

# WC10 Customizing

Eric Kim ([yckim@ptc.com](mailto:yckim@ptc.com))



## Document Properties

<b>File Name</b>	WC10 UI Customization
<b>Status</b>	Published

## Change History

Date	Author	Version	Description
06/24/2013	Eric Kim	1.0	Initial
07/16/2013	Indira Dash & Ritu Kathuria	1.1	Editorial Review
07/30/2013	Ketan Koli	1.2	Final touchup

## PTC Product Version

PTC Product	Version
Windchill PDMLink	Windchill 10.0 M040

## Modules

- [Eclipse Configuration](#)
- [Useful Eclipse Configuration](#)
- [Modeling with Annotation](#)
- [Action and Property Report](#)
- [Log and Navigation](#)
- [MVC](#)
- [Table](#)
- [Search and Result Table](#)
- [Other Table Features](#)
- [Tree](#)
- [Info Page Customization](#)
- [Wizard Customization](#)
- [Picker Customization](#)
- [Attribute Panel](#)
- [Data Utility](#)
- [Suggestion Text Box](#)

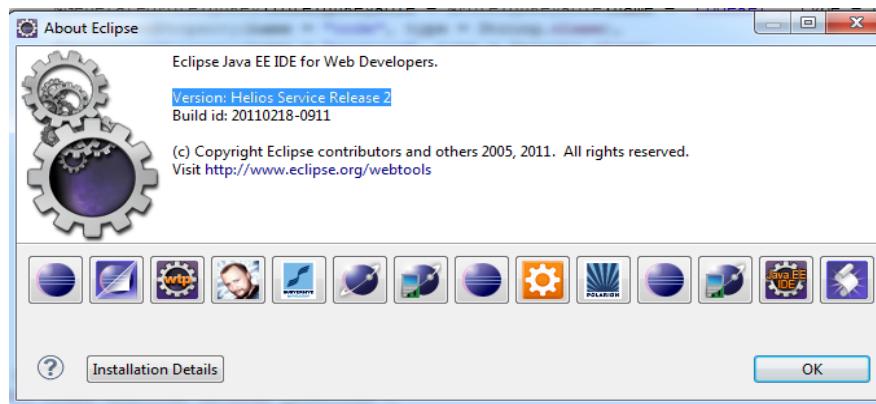
# Eclipse Configuration

# 1. Download and Install Eclipse

## Version List and Download Site

- **Eclipse (Version: Helios 3.6 R2)**

- [http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/helios/SR2/eclipse-jee-helios-SR2-win32-x86\\_64.zip](http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/helios/SR2/eclipse-jee-helios-SR2-win32-x86_64.zip) - Windows 64Bit
- <http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/helios/SR2/eclipse-jee-helios-SR2-win32.zip> - Windows 32Bit



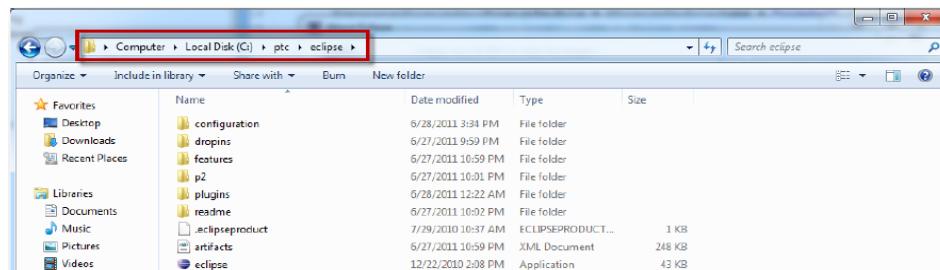
- We need to install add-on program for WC10.

- Aptana
- FileSync
- SVN Subversive

# 1. Download and Install Eclipse

## Install Eclipse

- Extract compressed file and copy to any directory
  - Since Eclipse doesn't need to execute installation program, you can copy it in any directory you are comfortable with.



- Create shortcut and set virtual memory for performance
  - Open INI file in same directory of eclipse.exe
  - Update memory size for Eclipse

```
-startup
plugins/org.eclipse.equinox.launcher_1.1.1.R36x_v20101122_1400.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.2.R36x_v20101222
--product
org.eclipse.epp.package.jee.product
--launcher.defaultAction
openFile
--launcher.XXMaxPermSize
256M
--showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
--vargs
-Dosgi.requiredJavaVersion=1.5
-Xms512m
-Xmx512m
```

## 2. Set Environment for Eclipse

Make sure the Eclipse environment for performance is ready and is correctly executed

### 1. Set Java path for Eclipse

- Add Java path \$WT\_INSTALL\_PATH/Java/bin to system path c:\ptc\Windchill\_10\Java\bin

### 2. Set memory size for Eclipse

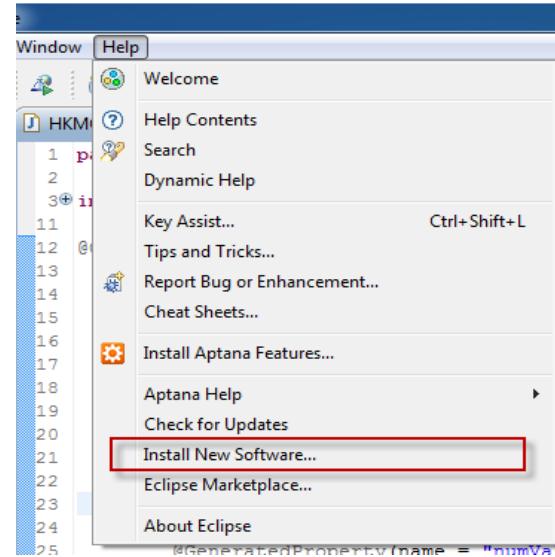
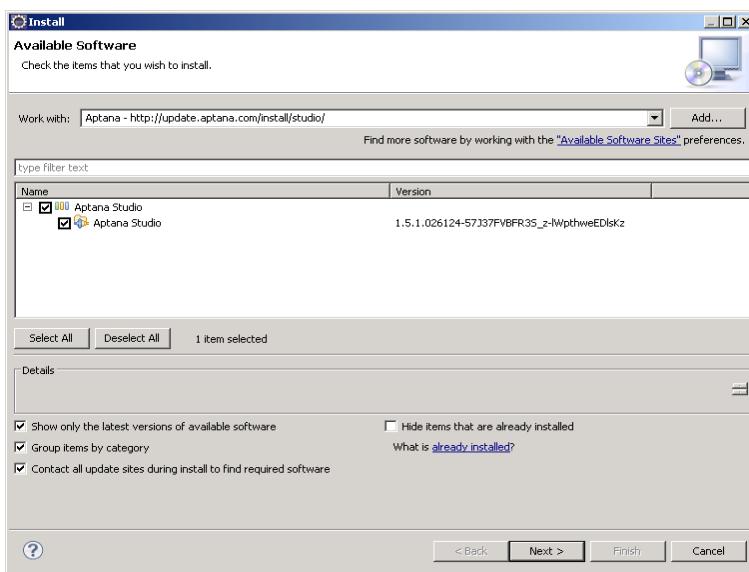
- Change directory to install Eclipse
- Open eclipse.ini file for editing mode.
- Change to correct memory size on INI file

```
1 -startup
2 plugins/org.eclipse.equinox.launcher_1.1.1.R36x_v20101122_1400.jar
3 --launcher.library
4 plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.2.R36x_v20101222
5 -product
6 org.eclipse.epp.package.jee.product
7 --launcher.defaultAction
8 openFile
9 --launcher.XXMaxPermSize
10 512M
11 -showsplash
12 org.eclipse.platform
13 --launcher.XXMaxPermSize
14 512m
15 --launcher.defaultAction
16 openFile
17 -vmargs
18 -Dosgi.requiredJavaVersion=1.5
19 -Xms40m
20 -Xmx512m
21
```

### 3. Install Aptana

#### Install Aptana Using Eclipse

- Aptana makes updating ExtJS simple
- Similar to how Eclipse simplifies Java coding
- Install using Eclipse
- **Help > Install New Software ...**
- Point to the Aptana URL to install this Eclipse Plug-in
- Editing ExtJS on a Web Page with Aptana



Although pages on the Web will indicate an older URL, the correct URL is:

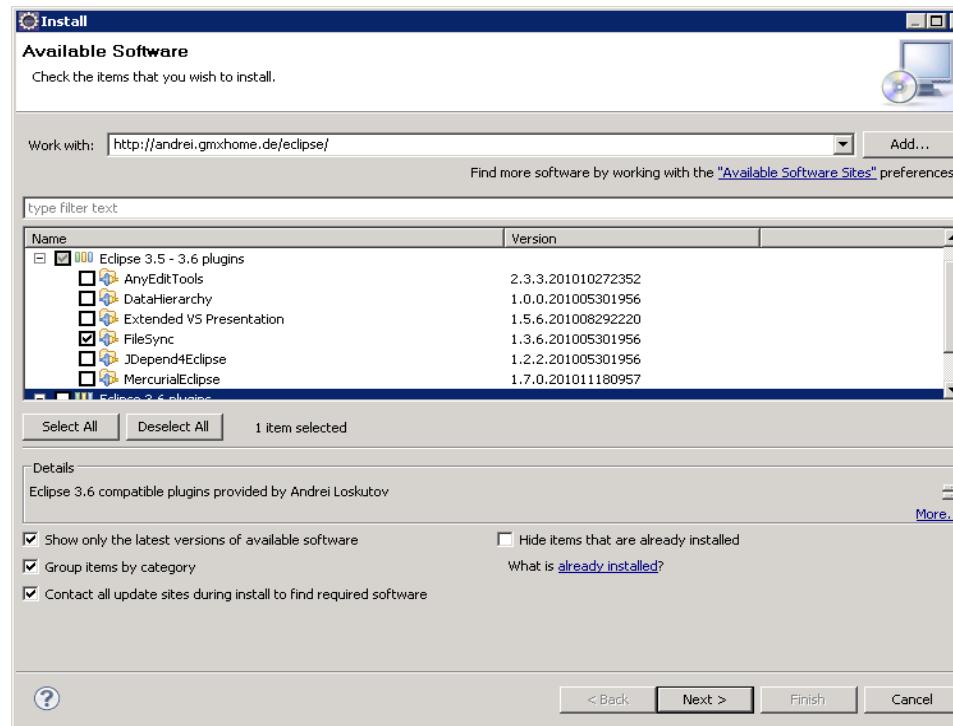
<http://download.aptana.com/tools/studio/plugin/install/studio>

The older version will display an error when Eclipse starts.

## 4. Install FileSync

### FileSync is Used to Synchronize Directories

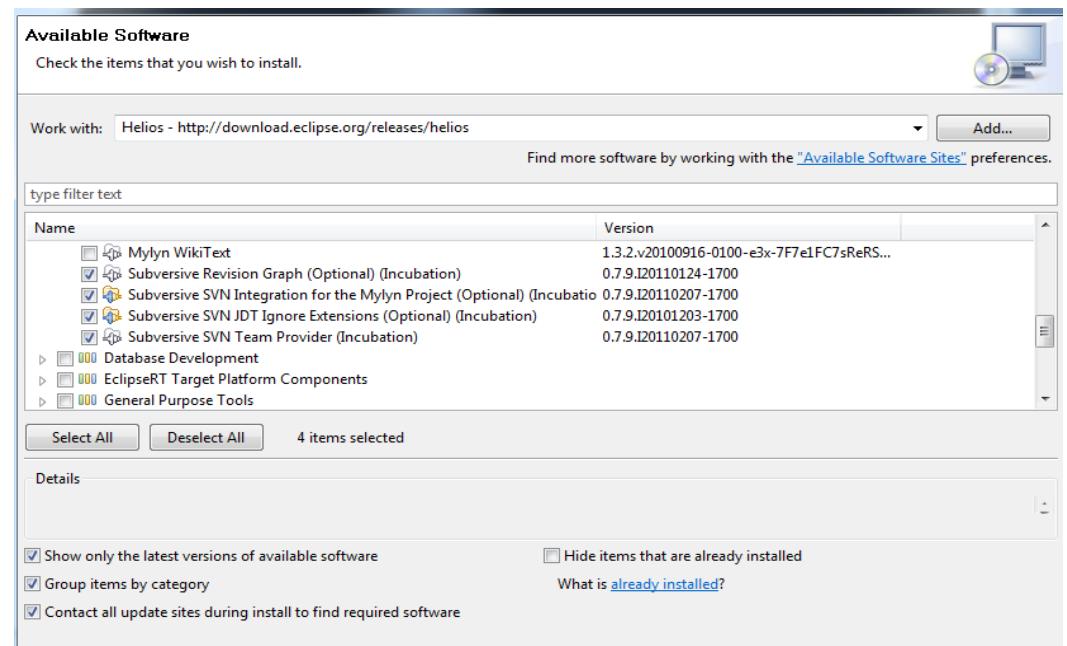
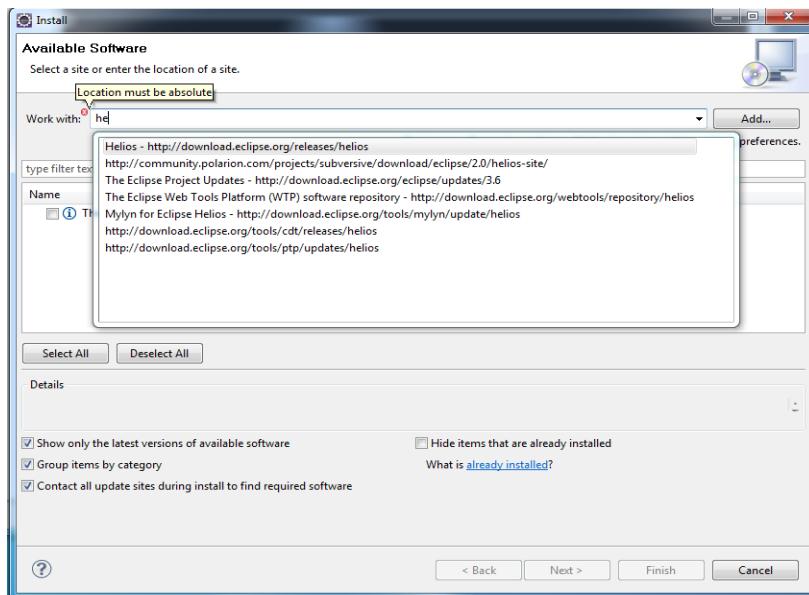
- In this case, we will use it to synchronize our build directory with the Windchill codebase
  - Help > Install New Software
  - Enter the following URL <http://andrei.gmxhome.de/eclipse/>



# 5. Install SVN Repository

## Subversive

- **Help > Install New Software**
- For Subversive Plugin, we don't need to enter an URL. You have to use Eclipse update site.
- On the work with line, you enter "Helios"
- Select Collaboration > Subversive software
- After you have finished installing and restarting, Eclipse will ask you to install connector for Subversive. This time, if you are using Windows operating system, just select Toolkit.



## Miscellaneous Configurations

- **Window > Preferences**
  - Java > Compiler > Building
    - Unselect *Output folder* > *Scrub output folders when cleaning projects*
  - Java > Code Style > Formatter > Import...
    - Select the following file: %wt\_viewroot%\Windchill\src\devtools\eclipse\eclipseFormatterProfile.xml
  - General > Editors > Text Editors
    - Check "*Insert spaces for tabs*"
    - Optionally check *show print margin* and set to 120
    - Optionally check *show line numbers*
  - General > Editors > File Association
    - Add “\*.jsfrag” and add associated editors *JavaScript Editor*

## Configure Aptana as Shown

### If you have the Aptana plugin

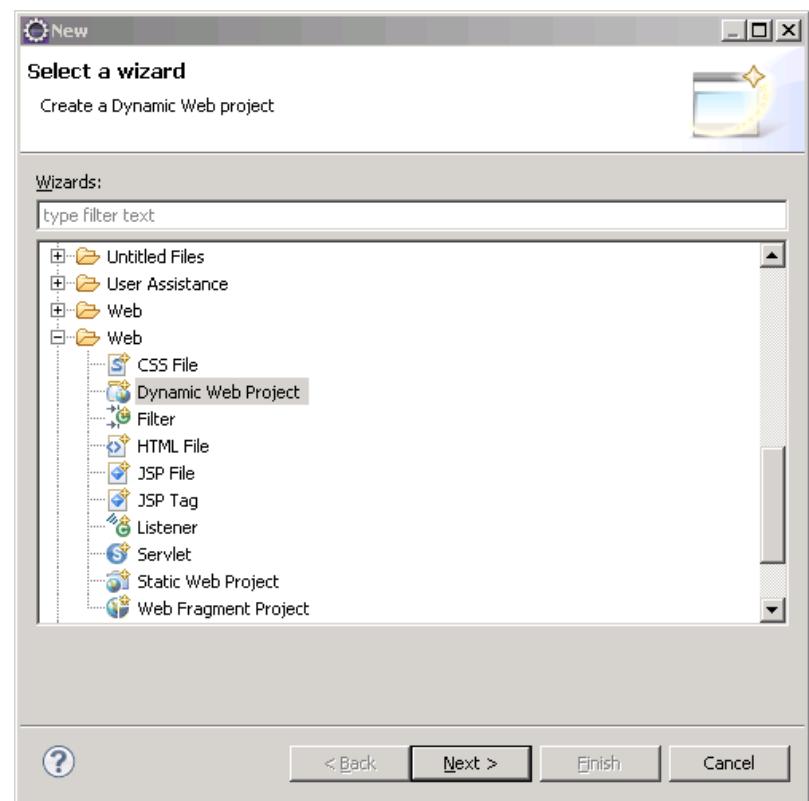
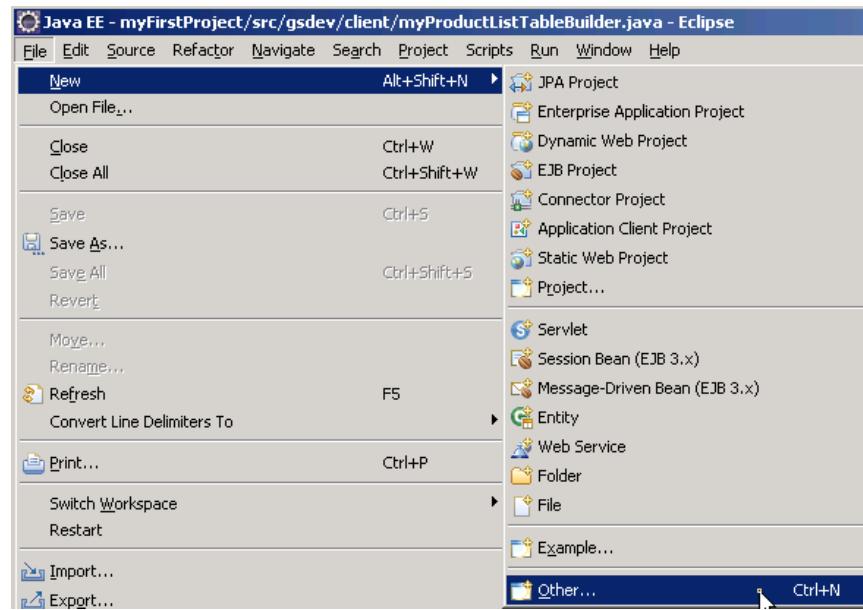
- Aptana > Editors > JavaScript > Formatting > Import...
  - Select the following (find in OpenGrok) file:  
%wt\_viewroot%\Windchill\src\devtools\eclipse\eclipseAptanaFormatterProfile.xml
- General > Editors > File Association
  - Update “\*.jsfrag” and “\*.js” to use associated editor Aptana JS Editor by default
- Aptana > Editors > General
  - Under Tab Insertion, select Use Spaces
- XML > XML Files > Editor
  - Select Indent using spaces and set Indentation size to 4

