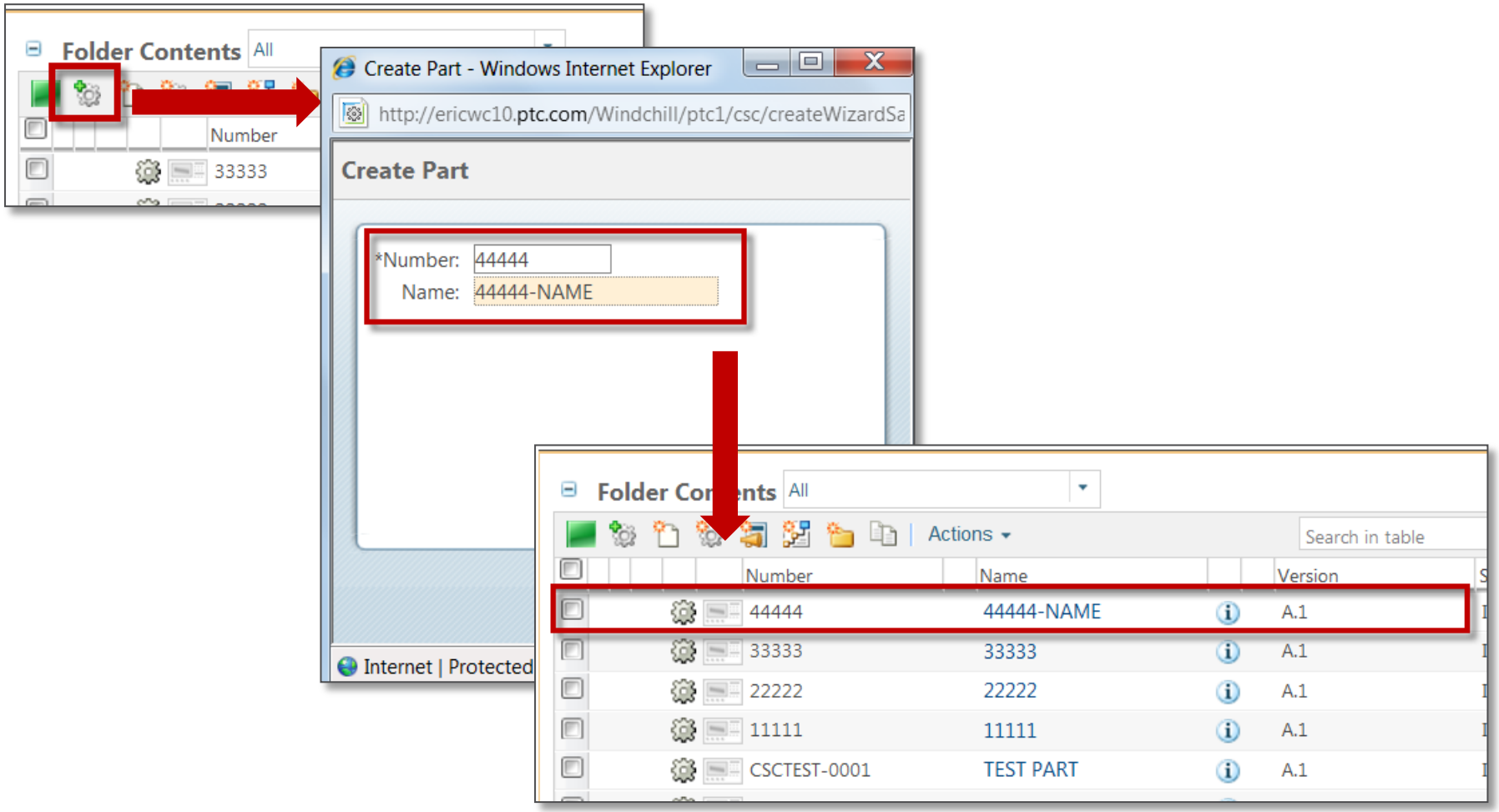
$WT_HOME/codebase/ext/csc/jca/wizard/CSCCreateWizardSampleProcess.java

```
package ext.csc.jca.wizard;
import java.util.HashMap;
…..
public class CSCCreateWizardSampleProcess extends DefaultObjectFormProcessor {
   @Override
   public FormResult doOperation(NmCommandBean bean, List list) throws WTException {
      FormResult result = new FormResult();
      try {
         WTPart part = WTPart.newWTPart();
         HashMap map = bean.getText();
         Object[] keys = map.keySet().toArray();
         for( Object oneSet : keys) {
               if (((String)oneSet).contains("number")) {
               part.setNumber((String)map.get(oneSet));
            } else if (((String)oneSet).contains("name")) {
               part.setName((String)map.get(oneSet));
            }
         }
         WTContainer container = bean.getContainer();
         part.setContainer(container);
         PersistenceHelper.manager.save(part);
      } catch(WTPropertyVetoException pve ) {
         result.setStatus(FormProcessingStatus.FAILURE);
         return result;
      } catch(WTException wte) {
         result.setStatus(FormProcessingStatus.FAILURE);
         return result;
      }
      result.setStatus(FormProcessingStatus.SUCCESS);
      return result;
   }
```

3. Excise 2 – Simple Wizard for Create part

Create Wizard Step

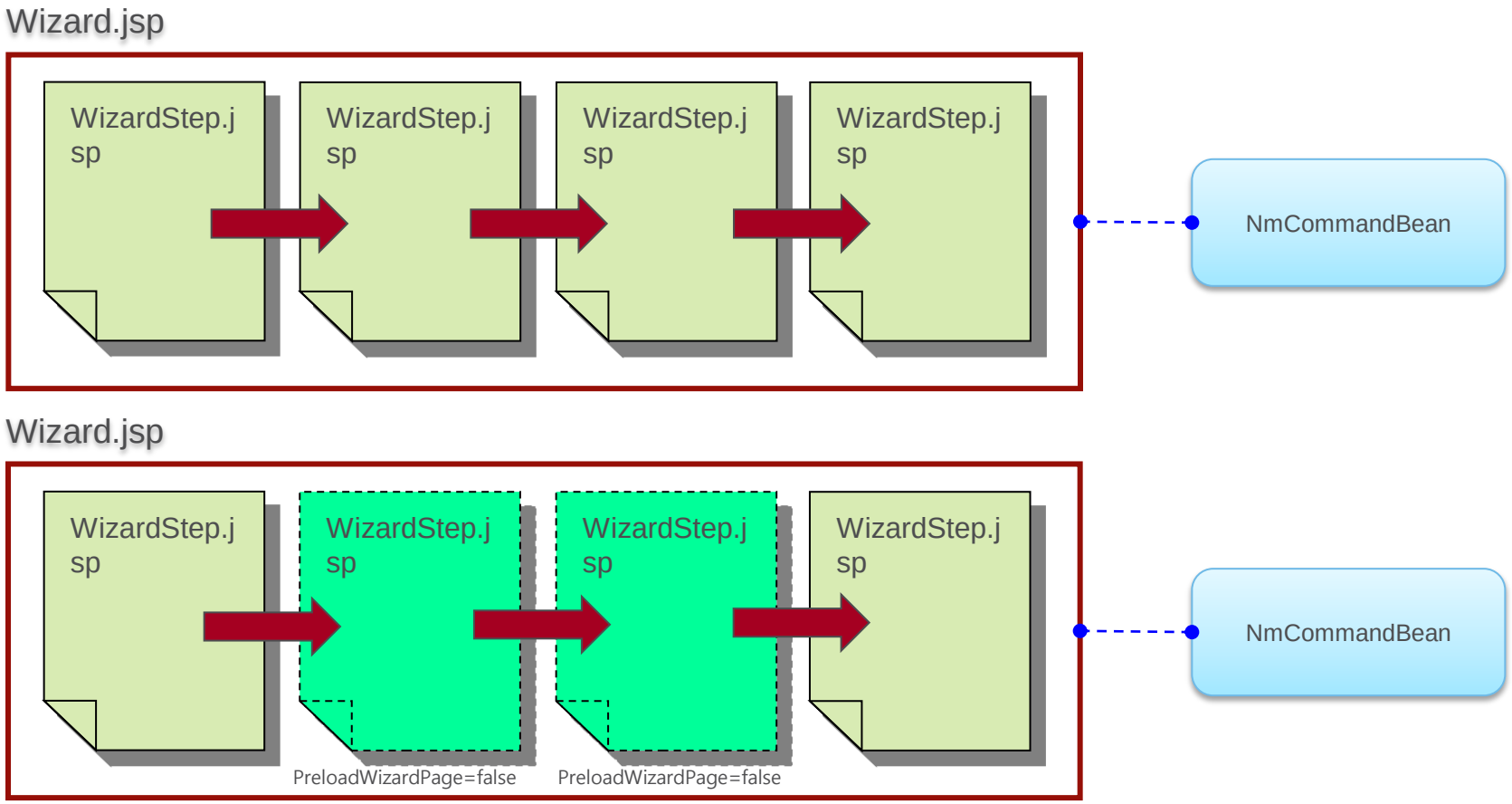
7. Restart Service and check result



3. Excise 3 – Receive previous step info

Understand for pre load option

When you set no configuration for wizard step, whole wizard step will loaded at opening wizard. Each of step can not transfer changed status to other step because next step page is already loaded on the servlet server. For transferring previous changed value and showing next page based on previous information, wizard can use "preloadWizardPage". But it is just for one time action.

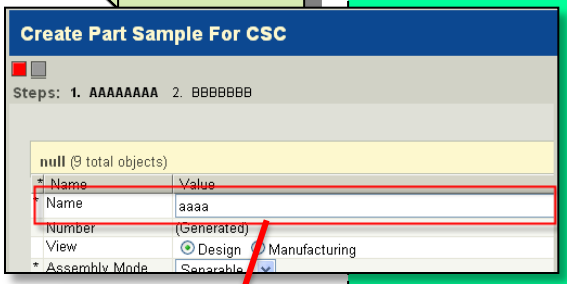


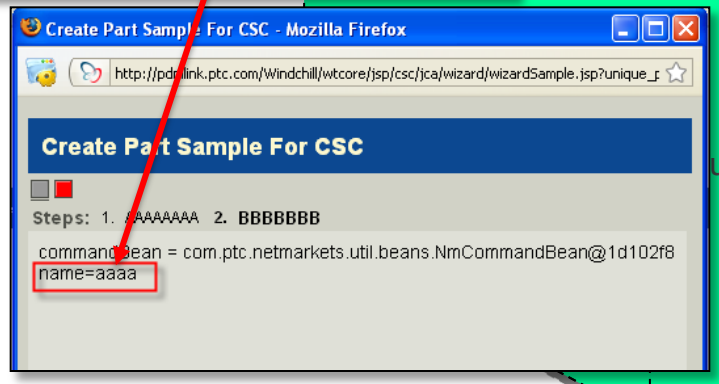
3. Excise 3 – Receive previous step info

Understand for pre load option

If step is pre-load option, Servlet will refresh NmCommandBean before loading next page. So next page can read changed information of previous pages

WizardStep.jsp






```
<%
out.println("commandBean = " + commandBean+"<BR>");
//HashMap pMap = commandBean.getParameterMap();
HashMap pMap = commandBean.getText();

Set aSet = pMap.keySet();
Object[] aArray = aSet.toArray();
Collection bSet = pMap.values();
Object[] bArray = bSet.toArray();

String elementName = "";

for( int i=0; i < aArray.length; i++) {
    if ( ((String)aArray[i]).indexOf("name") > -1 ) {
        out.println("name" + "=" + bArray[i] + "<BR>");
        elementName = (String)aArray[i];

ut.println("elementName="+elementName);
    }
```

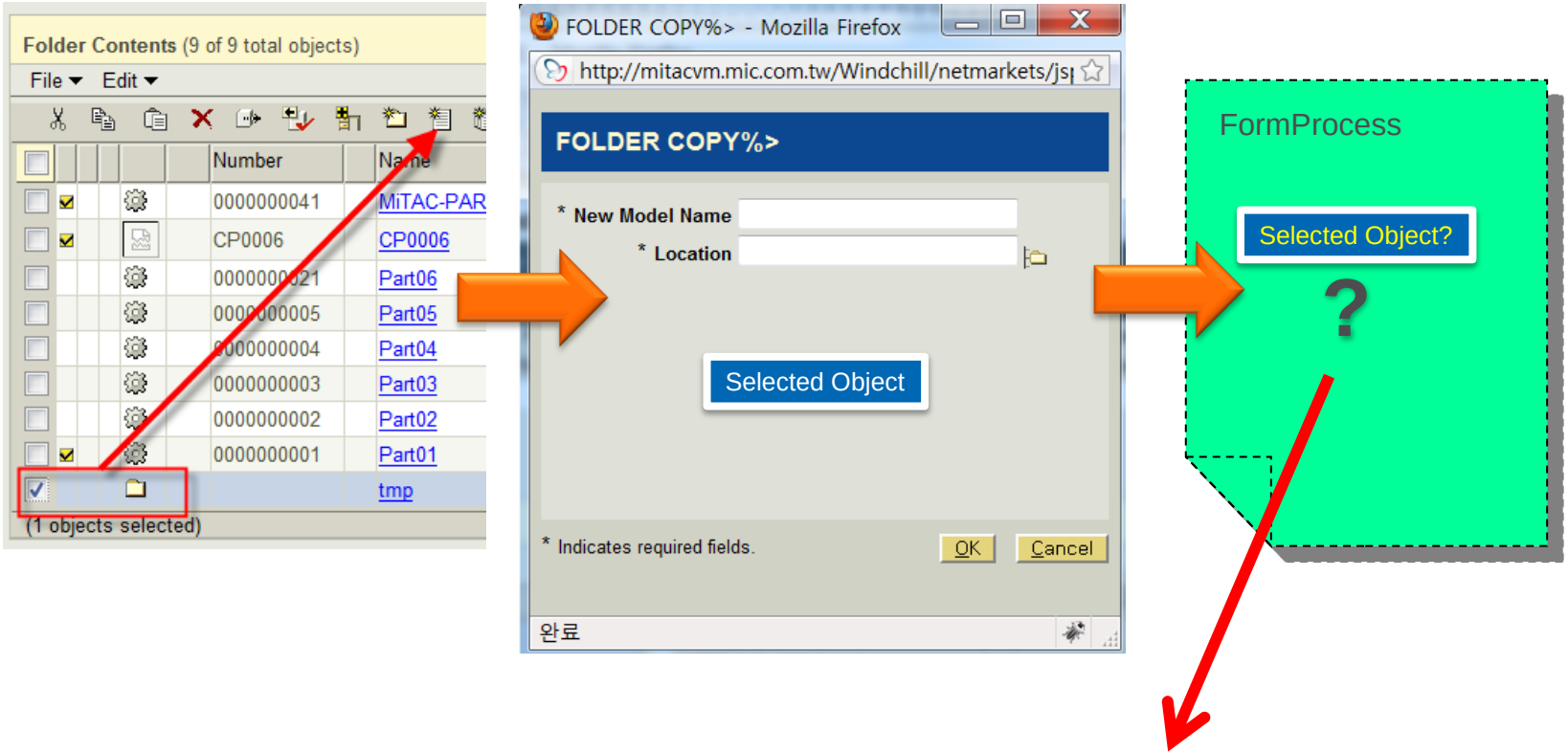
You have to pick up correct function
Base on previous pages, you have to change the function for each of components.

Finding using name
Windchill are not using inputted name by UI component engine. But basically the component include your inputted name, so you have to use "indexOf" function for inputted name.

3. Excise 4 – Get object instance on from processor

Getting selected object instance on form process

After selecting object on the table and then calling wizard, we need selected object instance. In this case, we can get instance object on wizard JSP. However we are not clear how to get on the form processor if we need to get selected object instance on form processor look following sample.



ArrayList aList = nmcommandBean.getNmOidSelectedInOpener();

Picker Customization

Understand WC10 picker

1. Customization of picker methodology basically wasn't changed on WC 10.
2. Picker is not customization deeply. It is most of configuration.
3. Windchill core has several picker component. You just choose usable picker for correct requirement. (If you want to make especial picker component, it is very complex customization. If possible, avoid that customization)
4. Following is the readied system component for generally used.
   - User Picker
   - Organization Picker
   - Context Picker
   - Item Picker
   - Type Picker
   - Participant Picker
5. For above picker, WC10 is using 3 kinds of tags like following.
   - <%@ taglib prefix="wctags" tagdir="/WEB-INF/tags" %>
   - <%@ taglib uri="http://www.ptc.com/windchill/taglib/picker" prefix="p"%>
   - <%@ taglib uri="http://www.ptc.com/windchill/taglib/components" prefix="jca"%>

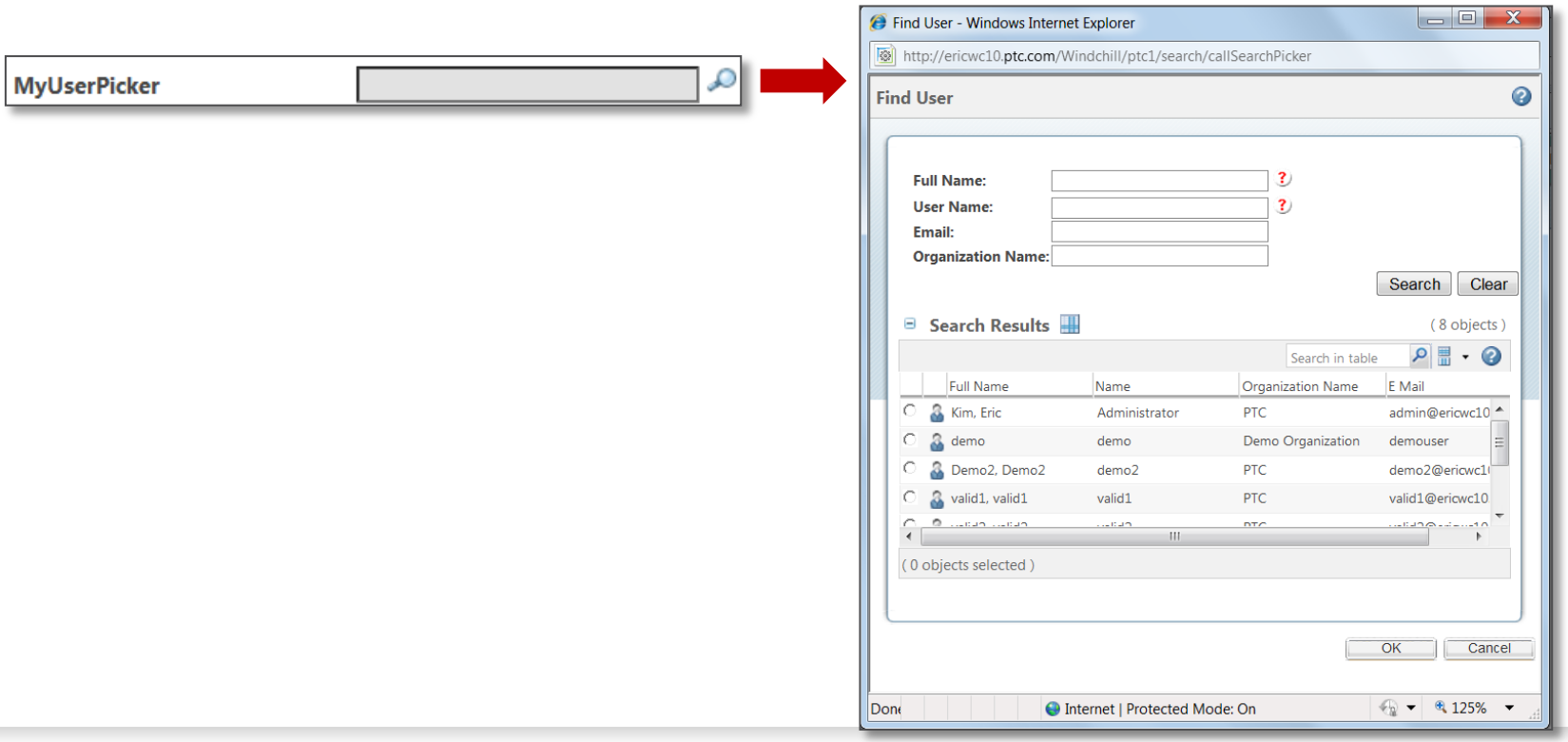
2. Review Picker Components

User Picker

The user picker is used when you have a requirement to select specific user(s) depending upon certain criteria and use them in your application. Typical use case could be that you may want to find parts which are created by certain user. In this case you can have a user picker and then through this you can select the user and pass it on to the search criteria to perform search for parts.

- **User Picker**

```
<wctags:userPicker id="testUserPicker" label="MyUserPicker" />
```



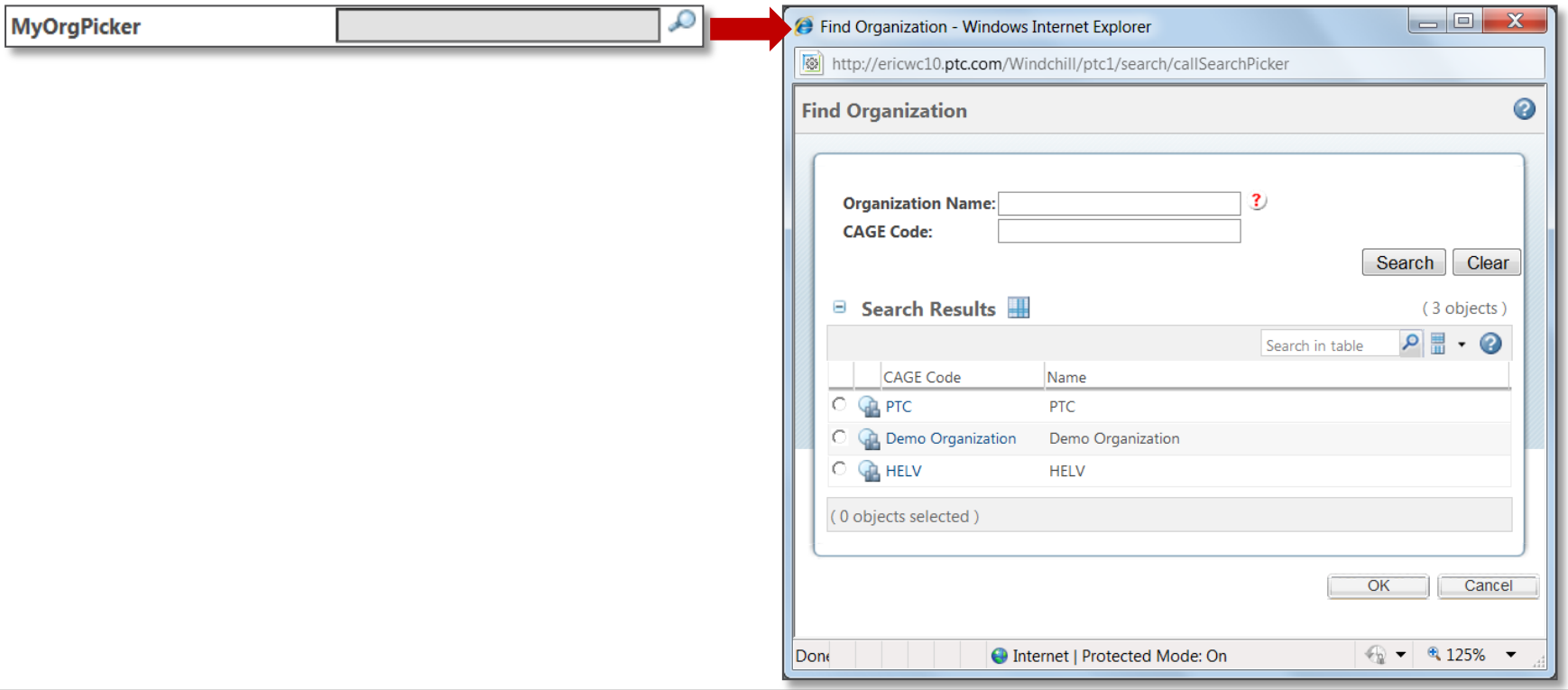
2. Review Picker Components

Organization Picker

The organization picker is used when you have a requirement to select specific organization(s) depending upon certain criteria and use them in your application. Typical use case could be that you may want to find parts that are available in particular organization. In this case, you can have a organization picker and then through this you can select the organization and pass it on to the search criteria to perform search for parts.

- **Organization Picker**

```
<wctags:organizationPicker id="orgPicker" label="MyOrgPicker"/>
```



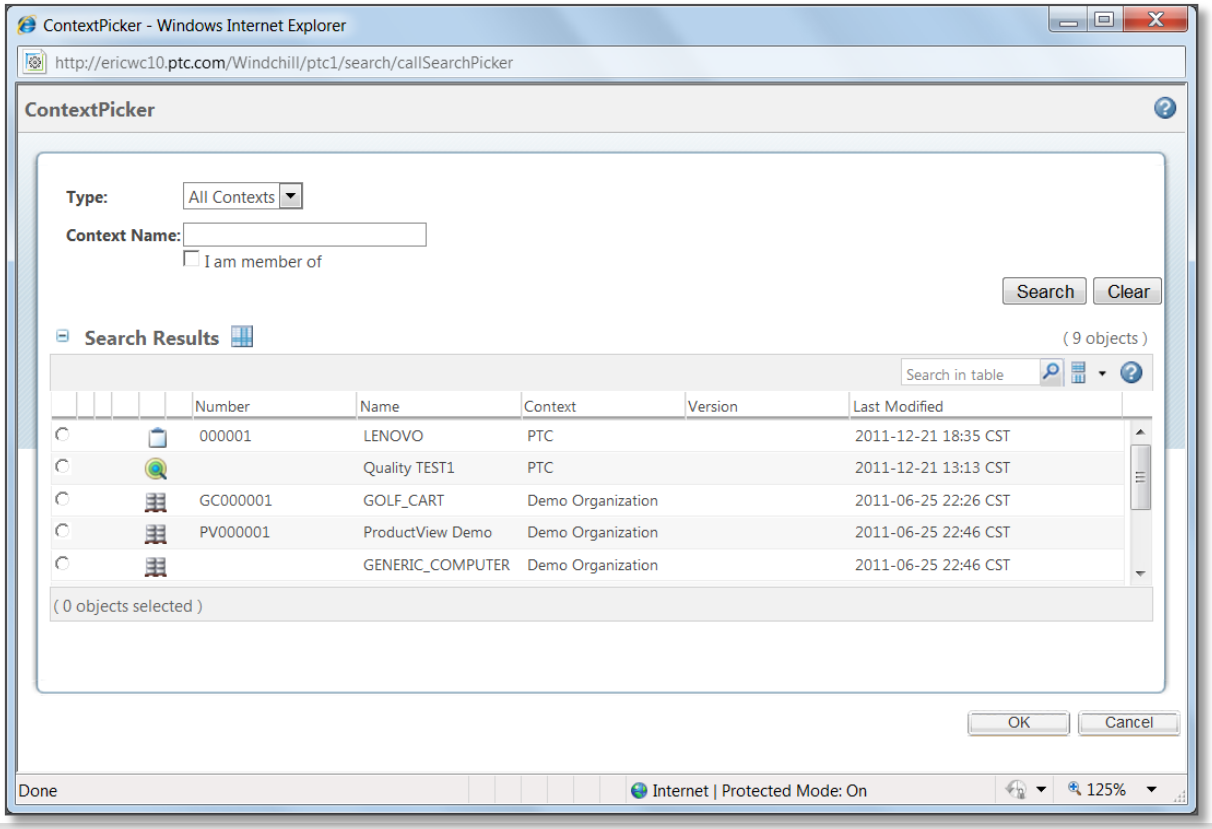
2. Review Picker Components

Context Picker

The context picker is used when you have a requirement to perform an operation that is based on certain context.

- **Context Picker**

```
<wctags:contextPicker id="contextPicker" label="MyContextPicker" pickerTitle="ContextPicker" />
```



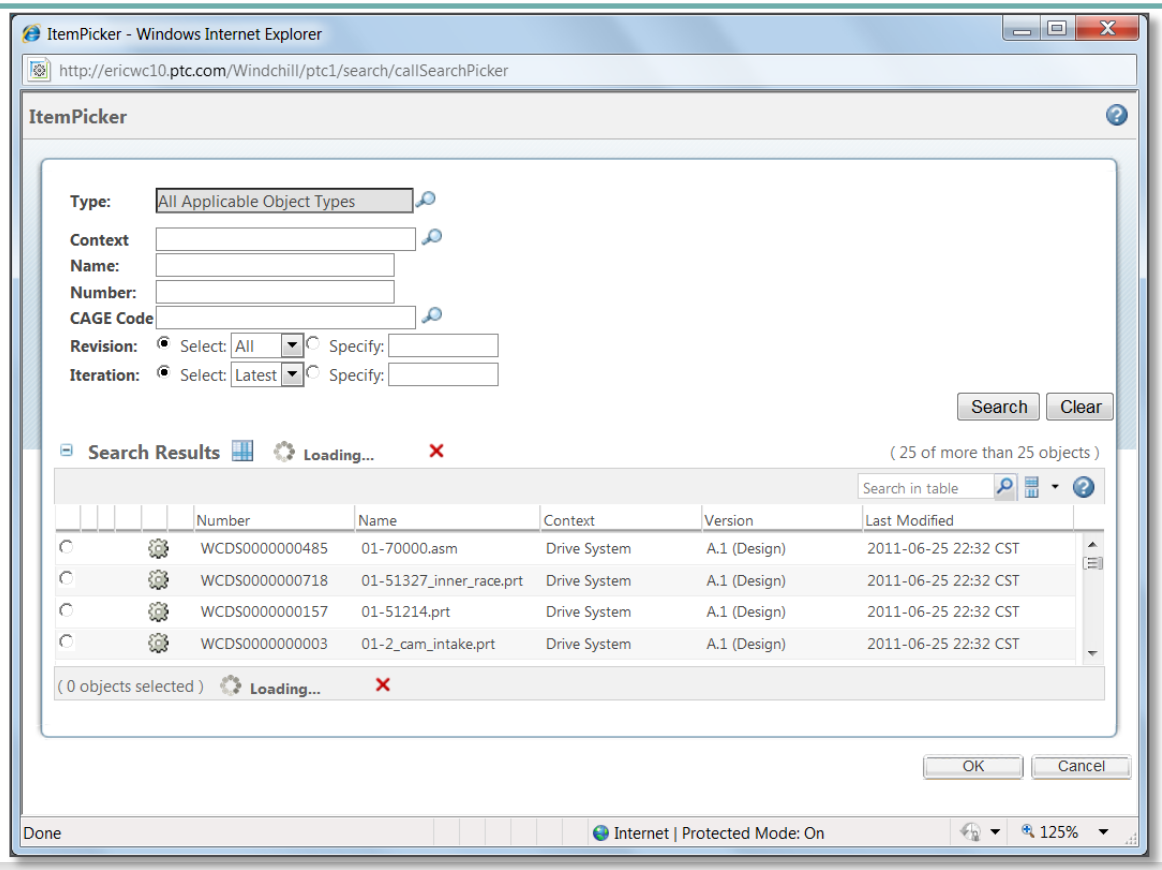
2. Review Picker Components

Item Picker

The item picker is used when you have a requirement to select specific business object(s) depending upon certain criteria and use them in your application.

- **Item Picker**

<wctags:itemPicker id="itemPicker" label="MyItemPicker" pickerTitle="ItemPicker"/>

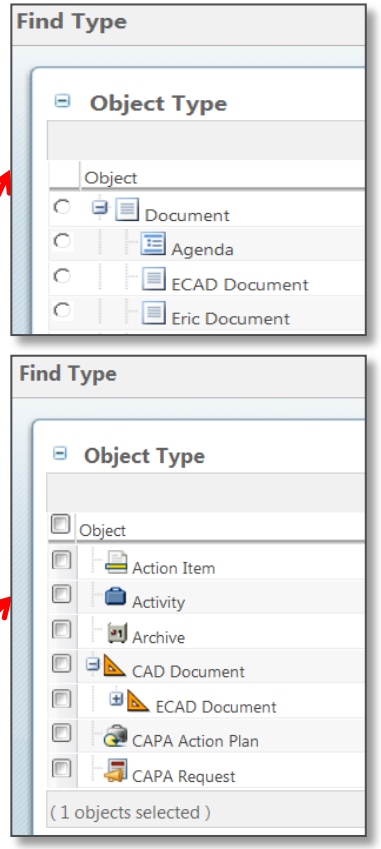


2. Review Picker Components

Type Picker

Type Picker Common Component is to be used either for display or for assignment of type-able items. For example in a search application to select the type of objects that you are interested to do a search on or in a create application to create an item of a specific type. It can be used to display the type in case of edit or view applications. The component can be used in the context/mode of CREATE, EDIT, SEARCH or VIEW.

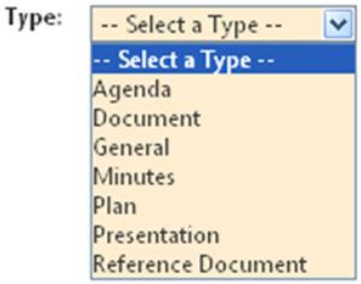
```
<%
    NmCommandBean cb = new NmCommandBean();
    cb.setCompContext(nmcontext.getContext().toString());
    String containerRef_name = null;
    if(cb.getContainerRef() != null )
        containerRef_name = cb.getContainerRef().toString();
%>
    <c-rt:set var="b" value="<%=containerRef_name%>"/>


<!-- TYPE PICKER TAG START -->

        <p:typePicker id="mytypepicker" label="MyTypePicker" mode="SEARCH">
            <p:pickerParam name="format"    value="tree" />
            <p:pickerParam name="componentId" value="Foundation.partDocSearch" />
            <p:pickerParam name="type"      value="BOTH" />
            <p:pickerParam name="displayHierarchy" value="true" />
            <p:pickerParam name="showRoot"  value="false" />
            <p:pickerParam name="containerRef"  value="${b}" />
        </p:typePicker>

    </tr>

    <tr>
    <p:typePicker id="typepicker2" label="Multi TypePicker with seedType">
        <p:pickerParam name="format"    value="tree" />
        <p:pickerParam name="select"    value="multi" />
        <p:pickerParam name="displayHierarchy"  value="false" />
        <p:pickerParam name="showRoot"  value="false" />
        <p:pickerParam name="defaultType"   value="wt.doc.WTDocument" />
        <p:pickerParam name="seedType"  value="wt.fc.Persistable" />
        <p:pickerParam name="type"  value="BOTH" />
    </p:typePicker>
</tr>
```

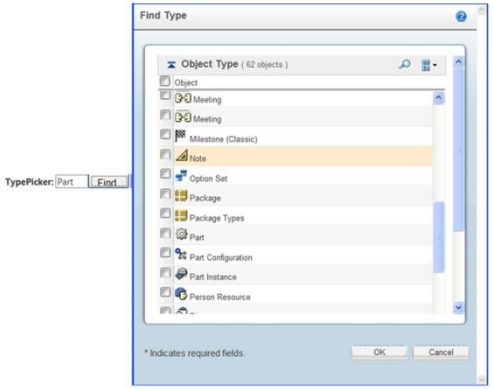


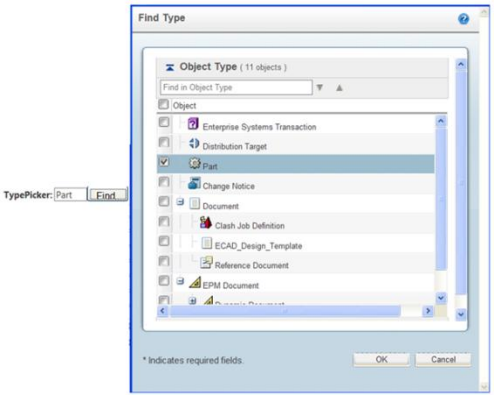
2. Review Picker Components

Format of Type Picker

Type Picker has several type of format like following Tree Type, Table Type and Drop Down







- **Type Picker in "Drop Down" format**

```
< p:typePicker id="typePickerTest" label="Type : "  mode="SEARCH" >
    <p:pickerParam name="format"   value="dropdown" />
</p:typePicker>
```

- **Type Picker in "table" format**

```
< p:typePicker id="typePickerTest" label="Type : "  mode="SEARCH" >
    <p:pickerParam name="format"   value="table" />
</pr:typePicker>
```

- **Type Picker in "tree" format**

```
< p:typePicker id="typePickerTest" label="Type : "  mode="SEARCH" >
    <p:pickerParam name="format"   value="tree" />
</p:typePicker>
```

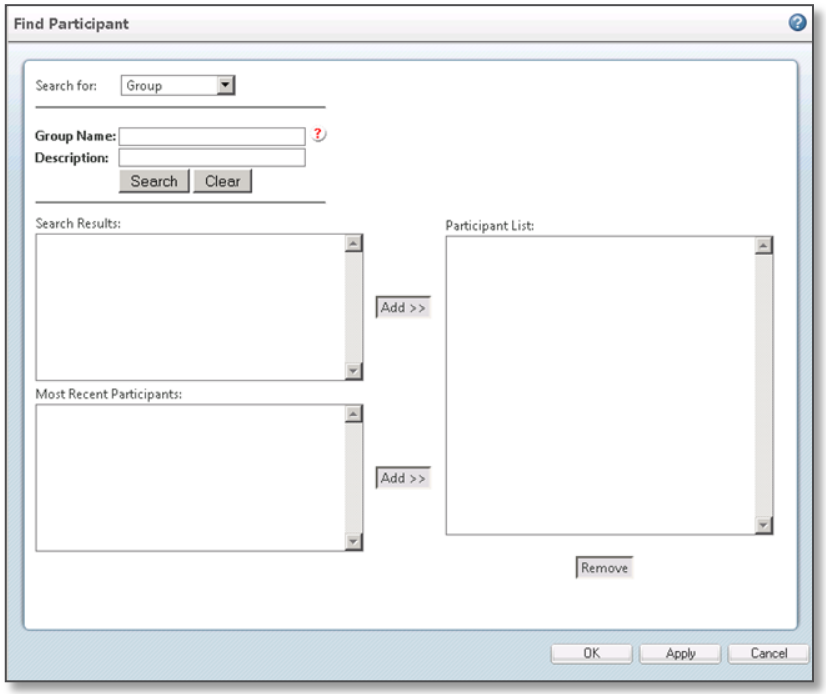
2. Review Picker Components

Participant Picker

The current way of picking of participants is limited to Users, Groups, and is not consistent across Windchill. The new Participant Picker Common Component gives consistent behavior across Windchill. Participant Picker gives you the ability to search Participants of type User, Group, and Organization. It provides a wide variety of search scope criteria using which you can narrow down the search.

- **Participant Picker**

<jca:participantPicker actionClass="com.ptc.netmarkets.principal.CustomPrincipalCommands" actionMethod="addPrincipal" participantType="<%= PrincipalBean.GROUP %>"> > </jca:participantPicker>



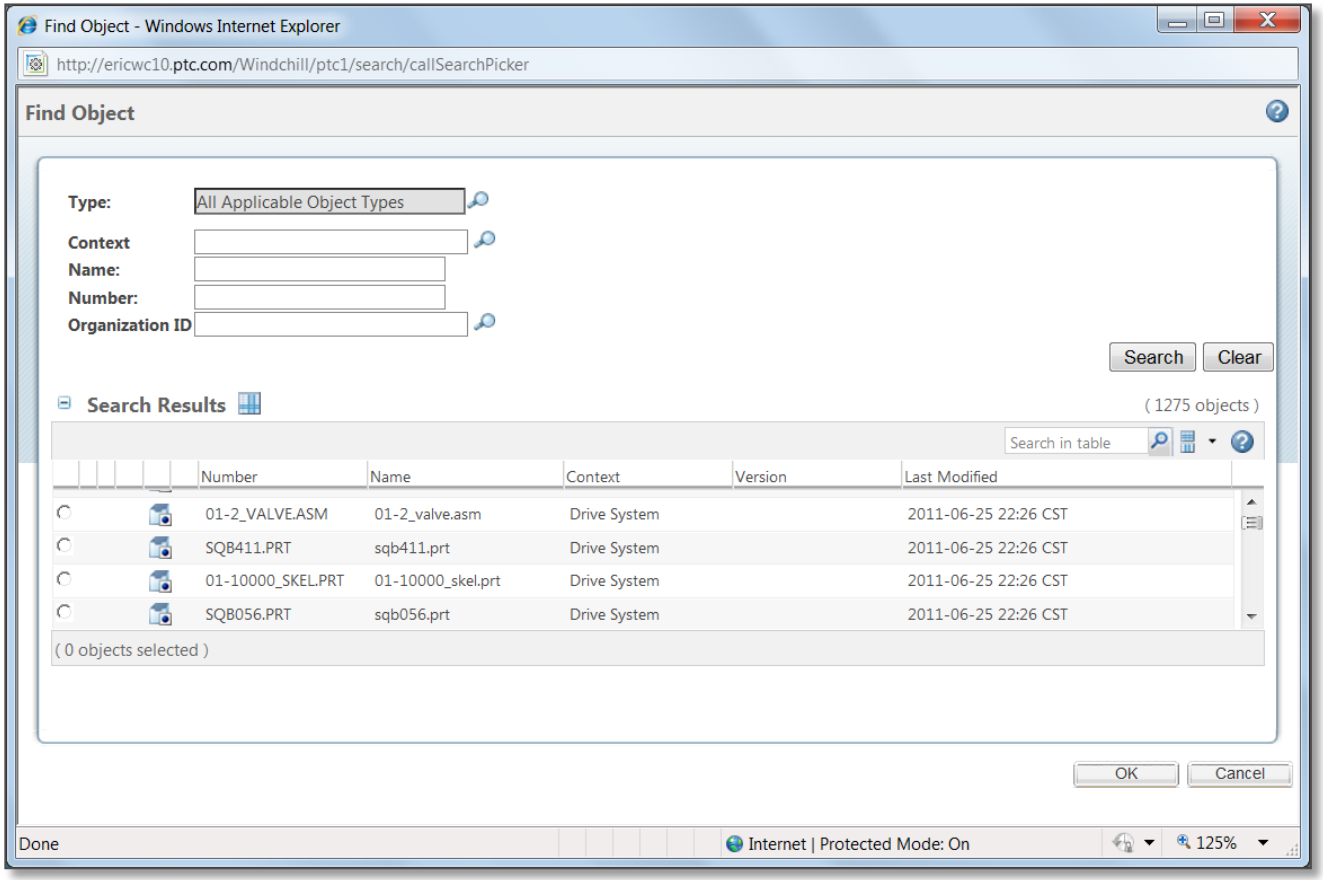
2. Review Picker Components

Item Master Picker

Item Master picker just try to find mastered object.

- **Item Master Picker**

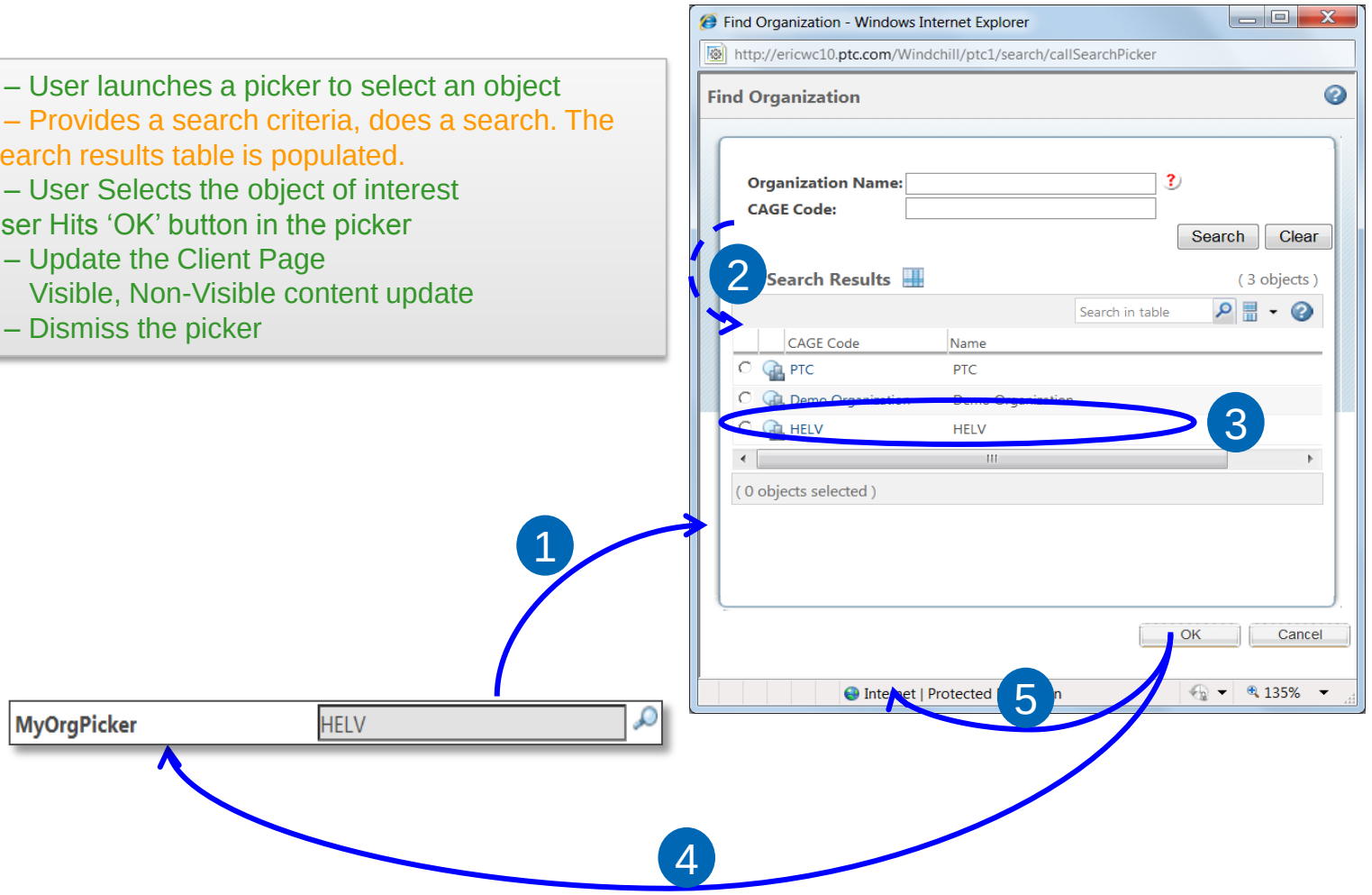
<wctags:itemMasterPicker id="itemMasterPicker" label="MyItemMasterPicker"/>



Picker Process

Following is the process of basic pickers. All of JCA picker using same architecture.

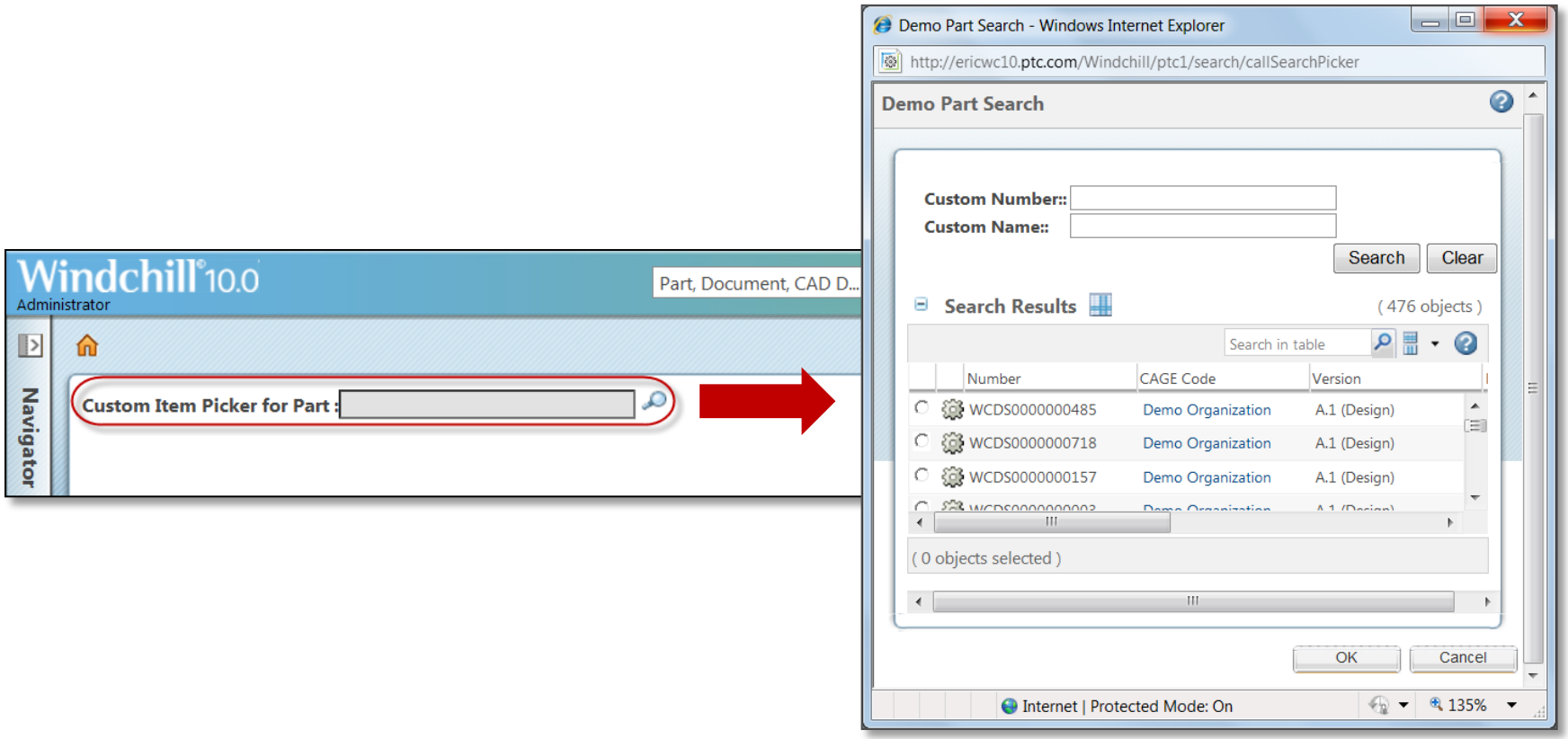
1 – User launches a picker to select an object
2 – Provides a search criteria, does a search. The Search results table is populated.
3 – User Selects the object of interest
User Hits 'OK' button in the picker
4 – Update the Client Page
Visible, Non-Visible content update
5 – Dismiss the picker



4. Excise 1 – Item Picker

Design of Item Picker

- Item Picker can be designed by customer like following area.
  - Search Criteria
  - Target Search Object
- Other pickers is designed by system, so you just call target picker using tag.



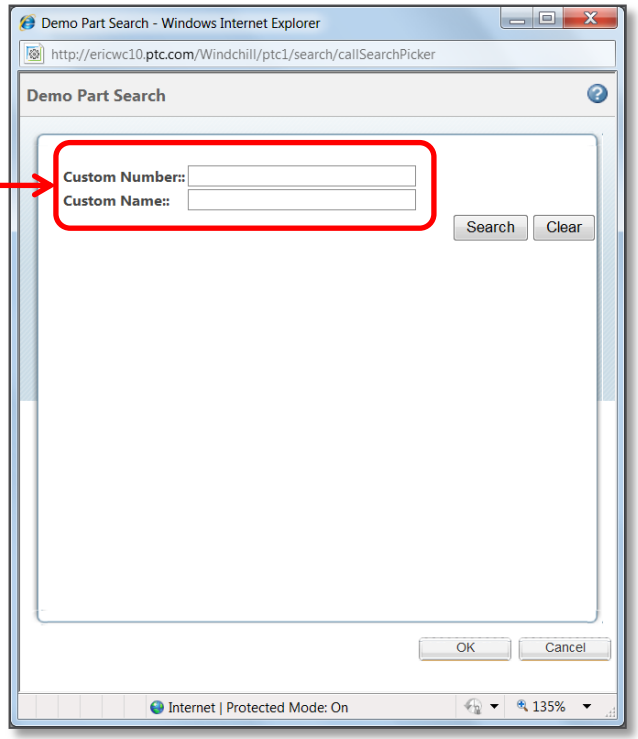
1. Create search criteria set – Item Picker

Before you create picker popup window, you should set search attribute set.

- The Set file is located in "**$WT_HOME/codebase**" and the file name is "**pickerAttributes.xml**".
- Following sample will be using for searching WTPart, so objectType is set "wt.part.WTPart".
- Attribute name should be used object's model or IBA logical name. The name can check at Property Report.
- Each of Display name is registered in ResourceBundle, so the filed name is the bundle's key.

```
<ComponentID id="cscPartPicker">
   <ObjectType id="wt.part.WTPart">
      <SearchCriteriaAttributes>
         <Attributes>
            <Name>number</Name>
            <DisplayName>NUMBER_LABEL</DisplayName>
            <IsSearchable>true</IsSearchable>
         </Attributes>
         <Attributes>
            <Name>name</Name>
            <DisplayName>NAME_LABEL</DisplayName>
            <IsSearchable>true</IsSearchable>
         </Attributes>
      </SearchCriteriaAttributes>
   </ObjectType>
</ComponentID>
```

This id will use in tag
"componentId=cscPartPicker"



4. Excise 1 – Item Picker

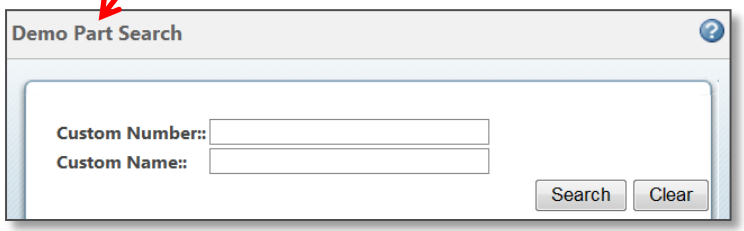
2. Create Picker page (1)

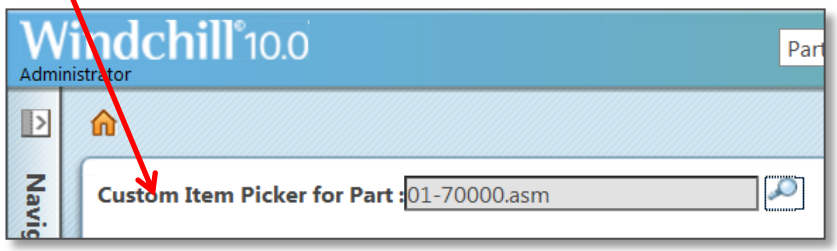
Previously you made picker component, so you will make the picker base(parent page) using this component.

- Picker base(parent page) is same architecture to create JCA page, so you should set all taglib and include.
- Additionally you should set “wctags” taglib.

```
<%@ taglib prefix="wctags" tagdir="/WEB-INF/tags" %>

<%@ taglib uri="http://www.ptc.com/windchill/taglib/components" prefix="jca"%>

<jca:tabToHighlight actionName="pickerSamples" objectType="csc" />
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

                                                    sp/util/begin.jspf"%>

<wctags:itemPicker id="cscPickerForPart" label="Custom Item Picker for Part"
        pickerTitle="Demo Part Search" showVersion="false"
        objectType="wt.part.WTPart" showTypePicker="false"
    componentId="cscPartPicker" pickerCallback="partPickerCallback“
/>

<%@ include file="/netmarkets/jsp/util/end.jspf"%>
```

This is the component in “pickerAttributes.xml”





2. Create Picker page (2)

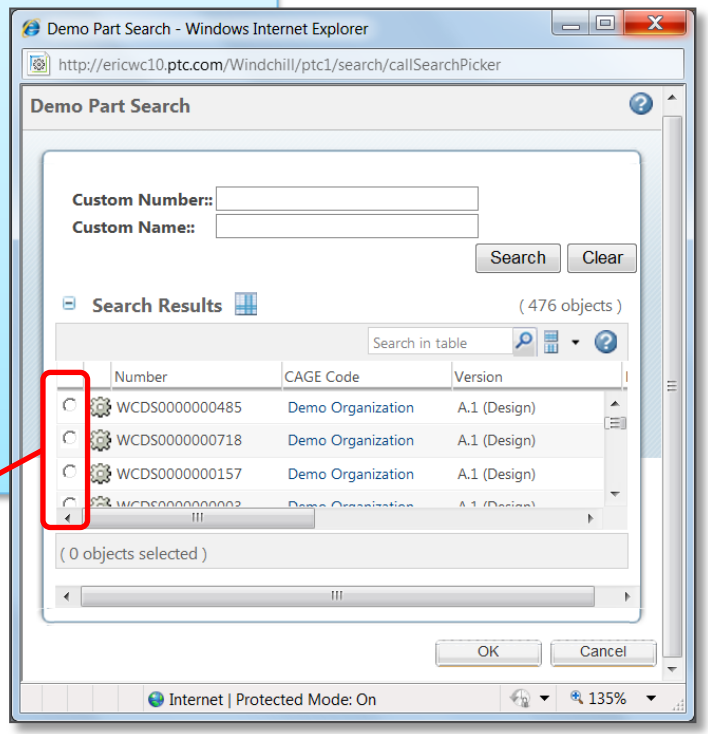
Previously you made picker component, so you will make the picker base(parent page) using this component.

- Picker base(parent page) is same architecture to create JCA page, so you should set all taglib and include.
- Additionally you should set “wctags” taglib.

```
<%@ taglib prefix="wctags" tagdir="/WEB-INF/tags" %>

<%@ taglib uri="http://www.ptc.com/windchill/taglib/components" prefix="jca"%>

<jca:tabToHighlight actionName="pickerSamples" objectType="csc" />
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ include file="/netmarkets/jsp/util/begin.jspf"%>

<wctags:itemPicker id="cscPickerForPart" label="Custom Part Picker"
      pickerTitle="Demo Part Search" showVersion="false"
      objectType="wt.part.WTPart" showTypePicker="false"
      componentId="cscPartPicker" pickerCallback="partPickerCallback“
/>

<%@ include file="/netmarkets/jsp/util/end.js
```

If you just want to find especial part, enter like “WCTYPE|wt.part.WTPart|com.ptc.kyc.Material”

If you want to use multi-select, you Should add “multiselect=true” inside of tag.



4. Excise 1 – Item Picker

3. Picker resource bundle

The resource bundle is located in “$WT_HOME/src/com/ptc/windchill/enterprise/search/client”. Resource Bundle file name is “searchClientResource.java”.