## 7. Create Project — (1/2)

### Create a Dynamic Web Project

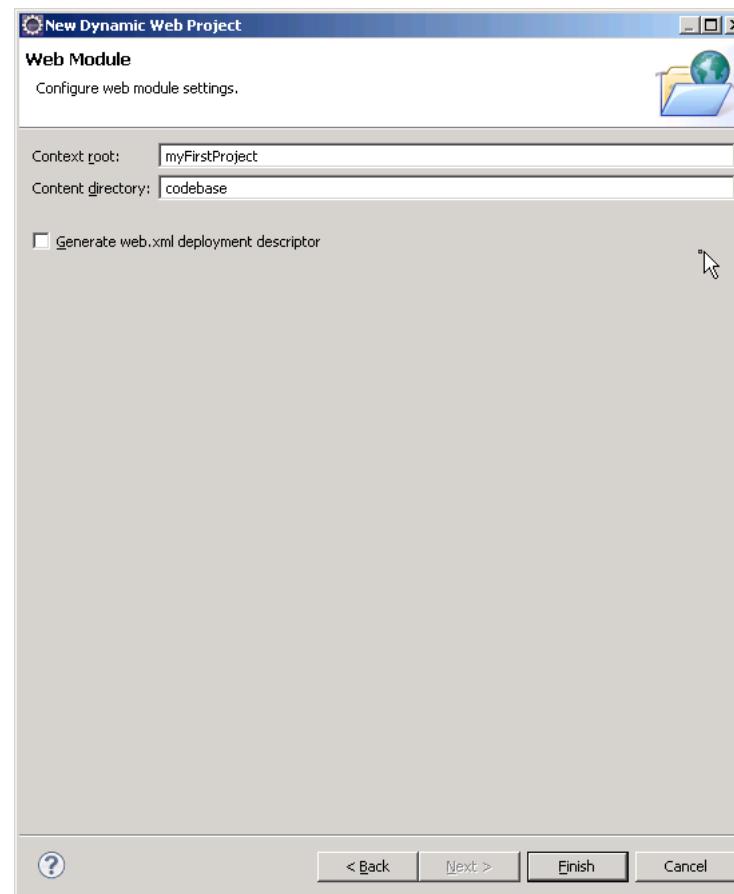
After Windchill 10, we use Web project because it is more comfortable for tag program and JSP program than using Java project.

- File > New > Other displays a list of Project and File Types
  - Varies depending on options installed
  - Dynamic Web project is useful for JSTL configurations



### Create a Dynamic Web Project

Every configuration will use default except on Web module.  
The configuration must create codebase content.



# 8. Set Environment of Project for Windchill

## Classpath Makes API Available

Classpath is configured using the Java Build Path in Eclipse

The Classpath consists of:

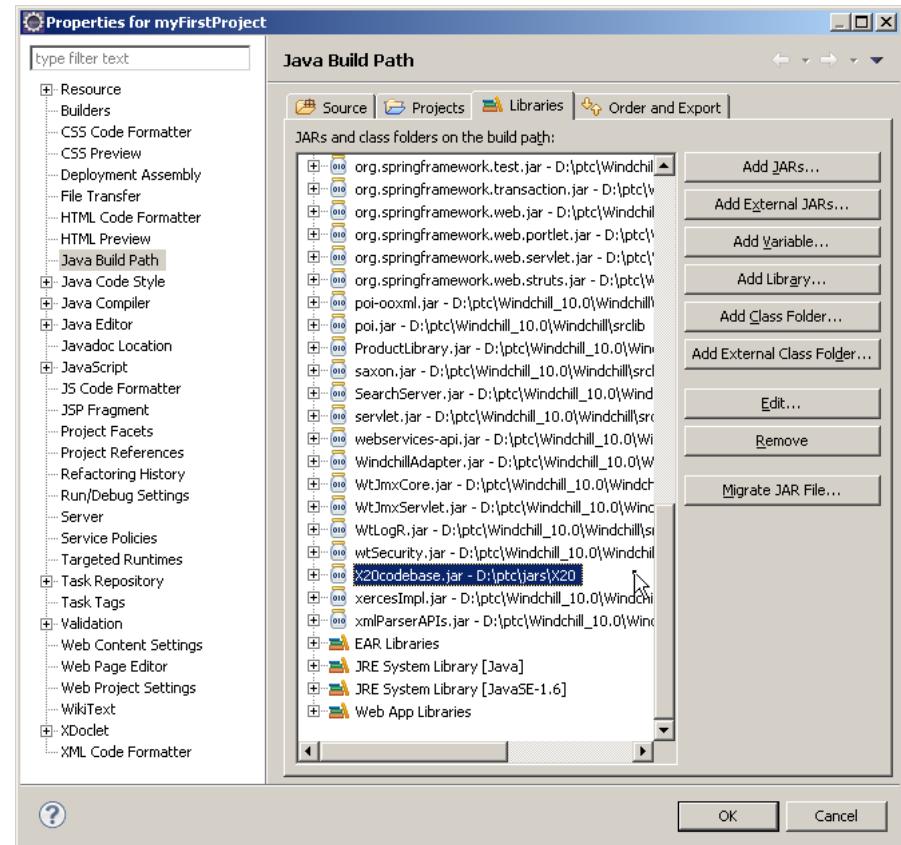
- Jars in the %WT\_HOME%\srclib directory

The srclib has many directories related to Windchill, so you can add more Jars for special implementation.

And a library named “srclib\tool\Annotations.jar” should be added

For example:

- Info\*Engine implementation must include srclib/ie libraries.
- Work Group Manager implementation needs to include srclib/prowt libraries.
- Class files in the codebase.



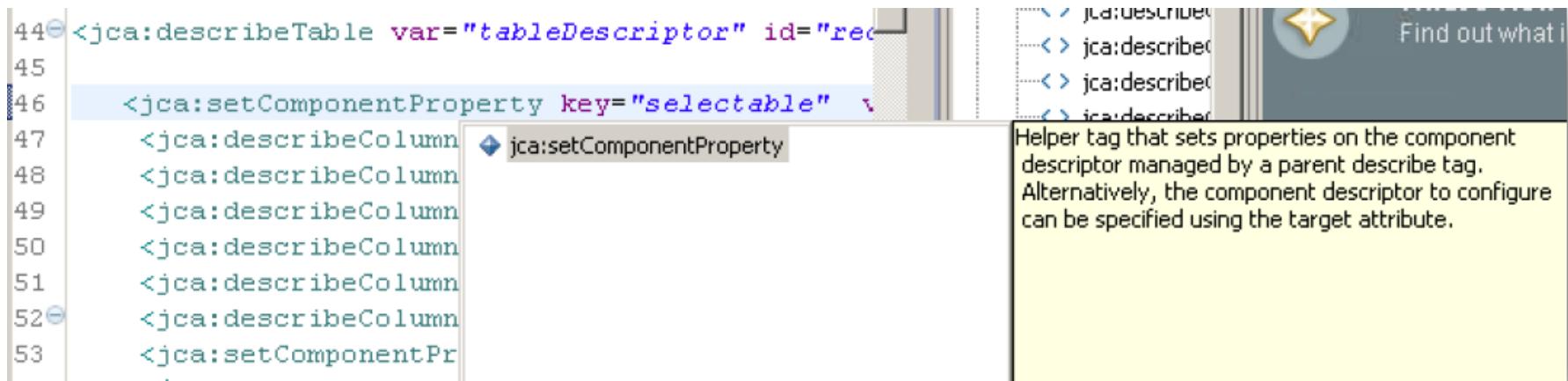
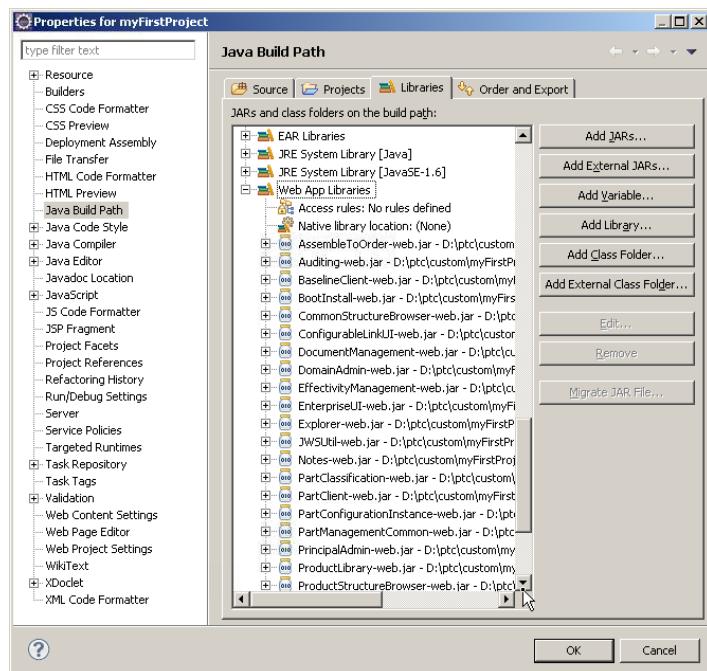
# 8. Set Environment of Project for Windchill

PTC®

## Code-Assist in JSP

To enable code-assist in JSP, copy the Web app libraries into the development root folders.

- From `%WT_HOME%\codebase\WEB-INF\lib\*web.jar`
- To `%DEV_ROOT%\codebase\WEB-INF\lib\*web.jar`



Code-Assist in JSP

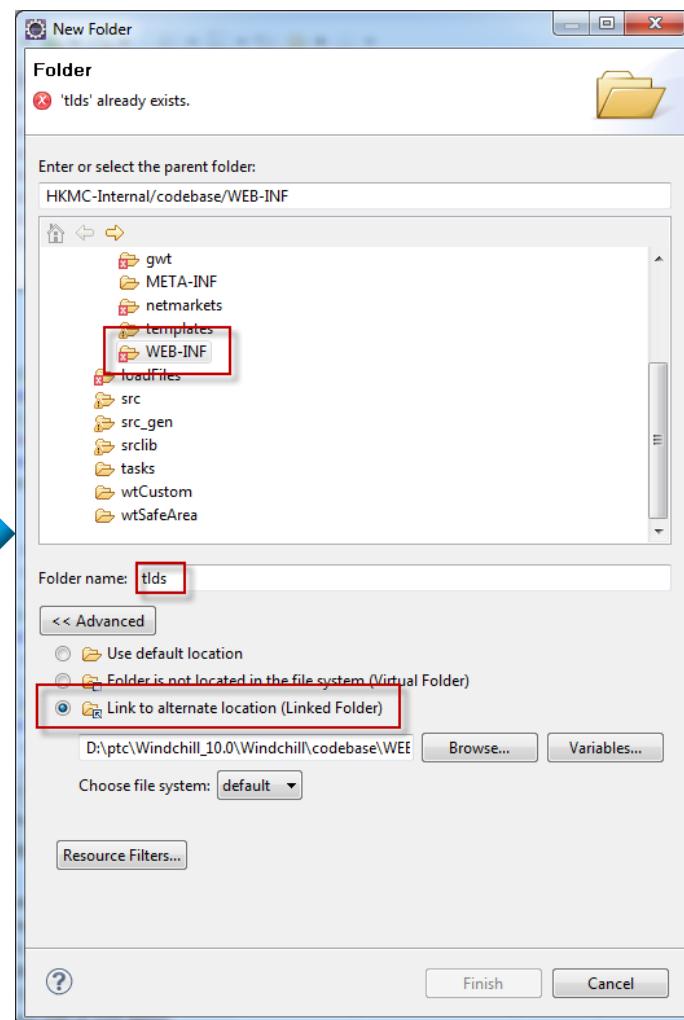
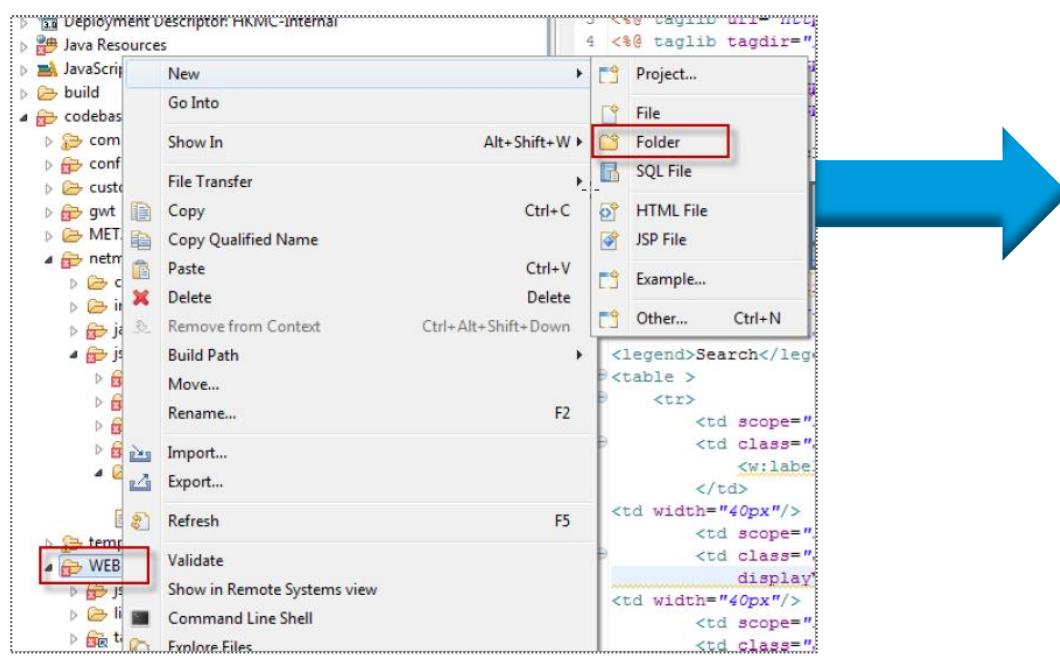
# 8. Set Environment of Project for Windchill

PTC®

## TLDs and TAGs under the WEB-INF Folder Are Required by Eclipse

### Configure a Second Web Content folder:

- Create a new folder under the Web Content (codebase) folder
  - tlds
  - tags
- Link it to the WEB-INF folder



# 8. Set Environment of Project for Windchill

## Annotation Processing

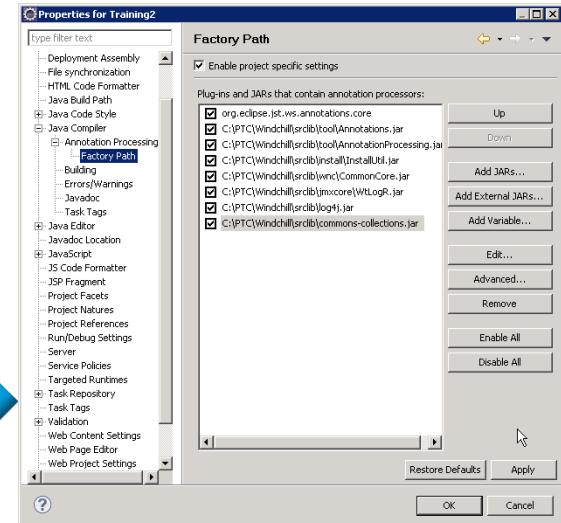
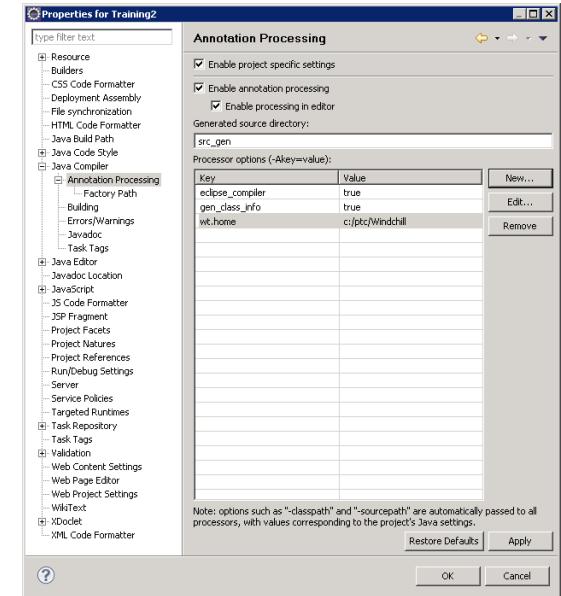
For annotation compiling, you must set the following environment.