- Copy all java code to eclipse compile directory
- If you are using not existed locale, add locale java code which is copied from “searchClientResource.java”
- Finally add custom resource on java code.

```
package com.ptc.windchill.enterprise.search.client;

import wt.util.resource.*;

@RBUUID("com.ptc.windchill.enterprise.search.client.searchClientResource")
public final class searchClientResource extends WTListResourceBundle {
        …….
        @RBEntry("Custom Number")
        @RBComment("Custom Number")
        public static final String CUSTOM_NUMBER_LABEL = "CUSTOM_NUMBER_LABEL";

        @RBEntry("Custom Name:")
        @RBComment("Custom Name")
        public static final String CUSTOM_NAME_LABEL = "CUSTOM_NAME_LABEL";
}
```

4. Picker call back function

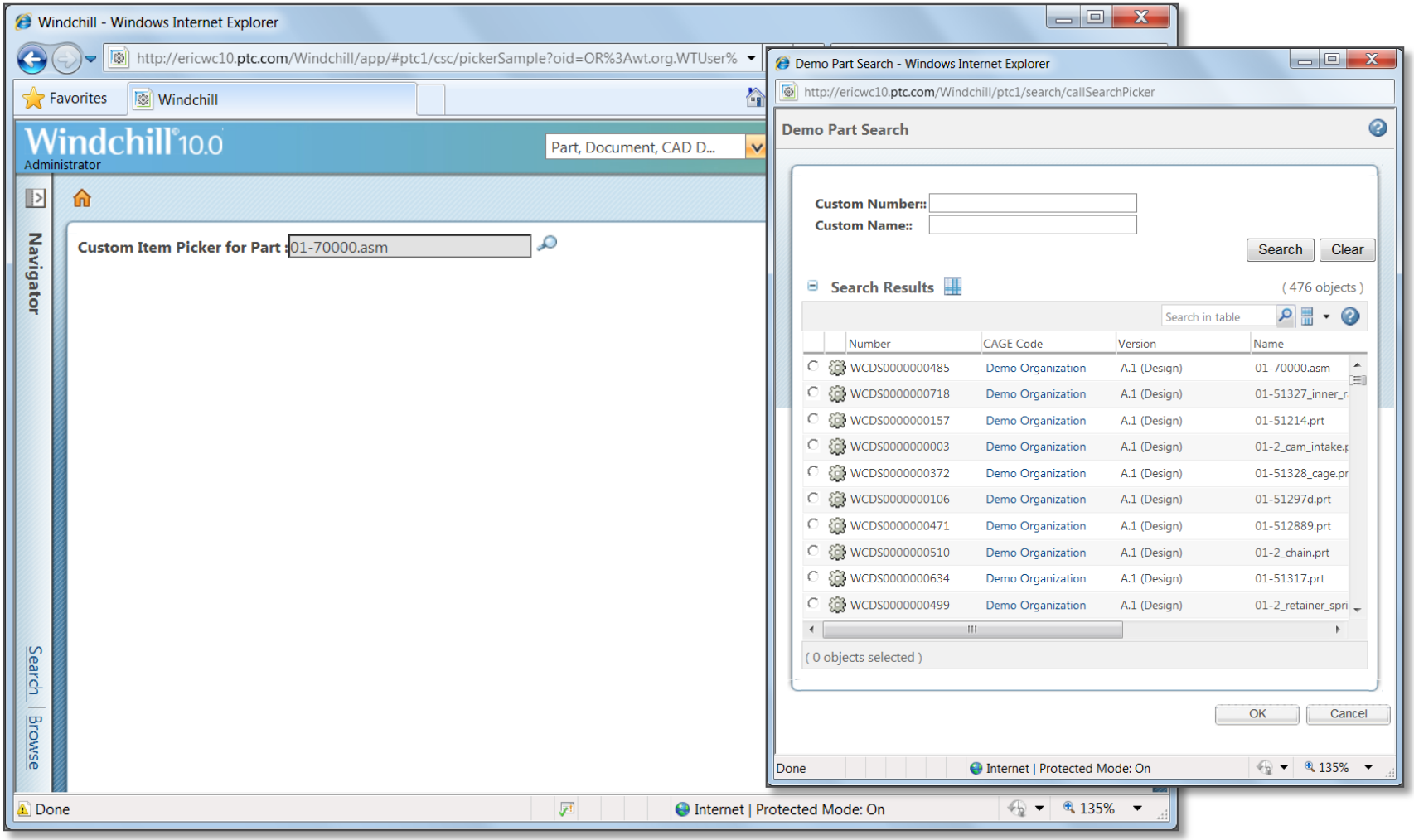
Previous JSP page should be included Javascript function for picker call back. The picker popup page will return to parent page's JS function about selected object information. The picker will call the function when you create picker tag within pickerCallBack's name.

```
<script>
function partPickerCallback (objects, pickerID)
{
                document.getElementById(pickerID +"$label$").setAttribute("value",objects.pickedObject[0].name);
                document.getElementById(pickerID +"$label$___old").setAttribute("value",objects.pickedObject[0].name);
}
</script>

<wctags:itemPicker id="cscPickerForPart" label="Custom Part Picker"
       pickerTitle="Demo Part Search" showVersion="false"
       objectType="wt.part.WTPart" showTypePicker="false"
      componentId="cscPartPicker" pickerCallback="partPickerCallback“
/>
```

4. Excise 1 – Item Picker

5. Result

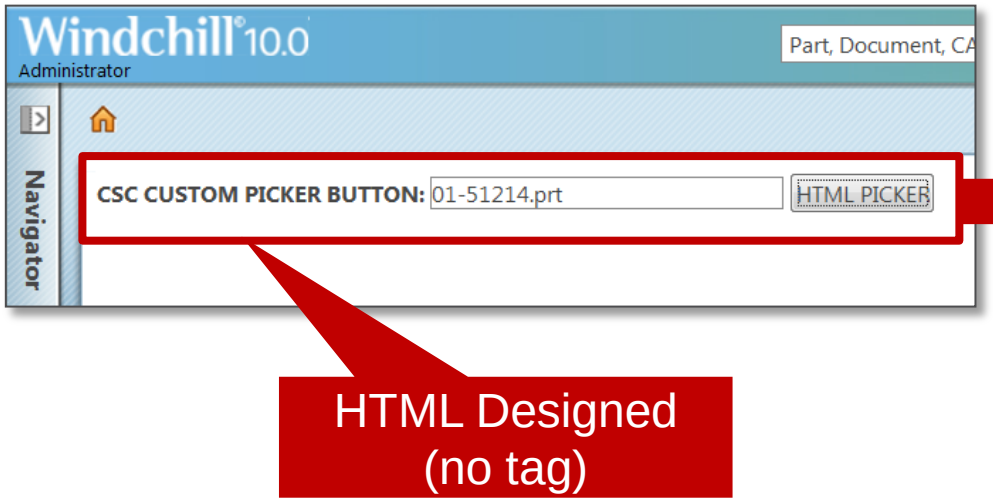
Following is the result.

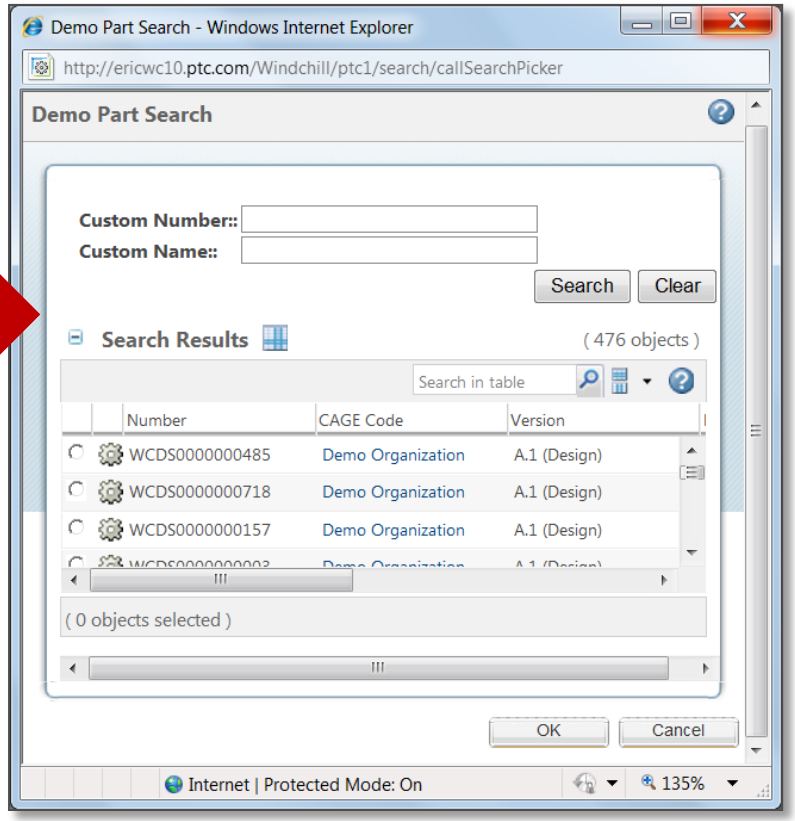


4. Excise 2 – Custom Picker button

Design of Custom picker button

- Sometimes customer want to change picker button and input field for customer standard.
- In this case we can not use Windchill tag rendering.
- This excise will explain for how custom button call picker window.



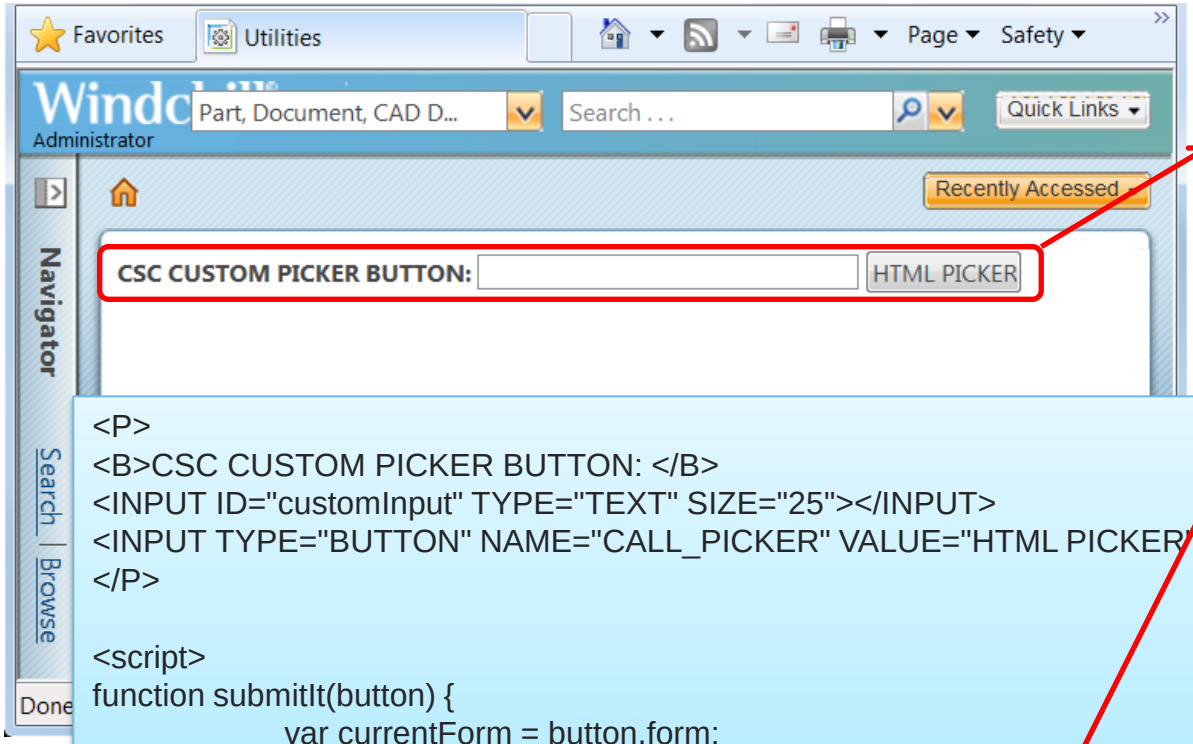


4. Excise 2 – Custom Picker button

1. Create main page

pickerSample.jsp

If you want to make following area by using HTML and openwindow Javascript function. You have to create picker window. But it is very simple, the picker window also are using same tag just add “inline” tag.



- When you call your picker page, add one required parameter need. “**portlet=poppedup**”.
- It means it is not display navigation bar and full Windchill page.
- Except on required, you can design by yourself.

```
<P>
<B>CSC CUSTOM PICKER BUTTON: </B>
<INPUT ID="customInput" TYPE="TEXT" SIZE="25"></INPUT>
<INPUT TYPE="BUTTON" NAME="CALL_PICKER" VALUE="HTML PICKER" ONCLICK="submitIt(this);"></INPUT>
</P>

<script>
function submitIt(button) {
        var currentForm = button.form;
        var customInput = document.getElementById("customInput").value;
        window.open('/Windchill/ptc1/csc/htmlPicker?portlet=poppedup','',width=500 height=400');
}
</script>
```

4. Excise 2 – Custom Picker button

2. Create Picker page

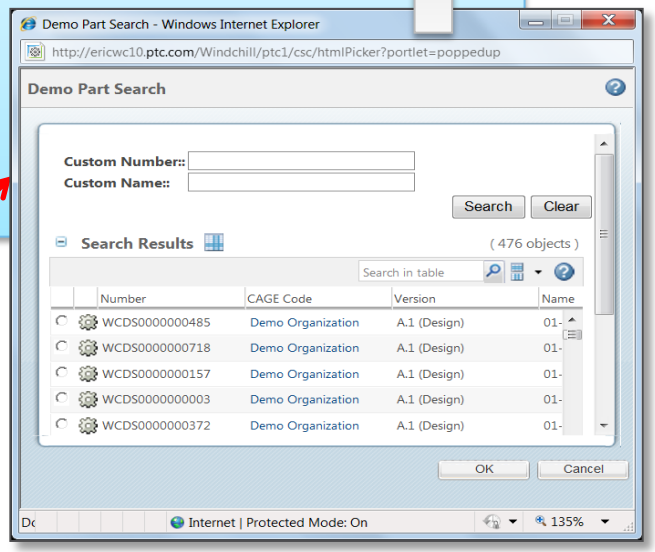
htmlPicker.jsp

Create your picker JSP page using “wctags”. Almost same to using previous material. But the tag do not display input and button. It just display picker main page directly.

- You should choose which kind of picker type using, and assign picker using **“wctags”.**
- Everything is same, but it does not need button and input filed. So you have to add one field “**inline=true**”.
- This page will be opened picker, so this page doesn’t need to render navigation bar. By this result you have to use the begin page of wizard. **“<%@ include file="/netmarkets/jsp/components/beginWizard.jspf"%>”**

```
<%@ taglib prefix="wctags" tagdir="/WEB-INF/tags" %>
<%@ include file="/netmarkets/jsp/components/beginWizard.jspf"%>

<wctags:itemPicker id="cscPickerForPart1" inline="true" label="Custom Part Picker"
       pickerTitle="OTHER DEMO Part Search" showVersion="false"
       objectType="wt.part.WTPart" showTypePicker="false"
      componentId="cscPartPicker" pickerCallback="partPickerCallback1“
/>

<%@ include file="/netmarkets/jsp/util/end.jspf"%>
```

r/open_picker.jsp?portlet=poppedup



4. Excise 2 – Custom Picker button

3. Callback function on main page.

The Callback Javascript can make on parent page, and the function name will use in pickerCallback parameter in “wctags”.

- The callback function is same type.

```
<script type="text/javascript">
<!--
function partPickerCallback1(objects, pickerID)
{
		alert(objects.pickedObject[0].name);
		alert(objects.pickedObject[0].oid);
		alert(pickerID);
		document.getElementById("customInput").setAttribute("value",objects.pickedObject[0].name);
}

//-->
</script>
```

```
<P>
<B>CSC CUSTOM PICKER BUTTON: </B>
<INPUT ID="customInput" TYPE="TEXT" SIZE="25"></INPUT>
<INPUT TYPE="BUTTON" NAME="CALL_PICKER" value="CALL PICKER" ONCLICK='submitIt(this);'>
</P>
```

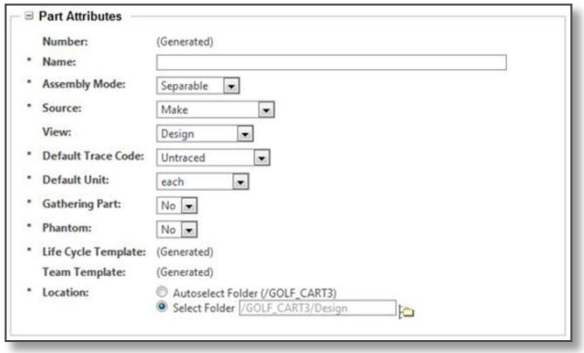
Attribute Panel

Background

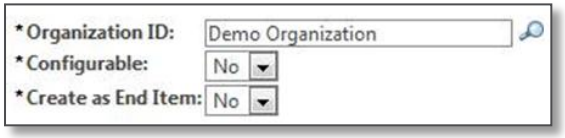
Windchill provides an attribute panel component that can be used to display attributes of a Windchill object. Typically, these are used to display attributes in read-only mode on object info pages or in edit mode in wizards. Attribute panels can be configured to be a simple list of name and value pairs, or they can be configured to have a more complex layout with:

- A border
- Attribute groups, each with a title and expand-collapse capability
- Multiple columns

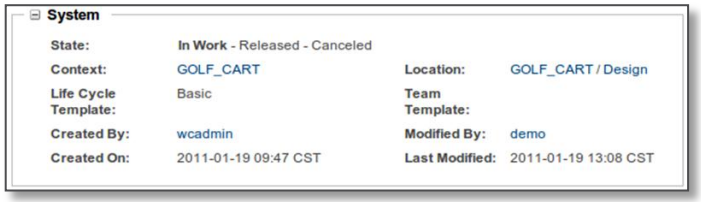
Read-Only attribute Panel



Editable advanced attribute panel



Editable attribute panel



Advanced view-only attribute panel

Outcome

- **Scope/Applicability/Assumptions**

  The best practices described in this document should be used when you want to display attributes of a single Windchill object for viewing or editing. Both hard and soft attributes can be displayed. The attribute panel object may or may not be TypeManaged, however to configure the panel from a layout it must be TypeManaged.

- **Intended Outcome**

  After reading this document you should be able to create simple and advanced attribute panels in either edit or view mode. Specifically, you will learn how to:

  - create advanced attribute panels using Type Manager layouts
  - create simple and advanced attributes panels by manually configuring the panels in a Java builder class
  - create simple attribute panels using JSP tags
  - include your panel in a JSP page
  - convert custom attribute panels/tables created in Windchill 9.0 or later to Windchill 10.0 attribute panels.

2. Implementation of Attribute panel

Overview(1)

Both simple and advanced attribute panels can be created using Java builder classes. Simple attribute panels can also be created using only JSP tags in certain limited cases. When a builder class is used to create a panel for a TypeManaged object, the contents and configuration of the panel can be defined in a layout in the Type and Attributes Manager and retrieved by the builder class. Alternatively, the contents and configuration of the panel can also be defined in the builder itself.

- **Use a layout-based based panel created by a Java builder whenever possible. This provides:**
  - easier customization
  - a UI for ease of configuration
  - greater reusability
  - a more consistent product
- **Use a non-layout based panel created by a Java builder if:**
  - The object type being displayed is not TypeManaged or
  - The type is TypeManaged but layouts are already defined for all related screen types and none are appropriate for your purpose
- **Use a JSP-tag based panel only if:**
  - You want to create a simple attributes panel and
  - You only want to display a few attributes which don't require complex configurations and
  - You don't want the overhead of creating a builder class

2. Implementation of Attribute panel

Overview(2)

If you are creating a new Java business object type, we recommend your class implement the TypeManaged interface to take advantage of the layouts feature.

The system provides OOTB builders for several layout-based advanced attribute panels used for common, shared actions. These actions are:

Action name	Action Object Type	Component Builder Id	Description
visualizationAndAttributes	object	ComponentId.VIS_AND_ATTRIBUTES	Used to display the “Visualization and Attributes” panel on information pages
primaryAttrbutes	object	ComponentId.PRIMARY_ATTRIBUTES	Used to display the “Visualization and Attributes” panel on information pages
attributes	object	ComponentId.ATTRIBUTES_ID	Used to display the “More Attributes” panel on information pages

To display these panels you just specify the appropriate component builder id in the URL or action used to generate the panel on your page. You do not need to write any Java code or JSPs. The system also provides OOTB builders for the layout-based “Attributes” panels in object create and edit wizards that can be displayed by using the common step actions for wizards.

To display a panel for a different layout or create a manually-configured panel you can extend an existing Java builder class and make new MVC class for attribute display. (Next Page)

Custom Decision

The decision diagram below provides general guidelines for how you should determine which builder of MVC to use for your panel, with references to the section of this document that describes the builder

Attribute Panel custom type:

1. **MVC implementation**
2. **JCA implementation (JSP tag)**

Sample code for super class:

VisualizationAttributesBuilder

CadPrimaryAttributesBuilder.java

PrimaryAttributesBuilder

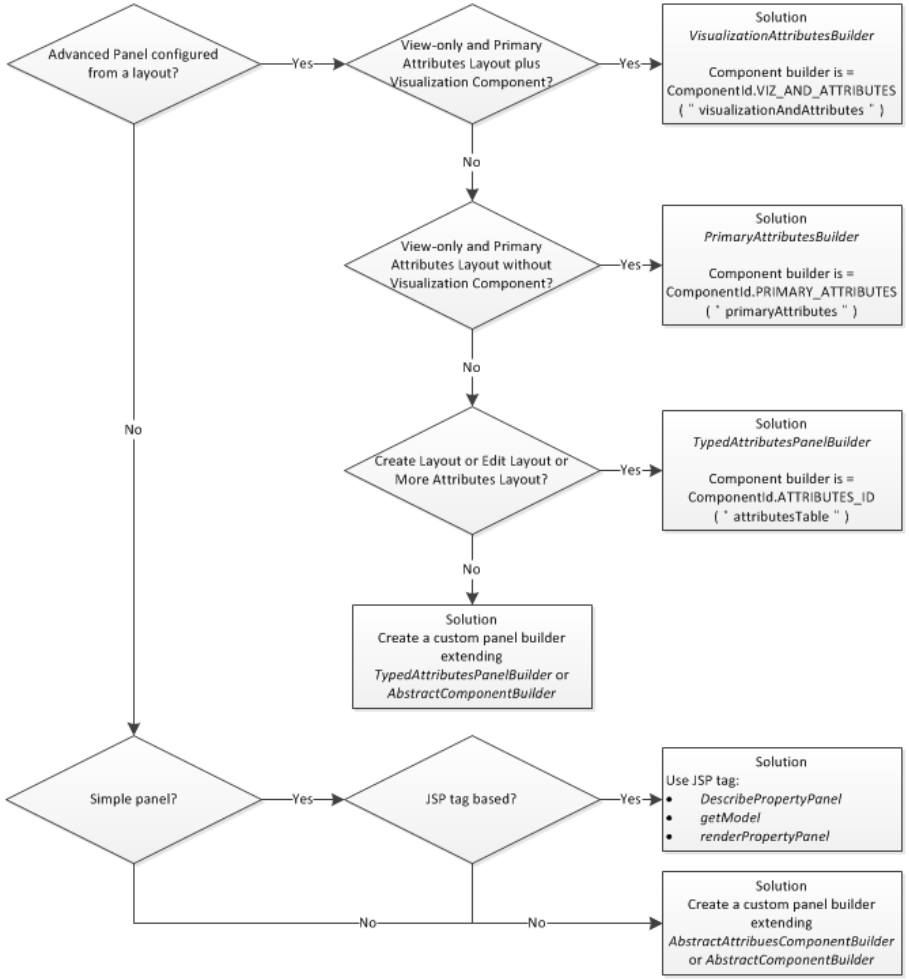
ProjectResourcePrimaryAttributeBuilder.java

TypedAttributesPanelBuilder

ApplicationDataContentAttributeBuilder.java

AbstractAttributesComponentBuilder

AbstractComponentBuilder



3. Excise 1 – Simple attributes panel

Design Simple attribute panel

Simply we will try to show attributes like following design.

- Show simple Attribute lists
- Delete group outline (System show group outline generally)
- Show non-OOTB instance data
- Using MVC architecture
- Data type use HashMap (Persistable object also can return)

Create MVC class

1. Create MVC class.

- Class name: CSCSimpleAttributesPanelBuilder
- Super Class: AbstractComponentBuilder
- Override: buildComponentConfig, buildComponentData
- Additional Private function: getAttributeConfig
- MVC Component name: eric.simple.attr

```
package ext.csc.training.mvc;

@ComponentBuilder("eric.simple.attr")
public class CSCSimpleAttributesPanelBuilder extends AbstractComponentBuilder {

    @Override
    public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTException {}

    private AttributeConfig getAttributeConfig(String id, String label) {}

    @Override
    public Object buildComponentData(ComponentConfig config, ComponentParams params) throws WTException {}
}
```

Design attributes panel

2. Design attributes panel (buildComponentConfig)

This function is a design area for attribute panel.

```
@Override
  public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTException {
     ComponentConfigFactory factory = getComponentConfigFactory();
     AttributePanelConfig panelConfig = factory.newAttributePanelConfig();

     // There are two ways of adding components to the AttributePanelConfig:
     // 1. Add AttributeConfigs directly to the AttributePanelConfig.
     // 2. Create a Group, add AttributeConfigs to the Group, and then add the Group to the AttributePanelConfig.

     // It is not recommended to use both methods on the same AttributePanelConfig.
     // This example shows adding the AttributeConfig directly to the AttributePanelConfig
     panelConfig.addComponent(getAttributeConfig("name", "Name"));
     panelConfig.addComponent(getAttributeConfig("address1", "Address"));
     panelConfig.addComponent(getAttributeConfig("address2", ""));
     panelConfig.addComponent(getAttributeConfig("phone", "Phone"));
     panelConfig.addComponent(getAttributeConfig("fax", "Fax"));

     // In production, we would set the view to "simpleAttributePanel.jsp", which would hide the border
     // and expand/collapse icon.
     panelConfig.setView("/components/simpleAttributePanel.jsp");

     return panelConfig;
  }
```

This is OOTB JSP component which is for delete group outline.

Query Data

3. Query data for attribute panel (buildComponentData)

Generally this function would return a persistable object and the panel would be showing properties of the persistable object. But this example will use hash map data.

Hash map keys or persistable attributes name must be matched with configured component key.

```
@Override
  public Object buildComponentData(ComponentConfig config, ComponentParams params) throws WTException {
     // returning a Map full of the properties shown in the example
     // Usually builders would return a persistable and the panel would be showing properties off the persistable
     Map datum = new HashMap();

     datum.put("name", "ERIC KIM1111");
     datum.put("address1", "Gaoxin-6-lu Keji-2-lu Xian");
     datum.put("address2", "Shanxi province");
     datum.put("country", "China");
     datum.put("phone","(029) 0000-0000");
     datum.put("fax","(029) 0000-6000");

     return datum;
  }
```

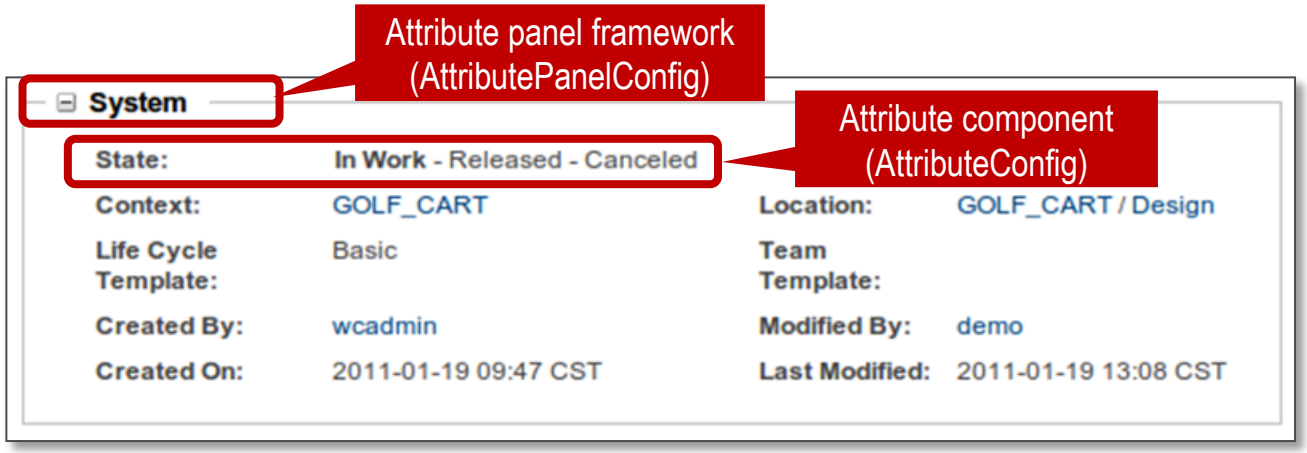
Internal function for AttributeConfig

4. Implement internal function to get AttributeConfig (getAttributeConfig)

AttributePanelConfig component must have AttributeConfig component, because AttributePanelConfig is just for framework of panel. So we have to design which attribute component will exist in the framework component.

This function is working for getting AttributeConfig component when receiving key and label name.

```
private AttributeConfig getAttributeConfig(String id, String label) {
    AttributeConfig attributeConfig = getComponentConfigFactory().newAttributeConfig();
    attributeConfig.setId(id);
    attributeConfig.setLabel(label);

    return attributeConfig;
}
```



3. Excise 1 – Simple attributes panel

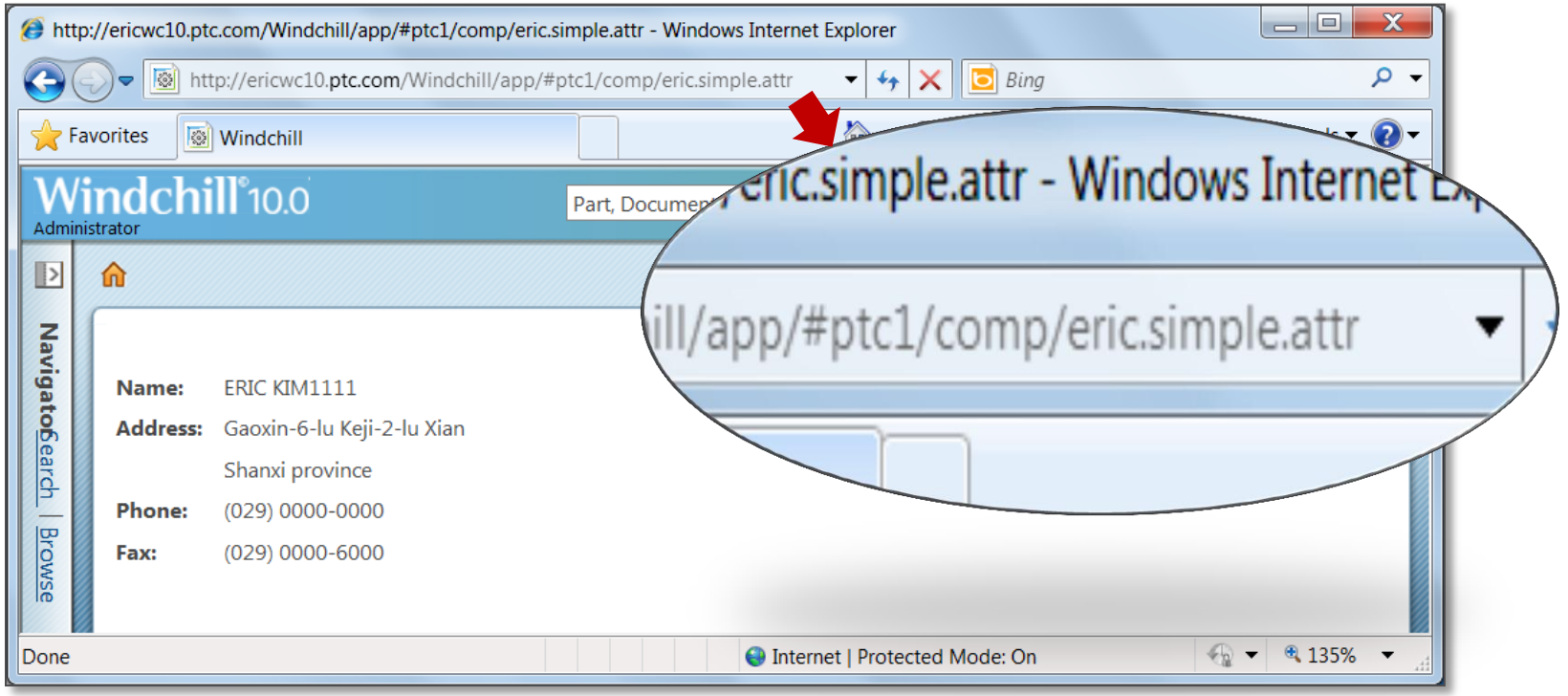
Test Result

CSCSimpleAttributesPanelBuilder.java

5. Restart all service and test

We had been created new MVC class for attributes panel, so we can call MVC component directly on web browser. If the MVC component was not work, you have to check following:

- Is the package of existing MVC component registered MVC Dispatcher?
- Doesn't the MVC component have error?