- Set annotation configuration

- Go to Project properties > Java Compiler > Annotation Processing
- Set “Enable project specific setting”
- Set “Enable annotation processing”
- Set “Enable processing In editor”
- Change Generated source directory – “src\_gen”
- Set three keys
  - eclipse\_compiler = true
  - gen\_class\_info = true
  - wt.home = \$WT\_HOME

- Set Factory path

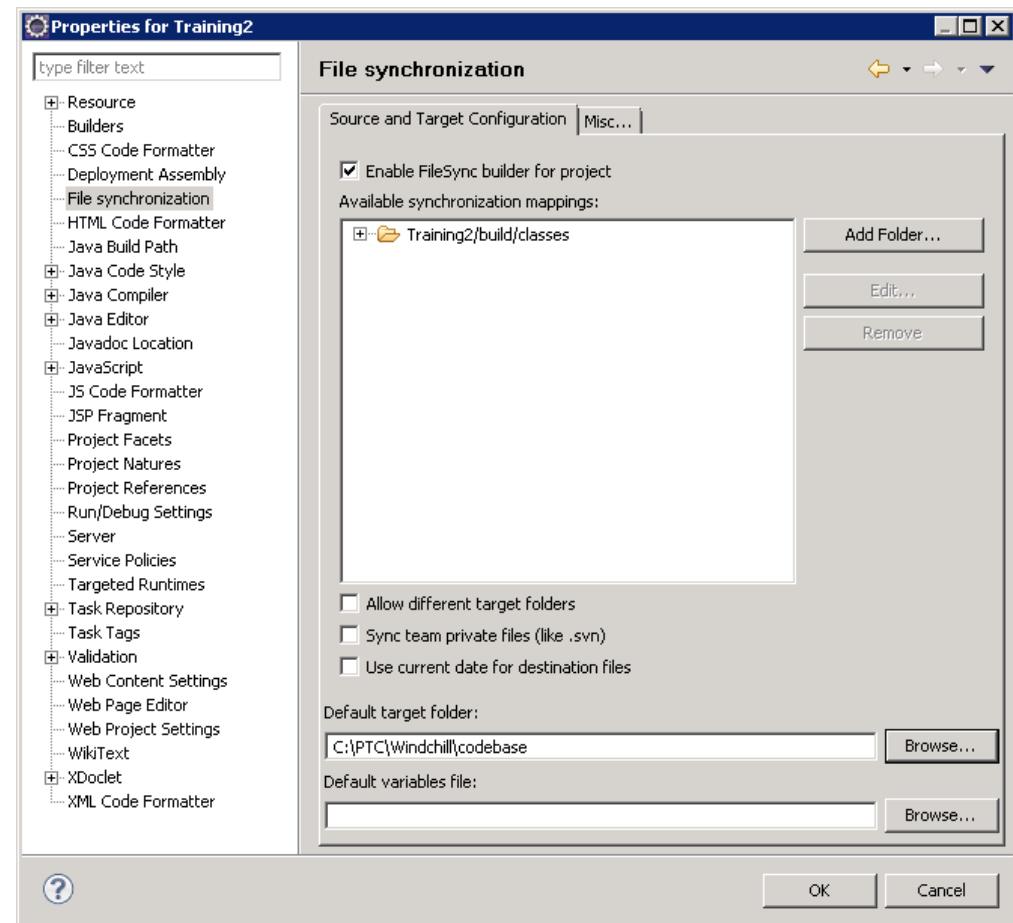
- Goto Project properties > Java Compiler > Annotation Processing > Factory Path
- Add JAR file as shown in the so



## FileSync Preference

For annotation, compile before using the ant-based annotation class that was deployed on Windchill class path.

- This configuration will be effected when you create new modeling within the annotation, and the system will automatically synchronize to Windchill.
- It means, Eclipse will also update registry files (should be kept in mind)
- If you do not want to automatically update, please do not set this option.



# 10. Create Project Using Command Line

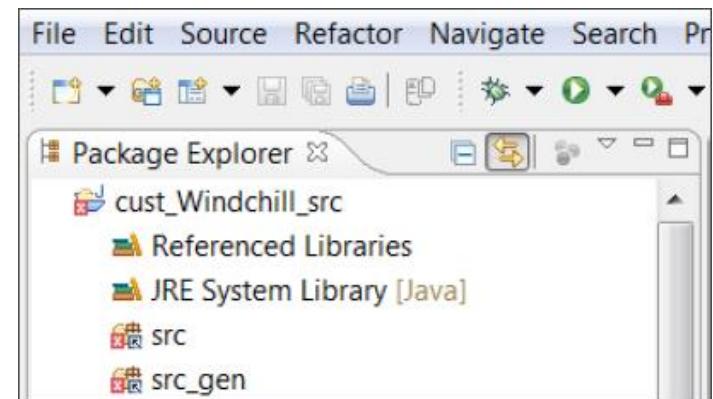
## Create Eclipse Project Using Command

Windchill support one command for making eclipse project automatically.

1. Open Windchill shell
2. Execute “**ant -f bin/tools.xml eclipse\_project**”
3. The workspace project will be created on “\$Windchill\_LoadPoint/eclipse/cust\_Windchill\_src”  
Ex.) c:\ptc\Windchill\_10.0\eclipse\cust\_Windchill\_src
4. Open eclipse
5. Import created project “**File > Import... > General > Existing Projects into Workspace**”
  - A. Select created root directory

If you want to see help message on how to use the command:

1. Open Windchill shell
2. Execute “**ant -f bin/tools.xml eclipse\_project.help**”



# Useful Eclipse Configuration

## Windchill Configuration

For UI customization, sometimes we need to debug the server-side execution. Basically, Eclipse can capture and debug a server event. For this one, we need some configuration for server-side debugging.

To debug Method Server, add the following to site.xconf and propagate:

```
<Property name="wt.manager.cmd.MethodServer.java.extra.args" overridable="true"  
targetFile="codebase/wt.properties" value="-Xnoagent -Xdebug -  
Xrunjdwp:transport=dt_socket,address=9999,server=y,suspend=n"/>
```

Or you can use command line in Windchill shell.

```
xconfmanager -s wt.manager.cmd.MethodServer.java.extra.args="-Xnoagent -Xdebug -  
Xrunjdwp:transport=dt_socket,address=9999,server=y,suspend=n" -p
```

Then run remote debugging in eclipse/netbeans (port 9999)

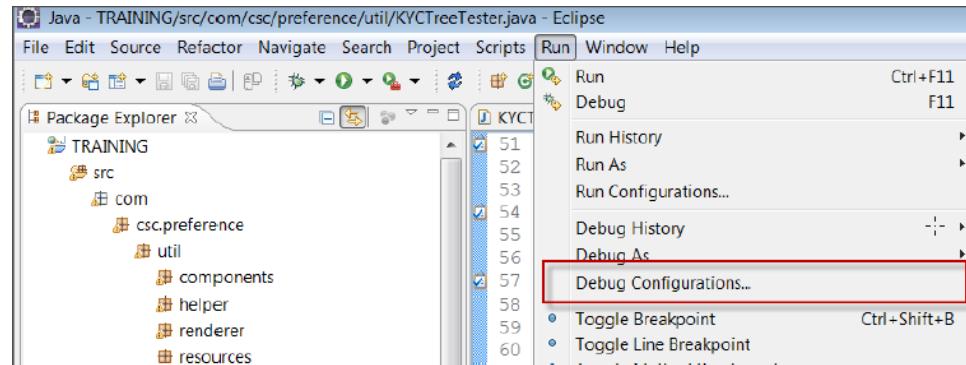
That is: in eclipse open “Debug Configurations” and create “Remote Java application” with socket attached pointing to provided port (9999) and Windchill host

**Windchill Services must be restarted for the changed configuration.**

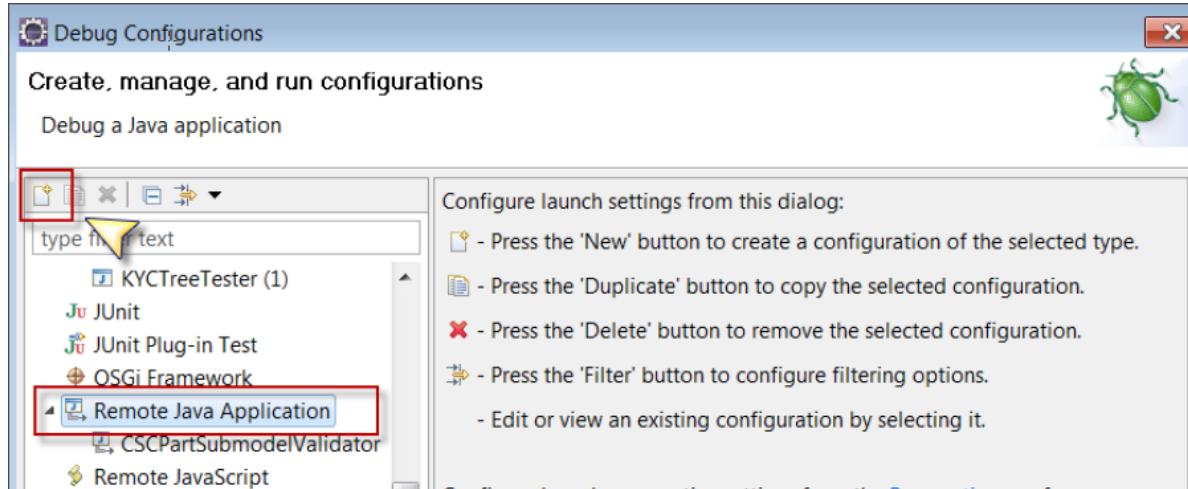
# 1. Remote Debug

## Eclipse configuration

### 1. Open Eclipse debug configuration



### 2. Create one remote Java application item

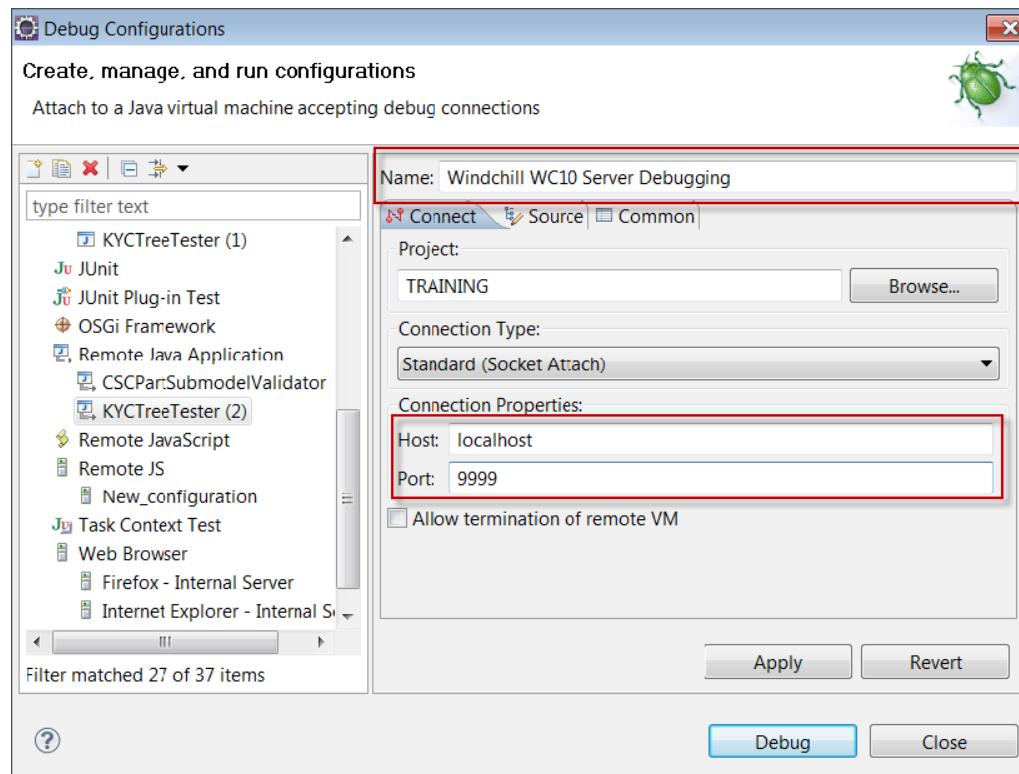


# 1. Remote Debug

## Eclipse Configuration

### 3. Configure for remote Windchill Method Server (port number: 9999)

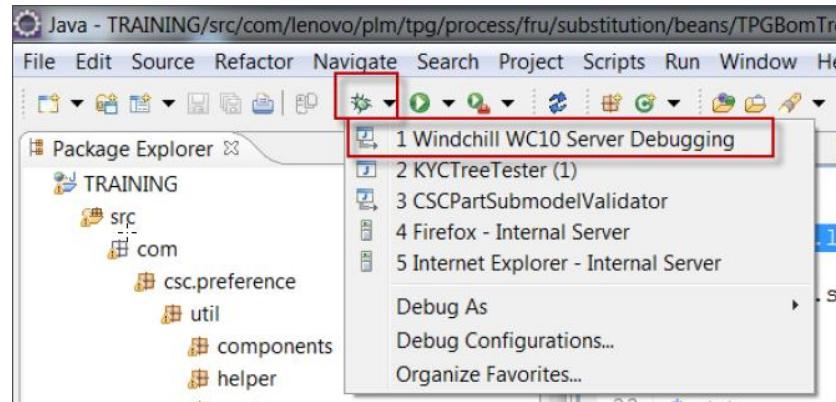
- Set the debugging managed name
- Set host and port about remote Method Server



# 1. Remote Debug

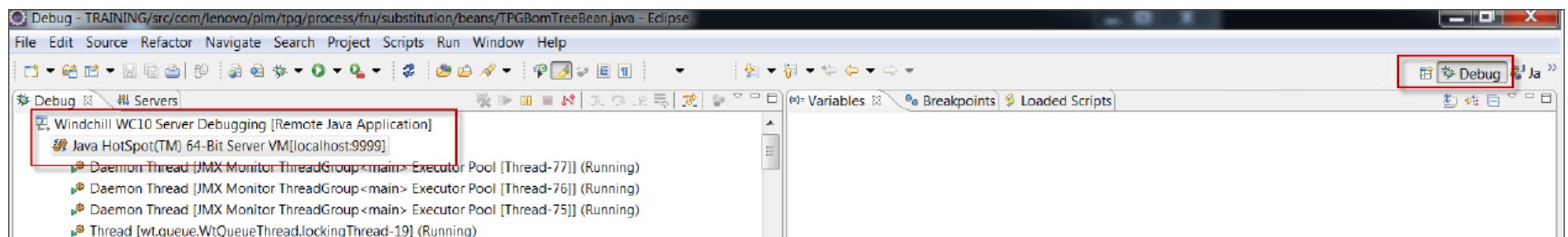
## Execution

You can see the configuration item on the debug execution menu.



Finally, you can see the execution status on debug area.

This manual will not explain how to debug.



## 2. Reload Server-Side Class (No Restarting of Method Server)

### Windchill Configuration

When we implement server-side class, if the class is changed we must restart Windchill server for testing. However, it takes much time because restarting server needs a longer time. So, if you are already registered on the server-side for the first time, you can use the reload tool next time.

You don't need to restart the server. You just update the changed class on the server and refresh the acting server.

For this tool, you also need to port configuration, but if you are already configured, this step is not needed. This tool also uses the same port and configuration with the debug tool. If you didn't set any for the debug, set the following configuration:

To debug Method Server, add following to site.xconf and propagate:

```
<Property name="wt.manager.cmd.MethodServer.java.extra.args" overridable="true" targetFile="codebase/wt.properties" value="-Xnoagent -Xdebug -Xrunjdwp:transport=dt_socket,address=9999,server=y,suspend=n"/>
```

Or you can use command line in Windchill shell.

```
xconfmanager -s wt.manager.cmd.MethodServer.java.extra.args="-Xnoagent -Xdebug -Xrunjdwp:transport=dt_socket,address=9999,server=y,suspend=n" -p
```

**Windchill Services must be restarted for the changed configuration.**

## 2. Reload Server-Side Class (No Restarting of Method Server)

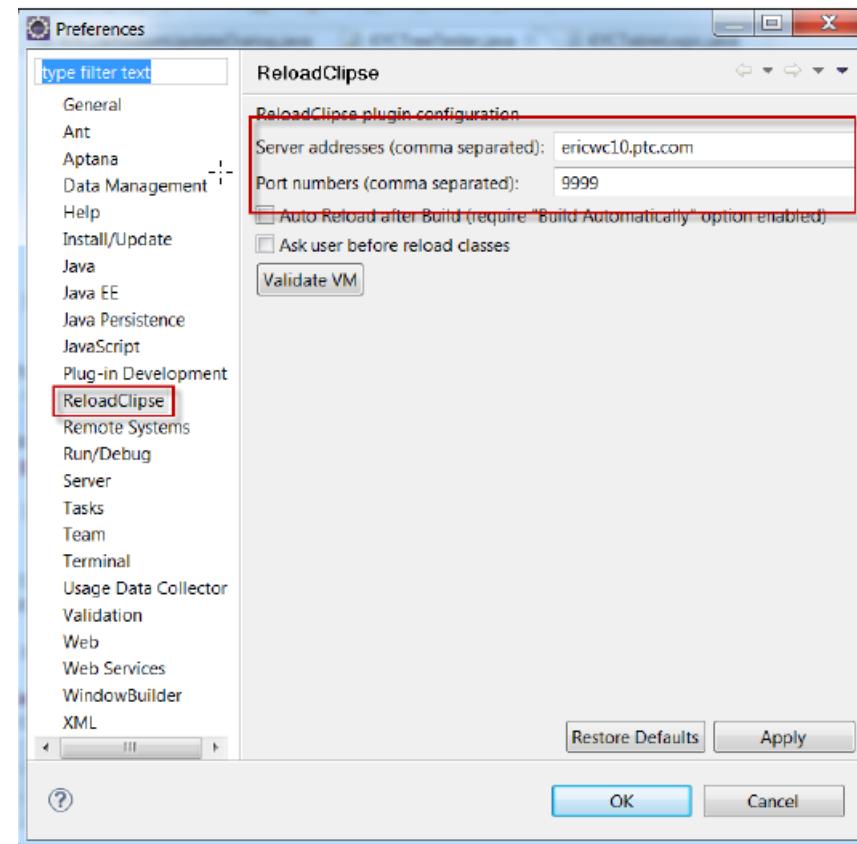
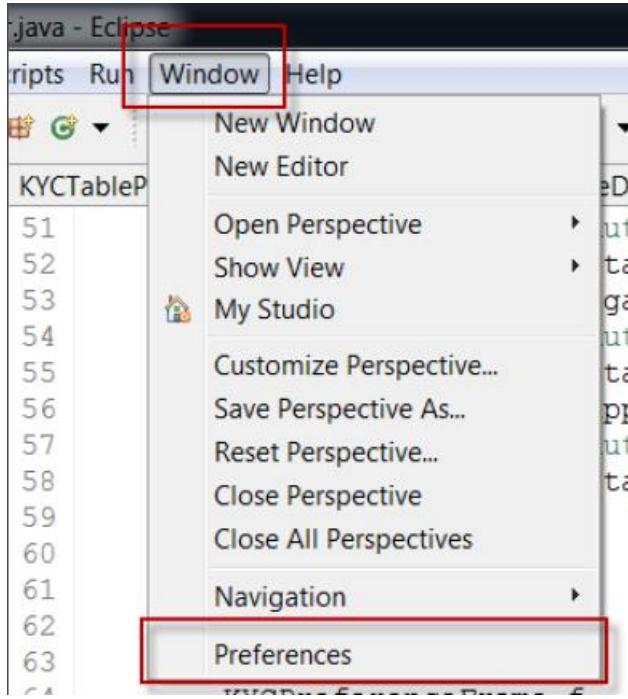
PTC®