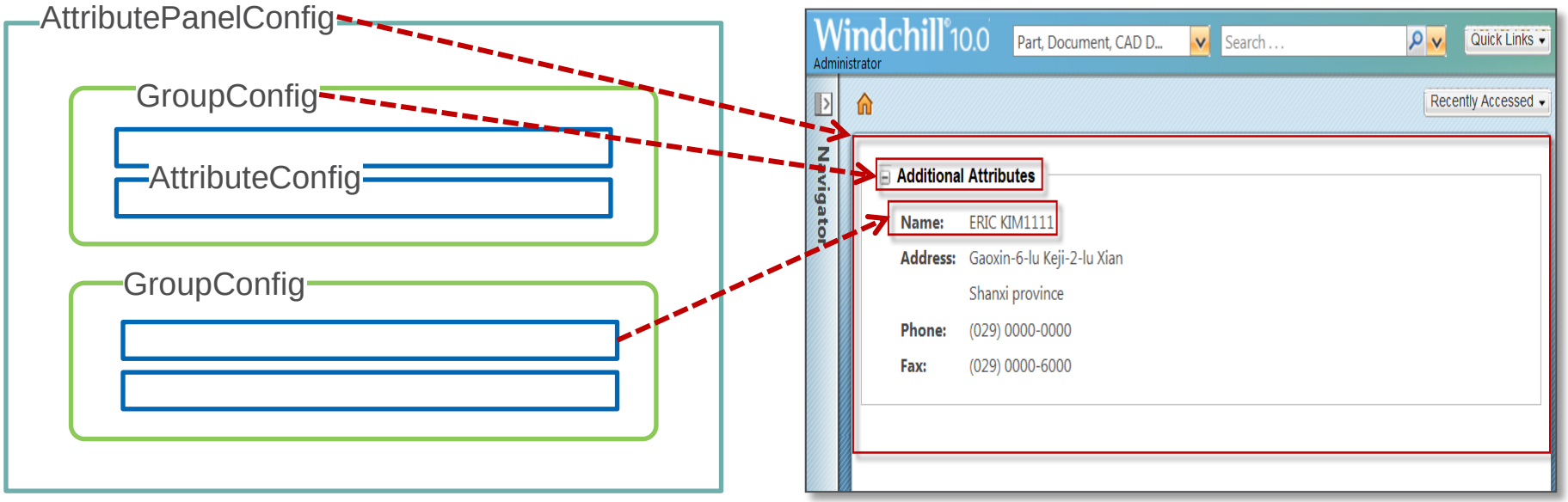
Design of Group framework

In the excise1, we deleted group framework, so this excise2 will explain how to add group framework.

- Show Group framework

When you want to add group framework, it is most similar previous excise. But we have to add one more component for group framework. Before adding group framework you must understand the each component's role.

- AttributePanelConfig : Set attributes all layout configuration. It controls for all environments.
- GroupConfig : Set one group layout.
- AttributeConfig : Set one element for attribute



Design attributes panel

1. Design attributes panel (buildComponentConfig)

We will update excise 1 source code for adding group framework. Copy excise 1's class and modify new class name and MVC id. We do not need to change other area except on "**buildComponentConfig**" function.

```
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTException {
    ComponentConfigFactory factory = getComponentConfigFactory();
    AttributePanelConfig panelConfig = factory.newAttributePanelConfig();

    // Set One group config
    GroupConfig groupConfig = factory.newGroupConfig("cscExcise2Config");
    groupConfig.setLabel("Additional Attributes");

    // Set Attribute on the group config
    groupConfig.addComponent(getAttributeConfig("name", "Name"));
    groupConfig.addComponent(getAttributeConfig("address1", "Address"));
    groupConfig.addComponent(getAttributeConfig("address2", ""));
    groupConfig.addComponent(getAttributeConfig("country", "Country"));
    groupConfig.addComponent(getAttributeConfig("phone", "Phone"));
    groupConfig.addComponent(getAttributeConfig("fax", "Fax"));

    // Add group config to panel
    panelConfig.addComponent(groupConfig);

    return panelConfig;
}
```

Deleted setView() function

3. Excise 2 – Group framework

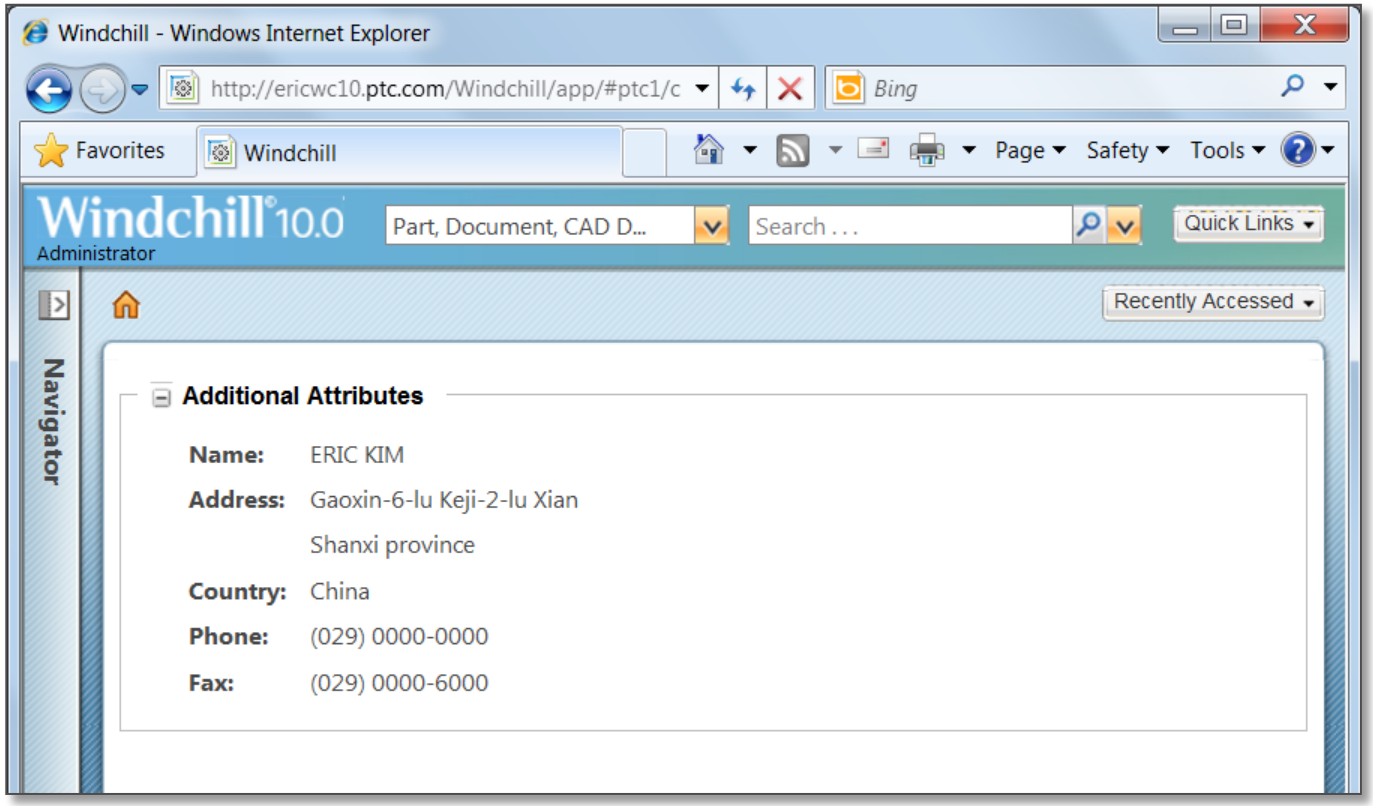
Test Result

CSCSimpleAttributesPanelBuilder1.java

2. Restart all service and test

We had been created new MVC class for attributes panel, so we can call MVC component directly on web browser. If the MVC component was not work, you have to check following:

- Is the package of existing MVC component registered MVC Dispatcher?
- Doesn't the MVC component have error?

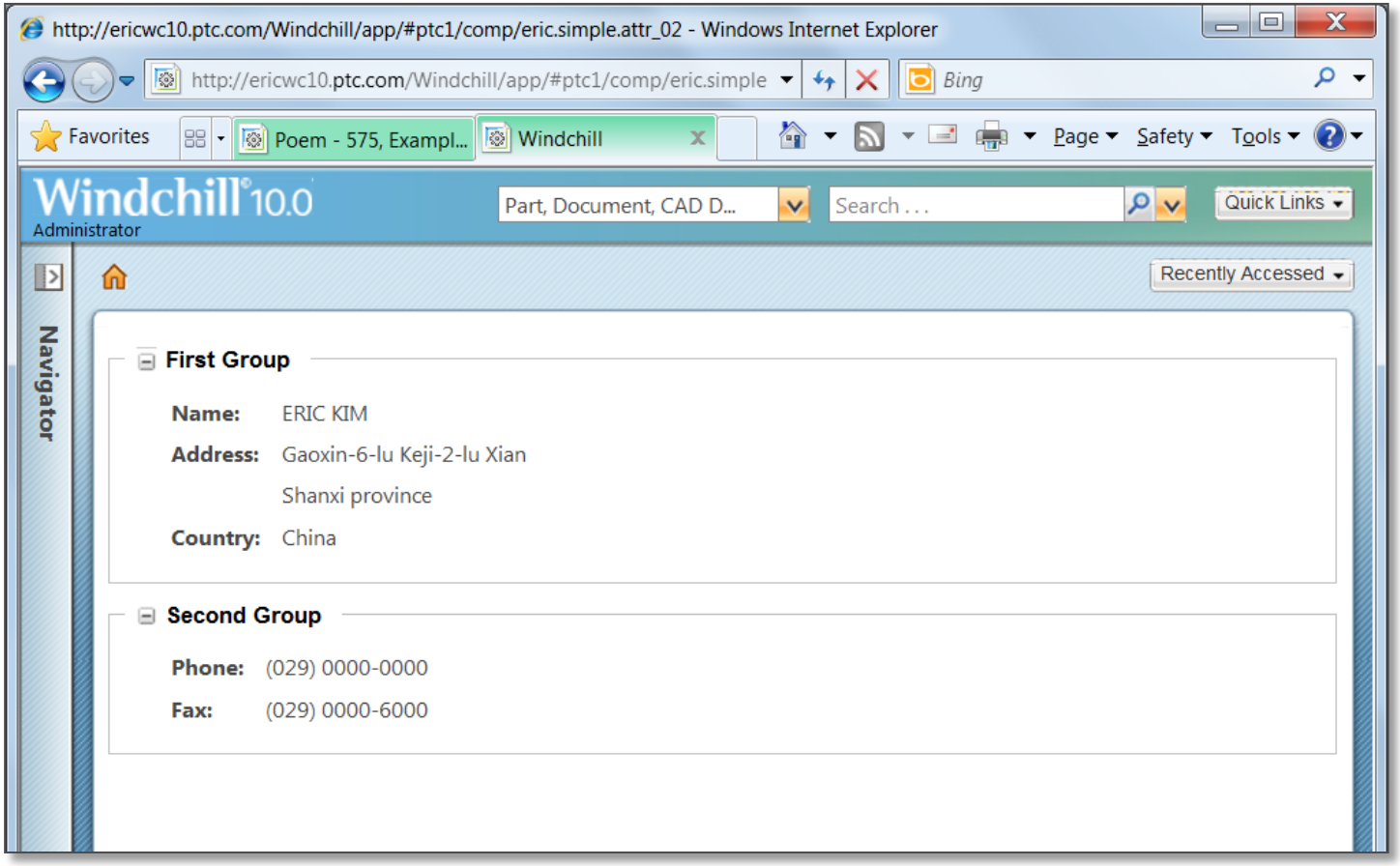


3. Excise 3 – Multi group attributes panel

Design Multi attributes panel

Using excise 2, we will make multi group attributes panel.

- Separate and show attributes group
- You can understand how to make multi group attributes panel.



3. Excise 3 – Multi group attributes panel

PTC

Design attributes panel

1. Design attributes panel (buildComponentConfig)

We will update excise 2 source code for making multi group framework. Copy excise 2's class and modify new class name and MVC id. We do not need to change other area except on "**buildComponentConfig**" function.

```
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTException {
    ComponentConfigFactory factory = getComponentConfigFactory();
    AttributePanelConfig panelConfig = factory.newAttributePanelConfig();

    // First Group
    GroupConfig groupConfig = factory.newGroupConfig("testGrp1");
    groupConfig.setLabel("First Group");
    groupConfig.addComponent(getAttributeConfig("name", "Name"));
    groupConfig.addComponent(getAttributeConfig("address1", "Address"));
    groupConfig.addComponent(getAttributeConfig("address2", ""));
    groupConfig.addComponent(getAttributeConfig("country", "Country"));
    panelConfig.addComponent(groupConfig);

    // Second Group
    GroupConfig groupConfig2 = factory.newGroupConfig("testGrp2", "Second Group", 2);
    groupConfig2.addComponent(getAttributeConfig("phone", "Phone"));
    groupConfig2.addComponent(getAttributeConfig("fax", "Fax"));
    panelConfig.addComponent(groupConfig2);

    return panelConfig;
}
```

3. Excise 3 – Multi group attributes panel

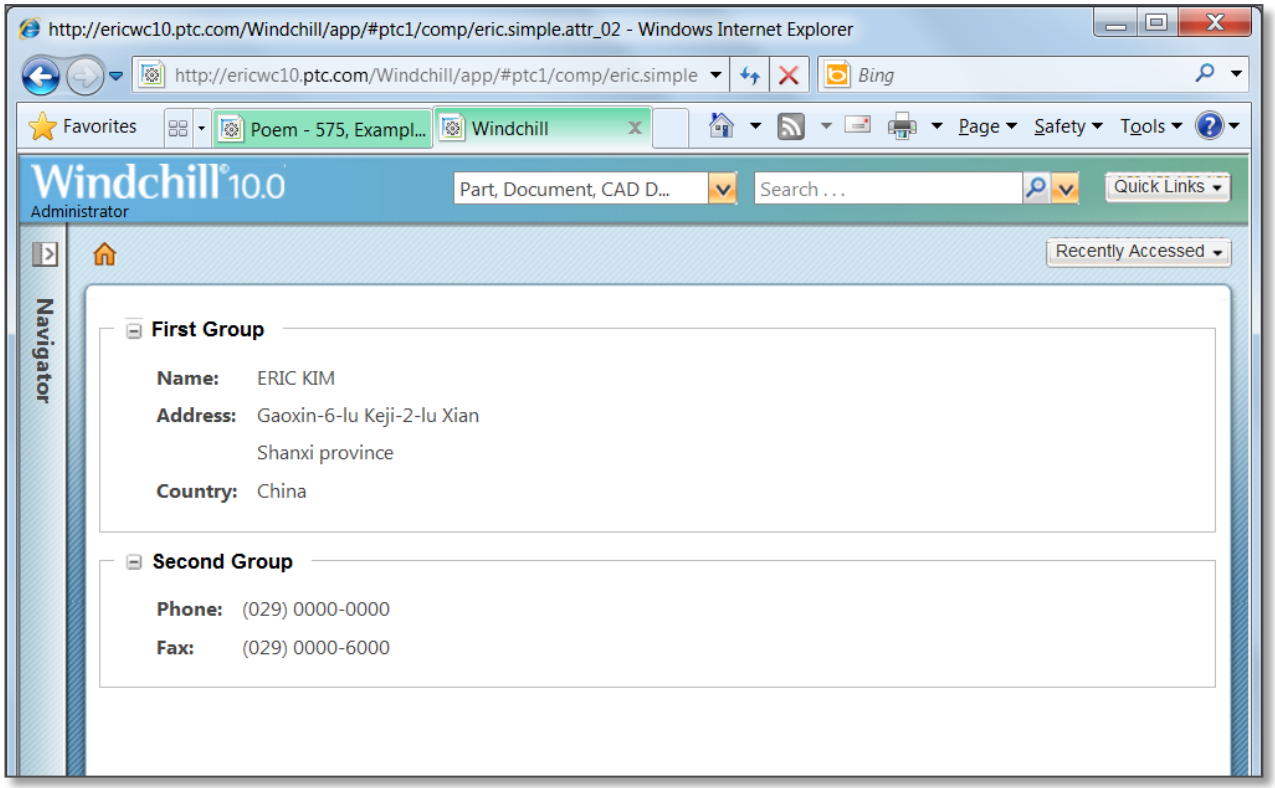
Test Result

CSCSimpleAttributesPanelBuilder2.java

2. Restart all service and test

We had been created new MVC class for attributes panel, so we can call MVC component directly on web browser. If the MVC component was not work, you have to check following:

- Is the package of existing MVC component registered MVC Dispatcher?
- Doesn't the MVC component have error?

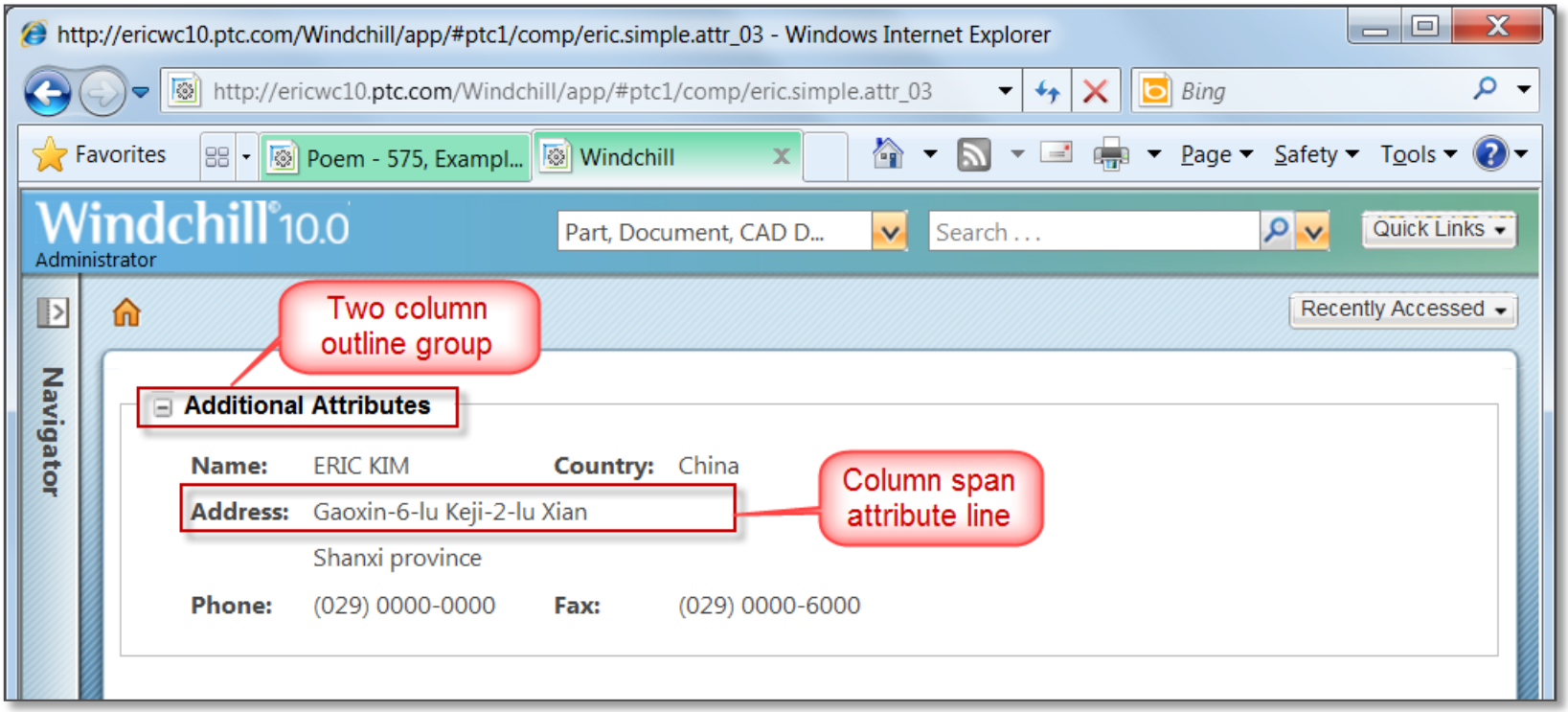


3. Excise 4 – Multi column and column span

Design Multi column and column span

Using excise 2, we will try to make column and column span.

- Basically make two column at each of lines.
- Some column needs to show one column, so this line needs to set column span.
- After this excise, you will understand how to make column in line and use column span



3. Excise 4 – Multi column and column span

Design attributes panel

1. Design attributes panel (buildComponentConfig)

We will update excise 2 source code for making multi group framework. Copy excise 2's class and modify new class name and MVC id. We do not need to change other area except on "**buildComponentConfig**" function.

```
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTException {
    ComponentConfigFactory factory = getComponentConfigFactory();
    AttributePanelConfig panelConfig = factory.newAttributePanelConfig();

    // Set One group config
    GroupConfig groupConfig = factory.newGroupConfig("cscExcise2Config");
    groupConfig.setLabel("Additional Attributes");
    groupConfig.addComponent(factory.newAttributeConfig("name", "Name", 0, 0)); // Line 1, column 1
    groupConfig.addComponent(factory.newAttributeConfig("country", "Country", 0, 1)); // Line 1, Column 2
    AttributeConfig address1 = factory.newAttributeConfig("address1", "Address", 1, 0); // Line 2, Column 1, 2-column span
    address1.setColSpan(2);
    groupConfig.addComponent(address1);
    AttributeConfig address2 = factory.newAttributeConfig("address2", "", 2, 0); // Line 3, Column 1, 2-column span
    address2.setColSpan(2);
    groupConfig.addComponent(address2);
    groupConfig.addComponent(factory.newAttributeConfig("phone", "Phone", 3, 0)); // Line 4, Column 1
    groupConfig.addComponent(factory.newAttributeConfig("fax", "Fax", 3, 1)); // Line 4, Column 2

    panelConfig.addComponent(groupConfig);
    return panelConfig;
}
```

factory.newAttributeConfig() function can add more 2 parameters to set column and row

3. Excise 4 – Multi column and column span

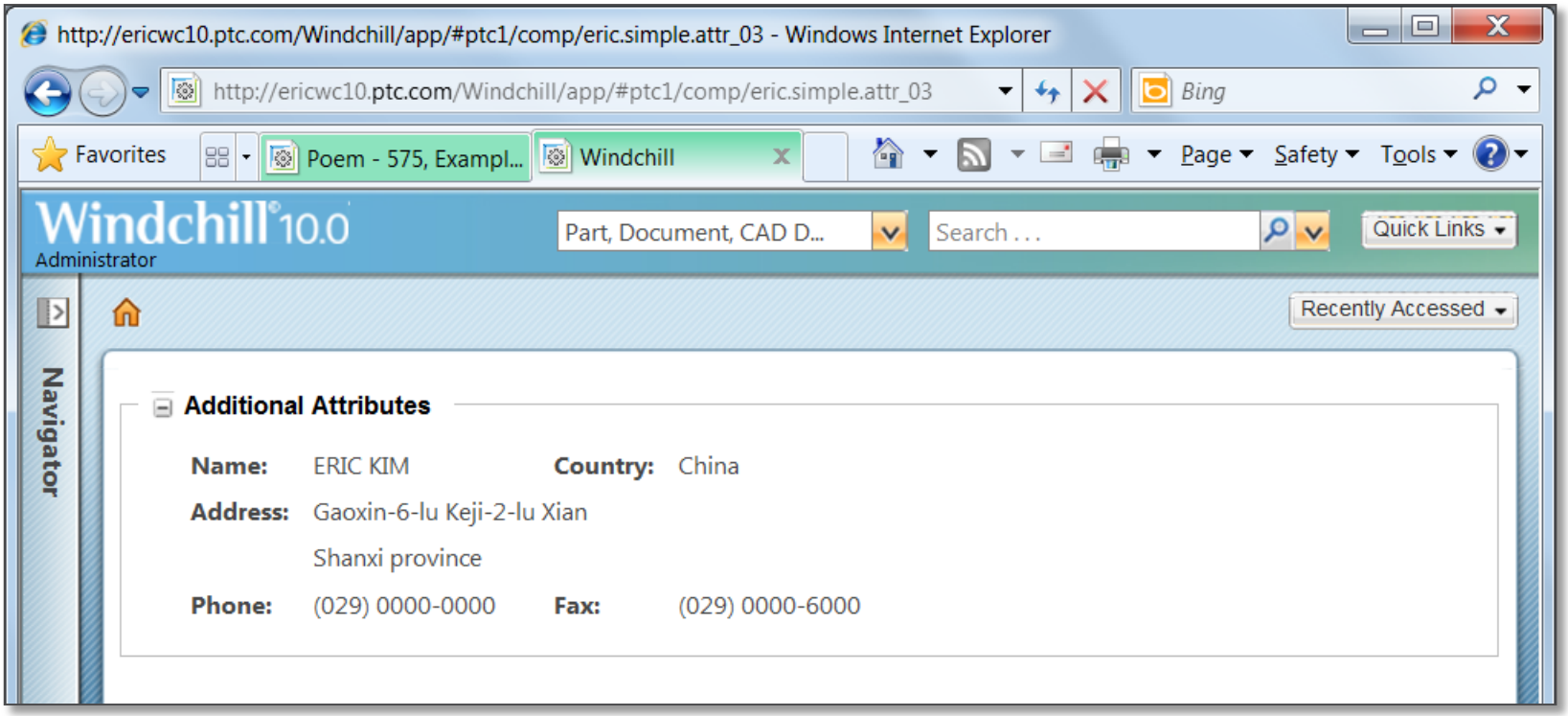
Test Result

CSCSimpleAttributesPanelBuilder3.java

2. Restart all service and test

We had been created new MVC class for attributes panel, so we can call MVC component directly on web browser. If the MVC component was not work, you have to check following:

- Is the package of existing MVC component registered MVC Dispatcher?
- Doesn't the MVC component have error?

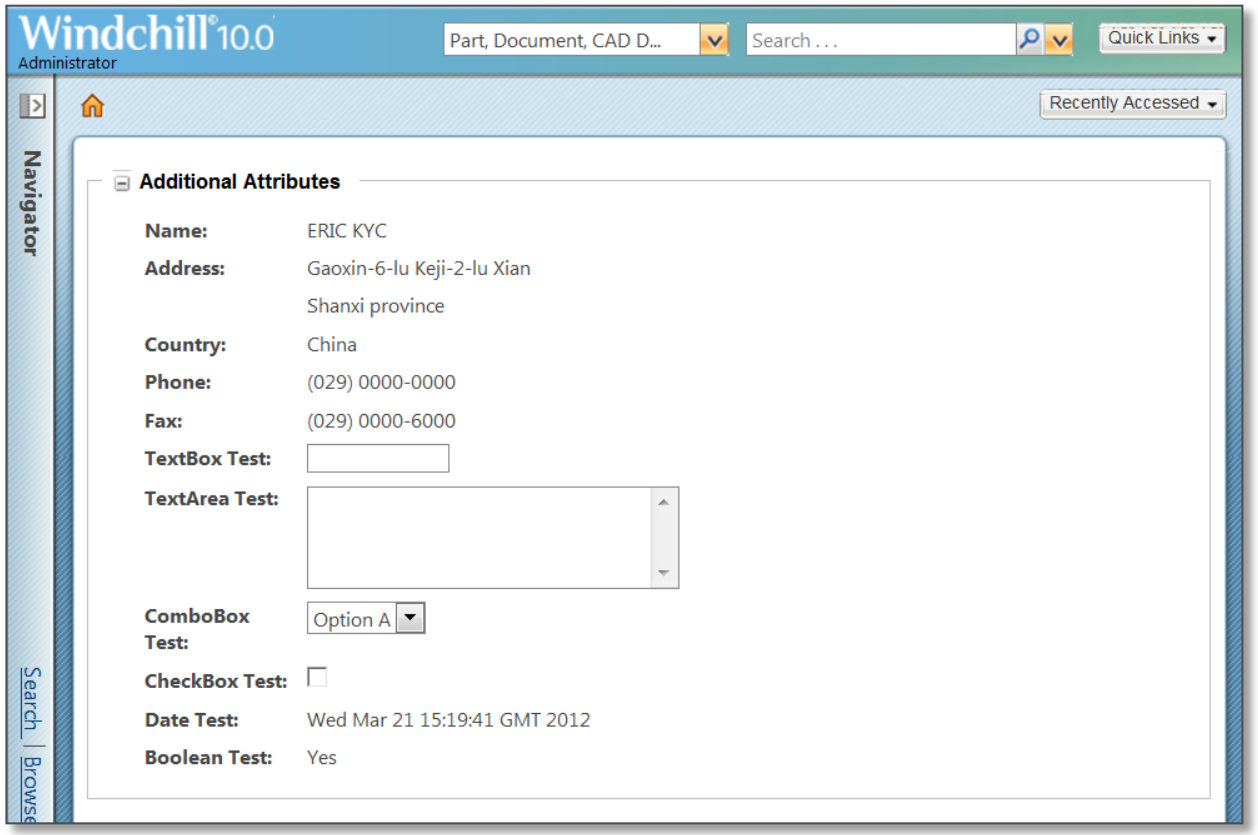


3. Excise 5 – Add Edit attributes

Design Multi column and column span

Using excise 2, we will try to add edit attributes for each types.

- Add edit attributes data for each of types (**buildComponentData**)
  : Make data model using UIComponent class
- Set edit mode for attribute components : **setComponentMode(ComponentMode.*EDIT*)**



3. Excise 5 – Add Edit attributes

PTC

Design attributes panel

1. Design attributes panel (buildComponentConfig)

This function just needs to set attributes layer and lists. **However if you don't set any mode for panel component, generally system will show default mode which is READ. So you must change panel component mode for EDIT**. When the component mode is EDIT, String of instance value will show read-only and UIComponent of instance value will show edit component.

```
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTException {
    ComponentConfigFactory factory = getComponentConfigFactory();
    AttributePanelConfig panelConfig = factory.newAttributePanelConfig();

   GroupConfig groupConfig = factory.newGroupConfig("cscExcise2Config");
    groupConfig.setLabel("Additional Attributes");
    groupConfig.addComponent(getAttributeConfig("myName", "Name"));
    groupConfig.addComponent(getAttributeConfig("address1", "Address"));
    groupConfig.addComponent(getAttributeConfig("address2", ""));
    groupConfig.addComponent(getAttributeConfig("country", "Country"));
    groupConfig.addComponent(getAttributeConfig("phone", "Phone"));
    groupConfig.addComponent(getAttributeConfig("fax", "Fax"));
    groupConfig.addComponent(factory.newAttributeConfig("textBox", "TextBox Test"));
    groupConfig.addComponent(factory.newAttributeConfig("textArea", "TextArea Test"));
    groupConfig.addComponent(factory.newAttributeConfig("comboBox", "ComboBox Test"));
    groupConfig.addComponent(factory.newAttributeConfig("checkBox", "CheckBox Test"));
    groupConfig.addComponent(factory.newAttributeConfig("date", "Date Test"));
    groupConfig.addComponent(factory.newAttributeConfig("booleanField", "Boolean Test"));

    panelConfig.addComponent(groupConfig);
    panelConfig.setComponentMode(ComponentMode.EDIT);
    return panelConfig;
}
```

Add attributes for testing each of UIComponents types

Set EDIT mode for show UIComponent value

3. Excise 5 – Add Edit attributes

PTC

Query Data

2. Query data for attribute panel (buildComponentData)

Previously we set String value. Now we have to show EDIT component on attributes panel. But the String value can not be shown for edit mode, although attribute panel is EDIT mode. So we have to return UIComponent value which you want to show edit mode attributes.

```
public Object buildComponentData(ComponentConfig config, Com
    Map datum = new HashMap();
    datum.put("myName", "ERIC KYC");
    datum.put("address1", "Gaoxin-6-lu Keji-2-lu Xian");
    datum.put("address2", "Shanxi province");
    datum.put("country", "China");
    datum.put("phone","(029) 0000-0000");
    datum.put("fax","(029) 0000-6000");

    GuiComponent textBox = new TextBox();
    ((TextBox) textBox).setId("testBox");
    ((TextBox) textBox).setName("textBox");
    datum.put("textBox",textBox);

    GuiComponent inputTextAreaField = new TextArea();
    ((TextArea) inputTextAreaField).setId("testTextAreaID");
    ((TextArea) inputTextAreaField).setName("testTextAreaName");
    ((TextArea) inputTextAreaField).setHeight(4);
    ((TextArea) inputTextAreaField).setWidth(30);
    datum.put("textArea",inputTextAreaField);
```

```
    GuiComponent comboBoxField = new ComboBox();
    ArrayList<String> internalValues = new ArrayList<String>();
    internalValues.add("Option A");
    internalValues.add("Option B");
    internalValues.add("Option C");
    ((ComboBox) comboBoxField).setInternalValues(internalValues);
    ((ComboBox) comboBoxField).setName("Test Combo box");
    ((ComboBox) comboBoxField).setValues(internalValues);
    datum.put("comboBox",comboBoxField);

    GuiComponent checkbox = new CheckBox();
    ((CheckBox) checkbox).setEnabled(true);
    ((CheckBox) checkbox).setName("Test Combobox");
    datum.put("checkBox",checkbox);

    Date dateField = new Date();
    datum.put("date",dateField);

    datum.put("booleanField", Boolean.TRUE);
    return datum;
}
```

3. Excise 5 – Add Edit attributes

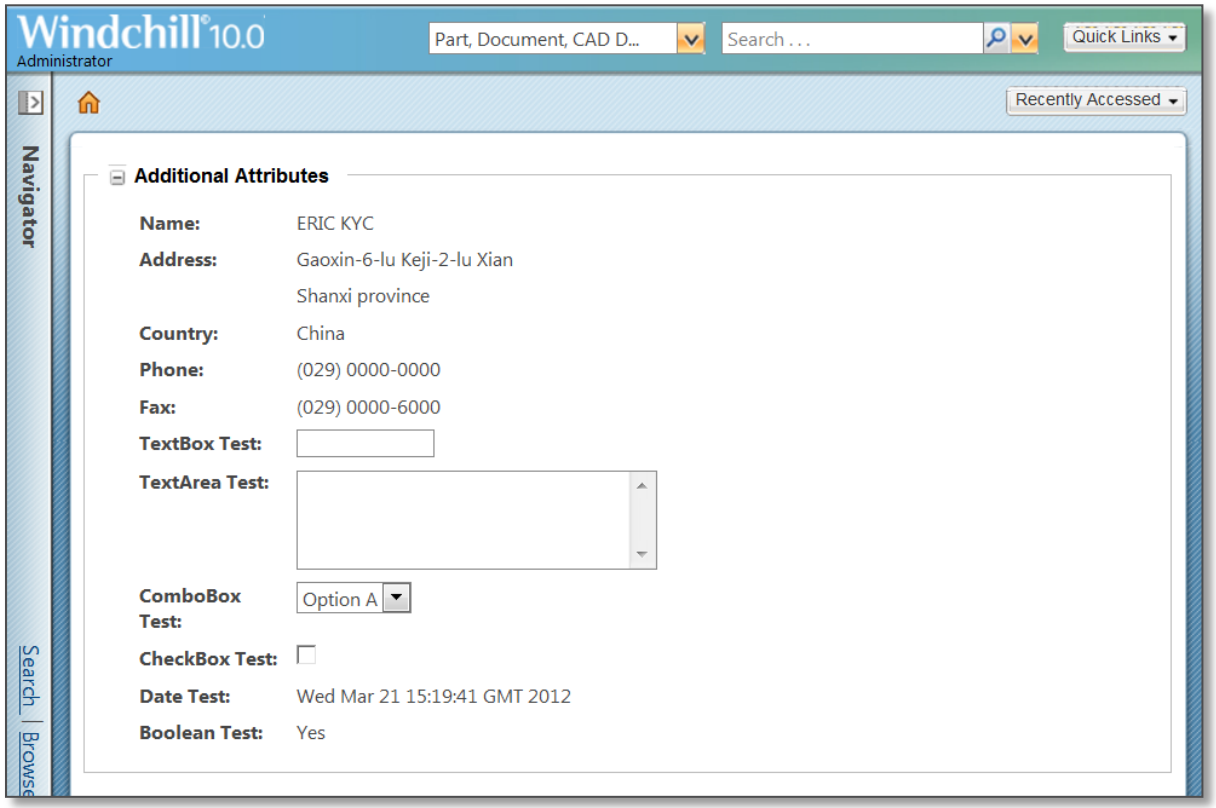
Test Result

CSCSimpleAttributesPanelBuilder4.java

2. Restart all service and test

We had been created new MVC class for attributes panel, so we can call MVC component directly on web browser. If the MVC component was not work, you have to check following:

- Is the package of existing MVC component registered MVC Dispatcher?
- Doesn't the MVC component have error?



Agenda

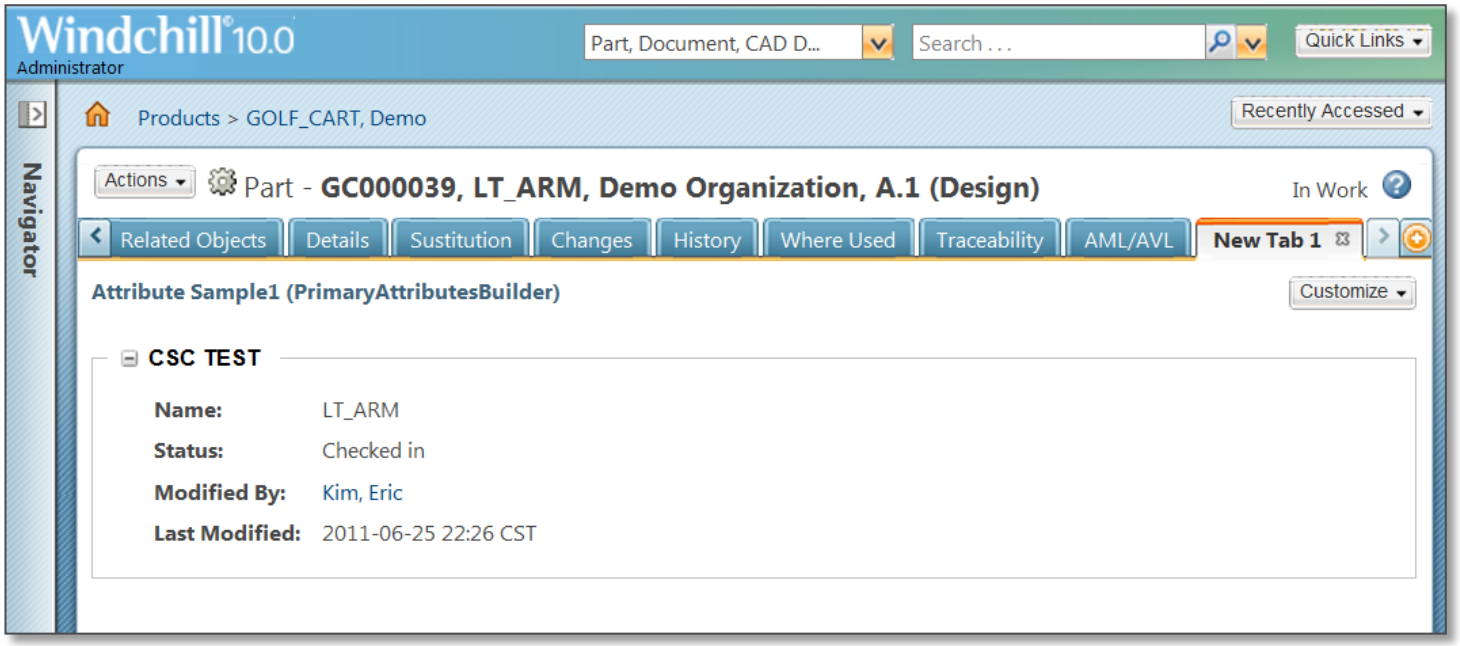
- Excise6 – AbstractAttributesComponentBuilder
- Excise7 – PrimaryAttributesBuilder
- Excise8 – TypedAttributesPanelBuilder

3. Excise 6 – PrimaryAttributesBuilder

Design for PrimaryAttributesBuilder

PrimaryAttributesBuilder is calling “primary attributes” layout in the type manager, so you do not need to make any attributes component design. It is very simple component to use standard typed object.

- Generally when using this component we have to get target object from parameter.
- PrimaryAttributesBuilder read display information from layout and make UIComponent.
- So we just change environment or add other information.
- Also we can delete already designed UIComponent based on layout.
- PrimaryAttributesBuilder must be inputted typed object from parameter, so this demo will add info page content in customize button.



3. Excise 6 – PrimaryAttributesBuilder

Update action model and action

1. Open “PartClient-actionmodels.xml” and copy “third_level_nav_part” block to custom action model. And the add custom action

custom-actionmodels.xml

```
<model name="third_level_nav_part" resourceBundle="com.ptc.windchill.enterprise.part.partResource">
      <submodel name="general"/>                          <!-- General -->
      <submodel name="relatedItems"/>                     <!-- Related Objects -->
      <submodel name="changes"/>                          <!-- Change -->
      <submodel name="history"/>                          <!-- History -->
      <submodel name="collaboration"/>                    <!-- Collaboration -->
      <submodel name="prodAnalytics"/>                    <!-- Product Analytics -->
      <action name="attrSamples1" type="csc"/>
</model>
```

2. Create new action (if you do not have the action)

custom-actions.xml

```
<objecttype name="csc" resourceBundle="com.ptc.core.ui.navigationRB">
       <action name="attrSamples1" ajax="row">
            <component windowType="page" name="eric.advanced.attr_01"/>
         </action>
</objecttype>
```

3. If no have resource bundle, you must create resource bundle for action

action.properties, action_[locale].properties

```
csc.attrSamples1.description=Attribute Sample1 (PrimaryAttributesBuilder)
```

3. Excise 6 – PrimaryAttributesBuilder

Create MVC class

4. Create MVC class.

- Class name: CSCAdvancedAttributesPanelBuilder1
- Super Class: PrimaryAttributesBuilder
- Override: buildPrimaryAttributePanelConfig, setTypedAttrLayOutFactory
- MVC Component name: eric.advanced.attr_01

```
package ext.csc.training.mvc;

@ComponentBuilder("eric.advanced.attr_01")
public class CSCAdvancedAttributesPanelBuilder1 extends PrimaryAttributesBuilder {

    @Override
    public AttributePanelConfig buildPrimaryAttributePanelConfig (ComponentParams params) throws WTException {}

    @Override
    public void setTypedAttrLayOutFactory(TypedAttrLayOutFactory factory) {}
}
```

3. Excise 6 – PrimaryAttributesBuilder

PTC

Design attributes panel

5. Design attributes panel (buildPrimaryAttributePanelConfig)

This function is a design area for attribute panel.

```
public AttributePanelConfig buildPrimaryAttributePanelConfig(ComponentParams params) throws WTException {
   // Set environment of Primary Attributes panel framework
   JcaAttributePanelConfig attrPanel = (JcaAttributePanelConfig)tfactory
                              .getAttributePanelConfig(getComponentConfigFactory(), params, ScreenDefinitionName.INFO);
   attrPanel.setComponentMode(ComponentMode.VIEW);

    // Set environment of group config
   JcaGroupConfig group = (JcaGroupConfig)attrPanel.getComponents().get(0);
   group.setLabel("CSC TEST");

   // Delete no used attribute (example: delete classification)
   List<ComponentConfig> attributestoHide = new ArrayList<ComponentConfig>();
   List<ComponentConfig>Attributes = group.getComponents();
   for(ComponentConfig attribute:Attributes){
      JcaAttributeConfig attributeConfig =(JcaAttributeConfig) attribute;
      if ( attributeConfig.getId().equals("classification.id")) attributestoHide.add(attribute);
   }
   for(ComponentConfig attribute:attributestoHide){
      group.removeComponent(attribute);
   }

   return attrPanel;
}
```

3. Excise 6 – PrimaryAttributesBuilder

Get Layout object instance and test

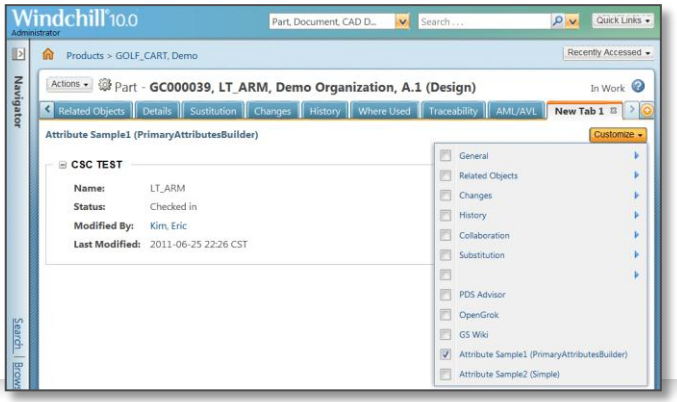
6. Get layout object instance for primary attributes(setTypedAttrLayOutFactory)

This function must be overridden for PrimaryAttribuesBuilder. If it is not initialized which means using super function, the layout information of target instance will be null.

```
// Global variable for layout factory
TypedAttrLayOutFactory tfactory;

@Override
public void setTypedAttrLayOutFactory(TypedAttrLayOutFactory factory) {
   this.tfactory = factory;
}
```

7. Restart Method Server and Test

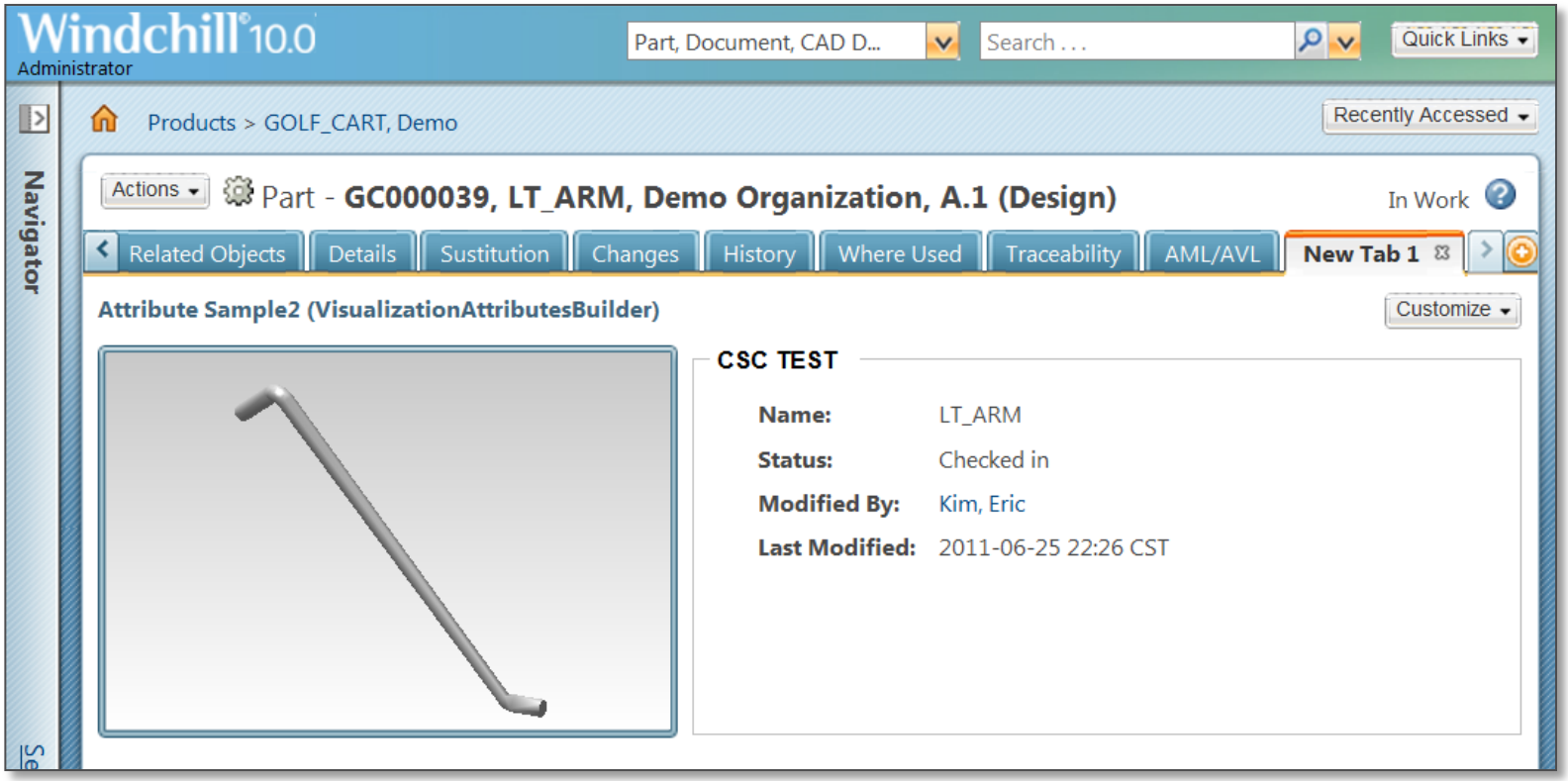
The menu will be set on the info page customize button on part information. You have to create one new tab and select add new item for PrimaryAttributesBuilder on the new tab page.



CSCAdvancedAttributesPanelBuilder1.java

Design for VisualizationAttributesBuilder

VisualizationAttributesBuilder is calling “primary attributes” layout in the type manager with visualization framework, so you do not need to make any attributes component design. It is very simple component to use standard typed object.



3. Excise 7 – VisualizationAttributesBuilder

Update action model and action

1. Open “PartClient-actionmodels.xml” and copy “third_level_nav_part” block to custom action model. And the add custom action

custom-actionmodels.xml

```
<model name="third_level_nav_part" resourceBundle="com.ptc.windchill.enterprise.part.partResource">
      <submodel name="general"/>                          <!-- General -->
      <submodel name="relatedItems"/>                     <!-- Related Objects -->
      <submodel name="changes"/>                          <!-- Change -->
      <submodel name="history"/>                          <!-- History -->
      <submodel name="collaboration"/>                    <!-- Collaboration -->
      <submodel name="prodAnalytics"/>                    <!-- Product Analytics -->
      <action name="attrSamples2" type="csc"/>
</model>
```

2. Create new action (if you do not have the action)

custom-actions.xml

```
<objecttype name="csc" resourceBundle="com.ptc.core.ui.navigationRB">
       <action name="attrSamples2" ajax="row">
            <component windowType="page" name="eric.advanced.attr_02"/>
         </action>
</objecttype>
```

3. If no have resource bundle, you must create resource bundle for action

action.properties, action_[locale].properties

```
csc.attrSamples2.description=Attribute Sample2 (VisualizationAttributesBuilder)
```

3. Excise 7 – VisualizationAttributesBuilder

Create MVC class

4. Create MVC class.

- Class name: CSCAdvancedAttributesPanelBuilder2
- Super Class: VisualizationAttributesBuilder
- Override: buildComponentConfig
- MVC Component name: eric.advanced.attr_02

```
package ext.csc.training.mvc;

@ComponentBuilder("eric.advanced.attr_02")
public class CSCAdvancedAttributesPanelBuilder2 extends VisualizationAttributesBuilder {

    @Override
    public ComponentConfig buildComponentConfig (ComponentParams params) throws WTException {}
}
```

Design attributes panel

5. Design attributes panel (buildComponentConfig)

This function is a design area for attribute panel. VisualizationAttributesBuilder is using MultiComponentConfig for showing two panel which are visualization panel and primary attributes panel. However the visualization panel will be added the end of rendering, actually after finishing buildComponentConfig, so MultiComponentConfig just has one ComponentConfig of primary panel.

```
@Override
public ComponentConfig buildComponentConfig(ComponentParams params) throws WTException {
    // Get multi component config
    MultiComponentConfig multiComponentConfig = (MultiComponentConfig) super.buildComponentConfig(params);
    multiComponentConfig.setComponentMode(ComponentMode.VIEW);

    // Get primary attributes panel and update Label
    JcaAttributePanelConfig primaryPanelConfig =
                                        (JcaAttributePanelConfig)multiComponentConfig.getComponents().get(0);
    JcaGroupConfig group = (JcaGroupConfig)primaryPanelConfig.getComponents().get(0);
    group.setLabel("CSC TEST");

    return multiComponentConfig;
}
```

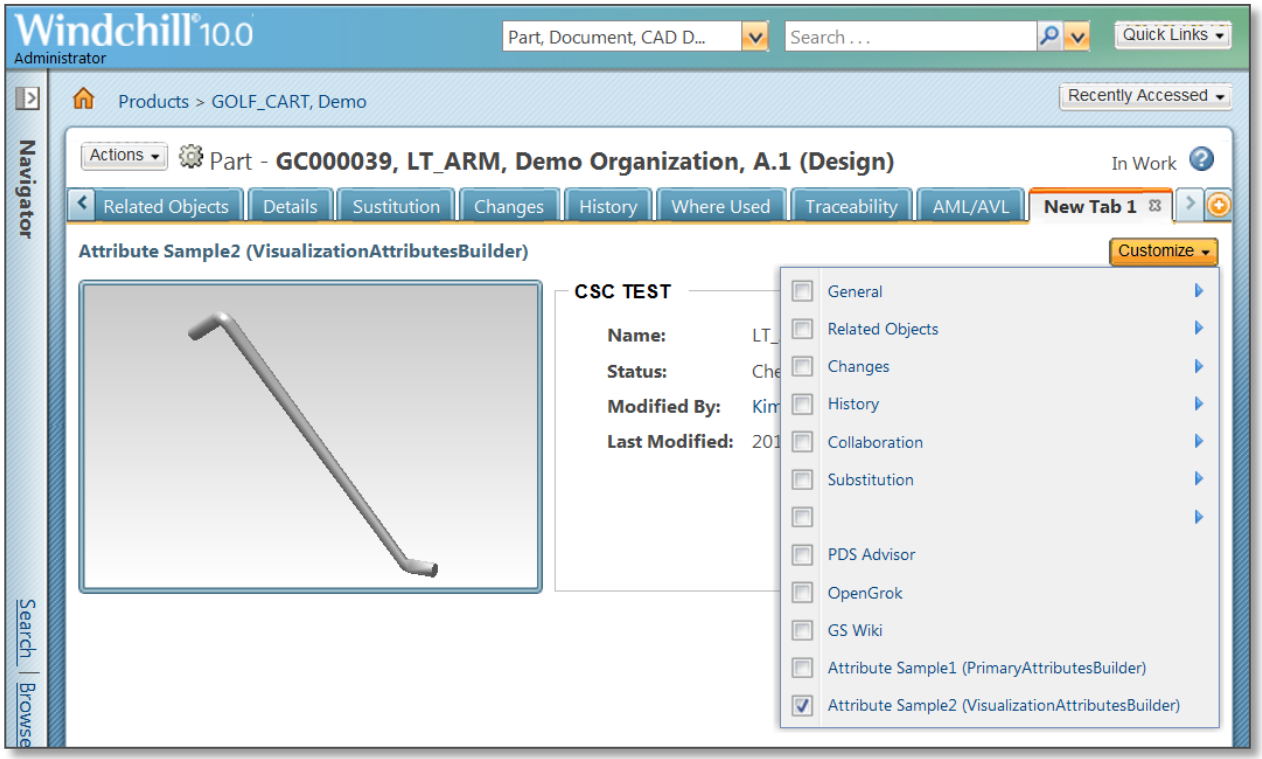
VisualizationAttributesBuilder doesn't need to be implemented "**setTypedAttrLayOutFactor**" function because it will be initialized on the super class.

3. Excise 7 – VisualizationAttributesBuilder

Test Result

6. Restart Method Server and Test

The menu will be set on the info page customize button on part information. You have to create one new tab and select add new item for VisualizationAttributesBuilder on the new tab page.



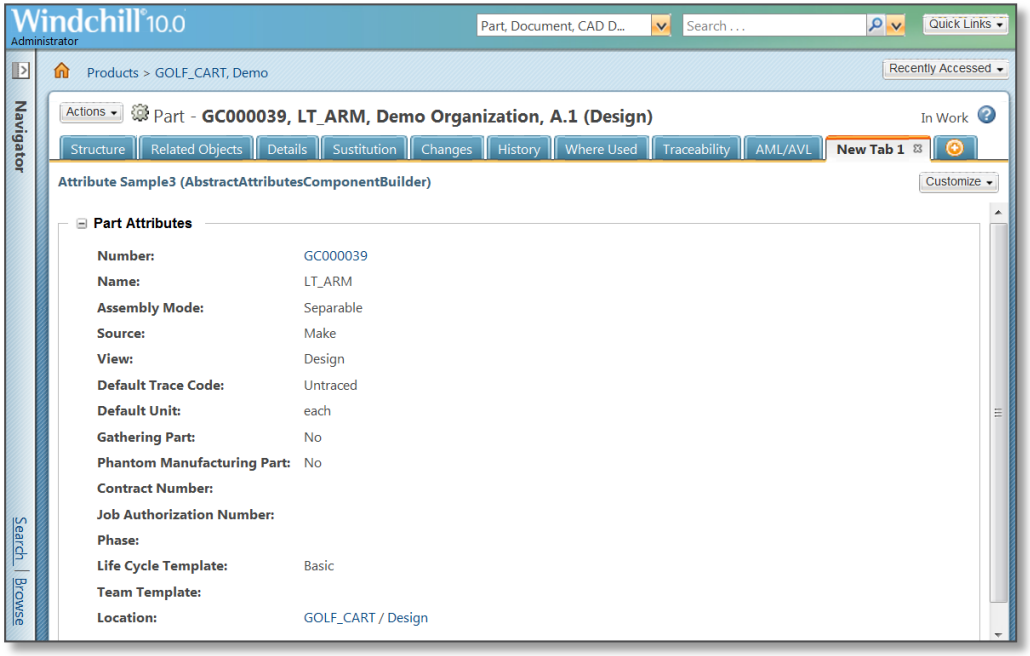
CSCAdvancedAttributesPanelBuilder2.java

3. Excise 8 – AbstractAttributesComponentBuilder

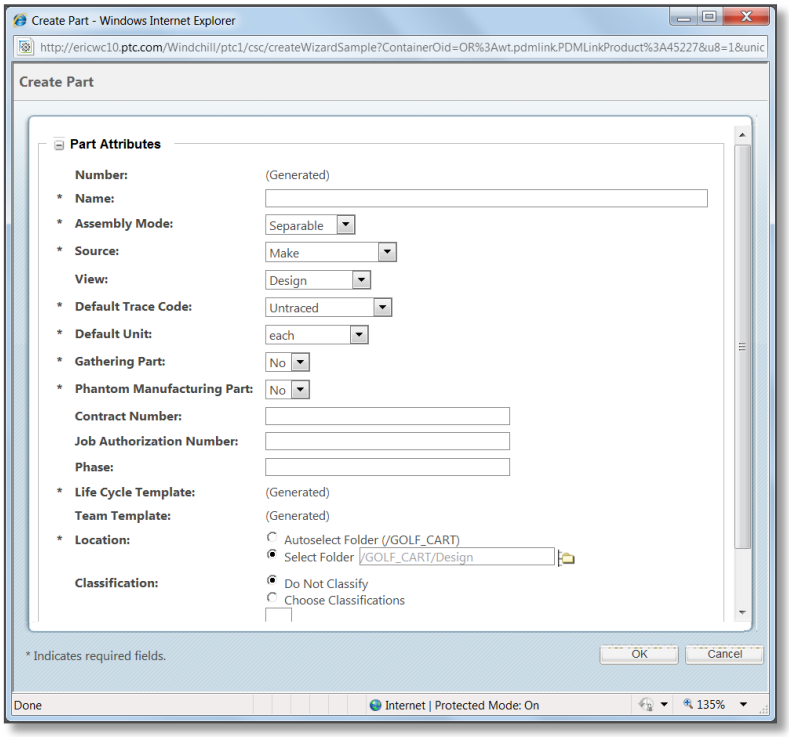
Design for AbstractAttributesComponentBuilder

AbstractAttributesComponentBuilder can use existed all layout on type manager. So you have to decide which kind of layout you want to use for customized pages.