### Eclipse configuration

#### 1. Download ReloadClipse plugin from PTC Service Wiki

[https://ssp.ptc.com/wiki/download/attachments/41488931/ext.psc.reloadclipse\\_1.1.0.zip?version=1&modificationDate=1285063901000](https://ssp.ptc.com/wiki/download/attachments/41488931/ext.psc.reloadclipse_1.1.0.zip?version=1&modificationDate=1285063901000)

#### 2. Configure ReloadClipse



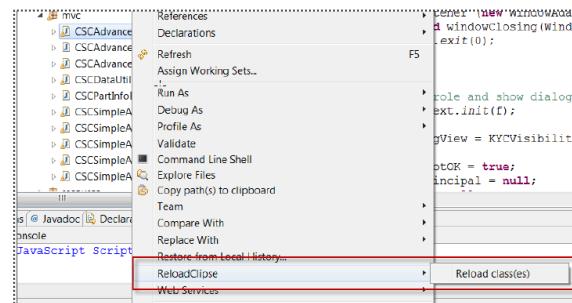
## 2. Reload Server-Side Class (No Restarting of Method Server)

PTC®

### Execution

When you finish server-side class for implementing:

- Click target class on class navigation
- Right-click and open Action menu
- Select ReloadClipse > Reload Class menu



If the action was complete without an error message, it is OK. (The server doesn't need to be restarted)

### Success

```
ReloadClipse
Sat Mar 31 14:45:38 CST 2012 - ReloadClipse INFO: com.lenovo.plm.tpg.process.fru.substitution.beans.TPGBomTreeBean successfully reloaded;
```

### Error

```
ReloadClipse
Sat Mar 31 14:46:10 CST 2012 - ReloadClipse WARNING: ext.csc.training.mvc.CSCAdvancedAttributesPanelBuilderl - Cannot reload class, not loaded yet;
```

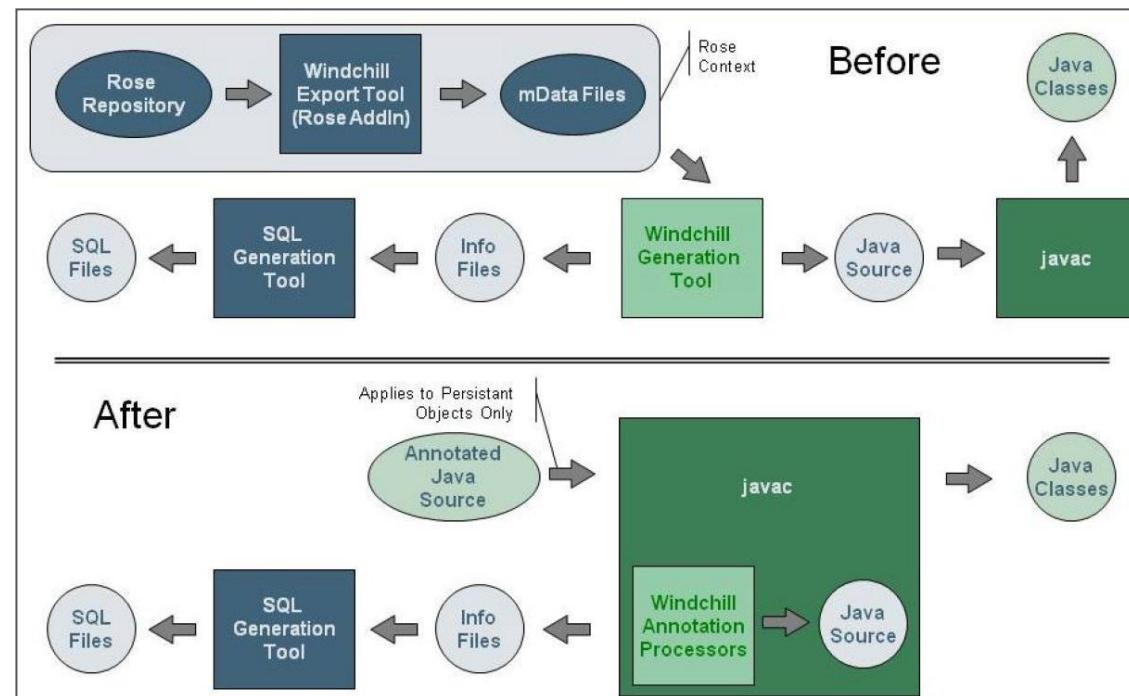
# Modeling with Annotation

# 1. Model Generation Process

## WC10's Generation

WC10 started to use annotation, so the modeling and generating process was changed. The following is the changed view.

- Rational Rose is not used anymore (no visual tool)
- All modeled objects must be generated from the annotation file (previous classes didn't have new annotation file except some important objects)



## Generation Command

When you want to generate annotation file, you will have to use Windchill command. The command is supported by the tools.xml

- Help!
  - ant -f bin/tools.xml help – or – ant -f bin/tools.xml <target>.help
- Compile
  - ant -f bin/tools.xml class -Dclass.includes=com.ptc/training/\*\*
- SQL scripts
  - ant -f bin/tools.xml sql\_script -Dgen.input=com.ptc.training.\*
- Bundles
  - ant -f bin/tools.xml bundle -Dbundle.input=com.ptc.training.\*

All the annotation files must exist within the “\$WT\_HOME/src” directory.

### Important

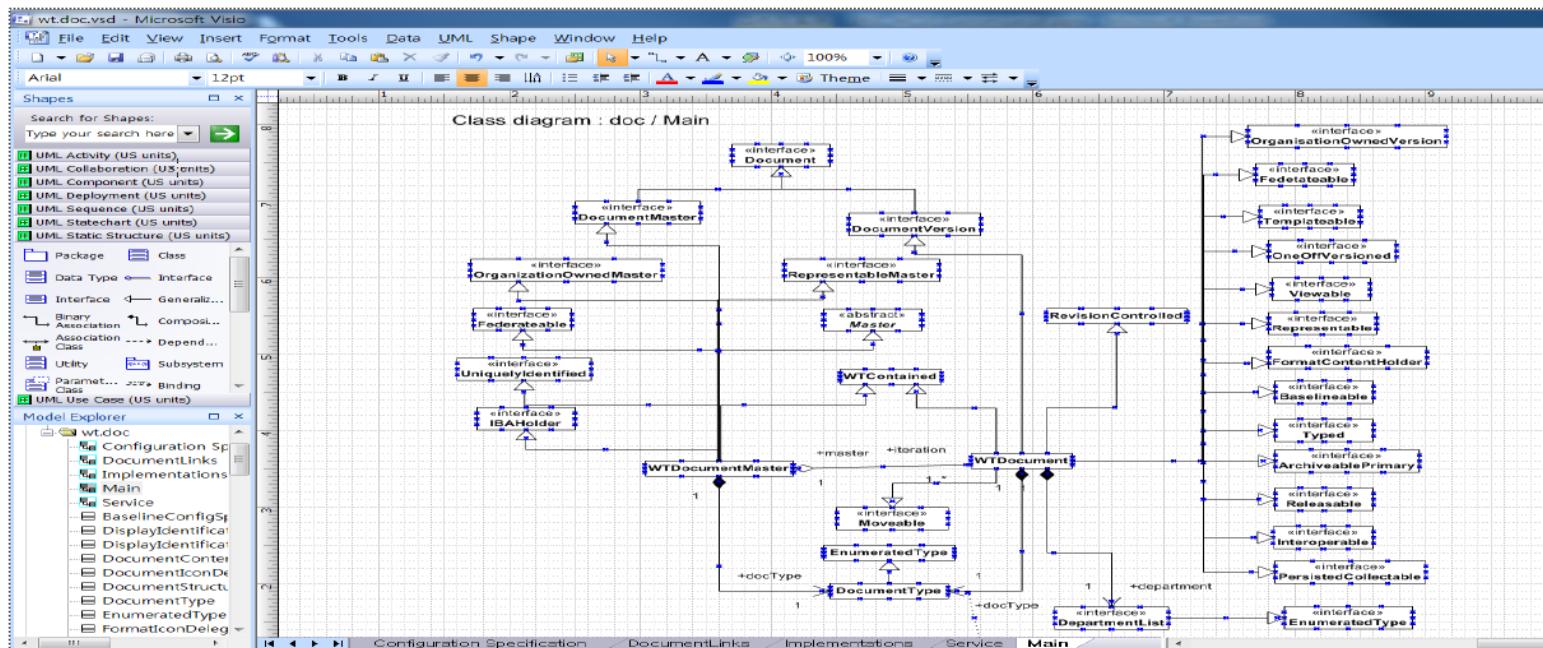
When you have completely compiled Windchill annotation on Eclipse, you don't need to copy source code at “src” directory for generating SQL script. (FileSync should be installed)

The bundle commands is just using for “\*.rbInfo” text file. If you are using class file, the command doesn't need to be executed. It is enough to annotation compile. (class compile)

## 2. Modeling

### Visual Tool (Just Imagination)

Because Rational Rose is not used anymore, PTC started using *Microsoft Visio* for visual tool before designing annotation.



You can get Windchill basic model from OpenGrok.

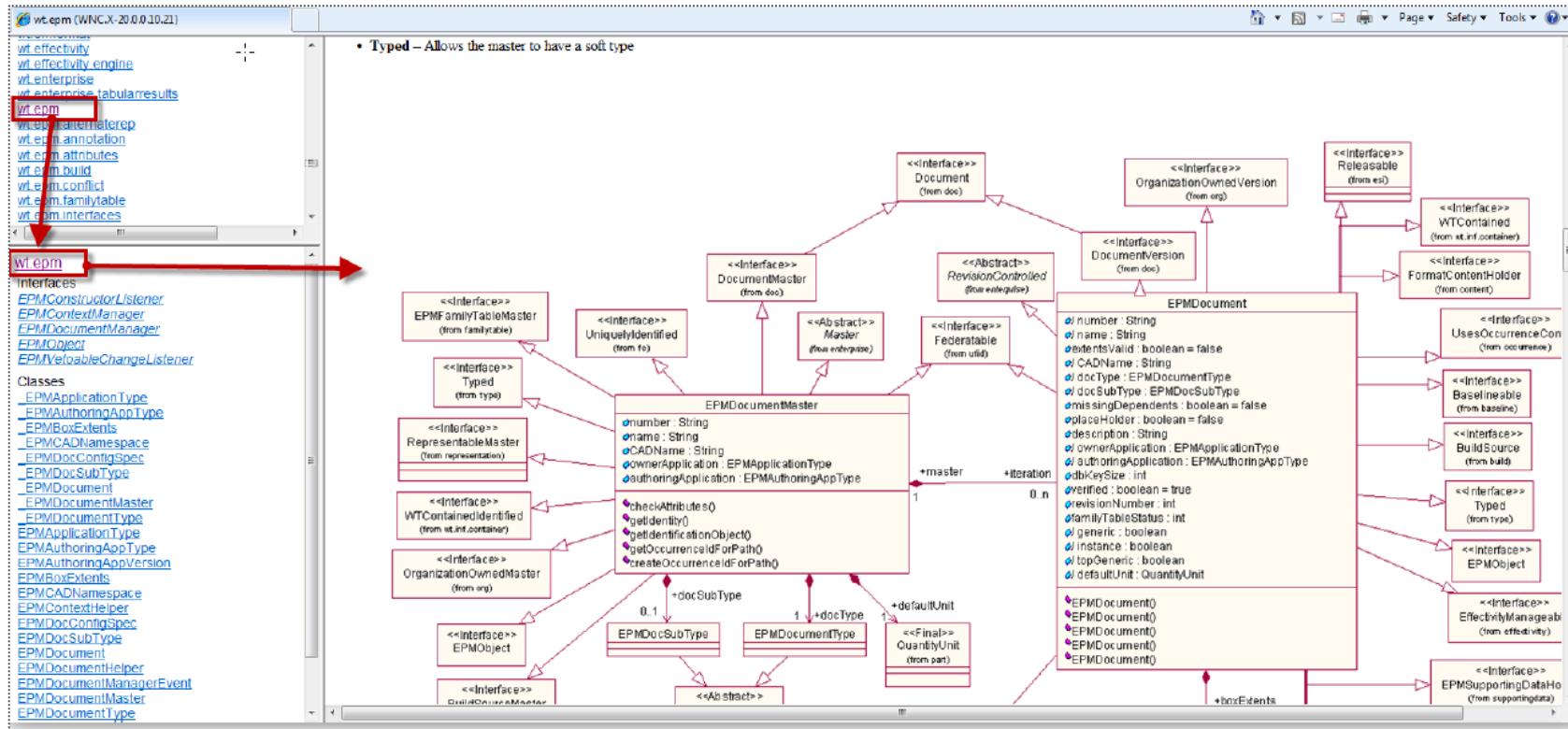
- <http://ah-grok.ptcnet.ptc.com/search?q=&project=x-20-M020&defs=&refs=&path=vsd&hist=>

## 2. Modeling

### Using Windchill Java Doc ([\\$WT\\_HOME/codebase/wt/clients/library/api](#))

Java Doc has some visual picture of modeling for each package. (It is not supported for all the packages.)

- For example, if you want to know EPMDocument modeling, have a look at “[wt.epm](#)” package.



### Get Annotation Sample from Windchill Javadoc

You can easily get annotation sample from Windchill Javadoc.

The screenshot shows a Java class hierarchy and its annotations. The hierarchy is as follows:

```
java.lang.Object
  └─ wt.fc._WTObject
    └─ wt.fc.WTObject
      └─ wt.vc.views. View
        └─ wt.vc.views. View
```

**All Implemented Interfaces:**

[Externalizable](#), [Serializable](#), [wt.fc.\\_NetFactor](#), [wt.fc.\\_ObjectMappable](#), [wt.fc.\\_Persistable](#), [NetFactor](#), [ObjectMappable](#), [Persistable](#), [DisplayIdentification](#)

```
@GenAsPersistable(superClass=WTObject.class,
    versions=-433302533224513071L,
    properties={
        @GeneratedProperty(name="name", type=java.lang.String.class, supportedAPI=PUBLIC,
            javaDoc="The name of the View. Must be unique.", constraints=@PropertyConstraints(upperLimit=30,
                required=true), columnProperties=@ColumnProperties(unique=true)),
        @GeneratedProperty(name="sortId", type=int.class, accessors=@PropertyAccessors(setAccess=PACKAGE)),
        tableProperties=@TableProperties(tableName="WTView"))
public final class View
extends View
```

### General Sample

While making a new object, designers mostly use the following general designing:

- [wt.part.WTPart](#)
  - derivedProperties and foreignKeys
- [wt\\_vc\\_baseline.BaselineMember](#)
  - Example association
- [wt.part.PartType](#)
  - Example enumerated type
- [wt.part.partResource](#)
  - Example resource bundle
- [wt.fc.PersistenceManager](#) and [wt.fc.StandardPersistenceManager](#)
  - Example service
- [com.ptc.windchill.annotations.metadata](#)
  - Explanation of annotations commands

### Meet Annotation

#### Primary Annotations (you'll start with these)

- **GenAsUnPersistable** non-persistent interfaces implemented by persistent classes and Evolvable classes (usage should be rare)
- **GenAsObjectMappable** persistent objects stored as columns (aka “cookies”)
- **GenAsPersistable** persistent objects stored as tables (aka “persistables”)
- **GenAsBinaryLink** specialize persistables associating a role A/B persistables
- **GenAsEnumeratedType** extensions of wt.fc.EnumeratedType
- **GenAsPrimitiveType** classes that can be reduced to a simple Java “primitive” type
- **GenAsDatastoreSequence** database sequences
- **GenAsDatastoreStruct** database structures
- **GenAsDatastoreArray** arrays of database column types, including structs
- **GenAsDatastoreTable** non-persistable persistables

### Meet Annotation

#### Secondary annotations (properties of primary annotations)

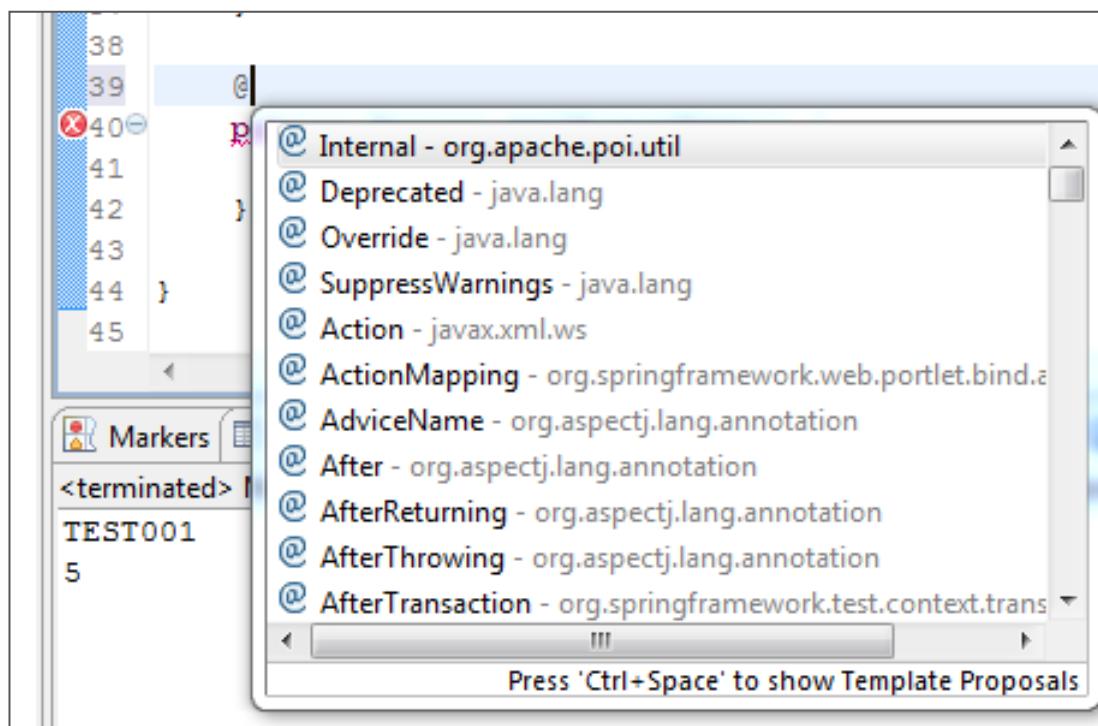
- **GeneratedProperty** field consisting of Java “primitives” and cookies
- **GeneratedForeignKey** reference to persistable stored locally in this persistable
- **GeneratedRole** role A/B of a binary link
- **DerivedProperty** convenience accessors to above
- **TableProperties** table name, composite indexes, and composite unique indexes
- **IconProperties** standard/open icon (this may be obsolete)

#### Tertiary annotations (properties of secondary annotations)

- **PropertyAccessors** getter/setter access/exceptions
- **PropertyConstraints** string case, upper/lower limits, required, changeability
- **ColumnProperties** column name/type, indexing, uniqueness, persistence
- **ForeignKeyRole** target persistable’s role (master in master/iteration)
- **MyRole** my role as the target sees me (iteration in master/iteration)

### Look Annotation List on the Eclipse

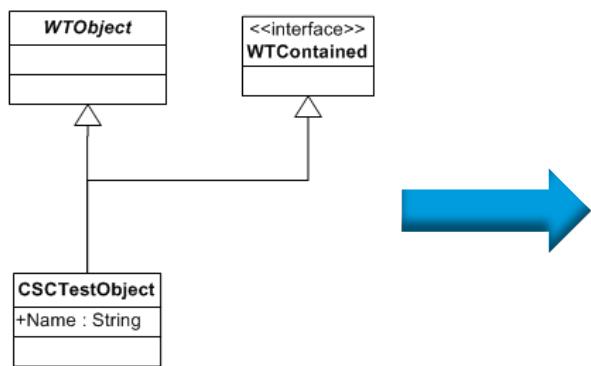
On the Eclipse, you can also see the annotation list by using CTRL+Space.