- Using CREATE layout
- Show info page and wizard the CREATE layout (using same MVC class)



Info page used CREATE layout



Create wizard used CREATE layout

3. Excise 8 – AbstractAttributesComponentBuilder

Update action model and action

1. Open “PartClient-actionmodels.xml” and copy “third_level_nav_part” block to custom action model. And the add custom action

custom-actionmodels.xml

```
<model name="third_level_nav_part" resourceBundle="com.ptc.windchill.enterprise.part.partResource">
      <submodel name="general"/>                          <!-- General -->
      <submodel name="relatedItems"/>                     <!-- Related Objects -->
      <submodel name="changes"/>                          <!-- Change -->
      <submodel name="history"/>                          <!-- History -->
      <submodel name="collaboration"/>                    <!-- Collaboration -->
      <submodel name="prodAnalytics"/>                    <!-- Product Analytics -->
      <action name="attrSamples3" type="csc"/>
</model>
```

2. Create new action (if you do not have the action)

custom-actions.xml

```
<objecttype name="csc" resourceBundle="com.ptc.core.ui.navigationRB">
    <action name="attrSamples3" ajax="row">
        <component windowType="page" name=" csc.test.attributes.component.part "/>
     </action>
</objecttype>
<!-- Reuse previous example -->
<objecttype name="cscwizard_create_sample" class="wt.part.WTPart" resourceBundle="ext.csc.training.resource.CSCTrainingRB">
    <action name="createWizardSamples" ajax="component">
        <command windowType="popup" class="ext.csc.jca.wizard.CSCCreateWizardSampleProcess" method="execute" url="netmarkets/jsp/csc/createWizardSample.jsp"/>
    </action>

    <action name="setPartAttribute">
        <command windowType="wizard_step" url="netmarkets/jsp/csc/setPartAttribute.jsp"/>
    </action>
</objecttype>
```

3. Excise 8 – AbstractAttributesComponentBuilder

PTC

Create resource bundle and MVC class

3. If no have resource bundle, you must create resource bundle for action

action.properties, action_[locale].properties

```
csc.attrSamples2.description=Attribute Sample2 (VisualizationAttributesBuilder)
```

4. Create MVC class.
   - Class name: CSCAdvancedAttributesPanelBuilder3
   - Super Class: AbstractAttributesComponentBuilder
   - Implements interface: TypedAttrLayOutFactoryAware
   - Override: buildAttributesComponentConfig, setTypedAttrLayOutFactory
   - MVC Component name: eric.advanced.attr_03

```
package ext.csc.training.mvc;

@ComponentBuilder("csc.test.attributes.component.part")
public class CSCAdvancedAttributesPanelBuilder3 extends AbstractAttributesComponentBuilder
                                                    implements TypedAttrLayOutFactoryAware {
   @Override
   protected CustomizableViewConfig buildAttributesComponentConfig(ComponentParams arg0) throws WTException {}

   @Override
   public void setTypedAttrLayOutFactory(TypedAttrLayOutFactory factory) {}
}
```

3. Excise 8 – AbstractAttributesComponentBuilder

PTC®

Design attributes panel

5. Design attributes panel (buildAttributesComponentConfig)

This function is a design area for attribute panel. In this function, you can select displayed layout existing on the type manager, and generate attribute panel using layout type.

```
protected CustomizableViewConfig buildAttributesComponentConfig(ComponentParams arg0) throws WTException {
    // Get inputted panel mode
    ComponentMode mode = getComponentMode(arg0);

    // Select displayed layout on the type manager and generate attributes panel
    ScreenDefinitionName layout = ScreenDefinitionName.CREATE;
    JcaAttributePanelConfig table = (JcaAttributePanelConfig)tfactory.getAttributePanelConfig(
                                                    getComponentConfigFactory(), arg0, layout);

    // Panel display mode. But on info page or wizard is doesn't work.
    table.setComponentMode(ComponentMode.VIEW);

    return table;
}
```

6. Get layout object instance for primary attributes(setTypedAttrLayOutFactory)

```
// Global variable for layout factory
TypedAttrLayOutFactory tfactory;
public void setTypedAttrLayOutFactory(TypedAttrLayOutFactory factory) {
   this.tfactory = factory;
}
```

Update wizard step

7. Update wizard step (setPartAttribute.jsp)

Previous wizard sample will reuse. But it doesn't need any update. We just need to update wizard step page which will set new MVC class.

- Add JCA tag component
- Add JCA initialize Item for setting type, it is the value of type manager for getting layout.
- Change MVC component Id name.

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ taglib uri="http://www.ptc.com/windchill/taglib/components" prefix="jca"%>
<%@ taglib uri="http://www.ptc.com/windchill/taglib/mvc" prefix="mvc"%>

<%@ include file="/netmarkets/jsp/components/beginWizard.jspf" %>

<jca:initializeItem operation="${createBean.create}" baseTypeName="wt.part.WTPart"/>

<div id='addressAttributesPane'>
                <jsp:include page="${mvc:getComponentURL('csc.test.attributes.component.part')}" />
</div>

<%@include file="/netmarkets/jsp/util/end.jspf"%>
```

3. Excise 8 – AbstractAttributesComponentBuilder

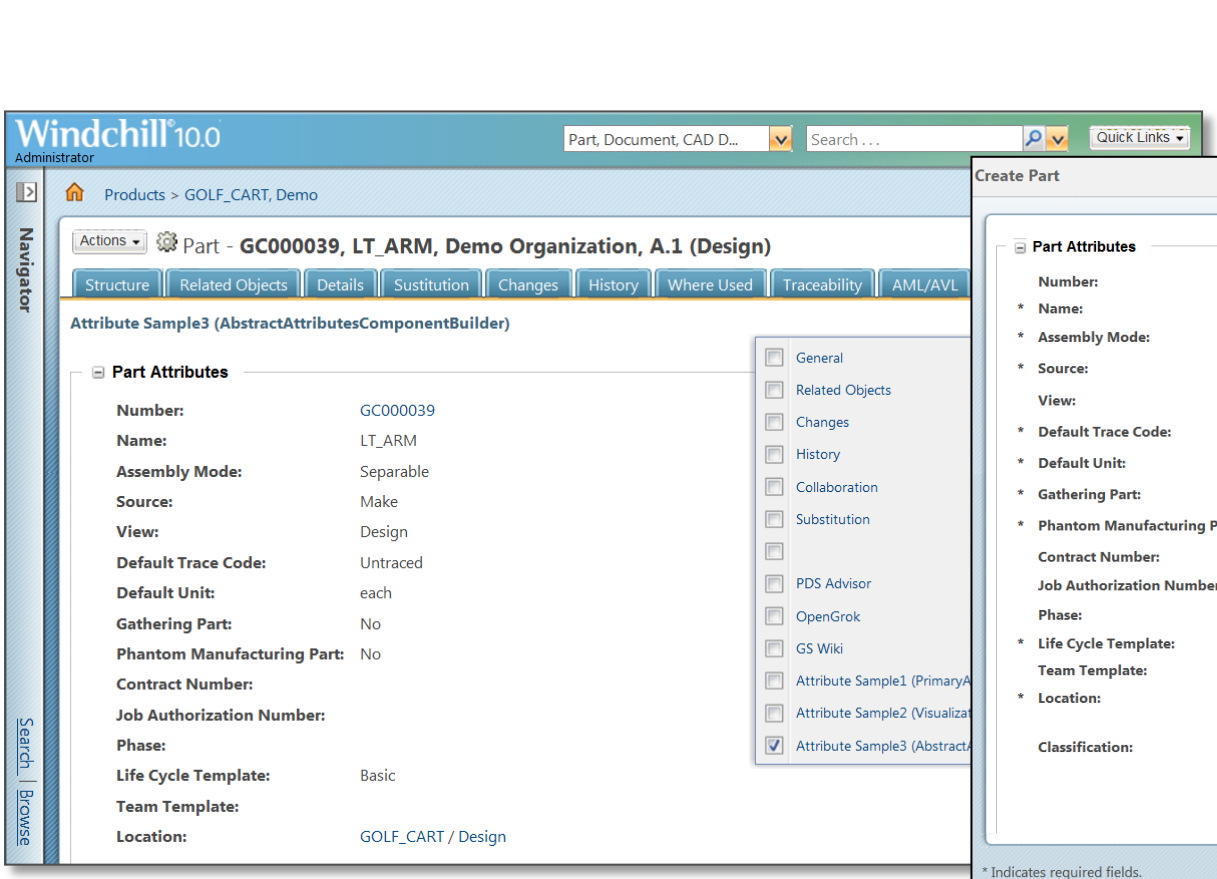
PTC

Test Result

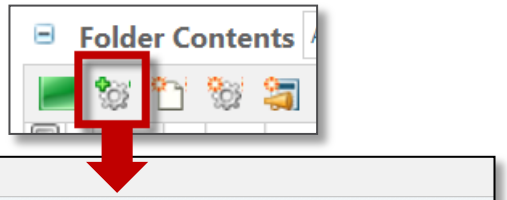
8. Restart Method Server and Test

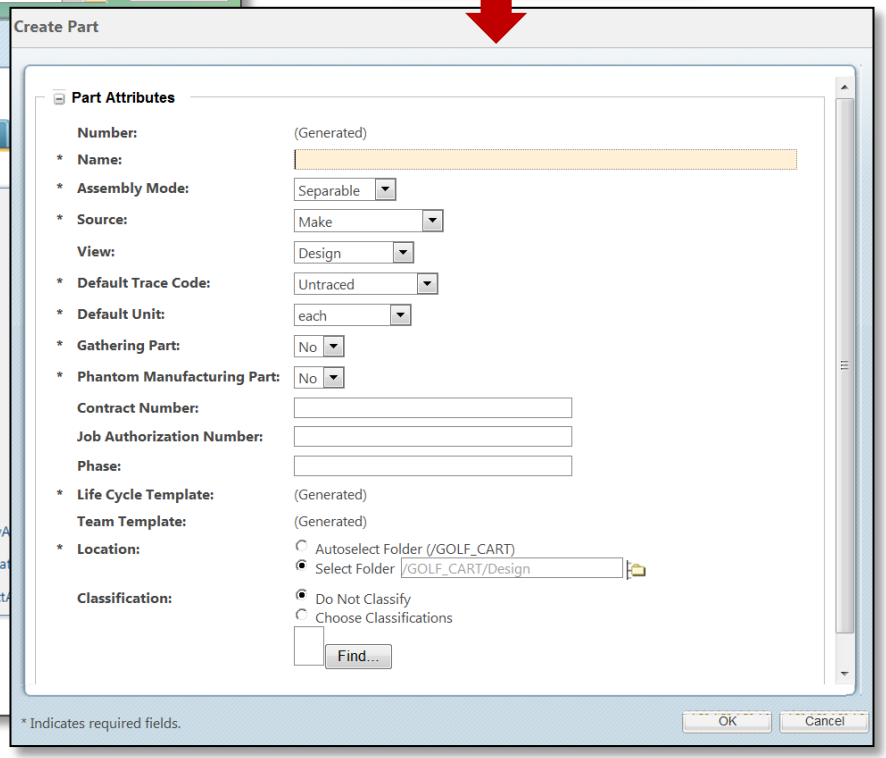
Check two types which are info page and wizard page.

CSCAdvancedAttributesPanelBuilder3.java



Info page used CREATE layout





Create wizard used CREATE layout

Data Utility

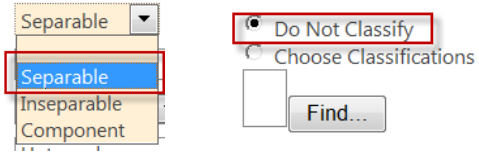
Background

Most of Windchill pages are constructed using GUI component which are table, tree or attributes panel, etc. Sometimes, we need to change part of UI Elements on GUI component like following.

- Need to change one item column style on the table.
- When creating some object, one input column want to control by other field on wizard page.
- Want to change from select box to check box design on create wizard.
- Want to show calculated value of especial attributes on info page.

What is UI Element?

Any visual element displayed on a Pages.



What is GUI component?

Object that carries information about a UI Element that is necessary to appropriately render the element on a page. It also includes the default renderer class that knows how to generate the appropriate display, based on the information in the GUI component.

What is renderer?

A Renderer is an object that knows how to generate HTML for a GUI Component.

All UI Component has been controlled by user interface renderer, so it is difficult to change partial user interface directly. In this time, if you want to change UI Element, generally we can use Data Utility.

What is Data Utility (1)

Data utilities are the glue that takes raw query data and converts it into the model data that a component can be built out of. In the case of tables, the data utilities are responsible for constructing particular cell values. Generally Data Utility is using Component Descriptor, Component Model and Component Bean for change UI Component.

What is Component Descriptor?

Encapsulates metadata about a UI component.

ComponentDescriptor objects can be nested. It is by convention that a table, tree, info page, attribute panel component descriptor has children that correspond to columns or properties.

Some of the primary properties of a ComponentDescriptor are:

- ✓ id — unique key
- ✓ label — Display text associated with the UI element
- ✓ mode — (VIEW, EDIT, SEARCH, CREATE)
- ✓ type — (INFO, PICKER, SEARCH, TABLE, WIZARD, WIZARD_TABLE)

What is Component Model?

The ComponentModel encapsulates three other models:

An **NmHTMLActionModel** that contains the actions for this component (toolbar for a table, third level navigation actions for an info page)

A **ComponentViewModel** that defines the view-related information for the component, if any

An **InfoEngine Group** that defines that actual data to be displayed in the component

What is Data Utility (2)

What is Component Bean?

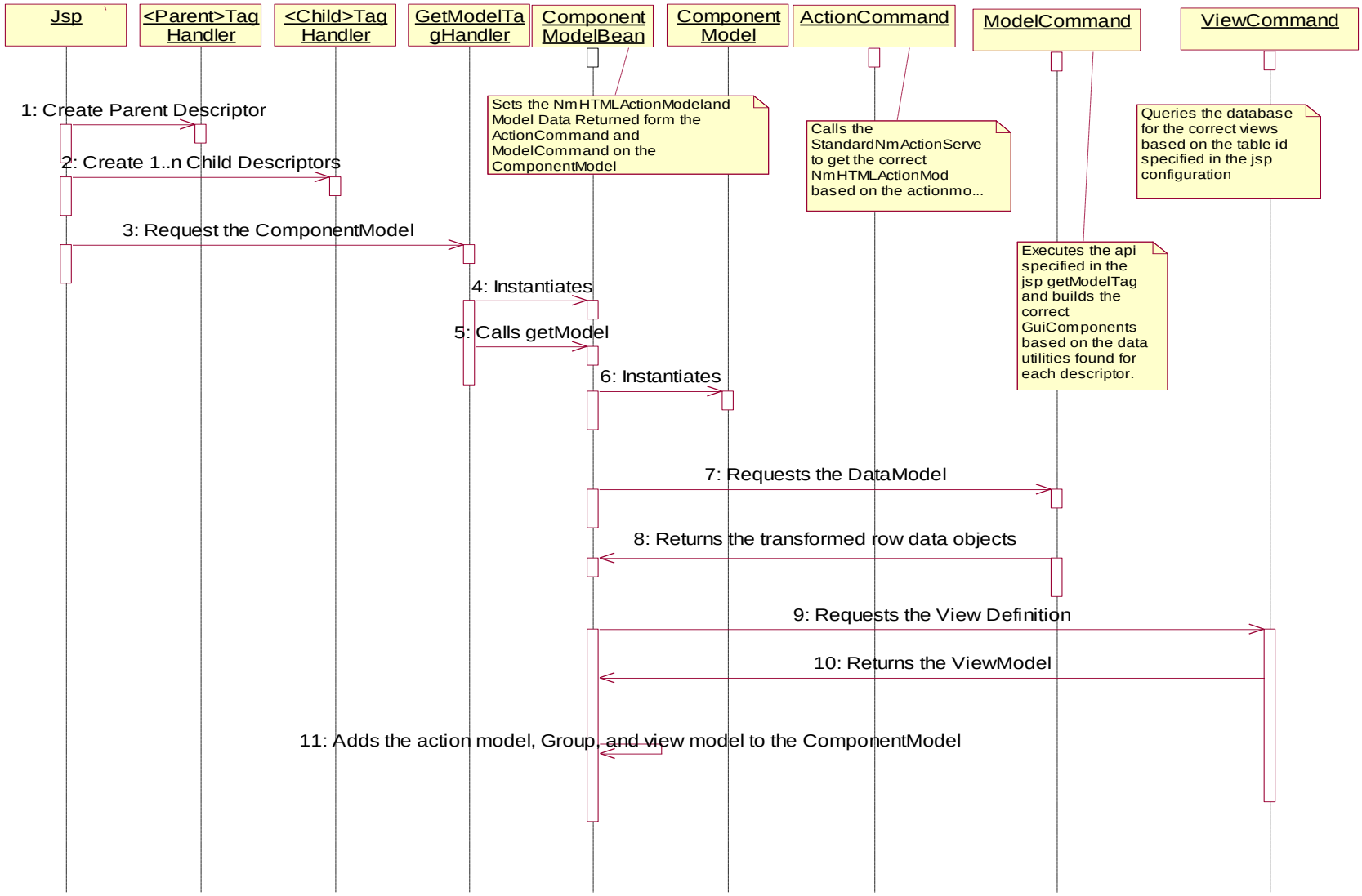
actually it does the work of building a ComponentModel.

- It constructs a BatchCommand with 3 subcommands to get each of the ComponentModel's 3 parts: an ActionCommand, a ComponentViewCommand, and a ModelCommand.
- It has hard-coded logic that creates one of the known ModelCommand implementations depending on the parameters specified in the jsp.

For example, it knows if the jsp specifies a service class and method name, to construct a ServiceModelCommand.

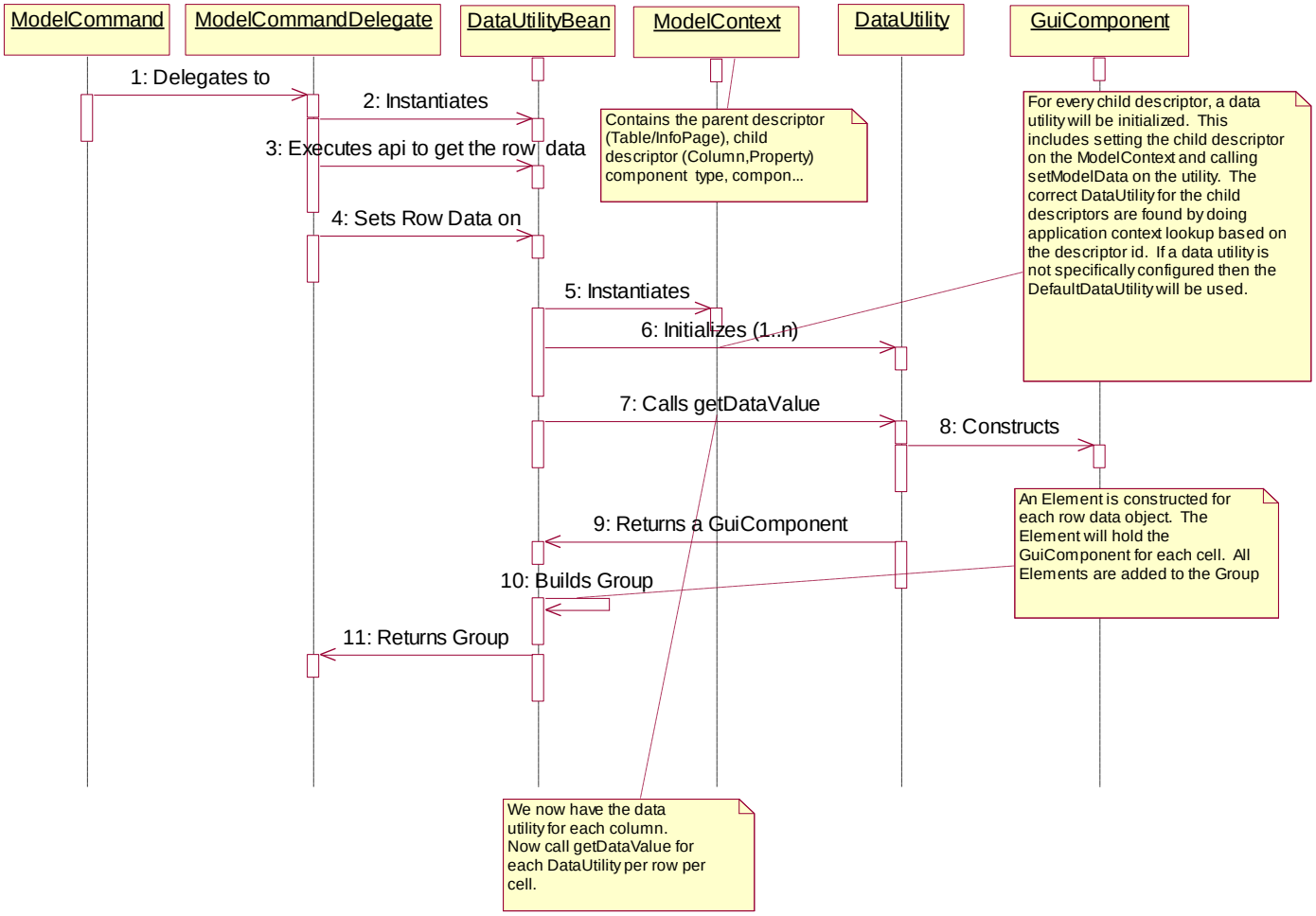
2. General Process for GUI Component

Getting data from request



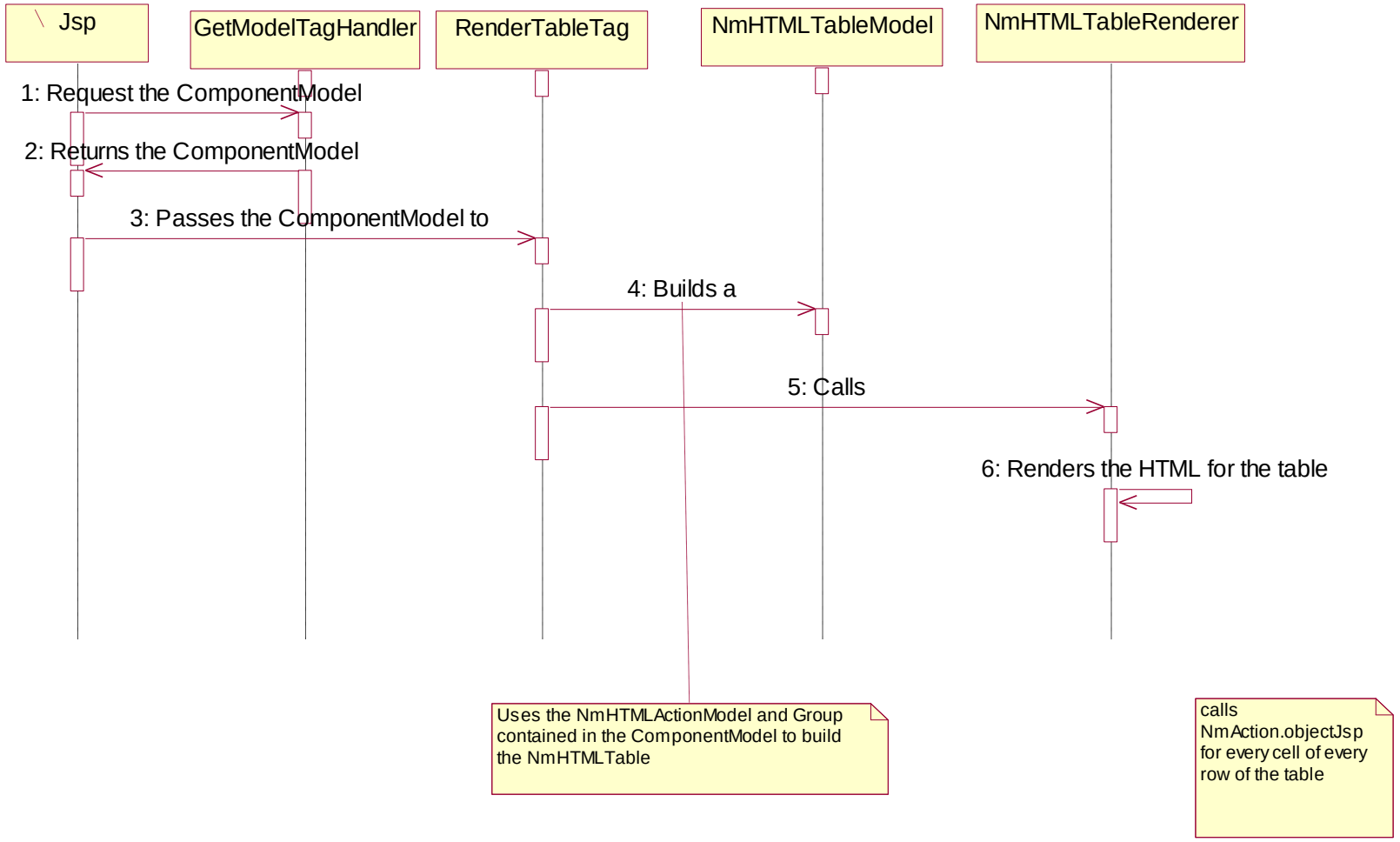
2. General Process for GUI Component

Transforming the data into GuiComponents



2. General Process for GUI Component

Displaying Data



GUI Components

This component will generally be constructed in the Method Server and then returned to the Servlet Container for rendering.

The GUI Component interface has one method that needs to be implemented:

public void draw(Writer out, RenderingContext renderContext) throws RenderingException;

→ *It just needs to understand, we will not touch OOTB function.*

Following is OOTB lists of GUI Components

- AttributeDisplayComponentCheckBox
- ComboBox
- DateDisplayComponent
- DateInputComponent
- EnumInputComponent
- UrlInputComponent
- IconComponent
- Label
- PushButton
- RadioButton
- RevisionInputComponent
- StringInputComponent
- LocationInputComponent
- AttributeGuiComponent
- AttributeInputComponent
- AttributeInputCompositeComponent
- BooleanInputComponent
- NumberInputComponent
- NumericDisplayComponent
- NumericInputComponent
- TextArea
- TextBox
- TextDisplayComponent
- PickerInputComponent

Create Custom DataUtility Class

The data utility interface (com.ptc.core.components.descriptor.DataUtility) has three core methods:

The ***getDataValue*** method gets the value that should be placed within a particular table cell. Typically, the object that is returned by this method should be an instance of GUIComponent. This interface has been retrofitted to NmString, NmDate.

Object getDataValue(String component_id, Object datum, ModelContext mc) throws WTException;

The ***setModelData*** method allows the data utility to prefetch data for all the row objects for a particular column that will be rendered. The method is called before getDataValue is called for any row, and is supplied a List of all the objects that will be processed.

void setModelData(String component_id, List objects, ModelContext mc) throws WTException;

The ***getLabel*** method allows the data utility to populate the descriptor with the correct label. There is a default implementation in the AbstractDataUtility for getting a label from a common ui rbinfo file.

String getLabel(String component_id, ModelContext mc) throws WTException;

When using DataUtility, we have to use a abstract super class which name is **AbstractDataUtility** generally. AbstractDataUtility is the child class of DataUtility.

PTC®

Sample DataUtility Class

```
public class CSCLifeCycleDataUtility extends AbstractDataUtility {
    public Object getDataValue(String component_id, Object datum, ModelContext mc) throws WTException {
        ...
        // Initialize ComboBox UI Component
        ComboBox comboBox = new ComboBox();
        comboBox.setName(component_id);
        comboBox.setColumnName(AttributeDataUtilityHelper.getColumnName(component_id, datum, mc));
        …
        ComponentMode mode = mc.getDescriptorMode();
        String selectClassName = CreateAndEditWizBean.getTypeNameFromTypePicker(mc.getNmCommandBean());
        …
        Class aClass = Class.forName(selectClassName);
        Vector vecLCTemplate = LifeCycleHelper.service.findCandidateTemplates(aClass, mc.getNmCommandBean().getContainerRef());
         for(int k=0; k<vecLCTemplate.size(); k++){
                lcTemplate = (LifeCycleTemplate)((LifeCycleTemplateReference)vecLCTemplate.get(k)).getObject();
                requstValue.add(lcTemplate.getName());
                displayValue.add(lcTemplate.getName());
         }
         …
         comboBox.setInternalValues(requstValue); //Key
         comboBox.setValues(displayValue); // display
         return comboBox;
    }

    private String getValue(String component_id, Object datum, ModelContext mc) throws WTException {
         return UtilityHelper.getStringValue(component_id, datum, mc);
    }
```

Register DataUtility

Customized data utility class can not use directly on GUI Component. All GUI Component can read service Identification which registered in MethodServer services cache – generally registered in “service.properties”.

Example: service.properties.xconf

```
<Service name="com.ptc.core.components.descriptor.DataUtility">
    <Option serviceClass="com.ptc.windchill.enterprise.classification.dataUtilities.DefaultClfPickerUtility"
        requestor="java.lang.Object"
        selector="classification.id"
        cardinality="duplicate"/>
</Service>
```

- serviceClass : DataUtility Class
- Selector : DataUtility Identification name (All GUI Component use this name)

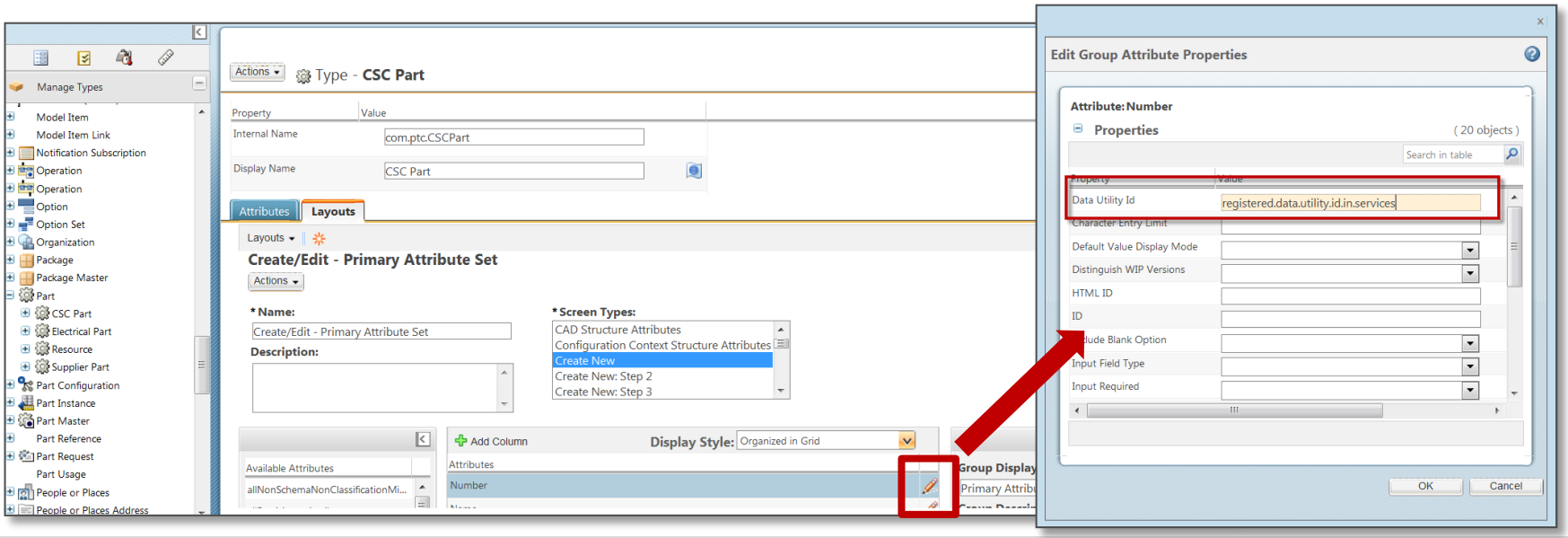
How to set data utility on the GUI Component

GUI Component will be rendered using several methodology, so we have to understand how to set custom data utility on each of GUI components.

- MVC designed – all MVC component has default function for setting data utility. (Tree/Table/Attribute Panel)