## 2. Modeling

### Sample of New Object

- WC10 doesn't use "preserve" marks
- Cannot change generated source code



```
package ext.csc.training.modeling;  
import org.apache.poi.util.Internal;  
...  
import com.ptc.windchill.annotations.metadata.GenAsPersistable;  
import com.ptc.windchill.annotations.metadata.GeneratedProperty;  
import com.ptc.windchill.annotations.metadata.PropertyConstraints;  
@GenAsPersistable(superClass=WTOObject.class, interfaces={WTContained.class},  
properties={  
    @GeneratedProperty(name="name", type=String.class, constraints=@PropertyConstraints(required=true))  
})  
)  
public class CSCCTestObject extends _CSCCTestObject {  
    static final long serialVersionUID = 1;  
    public static CSCCTestObject newCSCCTestObject() throws WTEException {  
        final CSCCTestObject instance = new CSCCTestObject();  
        instance.initialize();  
        return instance;  
    }  
    @Override  
    public void checkAttributes() throws InvalidAttributeException {  
        try {  
            nameValidate(name);  
        } catch (WTPPropertyVetoException wtpve) {  
            throw new InvalidAttributeException(wtpve);  
        }  
    }  
}
```

# Action and Property Report

## Property Report

Property report provides the detail of validate attribute lists for customization

<http://localhost/Windchill/app/#ptc1/netmarkets/jsp/property/propertyReport.jsp>

The screenshot shows the 'Property Report' page in the PTC Windchill interface. The top navigation bar includes 'Administrator', 'Part, Document, CAD D...', 'Search ...', and 'Quick Links'. The main content area is titled 'Property Report' with a 'Type' dropdown set to 'wt.part.WTPart' and a 'Flush Cache' button. Below this is a table titled 'Report Results' with a dropdown menu showing 'All Properties'. The table has columns: Name, Label, Conflicts, Data Utility, Table View, Validator, and Logical Attrib. A 'Sort by Data Utility' button is located in the header of the Data Utility column. The table lists numerous properties, many of which have checkmarks in the Data Utility column. The bottom of the table has navigation buttons for first, previous, next, last, and search.

Name	Label	Conflicts	Data Utility	Table View	Validator	Logical Attrib
adDescription	Comments	✓				
acceptedOf	Deliveries Accepted	✓				
accessDetailsIconAct...	View Access Information	✓				
accessRuleParticipan...		✓				
accessRulePermission...		✓				
accessRuleSource		✓				
accessRuleSourceIcon	Type	✓				
actInstruction		✓				
actionitemDatautilit...		✓				
actionitemInfoDataut...		✓				
actionItemListDataUt...		✓				
actionItemPriority	Priority	✓				

## Action Report

The Action Report is a search page that returns the details of actions matching the search criteria.

The Report can be accessed at:

<http://localhost/Windchill/app/#ptc1/netmarkets/jsp/carambola/tools/actionReport/action.jsp>

The screenshot shows the 'Action Report' page within a Windchill application. The top navigation bar includes 'Part, Document, CAD D...', 'Search ...', and 'Quick Links'. The left sidebar has 'Navigator' and 'Browse' tabs, with 'Search' selected. The main area has a title 'Action Report' and a 'Search By:' section with various filters like Label, Action Name, Object Type, etc. Below this is a table titled 'Actions:' showing 10 objects. The table columns are: icon, Label, Name, Type, Icon Path, Select Required, Dev Owner, and actionDetails. The data in the table is as follows:

icon	Label	Name	Type	Icon Path	Select Required	Dev Owner	actionDetails
□	Search	sharedTeamPickerSearch	containerteam		No		(10 objects)
□	Search	pickerSearch	search		No		
□	Search	search_for_attachments	forumPosting		No		
□	Search	commonSearch	object		No		
□	Search	queryPickerSearchButton	datasops		No		
□	Search	relatedDoc_search	part		No		
(0 objects selected)							

## Action Registry

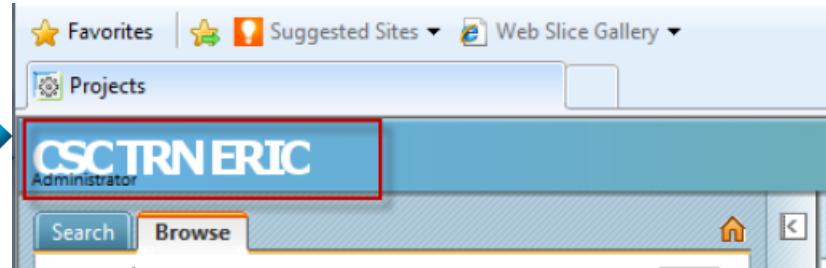
The action registry is a Windchill Service – including a cache – that:

- Reads the action and action model configuration files.
- Builds individual actions.
- Builds action models.
- It is initialized when the Method Server Starts.
- It can be reloaded without restarting the Method Server or Tomcat:
  - From a Windchill Shell, execute:  
*windchill com.ptc.netmarkets/util.misc.NmActionServiceHelper*
  - This command only reloads actions and action models; not JSP pages, or Method Server properties.

# Log and Navigation

# 1. Custom Logo

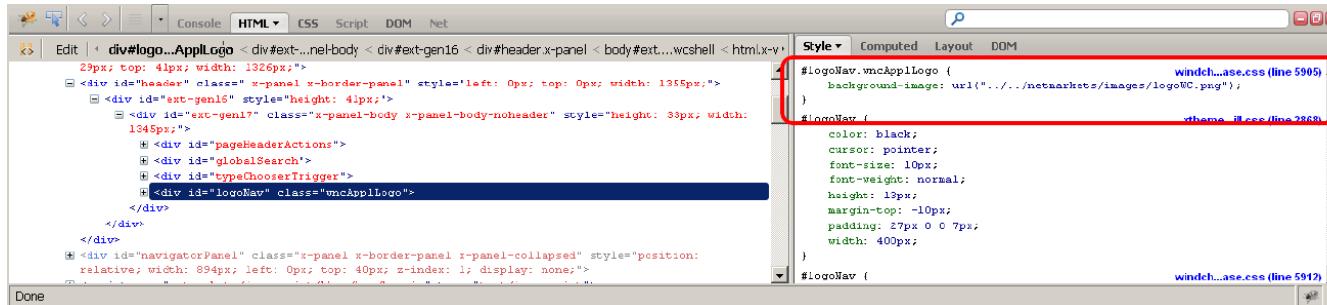
Custom Logo



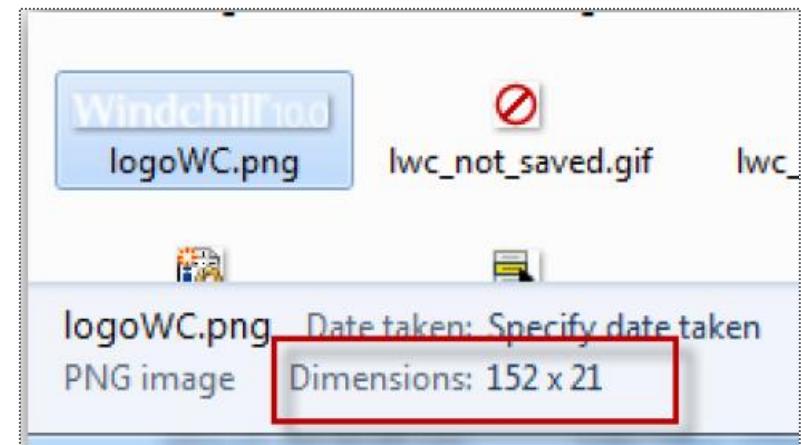
# 1. Custom Logo

## Review and Create Logo Image

1. We have to check which style sheet is being used and where the icon file is located.
2. This logo constitutes white text on a transparent background. As such, it will be difficult to see the logo on the white background of a File Manager window.



3. Check the size of original image on Windchill server.
4. Create custom image (for example, the file name is customLogo.png)



# 1. Custom Logo

## Update Style Sheet

1. Create a new style sheet called *custom.css* and place it in  
%WT\_HOME%\codebase\custom
2. Create *custom.css* on new folder
3. Copy the following block text to *custom.css*

```
#logoNav.wncApplLogo {  
    background-image: url("../custom/customLogo.png ");  
}
```

4. Register the *custom.css* file by updating  
*codebase\presentation.properties.xconf*

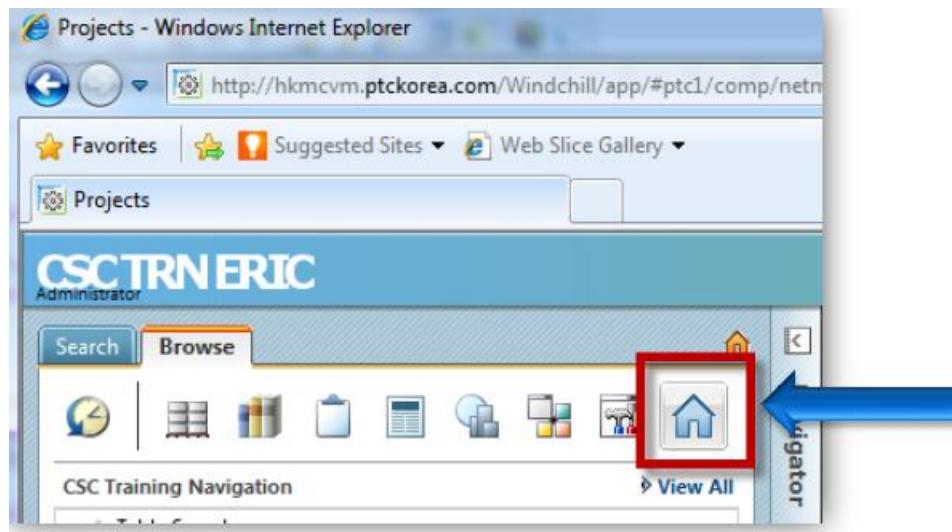
```
1 <?xml version="1.0" encoding="utf-8"?>  
2  
3 <!DOCTYPE Configuration  
4   SYSTEM "xconf.dtd">  
5 <Configuration targetFile="codebase/presentation.properties">  
6   <Property default="PTC" name="netmarkets.presentation.author"/>  
7   <Property default="custom/custom.css" name="netmarkets.presentation.cssFiles"/>  
8   <Property default="http://www.ptc.com" name="netmarkets.presentation.website"/>  
9 </Configuration>
```

Image path must be based on  
*custom.css* location

5. Execute *xconfmanager* and restart all services.



## 2. Navigation Menu (Icon)



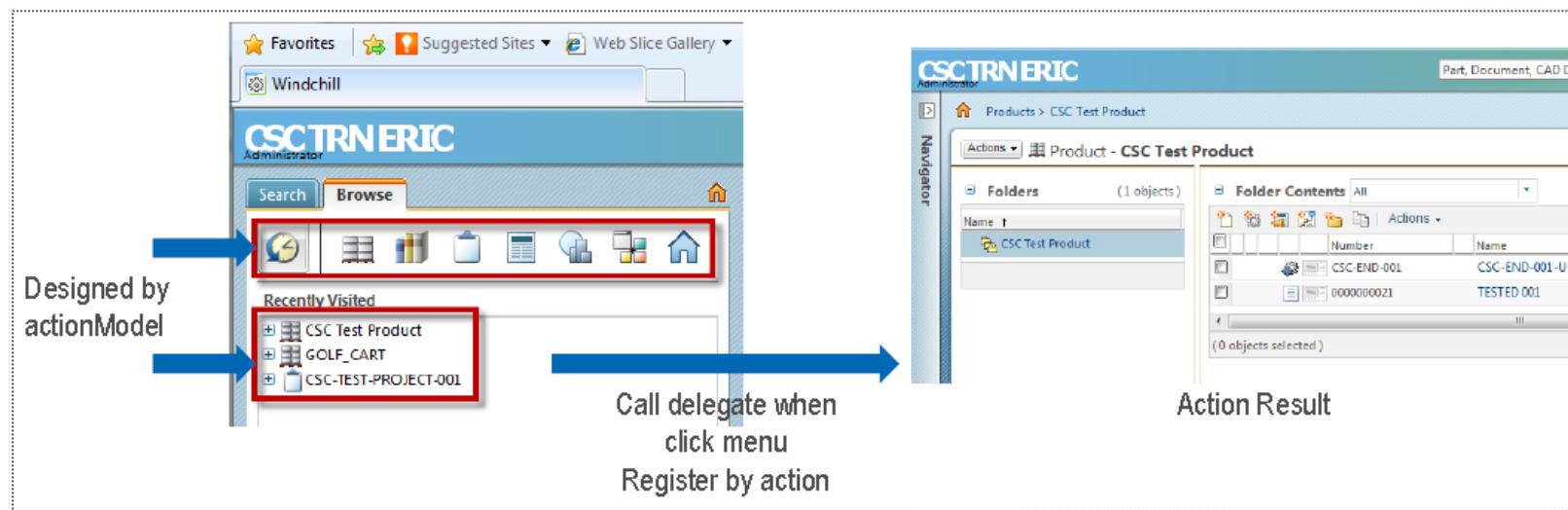
Navigation Icon

## 2. Navigation Menu

Action.xml and ActionModel.xml

Configuration of Windchill 10 navigation is similar to that of 9.1.

- All basic Windchill configuration files of action and action model are located in \$WT\_HOME/codebase/config/actions
- For navigation, icon and sub-menu use xxxAction.xml and xxxActionModel.xml
- ActionModel.xml includes a design of menu lists
- Action.xml includes activity of action model menu



## 2. Navigation Menu

### Register New XML Files

We can directly change the original XML, but it is not the best way because it can be changed when updating or deploying patches for Windchill. So, we try to use expanded XML files.

- OOTB has custom-actions.xml and custom-actionModels.xml
- Add new training XML in custom environment

**Add the following sentence in %WT\_HOME%\codebase\config\actions\custom-actions.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE listofactions SYSTEM "actions.dtd">
<listofactions>
    <include href="config/actions/csc/csc-actions.xml"/>
</listofactions>
```

**Add the following sentence in %WT\_HOME%\codebase\config\actions\custom-actionModels.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE actionmodels SYSTEM "actionmodels.dtd">
<actionmodels>
    <include href="config/actions/csc/csc-actionmodels.xml"/>
</actionmodels>
```

In this case, you must create one directory with the name “csc” in “\$WT\_HOME/codebase/config/actions”  
PTC Services Academy

## 2. Navigation Menu

### Create New Action

Create registered action file on the correct directory. Action and ActionModel files are in different format, so you have to watch out while making a new file.

#### Create and modify %WT\_HOME%\codebase\config\actions\csc\csc-actions.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE listofactions SYSTEM "actions.dtd">

<listofactions>
    <objecttype name="navigation" class="" resourceBundle="com.ptc.core.ui.navigationRB">
        <action name = "csc" uicomponent="TAB_ADMIN"/>
    </objecttype>
</listofactions>
```

- For Action file, you must add DTD configuration line such as line-2.
- Action-type file whose name is “navigation” is already assigned by Windchill, so we will just reuse the object-type action.
- If you want to add top menu (button), it is enough.

## 2. Navigation Menu

### Create New Action Model

Create registered action file on correct directory. Action and ActionModel files are in different format, so you have to watch out while creating new file.

### Create and modify %WT\_HOME%\codebase\config\actions\csc\csc-actionmodels.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<actionmodels>
    <model name="main navigation">
        <description>Main navigation (tabs)</description>
        <action name="home"      type="navigation"/>
        <action name="program"   type="navigation"/>
        <action name="product"   type="navigation"/>
        <action name="project"   type="navigation"/>
        <action name="change"    type="navigation"/>
        <action name="library"   type="navigation"/>
        <action name="org"       type="navigation"/>
        <action name="site"      type="navigation"/>
        <action name="supplier"  type="navigation"/>
        <!-- entry for customization tab -->
        <action name="customization" type="navigation"/>
        <action name="csc"       type="navigation"/>
    </model>

    <model name="csc navigation">
        <description>For CSC navigation</description>
    </model>
</actionmodels>
```

It is overriding model. It can be copied from OOTB action models, and we just add our action

Do not copy sample description because each of installation is different , so the elements will be different.  
Copy your system model block.

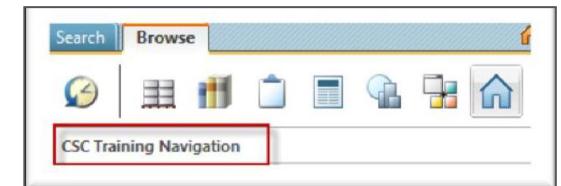
This must be added although sub-menu isn't existed

## 2. Navigation Menu

### Add Resource Bundle

The resource bundles of menu which is displayed on the UI are managed the other way comparing it with original OOTB resource bundle architecture.

- The resource file is managed on “\$WT\_HOME/codebase/action\_[Locale code].properties
- All of locale character must be changed UTF-8 format except on English
- Also you must update default property file. (action.properties)



### Update %WT\_HOME%\codebase\action\_LOCALE.properties and action.properties

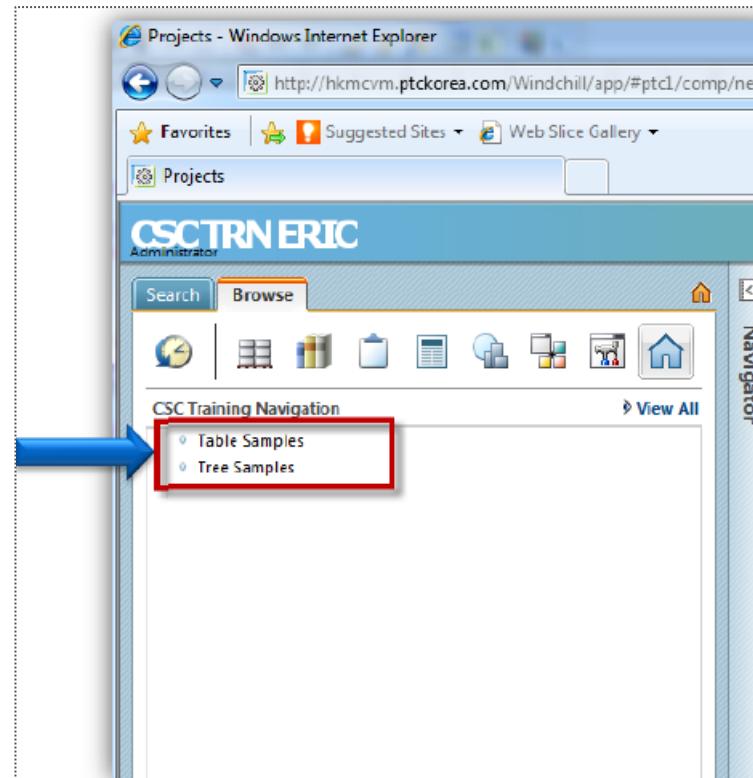
```
object.search_resulttable_authorizationagreement_file.description=\ud30c\uc77c  
object.search_resulttable_authorizationagreement_edit.description=\ud3b8\uc9d1  
  
navigation.csc.description=CSC  
navigation.csc.icon=help_32x32_manual.gif
```

The physical path is  
\$WT\_HOME/codebase/ne  
tmarkets/images

- Finally, you must restart all Windchill services.
- If you want to adapt your configuration without the restart service, use the following Windchill command:  
***windchill com.ptc.netmarkets.util.misc.NmActionServiceHelper***

### 3. Sub-Navigation Menu

Sub-Navigation Menu



### 3. Sub-Navigation Menu

#### Modify ActionModels.xml

Designing sub-menu using actions on %WT\_HOME%\codebase\config\actions\csc\csc-actionModels.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<actionmodels>
    <model name="main navigation">
        <description>Main navigation (tabs)</description>
        <action name="home" type="navigation"/>
        <action name="program" type="navigation"/>
        <action name="product" type="navigation"/>
        <action name="project" type="navigation"/>
        <action name="change" type="navigation"/>
        <action name="library" type="navigation"/>
        <action name="org" type="navigation"/>
        <action name="site" type="navigation"/>
        <!-- <action name="supplier" type="navigation"/> -->
        <!-- entry for customization tab -->
        <action name="customization" type="navigation"/>
        <action name="csc" type="navigation"/>
    </model>
    <model name="csc navigation" defaultActionName="tableSamples" defaultActionType="csc">
        <action name="tableSamples" type="csc" />
        <action name="treeSamples" type="csc" />
    </model>
</actionmodels>
```

- Sub-model name is created by a rule which combines name and type values of the main navigation's action.
- Action is generated to find action from action.xml. Type is objecttype name, and name is an element name of objecttype.

### 3. Sub-Navigation Menu

#### Modify Actions.xml

Sub-menu is similar to adding navigation

**Adding designed action for sub-menus on %WT\_HOME%\codebase\config\actions\csc\csc-actions.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE listofactions SYSTEM "actions.dtd">
<listofactions>
    <objecttype name="navigation" class="" resourceBundle="com.ptc.core.ui.navigationRB">
        <action name = "csc">
        </action>
    </objecttype>
    <objecttype name="csc" class="" resourceBundle="com.ptc.core.ui.navigationRB">
        <action name="tableSamples">
            <component windowType="page" name="netmarkets.project.list.table"/>
        </action>
        <action name="treeSamples">
            <command windowType="page" url="wtcore/jsp/csc/jca/tree/tree.jsp"/>
        </action>
    </objecttype>
</listofactions>
```

**WindowType attribute on <command/>**  
"page" : Display directly on current page. Meaning is refresh.  
"popup" : Display new popup page. Meaning is open new popup page.

- WC10 was added a MVC architecture, so, in the action, you can call MVC builder using <component> tag instead of <command> tag. (MVC will be explained in the next phase.)

```
<action name="tableSamples">
    <component windowType="page" name="netmarkets.project.list.table"/>
</action>
```

### 3. Sub-Navigation Menu

#### Modify action.properties for Display

Set display text to action.properties. If you don't set this property, the menu will not show text description such as the navigation icon.

- The resource file is managed on “\$WT\_HOME/codebase/action\_[Locale code].properties
- All locale characters except English must be changed to UTF-8 format
- Also you must update default property file (action.properties)

#### Modify to %WT\_HOME%\codebase\action\_LOCALE.properties

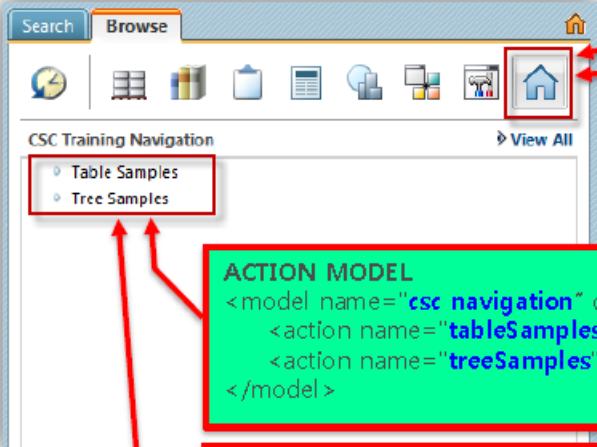
```
...
...
object.search_resulttable_authorizationagreement_file.description=\ud30c\uc77c
object.search_resulttable_authorizationagreement_edit.description=\ud3b8\uc9d1

##### NEW ADDED
#### TOP MENU
navigation.csc.description=CSC

#### SUB MENU (CSC)
csc.tableSamples.description=Table Samples
csc.treeSamples.description=Tree Samples
```

# 4. Summary for Navigation

## Modify action.properties for Display



The diagram illustrates the relationship between four components:

- ACTION** (Yellow Box):

```
<objecttype name="navigation" class="" resourceBundle="com.ptc.core.ui.navigationRB">
  <action name = "csc">
  </action>
</objecttype>
```
- ACTION MODEL** (Green Box):

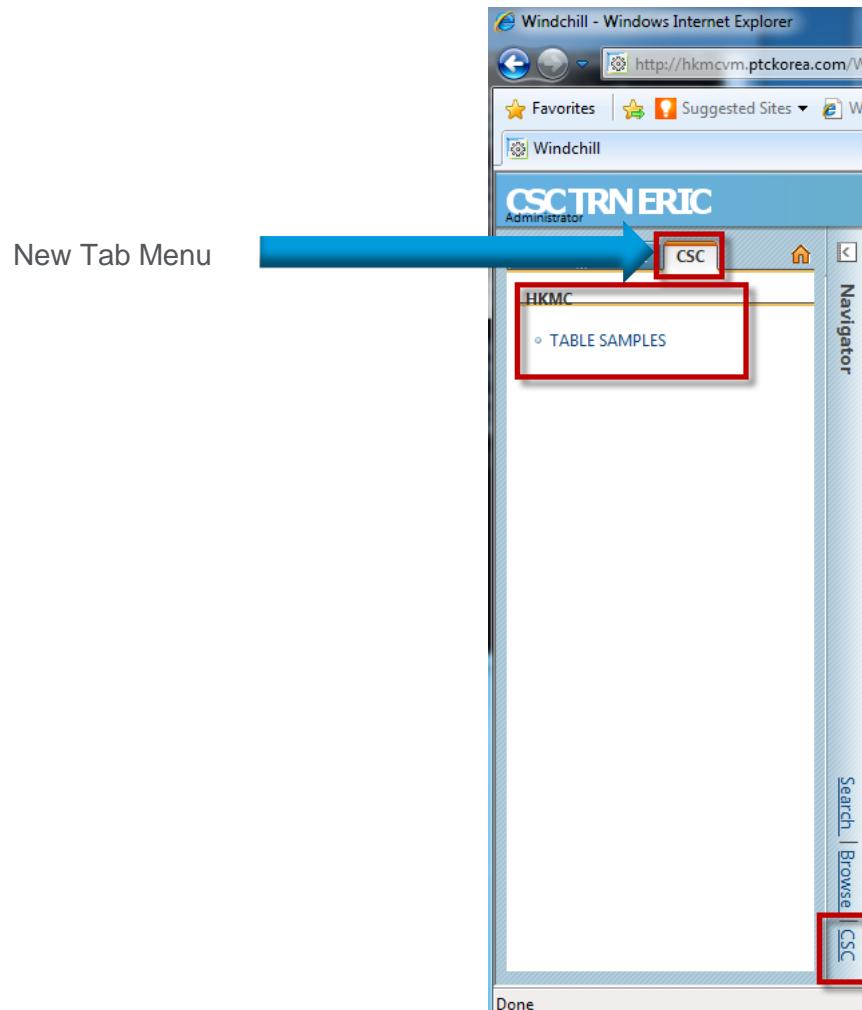
```
<model name="main navigation">
  ...
  ...
  <action name="csc" type="navigation"/>
</model>
```
- ACTION MODEL** (Green Box):

```
<model name="csc navigation" defaultActionName="tableSamples" defaultActionType="csc">
  <action name="tableSamples" type="csc" />
  <action name="treeSamples" type="csc" />
</model>
```
- ACTION** (Yellow Box):

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

Red arrows indicate the flow from the ACTION MODEL components to the ACTION components, and another red arrow points from the ACTION component back to the ACTION MODEL component in the center.

## 4. Create Tab



## 4. Create Tab

Add Tab on actionModels.xml

Basically, when we add new tabs, it will have different functionality between OOTB

- Navigation bar action must be included URL for sub-navigation lists.
- All sub-navigation will be designed on JSP pages
- Tab description (display) must use Resource bundle (do not use action.properties)

Copy <navigator> block from \$WT\_HOME/codebase/config/actions/navigation-actionModel.xml to csc-actionModels.xml. And then add new tab model.

```
<?xml version="1.0" encoding="UTF-8"?>

<actionmodels>
    <model name="navigator">
        <submodel name="search navigation"/>
        <submodel name="main navigation"/>
        <submodel name="addtab navigation"/>
    </model>
```

- Tab must be added on “navigator” block
- For tab, use <submodel> tag



## 4. Create Tab

### Add Navigation Bar for New Tab on actionModels.xml

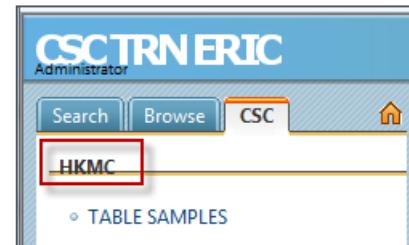
**Add navigation design for new tab. In this example, we will just add one navigation.**

```
<?xml version="1.0" encoding="UTF-8"?>

<actionmodels>
    <model name="navigator">
        <submodel name="search navigation"/>
        <submodel name="main navigation"/>
        <submodel name="addtab navigation"/>
    </model>

    <model name="addtab navigation" id="browseActions"
resourceBundle="ext.csc.training.navigation.navigationRB">
        <description>
            For CSC TAB
        </description>
        <action name="HKMC" type="support"/>
    </model>
```

- For displaying tab, you must create resource bundle



## Create Resource Bundle for Tab

**Create new resource bundle using annotation. If you want to use other languages, you have to create target language's resource bundle.**

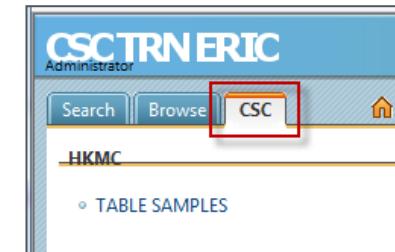
```
package ext.csc.training.navigation;

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

@RBUUID("ext.csc.training.navigation.navigationRB")
public final class navigationRB extends WTListResourceBundle {

    @RBEntry("CSC")
    @RBComment("CSC tab on Navigator.")
    public static final String PRIVATE_CONSTANT_364 = "object.addtab navigation.description";
}
```

- The constant value must include tab model name. For example, “object.addtab navigation.description.”
- Resource Bundle must be created using Windchill annotation.
- Resource Bundle must be extended from WTListResourceBundle.

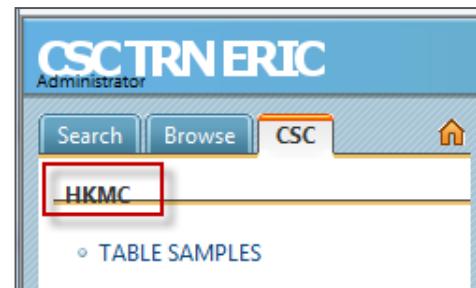


## 4. Create Tab

### Add Description of Navigation Bar to action.properties

**Add a description of navigation bar on the action.properties. If you want to use other language, you have to create target language's property file.**

```
support.HKMC.tooltip=HKMC  
support.HKMC.description=HKMC
```

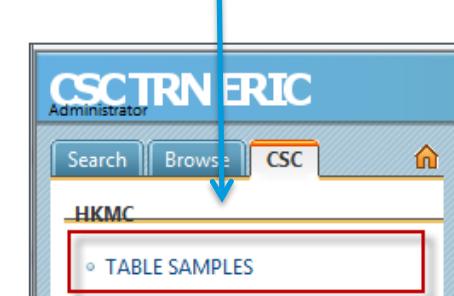


### Build Action for Each of Action Model

For Navigator tab and navigation bar, we define the action. Add the following block to the csc-actions.xml

```
<objecttype name="support" resourceBundle="com.ptc.core.ui.navigationRB">
  <action name="HKMC">
    <command windowType="page" url="wtcore/jsp/csc/hkmc/hkmc.jsp"/>
  </action>
</objecttype>
```

- “addtab navigation” action doesn’t need to be added on the action (tab).
- “HKMC” action must include URL of jsp for sub-navigation body.



### Build JSP for Sub-Navigation Body

You can create sub-navigation body using JSP which includes general HTML as the following.

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

<p>
<a
href="/Windchill/app/#wtcore/jsp/csc/jca/tree/tree.jsp">TABLE SAMPLES</a>

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

- JSP file must be included Windchill begin and end JSP.
- All link must use Windchill servlet path. For example “/Windchill/app/#wtcore/....”

MVC

## Understanding MVC

- Windchill 10 is using new architecture for UI
  - Look at the attached document
  - It is a document to understand MVC architecture used in Windchill 10
- Simple understanding of implementation
  - For implementation, we understand how to configure and call MVC function in action or JSP
  - We can use samples and tools for understanding WC10 implementation (Carambola)
- Set Carambola package
  - Open Site > Utilities > Preference Management
  - Change a value to true for Client customization

Preference Manager Standard

Name	Value	Description
Add to Baseline	Yes	Add to Baseline operation preferences
Arbortext	Yes	Arbortext Preferences
Attachments	Yes	
Attribute Handling	Yes	
Change Management	Yes	Change Management preferences
Client Customization	Yes	Value that determines if the client customization examples and tools are displayed.
Create and Edit	Yes	

Part Table (Sync) - Windows Internet Explorer  
https://himcmw.ptckorea.com/Windchill/app/#ptc1/carambola/tools/list?oid=0R%3Avrt.org.WTUser%3A11&uid=1

Favorites Suggested Sites Web Slice Gallery Windchill

CSCTRN ERIC

Tools

This is a list of various Windchill tools useful for developers and customizers. See the [Tools Overview Documentation](#) for more detailed info about each tool.

Action	Description
Action	Find an Action
Action Model	Find an Action Model
Application Context Service/Resource Properties	Generates a report for the classes, delegates, or resources configured to be used by services or factories.
Available Attributes	Generates a report on the attributes available for a Windchill type.
Logical Attributes Report	Generate a report on the attributes and their external forms for a given Windchill type.
Property Report	The Property Report lists all the attributes defined for a given object type and shows whether different configurations exist.
Reload Action(s)	Reload action configurations
MVC Builder Search	Finds MVC component builders by ID
MVC Builder Scan Report	MVC Builder Scan Report - Mini
MVC Builder Scan Report	MVC Builder Scan Report - Full
Modeled Objects	Find information about Modeled Objects
Log4j	Page where log4j loggers can be enabled. Must be Site administrator to adjust logging levels. Find the Click to enable log4j. See Tools Overview Documentation for more info.
jDebug	Click to enable a JavaScript logger. Just hit "Ok" to enable all loggers or enter a logger name to enable it.
jLog	Click to enable a log4j/javascript logger. Just hit "Ok" to enable all loggers or enter a logger name to enable it.
jcaDebug	Click to enable jcaDebug. See Tools Overview Documentation for more info.