```
JcaTreeConfig treeConfig = (JcaTreeConfig) factory.newTreeConfig();
ColumnConfig col = factory.newColumnConfig("number", true);
col.setDataUtilityId("registered.data.utility.id.in.services");
treeConfig.addComponent(col);
```

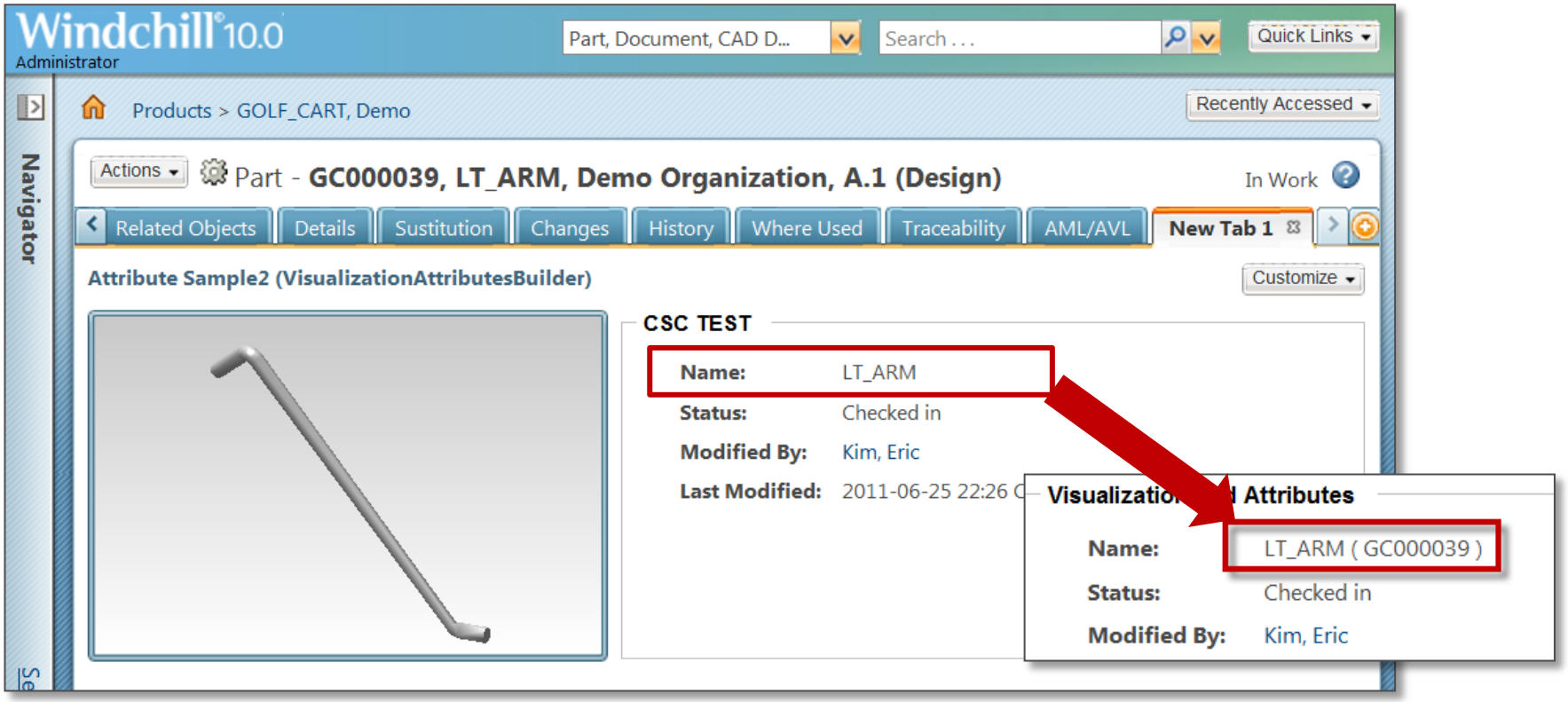
- Type Managed – Layout attributes. (Wizard / attribute panel in Info page)



5. Excise 1 – Change part attribute field (Layout)

Design part name value of info page

On the info page of WTPart, primary attribute group has name value. We will change the value using DataUtility. After gluing DataUtility, part name field will be shown name and number.



5. Excise 1 – Change part attribute field (Layout)

Create Data Utility

1. Create DataUtility class.

This class will get instance object from parameter for change label, and reset value using "**Label**" Component.

- Class name: CSCPartNameDataUtility
- Super Class: AbstractDataUtility
- Override: getDataValue

CSCPartNameDataUtility.java

```
package ext.csc.training.datautility;

Import wt.part.WTPart;
...
public class CSCPartNameDataUtility extends AbstractDataUtility {
   @Override
   public Object getDataValue(String component_id, Object datum, ModelContext mc) throws WTException {
      Label nameComponent = new Label("");

      nameComponent.setColumnName(AttributeDataUtilityHelper.getColumnName(component_id, datum, mc));
      nameComponent.setId(component_id);

      WTPart part = (WTPart)datum;
      nameComponent.setValue(part.getName() + " ( " + part.getNumber() + " )" );

      return nameComponent;
   }
}
```

5. Excise 1 – Change part attribute field (Layout)

Register data utility on MethodServer service

2. Update “service.properties.xconf” for registering custom data utility.

Open “$WT_HOME/codebase/service.properties.xconf” and add following block on the end of file. Actually it must be located the front of “</Configuration>” tag.

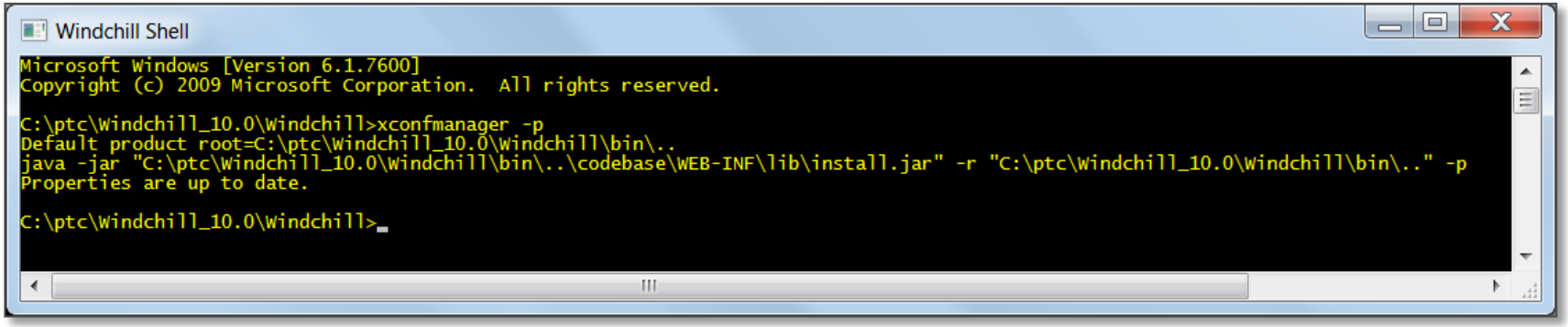
```
<Service context="default" name="com.ptc.core.components.descriptor.DataUtility">
        <Option serviceClass="ext.csc.training.datautility.CSCPartNameDataUtility"
                selector="cscPartNameDataUtility"
                requestor="java.lang.Object"
                cardinality="duplicate"  />
</Service>
```

DataUtility class

DataUtility Id

3. Register service on system configuration

Open Windchill shell and execute “xconfmanger –p”

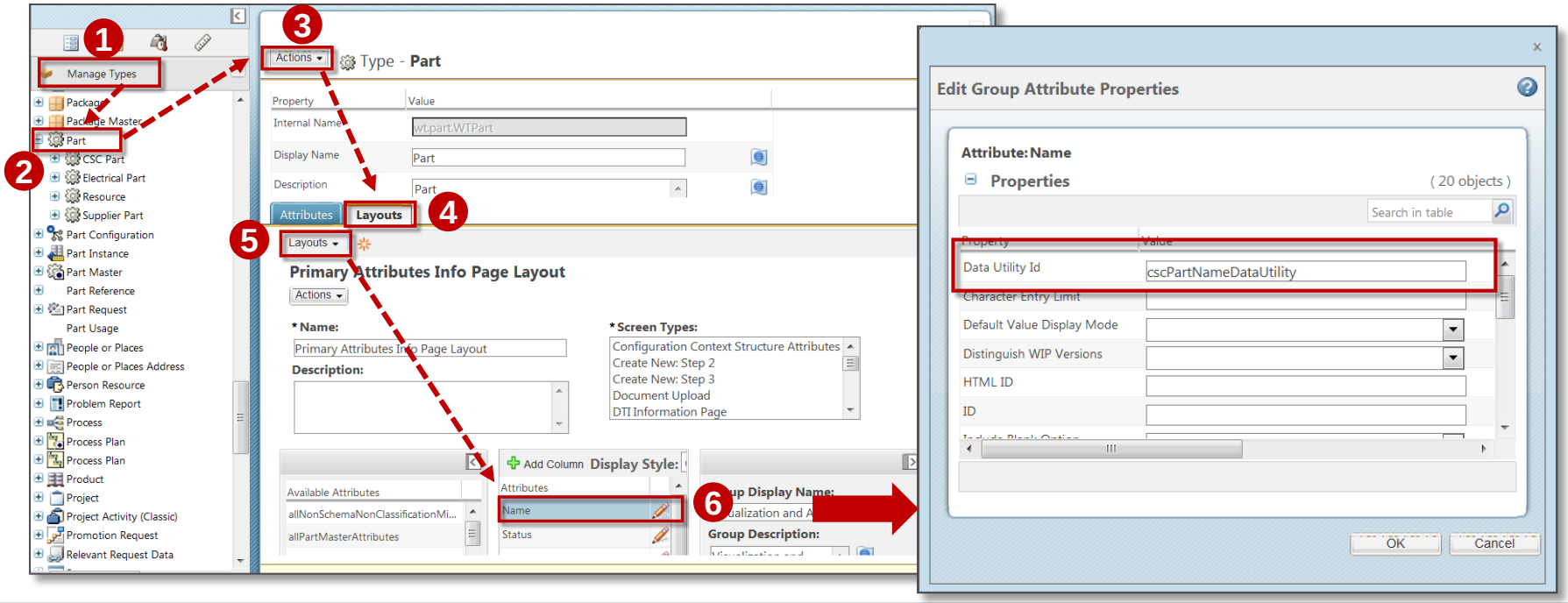


5. Excise 1 – Change part attribute field (Layout)

Set data utility id on part primary attributes layout

4. Open type manager and select part object. Finally set data utility Id on name attribute

A. Open Site > Utilities > Type and Attribute Management
B. Select Manage Types tab and select Part object
C. Select action menu on part detail page and select Edit menu
D. Select Layout tab and select "**Primary Attributes Info Page Layout**" of Layouts menu.
E. Select detail button of name attributes
F. Set data utility Id which name is "**cscPartNameDataUtility**" on the editing window of name

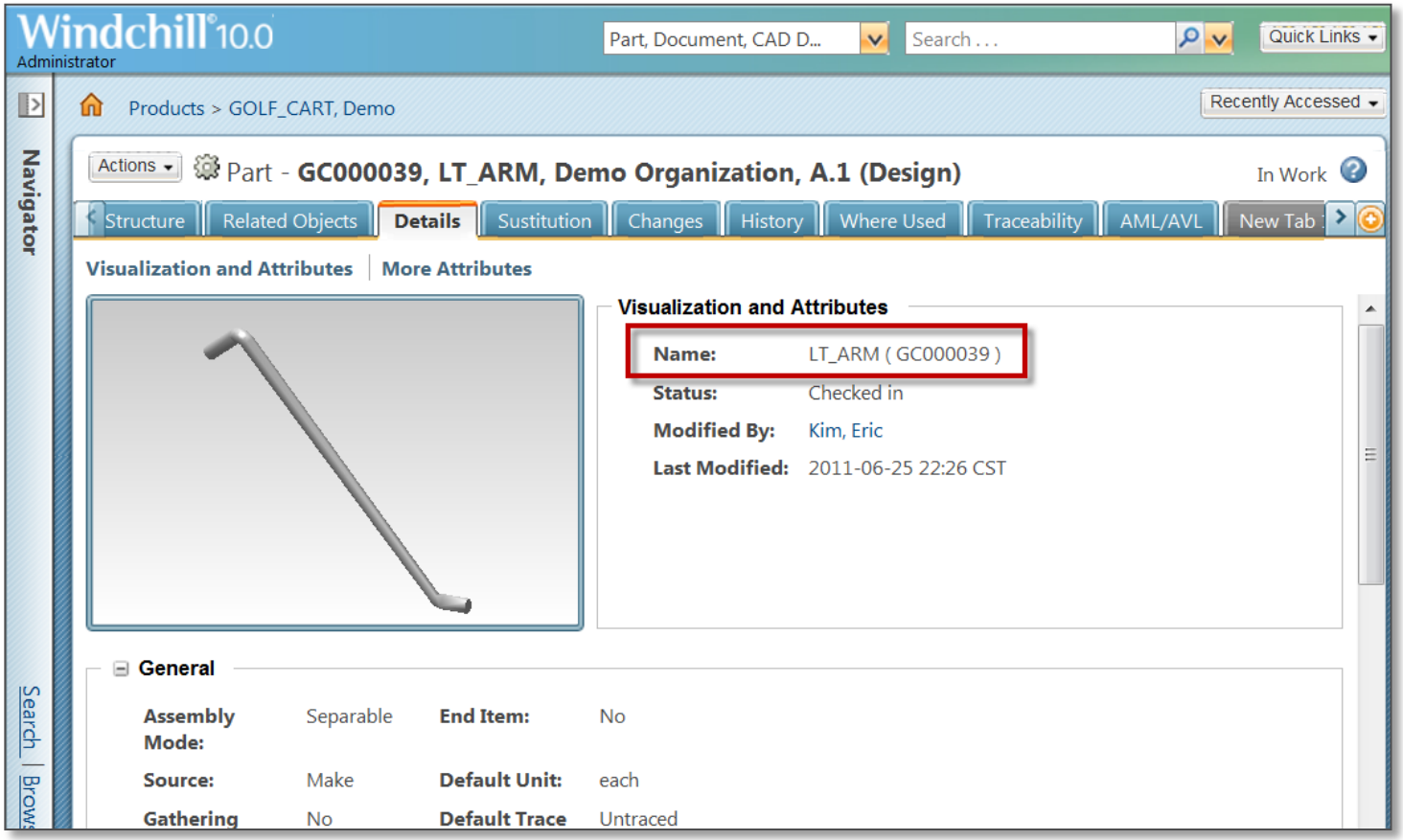


5. Excise 1 – Change part attribute field (Layout)

Test Result

6. Restart Method Server and Test

Open part information and check name attribute

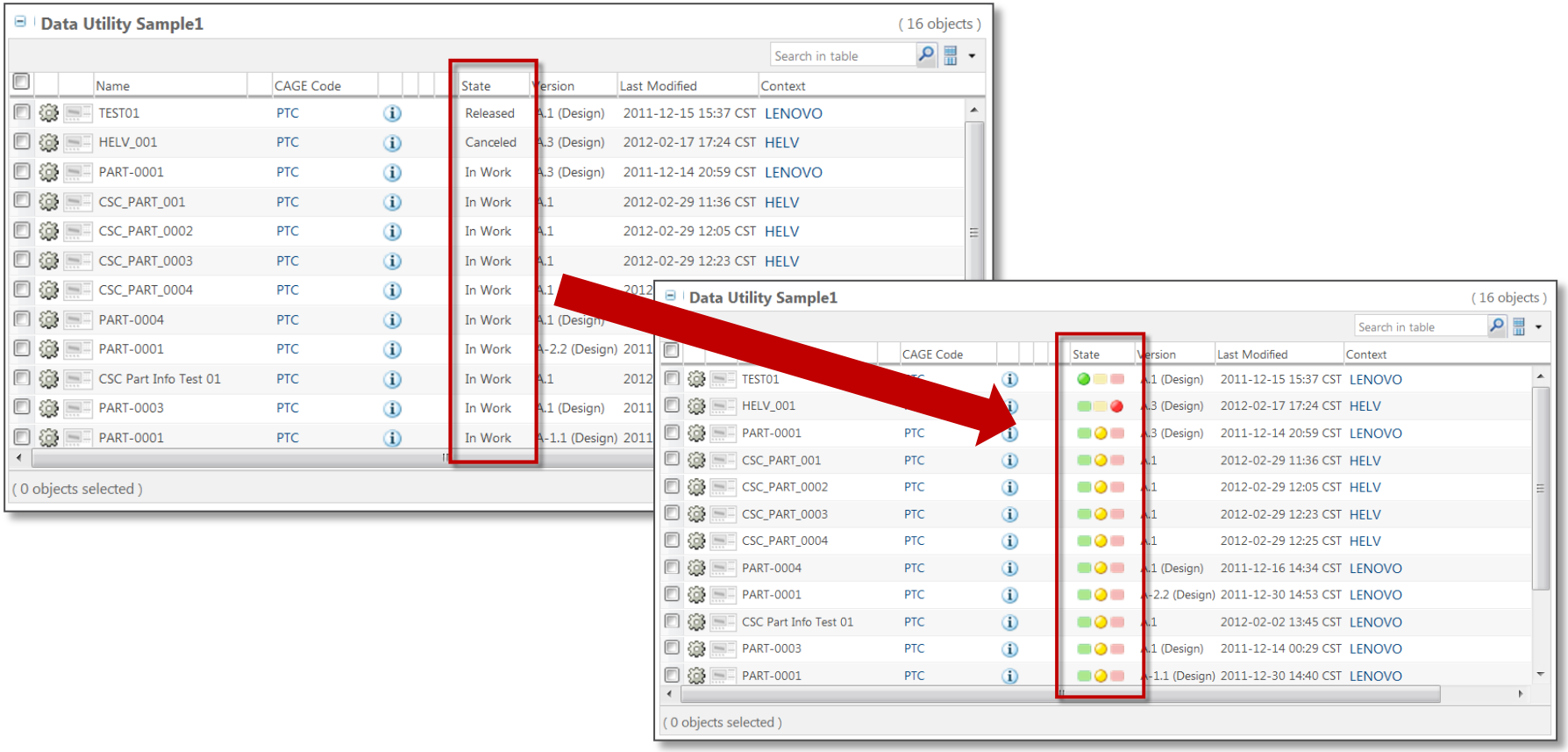


5. Excise 2 – Change table column (MVC)

Design table

We will change on column of table which was designed by MVC.

- Add lifecycle state column on table
- Change from state string to icon



Create MVC table class

1. Create table MVC class.

For testing, we need one simple MVC table class. When you create table class, you must set “STATE” column. Because we will try to change the column using data utility.

- Class name: CSCDataUtilitySampleTable
- MVC Id: eric.test.datautility_01

CSCDataUtilitySampleTable.java

```
package ext.csc.training.mvc;
...
@ComponentBuilder("eric.test.datautility_01")
public class CSCDataUtilitySampleTable extends AbstractComponentBuilder {
    public Object buildComponentData(ComponentConfig config, ComponentParams params) throws Exception {
        QuerySpec spec = new QuerySpec(WTPart.class);
        spec.appendWhere(new SearchCondition(WTPart.class, "master>number", SearchCondition.LIKE, "00000%"));
        LatestConfigSpec latestCSpec = new LatestConfigSpec();
        spec = latestCSpec.appendSearchCriteria(spec);
        return PersistenceHelper.manager.find(spec);
    }
    public ComponentConfig buildComponentConfig(ComponentParams params) throws WTException {
        …
        ColumnConfig col = factory.newColumnConfig(STATE, true);
        col.setDataUtilityId("cscLifecycleIconDataUtility");
        table.addComponent(col);
        …
    }
}
```

You must remember the data utility Id.

5. Excise 2 – Change table column (MVC)

PTC

Create Data Utility

2. Create DataUtility class.

This class will change image icon for each of state value, so it will use "**IconComponent**" class.

- Class name: CSCLifecycleIconDataUtility
- Super Class: AbstractDataUtility
- Override: getDataValue

CSCLifecycleIconDataUtility.java

```
package ext.csc.training.datautility;
…
public class CSCLifecycleIconDataUtility extends AbstractDataUtility {
   @Override
   public Object getDataValue(String component_id, Object datum, ModelContext mc) throws WTException {
      WTPart part = (WTPart)datum;
      String state = part.getState().getState().toString();

      String imgURL = null;
      if ( state.equals("RELEASED") ) { imgURL = "netmarkets/images/green.gif"; }
      else if ( state.equals("INWORK") ) { imgURL = "netmarkets/images/yellow.gif"; }
      else { imgURL = "netmarkets/images/red.gif"; }

      IconComponent iconComponent = new IconComponent(imgURL);
      return iconComponent;
   }
}
```

5. Excise 2 – Change table column (MVC)

Register data utility on MethodServer service

3. Update “service.properties.xconf” for registering custom data utility.

Open “$WT_HOME/codebase/service.properties.xconf” and add following block on the end of file. Actually it must be located the front of “</Configuration>” tag.

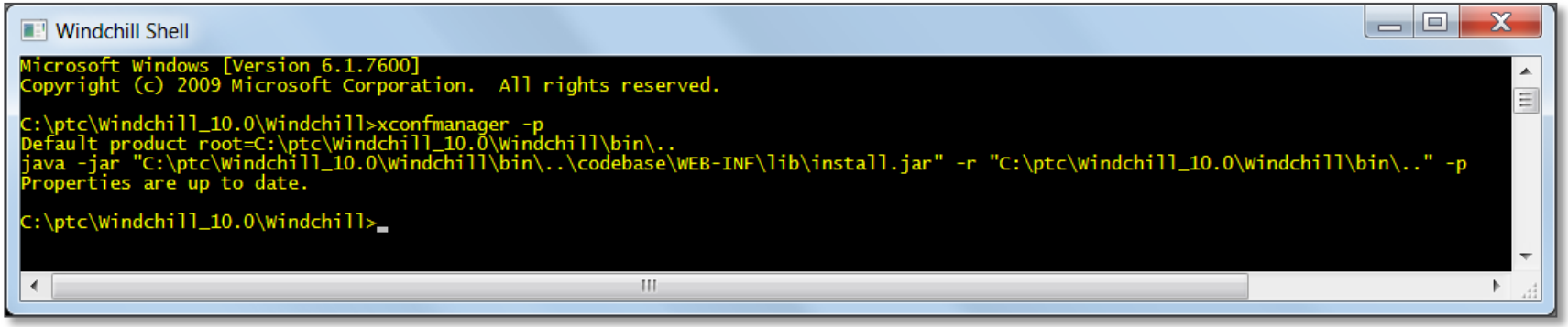
```
<Service context="default" name="com.ptc.core.components.descriptor.DataUtility">
        <Option serviceClass="ext.csc.training.datautility.CSCLifecycleIconDataUtility"
                selector="cscLifecycleIconDataUtility"
                requestor="java.lang.Object"
                cardinality="duplicate"  />
</Service>
```

DataUtility class

DataUtility Id

4. Register service on system configuration

Open Windchill shell and execute “xconfmanger –p”



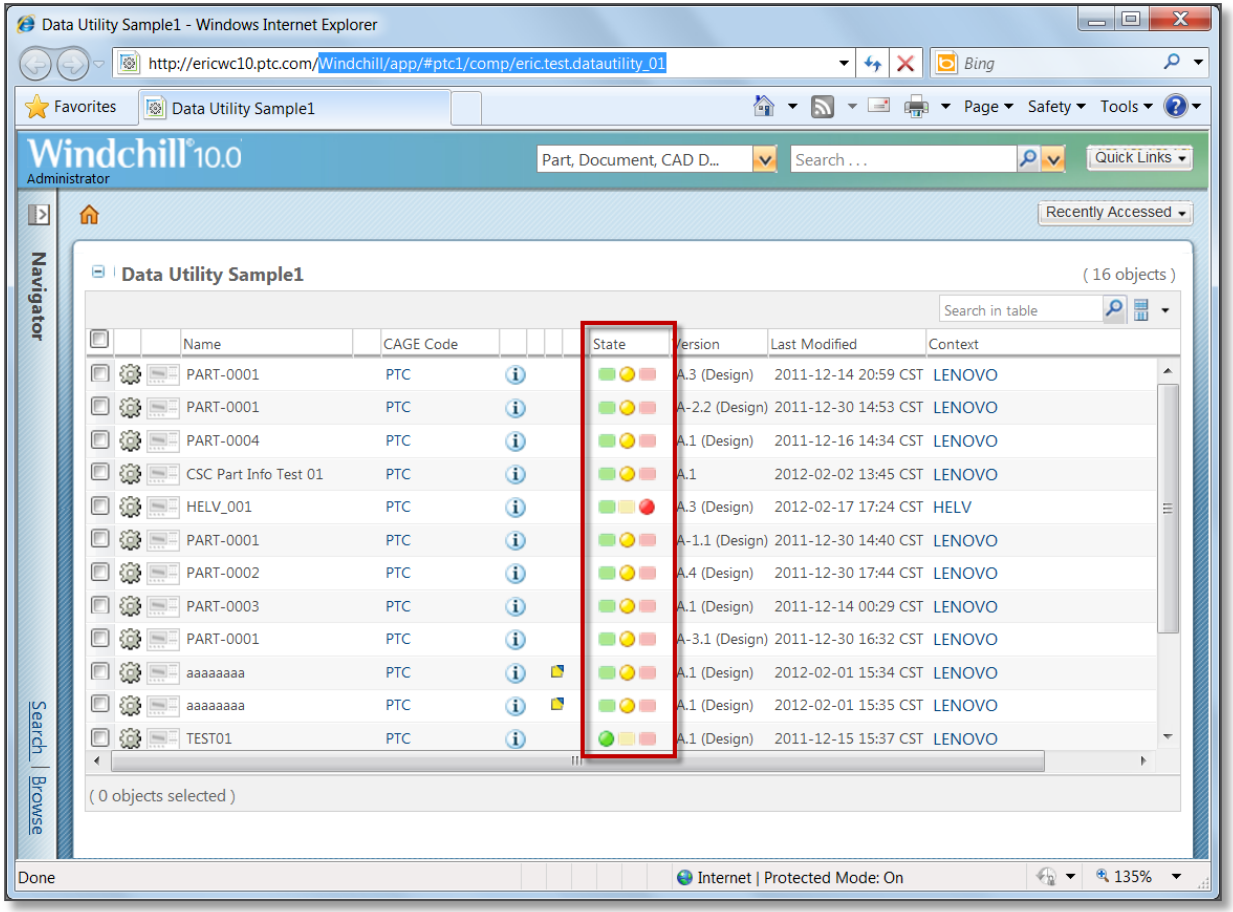
5. Excise 2 – Change table column (MVC)

Test Result

6. Restart Method Server and Test

Open Internet explorer and directly type on URL like following

http://localhost/Windchill/app/#ptc1/comp/eric.test.datautility_01



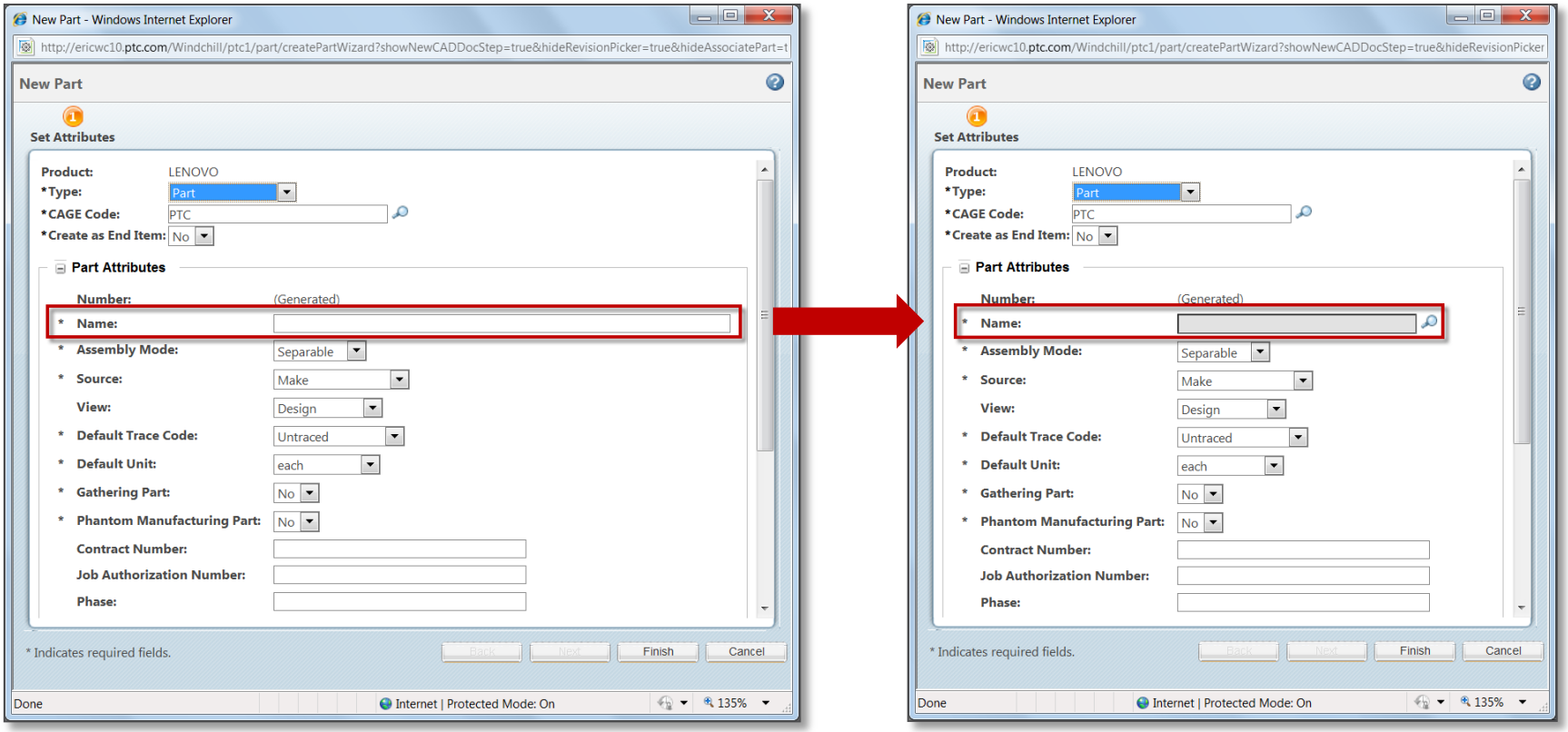
5. Excise 3 – Picker Data Utility Component

Design table

In data utility components, there exist several components. In this case, we will study some useful component which is picker component.

We had been created one custom wizard which is previous excise 8 of "Attributes Panel" section. We will reuse the sample and change name field to using picker data utility.

The picker component will use item picker, so picker configuration also will reuse excise 1 of "Picker".



5. Excise 3 – Picker Data Utility Component

Create Data Utility

CSCPickerSampleDataUtility.java

1. Create DataUtility class.

This class will make picker for name attribute. It will use "**PickerInputComponent**" class. If you want to make picker by yourself using general components which are "TextBox" and "ButtonComponent", It is very difficult to construct. In this case, Windchill core is supporting picker component. It is very simple to make picker data utility.