## MVC Configuration

- All the components for MVC will be registered when starting Method Server
  - The components list is registered in \$WT\_HOME/codebase/WEB-INF/MVCD Dispatcher-servlet.xml

```
<?xml Version="1.0" encoding="UTF-8"?>
<!-- Application context definition for "MVC" DispatcherServlet.
-->

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

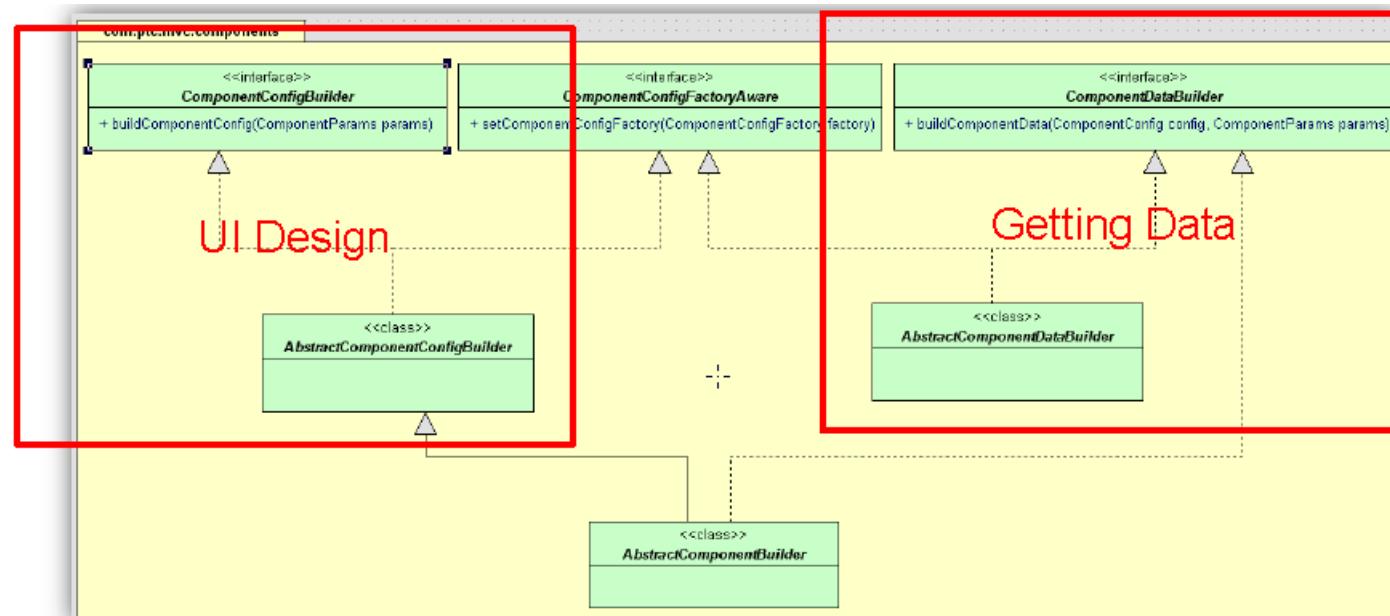
    <import resource="classpath:config/mvc/mvc.xml" />
    <import resource="classpath:config/mvc/jca-mvc.xml" />
    <import resource="classpath:config/mvc/*-configs.xml" />
    <import resource="classpath:config/mvc/custom.xml" />

    <bean id="defaultHandlerMappings"
        class="org.springframework.beans.factory.config.PropertiesFactoryBean">
        <property name="locations">
            <list>
                <value>
                    classpath:/config/mvc/*-urlMappings.properties
                </value>
                <value>classpath:/config/mvc/custom.properties</value>
            </list>
        </property>
    </bean>
```

- If you want to register your own XML, you can add in MVCD Dispatcher-servlet.xml
  - For training, you have to add **<import resource="classpath:config/mvc/csc-custom.xml" />**
- Generally, if you want to register your MVC component, update the custom.xml
  - For training, we will use additional XML with the name csc-custom.xml
  - Copy and rename custom.xml to csc-custom.xml
  - Add **<mvc:builder-scan base-package="com.csc.mvc" />**
  - Or directly register component class **<bean class="com.ptc.windchill.enterprise.product.mvc.builders.ProductListTableBuilder" />**
  - After that, when starting Method Server, all components will be loaded under “codebase/com/csc/mvc.”

## How to Create MVC Component

- There are several rules for creating MVC component:
  - All components must be extended to OOTB abstraction builder
    - com.ptc.mvc.components.AbstractComponentBuilder
    - The builder is including config and data builder, so generally we use that abstract builder



## How to Create MVC Component

- There are several rules for creating MVC component:
  - Component must be set component Id for using in action or JSP by annotation
    - Annotate it with [ComponentBuilder](#) to specify to which componentId its mapped for.
      - @ComponentBuilder(value = "<componentId>", type = ComponentBuilderType.CONFIG\_ONLY)
      - @ComponentBuilder(value = "<componentId>", type = ComponentBuilderType.DATA\_ONLY)
      - @ComponentBuilder(value = "{<componentId1>, <componentId2>}")

```
//Config Builder
@ComponentBuilder(value = "carambola.mvc.table.separateBuilders", type = ComponentBuilderType.CONFIG_ONLY)
public class MvcTableConfigBuilder extends AbstractComponentConfigBuilder {
}

//Data Builder
@ComponentBuilder(value = "carambola.mvc.table.separateBuilders", type = ComponentBuilderType.DATA_ONLY)
public class MvcTableDataBuilder implements ComponentDataBuilder, ComponentConfigFactoryAware {
}

//Config + Data Builder
@ComponentBuilder("carambola.mvc.tree")
public class MvcTreeBuilder extends AbstractConfigurableTableBuilder {
```

## How to Create MVC Component

- There are several rules for creating MVC component:
  - Overriding “*buildComponentData*” function for getting data
    - Input parameter:
      - ComponentConfig
      - ComponentParams
    - Return
      - Object
  - Overriding “*buildComponentConfig*” function for design UI
    - Input parameter:
      - ComponentParams
    - Return
      - ComponentConfig

# How to Call MVC Component in Action or JSP

- Sample of MVC component

```
@ComponentBuilder("com.hkmc.cad.builder.packageHistoryTableBuilder")
public class PackageHistoryTableBuilder extends AbstractComponentBuilder {
private final ClientMessageSource resource = getMessageSource("com.hkmc.cad.resource.CADModuleRB");
@Override
public Object buildComponentData(ComponentConfig paramComponentConfig, ComponentParams paramComponentParams) throws Exception {
NmHelperBean localNmHelperBean = ((JcaComponentParams) paramComponentParams).getHelperBean();
localNmHelperBean.getRequest().setAttribute("showContextInfo", "false");
NmCommandBean localNmCommandBean = localNmHelperBean.getNmCommandBean();
Workable localWorkable = (Workable) localNmCommandBean.getPrimaryOid().getRefObject();
QueryResult iterRev = HistoryTablesCommands.currentRevIterHistory(localWorkable);
return iterRev;
}
@Override
public ComponentConfig buildComponentConfig(ComponentParams paramComponentParams) throws WTEException {
ComponentConfigFactory localComponentConfigFactory = getComponentConfigFactory();
JcaTableConfig localJcaTableConfig = (JcaTableConfig) localComponentConfigFactory.newTableConfig();
localJcaTableConfig.setComponentMode(ComponentMode.VIEW);
localJcaTableConfig.setLabel(resource.getMessage(CADModuleRB.History));
localJcaTableConfig.setAutoGenerateRowId(true);
localJcaTableConfig.setConfigurable(true);
ColumnConfig localColumnConfig1 = localComponentConfigFactory.newColumnConfig(HKMCConstants.Epm.Attributes.PROJECT_CODE,
resource.getMessage(CADModuleRB.ProjectCode), true);
ColumnConfig localColumnConfig2 = localComponentConfigFactory.newColumnConfig(HKMCConstants.Epm.Attributes.PKG_TYPE, true);
ColumnConfig localColumnConfig3 = localComponentConfigFactory.newColumnConfig(HKMCConstants.Epm.Attributes.PKG_REGION, true);
ColumnConfig localColumnConfig5 = localComponentConfigFactory.newColumnConfig(HKMCConstants.Epm.Attributes.PKG_REGION, true);
ColumnConfig localColumnConfig4 = localComponentConfigFactory.newColumnConfig(HKMCConstants.Epm.Attributes.REVISION_NUMBER, true);
localColumnConfig4.setDataUtilityId("packageRevisionUtility");
ColumnConfig localColumnConfig7 = localComponentConfigFactory.newColumnConfig("versionInfoPageAction", false);
localColumnConfig7.setDataUtilityId("versionInfoPageAction");
localJcaTableConfig.addComponent(localColumnConfig1);
localJcaTableConfig.addComponent(localColumnConfig7);
localJcaTableConfig.addComponent(localColumnConfig2);
localJcaTableConfig.addComponent(localColumnConfig3);
localJcaTableConfig.addComponent(localColumnConfig4);
localJcaTableConfig.addComponent(localColumnConfig5);
localJcaTableConfig.setView("/components/table.jsp");
return localJcaTableConfig;
}
}
```

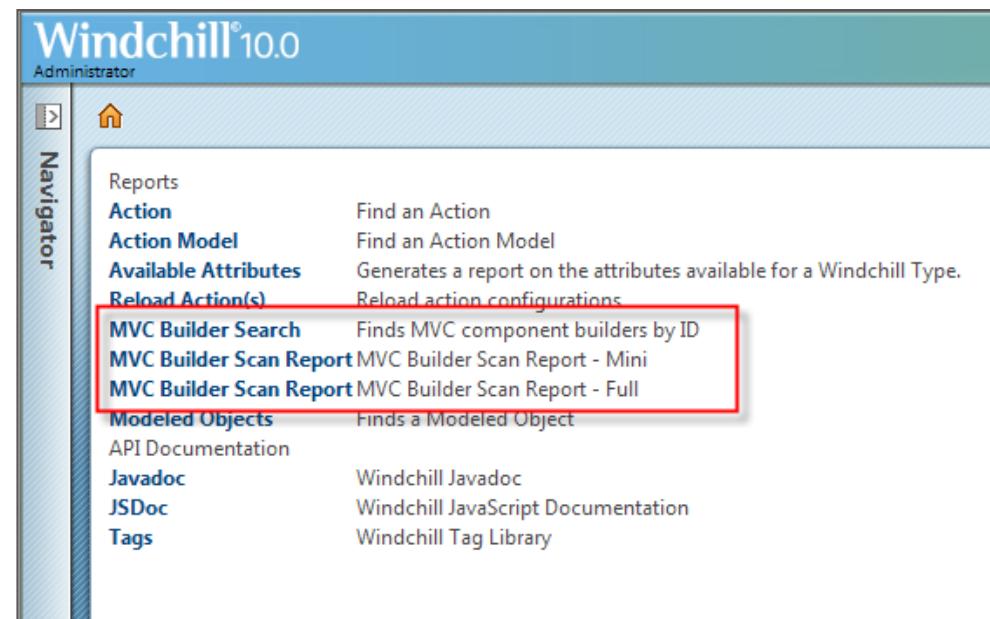
## How to Create MVC component

- In buildComponentConfig function, use JCA component for UI design.
- JCA Components that implement ComponentConfig interface:
  - JcaTableConfig
  - JcaAttributesTableConfig
  - JcaTreeConfig
  - JcaAttributePanelConfig
  - JcaColumnConfig
  - JcaInfoConfig
  - JcaPropertyConfig
  - JcaPropertyPanelConfig

```
● @ComponentBuilder("com.hkmc.cad.builder.packageHistoryTableBuilder")
  public class PackageHistoryTableBuilder extends AbstractComponentBuilder {
    @Override
      public ComponentConfig buildComponentConfig(ComponentParams paramComponentParams) throws WTEException {
        ComponentConfigFactory localComponentConfigFactory = getComponentConfigFactory();
        JcaTableConfig localJcaTableConfig = (JcaTableConfig) localComponentConfigFactory.newTableConfig();
        return localJcaTableConfig;
    }
}
```

## Navigate Customization > Tools

- MVC Builder Search: Find the Builder given a componentId
  - <demo>
- MVC Builder Scan Report: Find a report on mvc-scan
  - <demo>
  - log4j.logger.com.ptc.mvc.scan=INFO should be enabled



## More Resources

- [JCA MVC](#)
- [Javadoc](#)

# Table

## Customization Process

We will create MVC table by the following process:

- Register custom dispatcher
- Register custom MVC component library
- Create MVC table class for search product
- Register MVC ID to action

The screenshot shows a software interface titled 'CSC TRN ERIC' with a sub-header 'Administrator'. On the left, there's a vertical 'Navigator' bar with icons for search, home, and other navigation. The main area displays a table titled 'CSC Table Sample01' with a dropdown menu showing 'All'. The table has columns: Name, Owner, Last Modified, Description, Creator, Created On, and Private Access. The data rows are:

Name	Owner	Last Modified	Description	Creator	Created On	Private Access
CSC Test Product	① Kim, Eric	2011-07-12 20:41 CST		Kim, Eric	2011-07-12 20:41 CST	No
Drive System	① Kim, Eric	2011-07-04 21:27 CST	PDMLink Drive System demo	Kim, Eric	2011-07-04 21:27 CST	No
GENERIC_COMPUTER	① Kim, Eric	2011-07-04 21:45 CST	PDMLink Options and Variants demo	Kim, Eric	2011-07-04 21:45 CST	No
GOLF_CART	① Kim, Eric	2011-07-04 21:27 CST	PDMLink golf cart demo	Kim, Eric	2011-07-04 21:27 CST	No
ProductView Demo	① Kim, Eric	2011-07-04 21:45 CST	ProductView demos	Kim, Eric	2011-07-04 21:45 CST	No

( 0 objects selected )

# 1. Register MVC Builder Path

## Register MVC Builder Path

### 1. Open \$WT\_HOME/codebase/WEB-INF/MVCDispatcher-servlet.xml and add following

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    - Application context definition for "MVC" DispatcherServlet.
-->

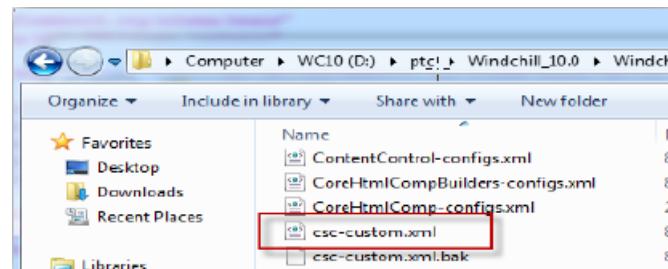
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <import resource="classpath:config/mvc/mvc.xml" />
    <import resource="classpath:config/mvc/jca-mvc.xml" />
    <import resource="classpath:config/mvc/*-configs.xml" />
    <import resource="classpath:config/mvc/custom.xml" />
    <import resource="classpath:config/mvc/csc-custom.xml" />

    <bean id="defaultHandlerMappings"
        class="org.springframework.beans.factory.config.PropertiesFactoryBean">
        <property name="locations">
            <list>
                <value>
                    classpath:/config/mvc/*-urlMappings.properties
                </value>
                <value>classpath:/config/mvc/custom.properties</value>
            </list>
        </property>
    </bean>

</beans>
```

### 2. Copy \$WT\_HOME/codebase/config/mvc/custom.xml and rename “csc-custom.xml”



# 1. Register MVC Builder Path

## Register MVC Builder Path

### 3. Open \$WT\_HOME/codebase/config/mvc/csc-custom.xml and add following

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.ptc.com/schema/mvc"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd
        http://www.ptc.com/schema/mvc http://www.ptc.com/schema/mvc/mvc-10.0.xsd">

    <!-- Configurations in this file override all other configurations -->
    <mvc:builder-scan base-package="com.csc.mvc" />

</beans>
```

## 2. Create MVC Component

### Create MVC Component Class

#### 4. Download the sample Java from OpenGrok which is listed in the Product list.

- <http://ah-opengrok/xref/x-20-M010/wcEnterprise/ProductLibrary/src/com/ptc/windchill/enterprise/product/mvc/builders/ProductListTableBuilder.java>

#### 5. Change the following in the downloaded class:

- Change Class name and File name (CSCSampleTable01)
- Change package (com.csc.mvc)
- Change the Component annotation (csc.mvc.sampleTable01)
- Change Table ID (csc.mvc.sampleTable01)
- Change Label (table.setLabel("CSC Table Sample01");)

```
1④ /* bcwt1 */
2 package com.csc.mvc;
3
4④ import wt.util.WIException;
5
6
7 /**
8  * MVC builder class for Product List table
9  */
10 @ComponentBuilder("csc.mvc.sampleTable01")
11 public class CSCSampleTable01 extends AbstractConfigurableTableBuilder {
12     private final ClientMessageSource messageSource = getMessageSource("com.ptc.windchill.enterprise.product.ProductClient");
13
14
15     /*
16      * (non-Javadoc)
17      * @see com.ptc.mvc.components.ComponentDataBuilder#buildComponentData(com.ptc.mvc.components.ComponentConfig, com.ptc.mvc.components.ComponentParams)
18      */
19     @Override
20     public Object buildComponentData(ComponentConfig config, ComponentParams params) throws WIException {
21
22         String tableId="csc.mvc.sampleTable01";
23         return ProductListCommand.getProducts(tableId);
24     }
25
26     @Override
27     public ComponentConfig buildComponentConfig(ComponentParams params) throws WIException {
28         String helpContext ="ProductsHelp";
29         ComponentConfigFactory factory = getComponentConfigFactory();
30         TableConfig table = factory.newTableConfig();
31         table.setId("csc.mvc.sampleTable01");
32         table.setConfigurable(true);
33         table.setType("wt.pdmlink.PDMLinkProduct");
34     }
35 }
```

## 2. Create MVC Component

### Create MVC Component Class

#### 6. Add new action model and action

csc-actionmodels.xml

```
<model name="csc navigation" defaultActionName="treeSamples" defaultActionType="csc">
    <description>
        For CSC navigation
    </description>
    <action name="tableSample01" type="csc" />
    <action name="tableSamples" type="csc" />
    <action name="treeSamples" type="csc" />
</model>
```

csc-actions.xml

```
<objecttype name="csc" resourceBundle="com.ptc.core.ui.navigationRB">
    <action name="tableSample01">
        <component windowType="page" name="csc.mvc.sampleTable01"/>
    </action>
```

#### 7. Update action property for display description

actions.properties and action\_\${locale}.properties

```
csc.tableSample01.description=MVC Table Sample01
```

#### 8. Restart All service

## 2. Create MVC Component

### Result

The screenshot shows two windows of the CSC Training Navigation application. The top window displays the 'CSC Training Navigation' menu with items: Search, Browse (which is selected), CSC, View All, and Navigator. Under 'Browse', there are three categories: MVC Table Sample01 (highlighted with a red box), Table Samples, and Tree Samples. A yellow arrow points from the 'MVC Table Sample01' item down to the second window. The bottom window shows the details of 'CSC Table Sample01'. It has a table with columns: Name, Owner, Last Modified, Description, Creator, Created On, and Private Access. The table contains five rows of data:

Name	Owner	Last Modified	Description	Creator	Created On	Private Access
CSC Test Product	Kim, Eric	2011-07-12 20:41 CST		Kim, Eric	2011-07-12 20:41 CST	No
Drive System	Kim, Eric	2011-07-04 21:27 CST	PDMLink Drive System demo	Kim, Eric	2011-07-04 21:27 CST	No
GENERIC_COMPUTER	Kim, Eric	2011-07-04 21:45 CST	PDMLink Options and Variants demo	Kim, Eric	2011-07-04 21:45 CST	No
GOLF_CART	Kim, Eric	2011-07-04 21:27 CST	PDMLink golf cart demo	Kim, Eric	2011-07-04 21:27 CST	No
ProductView Demo	Kim, Eric	2011-07-04 21:45 CST	ProductView demos	Kim, Eric	2011-07-04 21:45 CST	No

(0 objects selected)

## Search and Result Table

# 1. Create Populator Tag Library

## Populator Tag and Wrapper Tag

### What is wrapper tag?

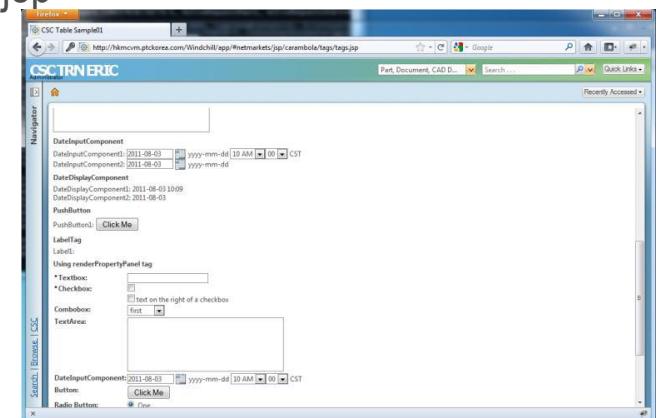
- Wrapper tag helps to simplify UI design on JSP source code.
- If using wrapper tag, JSP source code doesn't need to involve getting data source code.
- Wrapper tag is OOTB tag library for UI modeling.