```
package ext.csc.training.datautility;
...
public class CSCPickerSampleDataUtility extends AbstractDataUtility {
   public Object getDataValue(String component_id, Object datum, ModelContext modelcontext) throws WTException {
      ComponentDescriptor componentdescriptor = modelcontext.getDescriptor();
      Map<Object, Object> map = componentdescriptor.getProperties();
      Object value = modelcontext.getRawValue();
      String attLabelName = getLabel(component_id, modelcontext);

      PickerRenderConfigs.setDefaultPickerProperty(map, PickerRenderConfigs.PICKER_ID, component_id);
      PickerRenderConfigs.setDefaultPickerProperty(map, PickerRenderConfigs.OBJECT_TYPE, "wt.part.WTPart");  // search target object
      PickerRenderConfigs.setDefaultPickerProperty(map, PickerRenderConfigs.PICKER_TITLE, "Search Part");
      PickerRenderConfigs.setDefaultPickerProperty(map, PickerRenderConfigs.READ_ONLY_TEXTBOX, "true");
      PickerRenderConfigs.setDefaultPickerProperty(map, "showTypePicker", "false");
      PickerRenderConfigs.setDefaultPickerProperty(map, PickerRenderConfigs.COMPONENT_ID, "cscPartPicker");  // picker search criteria configuration
      PickerRenderConfigs.setDefaultPickerProperty(map, "defaultHiddenValue", (String)value);
      PickerRenderConfigs.setDefaultPickerProperty(map, "pickedAttributes", "name");  // Picker return value to input field
      PickerRenderConfigs.setDefaultPickerProperty(map, "includeTypeInstanceId", "true");
      PickerRenderConfigs.setDefaultPickerProperty(map, "pickerCallback", "partPickerCallback");  // Picker Call back function name

      PickerInputComponent pickerinputcomponent = new PickerInputComponent(attLabelName, (String)value, PickerRenderConfigs.getPickerConfigs(map), 30);
      pickerinputcomponent.setColumnName(AttributeDataUtilityHelper.getColumnName(component_id, datum, modelcontext));
      pickerinputcomponent.setId(component_id);
      pickerinputcomponent.setName(component_id);
      pickerinputcomponent.setRequired(AttributeDataUtilityHelper.isInputRequired(modelcontext));
      return pickerinputcomponent;
   }
}
```

5. Excise 3 – Picker Data Utility Component

Register data utility on MethodServer service

2. Update “service.properties.xconf” for registering custom data utility.

Open “$WT_HOME/codebase/service.properties.xconf” and add following block on the end of file. Actually it must be located the front of “</Configuration>” tag.

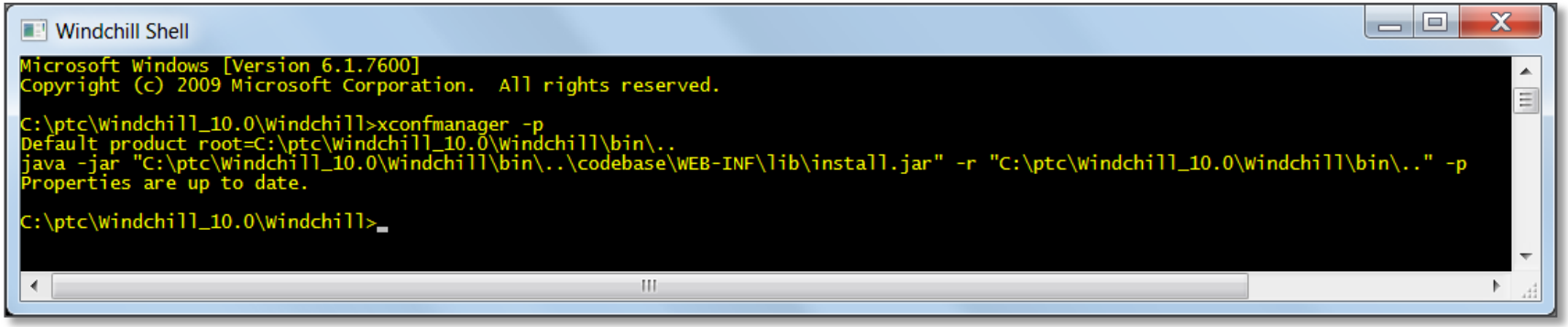
```
<Service context="default" name="com.ptc.core.components.descriptor.DataUtility">
        <Option serviceClass="ext.csc.training.datautility.CSCPickerSampleDataUtility"
                selector="cscPickerDataUtility"
                requestor="java.lang.Object"
                cardinality="duplicate"  />
</Service>
```

DataUtility class

DataUtility Id

3. Register service on system configuration

Open Windchill shell and execute “xconfmanger –p”

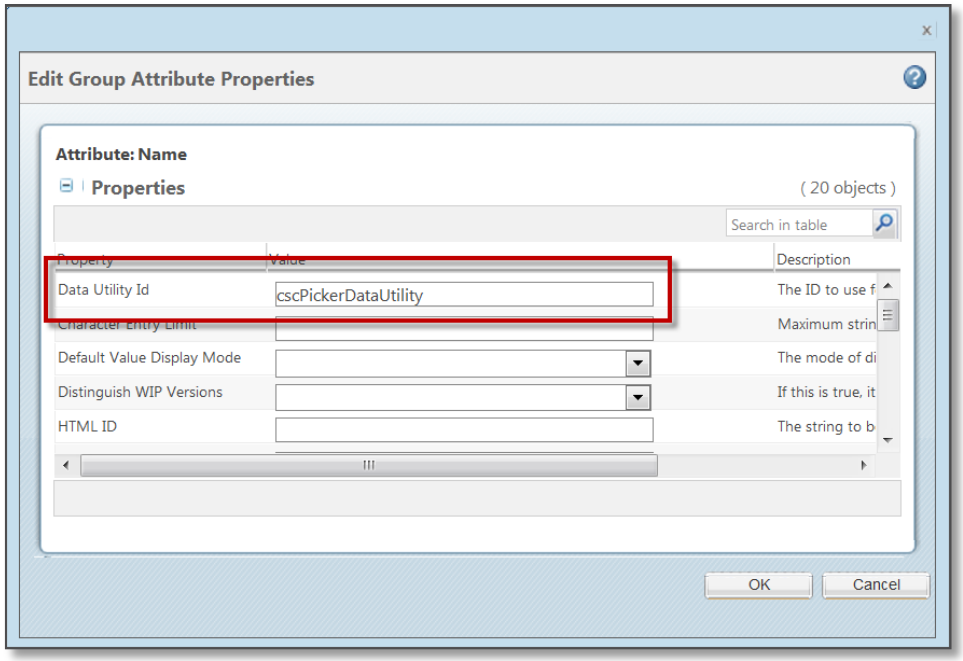


5. Excise 3 – Picker Data Utility Component

Set data utility id on part primary attributes layout

4. Open type manager and select part object. Finally set data utility Id on name attribute

A. Open Site > Utilities > Type and Attribute Management
B. Select Manage Types tab and select Part object
C. Select action menu on part detail page and select Edit menu
D. Select Layout tab and select "**Create New Layout**" of Layouts menu.
E. Select detail button of name attributes
F. Set data utility Id which name is "**cscPickerDataUtility**" on the editing window of name



4. Picker call back function

5. Add call back javascript on wizard body page.

When we use picker data utility, it will be constructed different way comparing using picker customization. So we need to implement other way to get data and set data to input field.

Add following javascript on *"$WT_HOME/codebase/netmarkets/jsp/csc/createWizardSample.jsp"*.

```
<script>
function partPickerCallback (objects, pickerID, attr, displayFieldId)
{
		var updateHiddenField = document.getElementsByName(pickerID)[0];
		var updateDisplayField = document.getElementsByName(displayFieldId)[0];
		updateHiddenField.value = objects.pickedObject[0].oid;
		updateDisplayField.value = objects.pickedObject[0].name;
}
</script>
```

Following is the javascript when using picker customization

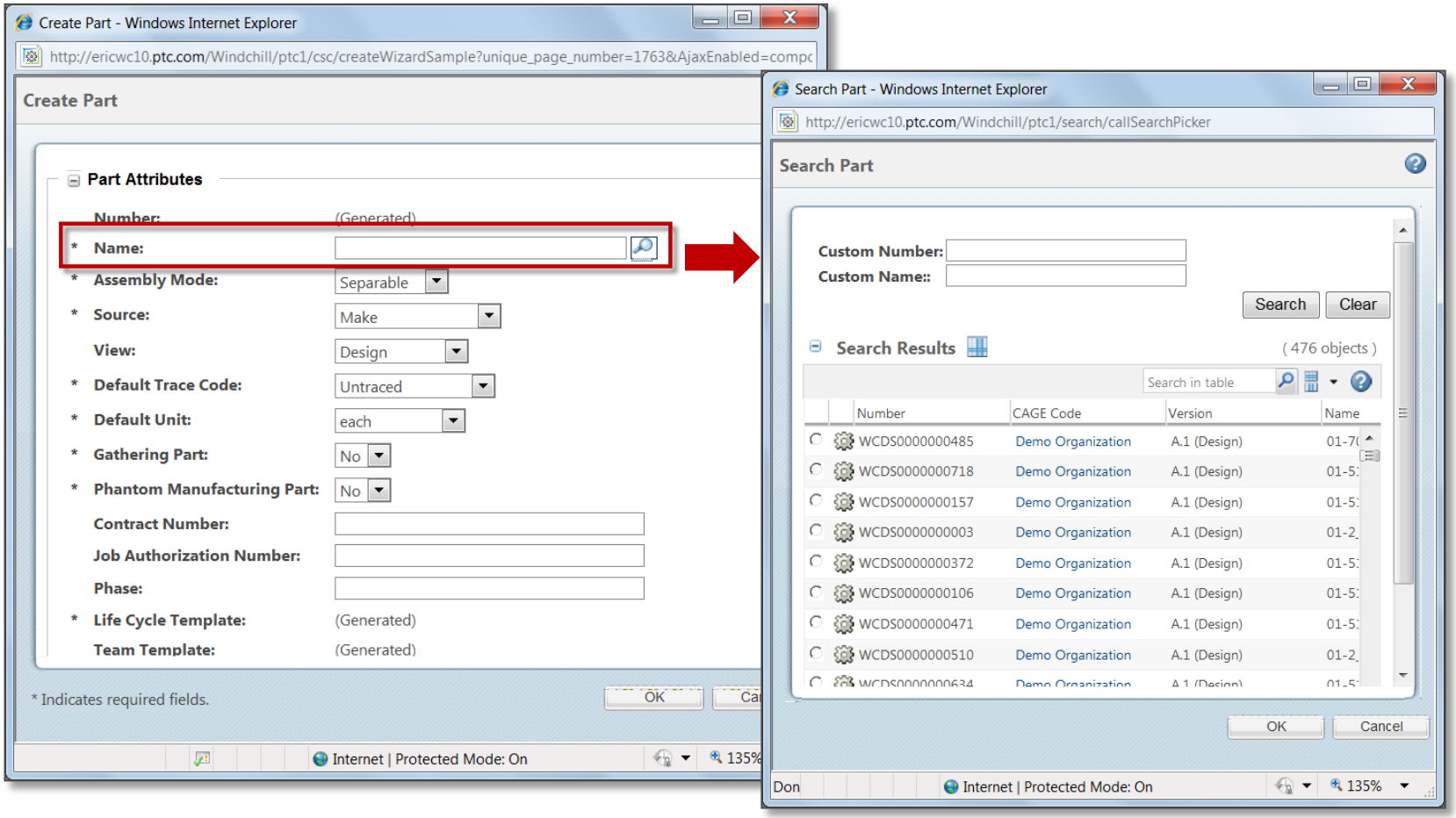
```
<script>
function partPickerCallback (objects, pickerID)
{
		document.getElementById(pickerID +"$label$").setAttribute("value",objects.pickedObject[0].name);
		document.getElementById(pickerID +"$label$___old").setAttribute("value",objects.pickedObject[0].name);
}
</script>
```

5. Excise 3 – Picker Data Utility Component

Test Result

6. Restart Method Server and Test

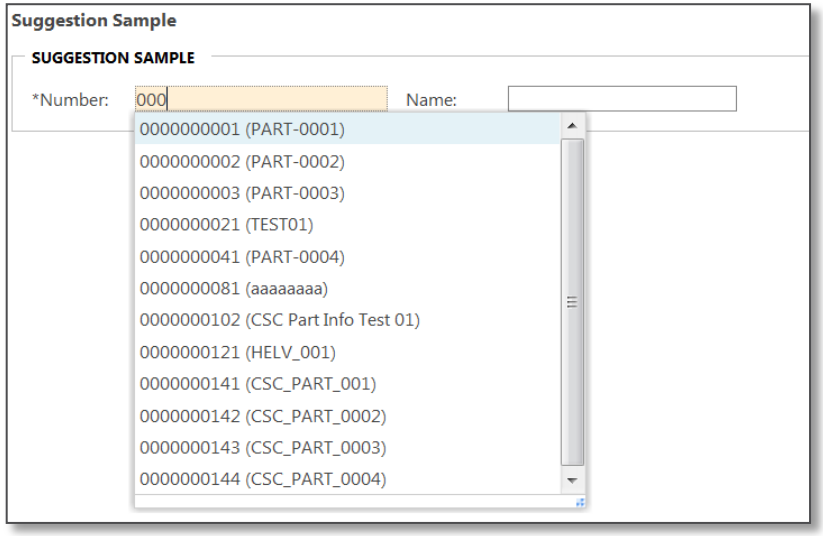
Open product folder and click custom button. (Excise 8 of Attributes panel)



Suggestion Text Box

Suggestion text box on wrapper tag

Previously we had been studied about wrapper tag. Windchill 10 has one useful tag which name is "**suggestTextBox**". It is same kinds of text input box, but it try to search target object and show the suggestion result like following when you enter text at input box.



The tag format is like following:

```
<wrap:suggestTextBox name="number" id="number" serviceKey="cscNumberSuggest" maxlength="30"
size="10" onblur="this.value=this.value.toUpperCase()" />
```

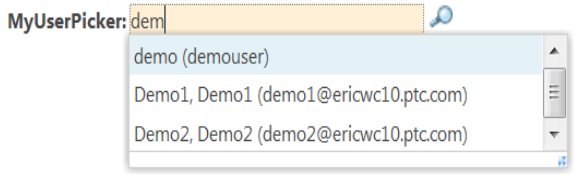
It can modify by ourselves which target object we will find and which search condition we will use for suggest box result. We will skip to explain wrapper tag. (Look Customization Guide – 1)

Suggestion box on picker

Some pickers also can use suggestion configuration, but unfortunately suggest configuration doesn't support on every picker. Following is the supported picker lists.

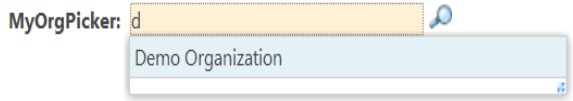
- User Picker

```
<wctags:userPicker id="testUserPicker" label="MyUserPicker: "
        readOnlyPickerTextBox="false" editable="true" showSuggestion="true"
        suggestMinChars="2" />
```



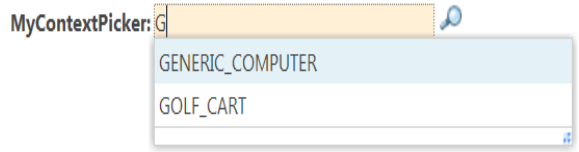
- Organization Picker

```
<wctags:organizationPicker id="orgPicker" label="MyOrgPicker: "
        readOnlyPickerTextBox="false" editable="true" showSuggestion="true"
        suggestMinChars="1" />
```



- Context Picker

```
<wctags:contextPicker id="contextPicker" label="MyContextPicker: "
        pickerTitle="ContextPicker"
        readOnlyPickerTextBox="false" editable="true" showSuggestion="true"
        suggestMinChars="1" />
```

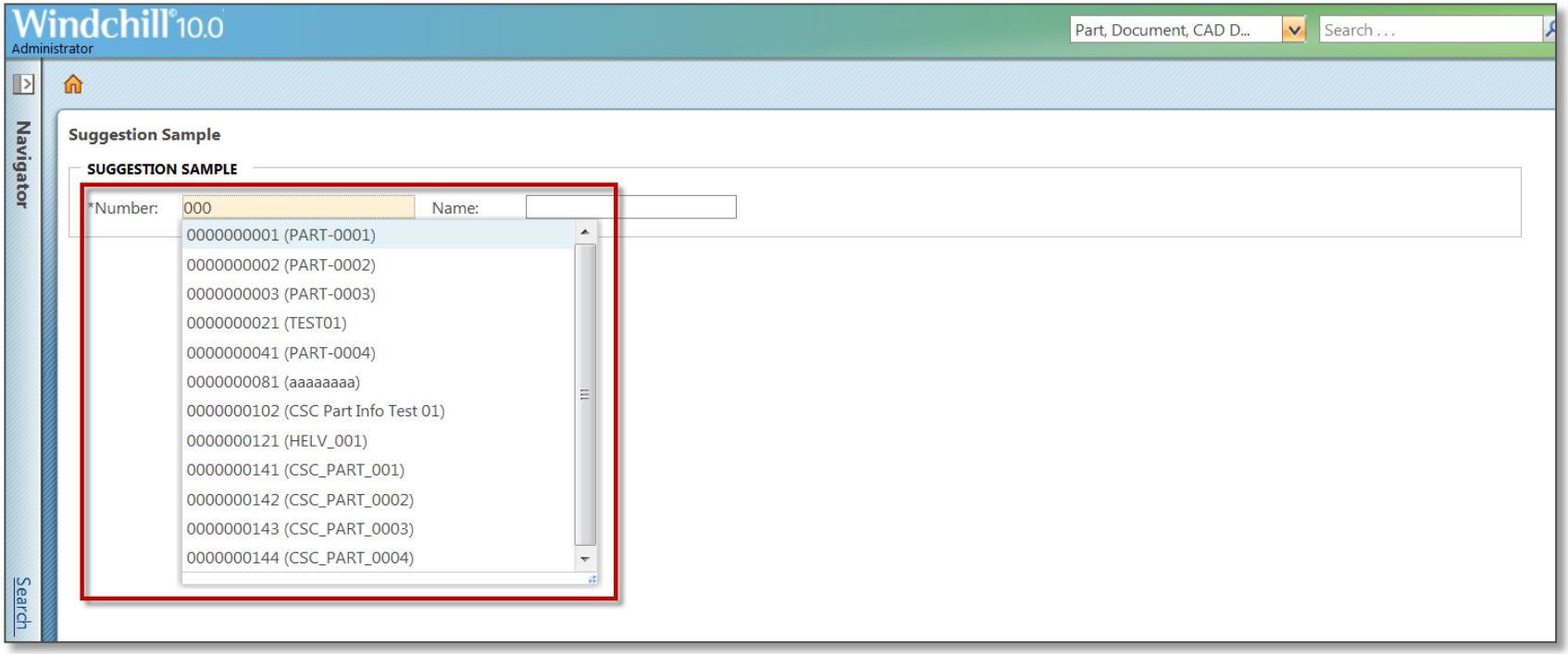


This picker also can use customized suggestion using "suggestSeriveKey" which is registered suggest class.

2. Excise 1 – Simple suggestion text box

Design table

Generally we use suggestion text box on search criteria, so we will create simple search criteria using wrapper tag. (No search action button) And then, finally, we will set suggestion class on wrapper tag.



2. Excise 1 – Simple suggestion text box

PTC

Suggestion Class

1. Create Suggestion class

When user enter some value on “suggestTextBox”, this class will be called and made a result for showing data.

- Class Name: **CSCNumberSuggest**
- Implements: **Suggestable**
- Override: **getSuggestions**

CSCNumberSuggest.java

```
package ext.csc.training.suggestion;
…
public class CSCNumberSuggest implements Suggestable {  //Implement Suggestable interface
   public Collection<SuggestResult> getSuggestions(SuggestParms suggestParams) {  // Override function
      List<SuggestResult> list = new ArrayList<SuggestResult>();
      try {
         String partNumber = suggestParams.getSearchTerm().toUpperCase() + "%";
         QuerySpec spec = new QuerySpec(WTPartMaster.class);
         SearchCondition sc = new SearchCondition(WTPartMaster.class, WTPartMaster.NUMBER, SearchCondition.LIKE, partNumber, false);
         spec.appendWhere(sc, new int[]{0});
         QueryResult result = PersistenceHelper.manager.find(spec);
         WTPartMaster oneMaster = null;
         while(result.hasMoreElements()) {
            oneMaster = (WTPartMaster)result.nextElement();
            list.add(SuggestResult.valueOf(oneMaster.getNumber(), oneMaster.getName()));
         }
      } catch(WTException wte) {}
      return list;
   }
}
```

2. Excise 1 – Simple suggestion text box

Register suggestion class on MethodServer service

2. Update “service.properties.xconf” for registering custom data utility.

Open “$WT_HOME/codebase/service.properties.xconf” and add following block on the end of file. Actually it must be located the front of “</Configuration>” tag.

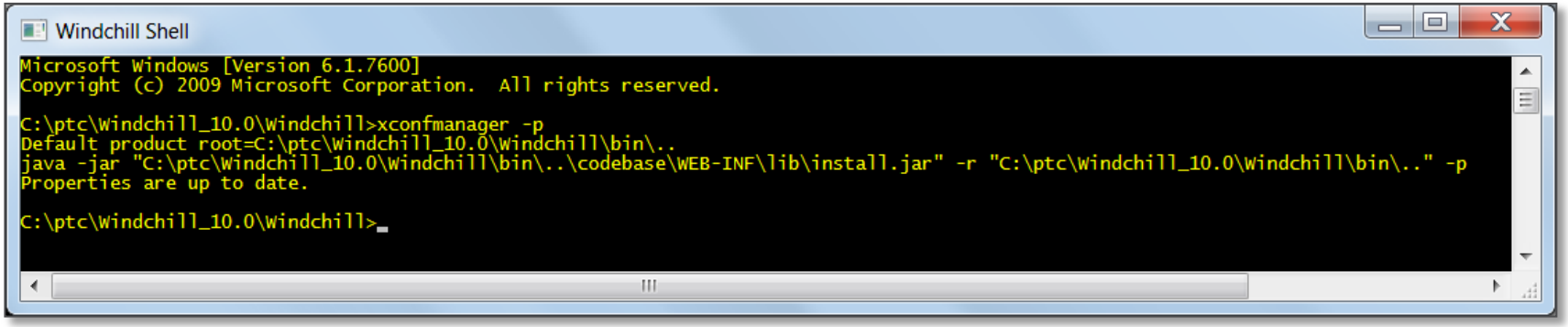
```
<Service context="default" name="com.ptc.core.components.descriptor.DataUtility">
        <Option serviceClass="ext.csc.training.suggestion.CSCNumberSuggest"
                selector="cscNumberSuggest"
                requestor="null"
                cardinality="duplicate"  />
</Service>
```

Suggestable class

Suggestable Id

3. Register service on system configuration

Open Windchill shell and execute “xconfmanger –p”



2. Excise 1 – Simple suggestion text box

Create JSP

4. Create JSP page including wrapper tag

Add two input fields. One is number field and the other is name field. The number field will use “suggestTextBox” wrapper tag. ($WT_HOME/codebase/netmarkets/jsp/csc/searchSample.jsp)

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ include file="/netmarkets/jsp/util/begin.jspf"%>
<%@ taglib prefix="wrap" uri="http://www.ptc.com/windchill/taglib/wrappers"%>

<b>Suggestion Sample</b>
<fieldset class="x-fieldset x-form-label-left“ id="Visualization_and_Attributes" style="width: 1024px;">
    <legend>SUGGESTION SAMPLE</legend>
    <table>
       <tr>
                  <td scope="row" width="50" class="tableColumnHeaderfont" align="left">*Number:</td>
                  <td class="tabledatafont" align="left"> 
                        <wrap:suggestTextBox
                                    name="number" id="number" serviceKey="cscNumberSuggest" maxlength="30" size="10“
                                    onblur="this.value=this.value.toUpperCase()" />
                  </td>
                  <td scope="row" width="50" class="tableColumnHeaderfont" align="left">Name:</td>
                  <td class="tabledatafont" align="left"> 
                         <wrap:textBox name="name" id="name" maxlength="100" size="20"/>
                  </td>
         </tr>
     </table>
</fieldset>
<%@ include file="/netmarkets/jsp/util/end.jspf"%>
```

2. Excise 1 – Simple suggestion text box

Set style on the JSP

5. Update style configuration on the JSP

searchSample.jsp

```
<style type="text/css">
.tabledatafont {
                        padding: 0px 10px 0px 10px;
}
select.ppdata{
width: 98%;
width: 150px;
align: left;
}
input.ppdata {
width: 150px;
align: left;
}
input {
width: 120px;
align: left;
}
.hlpTxt {
display:none;
}
</style>
```

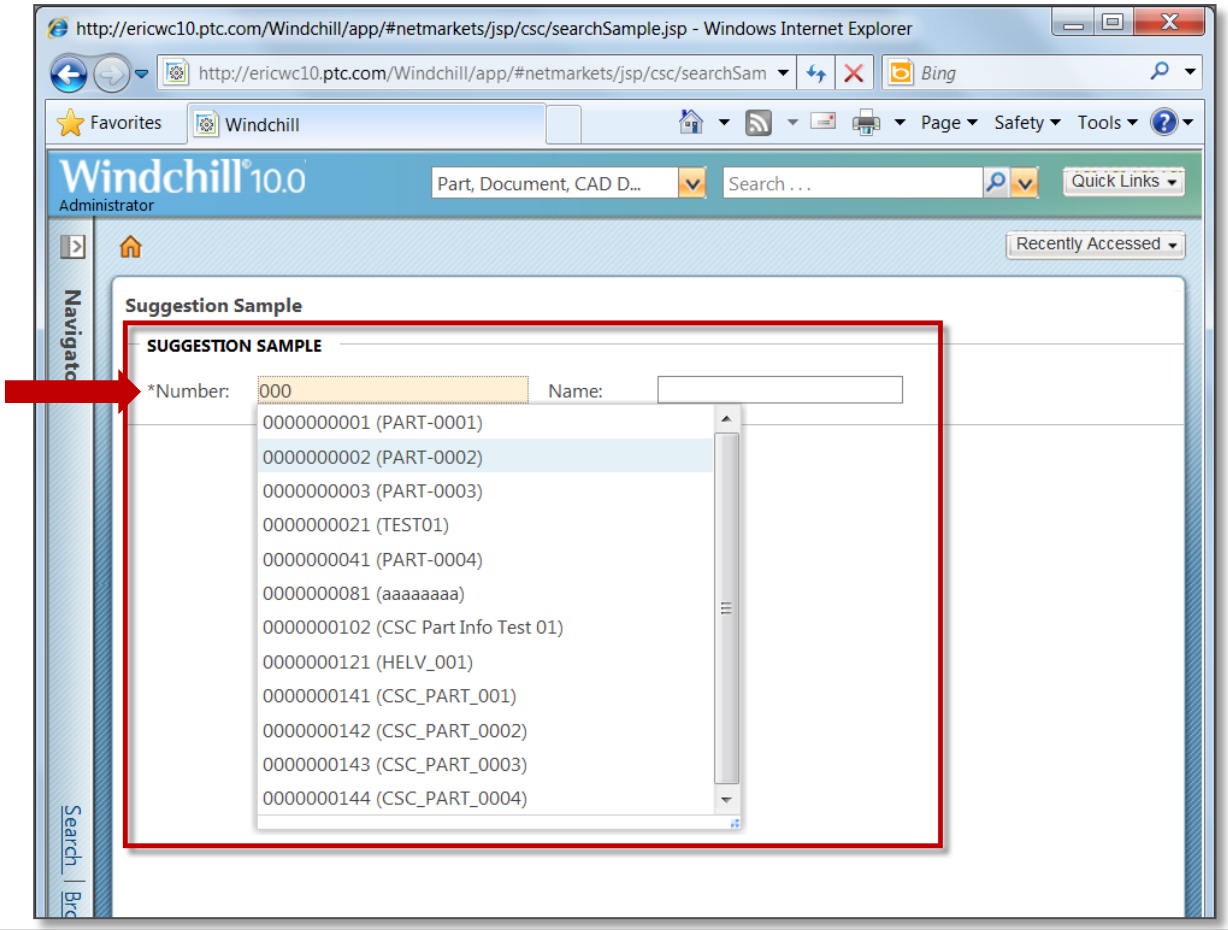
2. Excise 1 – Simple suggestion text box

Test Result

6. Restart Method Server and Test

Open explorer and enter following URL:

http://localhost/Windchill/app/#netmarkets/jsp/csc/searchSample.jsp



Epilogue

Thanks Letter

This document was started from my note for how to customized new Windchill user interfaces.

But I thought it was not everything, so I would like to gather higher information from all of PTC consultants. Finally I decided to share documents to my colleague, and want to communicate with all.

So if you have other experiences or best cases, please let me know. I will be really appreciate your support, and I will add your best cases on my document.

Following is my information. Don't be hesitate to call me:

Eric Kim (Principal Consultant)
Office : +86-29-6858-5337 (Can use English / Korean / Chinese)
Mobile: +86-183-9239-1492 (Can use English / Korean / Chinese)
E-mail: yckim@ptc.com (Can use English / Korean / Chinese / Japanese)

PTC®

the product development company

Thank You