### What is populator tag?

- Wrapper tag cannot get data, so it needs populator tag for getting data.
- Populator tag is not OOTB tag, so it must be created by developer. It is a custom tag.

### Sample page of wrapper tag

- <http://localhost/Windchill/app/#netmarkets/jsp/carambola/tags/tags.jsp>



# 1. Create Populator Tag Library

## Tag Type of Wrapper Tag

### The lists of wrapper tag

#### – Textbox

- <w:textBox name="textbox1" onblur="foo"/>
- <w:textBox name="textbox2" onblur="foo" required="true"/>
- <w:textBox name="textbox3" onblur="foo" styleClass="boo bar"/>
- <w:textBox name="textbox4" size="40" maxlength="50"/>
- <w:textBox name="textbox5" hidden="true" enabled="false" value="enabled=false"/>

TextBox

TextBox1:	<input type="text"/>
TextBox2:	<input type="text"/>
TextBox3:	<input type="text"/>
TextBox4:	<input type="text"/>
TextBox5: enabled=false	<input type="text"/>

#### – CheckBox

- <w:checkBox name="checkbox1" onblur="foo" renderExtra="whereDoesThisGo"/>
- <w:checkBox name="checkbox2" onblur="foo" required="true" submitAsNew="true"/>
- <w:checkBox name="checkbox3" onblur="foo" styleClass="boo bar" checked="true"/>
- <w:checkBox name="checkbox4" enabled="false" />
- <w:checkBox name="checkbox4" enabled="false" checked="true"/>

CheckBox

CheckBox1:	<input type="checkbox"/>
CheckBox2:	<input type="checkbox"/>
CheckBox3:	<input checked="" type="checkbox"/>
CheckBox4:	<input type="checkbox"/>
CheckBox5:	<input type="checkbox"/>

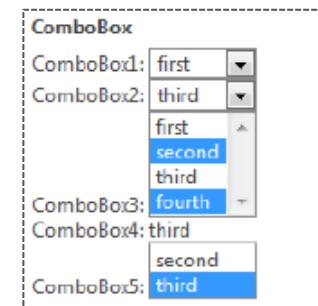
# 1. Create Populator Tag Library

## Tag Type of Wrapper Tag

### The lists of wrapper tag

- **ComboBox**

- <w:comboBox name="combobox" onselect="lauch()" internalValues="\${ivals}" displayValues="\${dvals}" />
- <w:comboBox name="combobox2" onmousedown="doFoo()" required="true" internalValues="\${ivals}" displayValues="\${dvals}" selectedValues="\${svals}"/>
- <w:comboBox name="combobox3" onmousemove="doBar()" styleClass="boo bar" multiSelect="true" internalValues="\${ivals}" displayValues="\${dvals}" selectedValues="\${multivals}"/>
- <w:comboBox name="combobox4" enabled="false" internalValues="\${ivals}" displayValues="\${dvals}" selectedValues="\${svals}"/>
- <w:comboBox name="combobox5" size="2" internalValues="\${ivals}" displayValues="\${dvals}" selectedValues="\${svals}"/>
- ComboBox needs populate value, for example, \${dvals}, \${svals} will set on populate tag process



# 1. Create Populator Tag Library

## Tag Type of Wrapper Tag

### The lists of wrapper tag

- TextArea

- `<w:textArea id="message" name="message" value="" cols="39" rows="5" required="false"/>`



- DateInputComponent

- `<w:dateInputComponent name="startdate" required="true" dateValueType="DATE_TIME"/>`
  - `<w:dateInputComponent name="startdate" required="true" dateValueType="DATE_ONLY"/>`



- DateDisplayComponent

- `<w:dateDisplayComponent name="datetime" dateDisplayFormat="${standardDateTime}" dateValue="${date}"/>`
  - `<w:dateDisplayComponent name="datetime2" dateDisplayFormat="${standardDate}" dateValue="${date}"/>`



# 1. Create Populator Tag Library

## Tag Type of Wrapper Tag

### The lists of wrapper tag

- PushButton

- `<w:button name="button1" value="Click Me" typeSubmit="false" toolTipText="Click Me" onclick="alert('clicked')"/>`



- Label

- `<w:label value="${labelCc}" styleClass="boo bar"/>`



# 1. Create Populator Tag Library

## Using RenderPropertyPanel JCA Tag

By using RenderPropertyPanel JCA tag action, all wrapper tags will stand in line.

```
<jca:renderPropertyPanel>
    <jca:addHeader text="Adding a bunch of different gui components..." />
    <w:textBox propertyLabel="Textbox A" name="textboxA" onBlur="foo" required="true"/>
    <w:checkbox propertyLabel="Checkbox" name="checkboxA" onBlur="foo" required="true" submitAsNew="true"/>
    <w:checkbox label="text on the right of a checkbox" name="checkboxB" onBlur="foo" required="false" submitAsNew="true" renderLabel="true" renderLabelOnRight="true" />
    <w:comboBox propertyLabel="Combobox" name="comboBoxA" onSelect="lauch()" internalValues="#{ivals}" displayValues="#{dvals}" />
    <w:textArea propertyLabel="TextArea" id="messageA" name="messageA" value="" cols="38" rows="5" required="false"/>
    <w:dateInputComponent propertyLabel="DateInputComponent" name="startdateA" required="true" dateValueType="DATE_TIME"/>
    <w:button propertyLabel="Button" name="button1" value="Click Me" typeSubmit="false" toolTipText="Click Me" onClick="alert('clicked')"/>
    <w:radioButton propertyLabel="Radio Button" label="One" value="123" name="radio" checked="true" onclick="doFoo()"/>
    <w:radioButton label="Two" value="456" name="radio" onclick="doFoo()"/>
    <w:dateDisplayComponent propertyLabel="Date" name="datetime" dateDisplayFormat="#{standardDateTime}" dateValue="#{date}"/>
    <jca:addSeparator/>
    <jca:addSeparator/>
    <jca:addElement text="Using separators..." />
    <w:textBox propertyLabel="Textbox A" name="textboxA" onBlur="foo" required="true"/>
    <jca:addSeparator style="blank"/>
    <w:textBox propertyLabel="Textbox B" name="textboxB" onBlur="foo" required="true"/>
    <jca:addSeparator style="line"/>
    <w:textBox propertyLabel="Textbox C" name="textboxC" onBlur="foo" required="true"/>
    </jca:renderPropertyPanel>
```

Using renderPropertyPanel tag

*Textbox:	<input type="text"/>
*Checkbox:	<input type="checkbox"/> text on the right of a checkbox
Combobox:	<input type="text" value="first"/>
TextArea:	<input type="textarea"/>
DateInputComponent:	2011-08-03 <input type="button" value="yyyy-mm-dd"/> 10 AM <input type="button" value="00"/> CST
Button:	<input type="button" value="Click Me"/>
Radio Button:	<input checked="" type="radio"/> One <input type="radio"/> Two
Date:	2011-08-03 10:09
*Textbox A:	<input type="text"/>
*Textbox B:	<input type="text"/>
*Textbox C:	<input type="text"/>
*Textbox E:	<input type="text"/>

# 1. Create Populator Tag Library

## Create Custom Tag for Populator

### 1. Create new file on \$WT\_HOME/codebase/WEB-INF/tlds, and modify

csc.tld

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
         version="2.0">
    <description>
        Search TLD for CSC Training
    </description>
    <display-name>CSC Tag Library</display-name>
    <tlib-version>1.1</tlib-version>
    <short-name>csc</short-name>
    <uri>http://www.ptc.com/windchill/taglib/csc</uri>
    <tag>
        <description>
            Search data tag for Document
        </description>
        <name>searchPopulator</name>
        <tag-class>com.csc.common.tags.SearchDataPopulatorTag</tag-class>
        <body-content>empty</body-content>
        <attribute>
            <description>
                Name of data populator
            </description>
            <name>name</name>
            <required>true</required>
            <rteprvalue>true</rteprvalue>
        </attribute>
    </tag>
</taglib>
```

# 1. Create Populator Tag Library

## Create Custom Tag for Populator

### 2. Create populator interface class

#### SearchDataPopulator.class

```
package com.csc.common.tags;

import javax.servlet.jsp.JspContext;

public interface SearchDataPopulator {
    public static final String VERSION = "$Id: $";

    public void populateSearchCriteria(JspContext jspContext);
}
```

# 1. Create Populator Tag Library

## Create Custom Tag for Populator

### 3. Create populator tag class

SearchDataPopulatorTag.class

```
package com.csc.common.tags;
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;
import com.csc.common.tags.SearchDataPopulator;

public class SearchDataPopulatorTag extends SimpleTagSupport {
    public static final String VERSION = "$Id: $";
    private String name;

    @Override
    public void doTag() throws JspException, IOException {
        SearchDataPopulator populator = null;
        try {
            final Class<?> clas = Class.forName(name);
            populator = (SearchDataPopulator) clas.newInstance();
        } catch (final ClassNotFoundException e) { e.printStackTrace(); }
        catch (InstantiationException e) { e.printStackTrace(); }
        catch (final IllegalAccessException e) { e.printStackTrace(); }

        if (populator != null) {
            populator.populateSearchCriteria(getJspContext());
        } else {
            throw new JspException("Unable to initialize search populator:\t" + name);
        }
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

## 2. Create Search and Table

### Scenario for Search and Table

Before we understand how to create search and table, we must have one scenario like the one shown.

- Search target will be all WTPart object.
- Search criteria need to be numbers, names, and Part types.
- Number field must be required field
- Part Type must be designed ComboBox
- Result table must be show like Part list.
  - Number, Name, Small thumbnail, and so on.

The screenshot shows a software interface for searching parts. At the top, there's a search bar labeled 'Search Sample01'. Below it is a 'SEARCH CONDITION' section with fields for 'Number' (containing 'W'), 'Name' (empty), 'Part Type' (set to 'Component'), and a 'Search' button. Below this is a 'Part Table' section with a header row containing columns for 'Name', 'Version', 'Last Modified', and 'Context'. There are seven rows of data, each representing a part file: '01-52200.prt', '01-51368.cprt', '01-51230.prt', '01-51101.prt', '01-51291.prt', and '01-32150.prt'. Each row includes a small thumbnail icon and a blue information icon.

	Name	Version	Last Modified	Context
1	01-52200.prt	①	A.1 (Design)	2011-07-04 21:28 CST Drive System
2	01-51368.cprt	①	A.1 (Design)	2011-07-04 21:28 CST Drive System
3	01-51230.prt	①	A.1 (Design)	2011-07-04 21:28 CST Drive System
4	01-51101.prt	①	A.1 (Design)	2011-07-04 21:28 CST Drive System
5	01-51291.prt	①	A.1 (Design)	2011-07-04 21:28 CST Drive System
6	01-32150.prt	①	A.1 (Design)	2011-07-04 21:28 CST Drive System

## 2. Create Search and Table

### Build JSP for Criteria

#### 1. Create JSP and set basic environment

sampleSearch01.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ include file="/netmarkets/jsp/util	begin.jspf"%>
<%@ taglib prefix="mvc" uri="http://www.ptc.com/windchill/taglib/jcaMvc"%>
<%@ taglib prefix="wrap" uri="http://www.ptc.com/windchill/taglib/wrappers"%>
<%@ taglib prefix="csc" uri="http://www.ptc.com/windchill/taglib/csc"%>
<%@ taglib tagdir="/WEB-INF/tags" prefix="wctags"%>

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

#### 2. Add populator tag for get criteria data set

sampleSearch01.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ include file="/netmarkets/jsp/util	begin.jspf"%>
<%@ taglib prefix="mvc" uri="http://www.ptc.com/windchill/taglib/jcaMvc"%>
<%@ taglib prefix="wrap" uri="http://www.ptc.com/windchill/taglib/wrappers"%>
<%@ taglib prefix="csc" uri="http://www.ptc.com/windchill/taglib/csc"%>
<%@ taglib tagdir="/WEB-INF/tags" prefix="wctags"%>

<csc:searchPopulator name="com.csc.search.SampleSearchDataPopulator01" />

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

## 2. Create Search and Table

### Build JSP for Criteria

#### 3. Create Populator Class for Sample Criteria

##### SampleSearchDataPopulator01.class

```
package com.csc.search;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.jsp.JspContext;
import com.csc.common.tags.SearchDataPopulator;
public class SampleSearchDataPopulator01 implements SearchDataPopulator {
    public static final String VERSION = "$Id: $";
    @Override
    public void populateSearchCriteria(JspContext jspContext) {
        // TODO Auto-generated method stub
        try {
            // Set all of key in here
            //Sample01 - Part Type Population
            List<String> partTypeKey = new ArrayList<String>();
            List<String> partTypeDisplay = new ArrayList<String>();
            partTypeKey.add("");
            partTypeDisplay.add("");
            PartType[] partSet = PartType.getPartTypeSet();
            for( int i=0; i < partSet.length; i++) {
                partTypeKey.add(partSet[i].getStringValue().substring(partSet[i].getStringValue().lastIndexOf(".") + 1));
                partTypeDisplay.add(partSet[i].getDisplay());
            }

            jspContext.setAttribute("partType_internal_vals", partTypeKey);
            jspContext.setAttribute("partType_display_vals", partTypeDisplay);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```



## 2. Create Search and Table

### Build JSP for Criteria

#### 4. Build criteria UI in JSP page

Add the following source in “sampleSearch01.jsp”

The screenshot shows a search interface titled "Search Sample01". It has a legend "SEARCH CONDITION". Below it are four input fields: "Number" containing "W", "Name" (empty), "Part Type" (set to "Component"), and a "Search" button.

```
<b>Search Sample01</b>
<fieldset class="x-fieldset x-form-label-left" id="Visualization_and_Attributes" style="width: 1024px;">
    <legend>SEARCH CONDITION</legend>
    <table>
        <tr>
            <td scope="row" width="50" class="tableColumnHeaderfont" align="left">*Number:</td>
            <td class="tabledatafont" align="left">&nbsp;
                <wrap:textBox name="number" id="number" maxlength="30" size="10"
                    nblur="this.value=this.value.toUpperCase()" />
            </td>
            <td scope="row" width="50" class="tableColumnHeaderfont" align="left">Name:</td>
            <td class="tabledatafont" align="left">&nbsp;
                <wrap:textBox name="name" id="name" maxlength="100" size="20"/>
            </td>
            <td scope="row" width="100" class="tableColumnHeaderfont" >Part Type:</td>
            <td class="tabledatafont" align="left">
                <wrap:comboBox id="partType" name="partType" multiSelect="false" size="1"
                    displayValues="${partType_display_vals}"
                    InternalValues="${partType_internal_vals}" />
            </td>
        </tr>
        <tr>
            <td colspan="6" scope="row" class="tableColumnHeaderfont" align="right">
                <wrap:button name="search" value='Search' onclick="submitDocSearch();"/>
            </td>
        </tr>
    </table>
</fieldset>
```

## 2. Create Search and Table

### Build JSP for Criteria

#### 5. Add MVC table and JavaScript function in JSP page

Add the following source in “sampleSearch01.jsp”

```
<mvc:tableContainer compId="csc.mvc.sampleSearch01" height="500" />

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

<script>
    function submitDocSearch() {
        if (document.getElementById('number').value=="") {
            JCAAlert("Number must be inputted");
        } else {
            submitParameters();
        }
    }
    function submitParameters() {
        var number = document.getElementById('number').value;
        var name = document.getElementById('name').value;
        var partType = document.getElementById('partType').value;
        var params = {
            number : number,
            name : name,
            partType : partType
        };
        PTC.jca.table.Utils.reload('csc.mvc.sampleSearch01', params, true);
    }
</script>
```

MVC table

Recall MVC table  
using parameters

## 2. Create Search and Table

## Build MVC Table

## 6. Create MVC Class

Before creating an MVC class, you must register the MVC directory. If you do not want to register the directory, you can directly register the new MVC class.

```
10 /* @bcut11 */
11 package com.csc.mvc; 1
12
13
14 import static com.ptc.core.components.descriptor.DescriptorConstants.ColumnIdentifiers.CHANGE_STATUS_FAMILY;
15
16 /**
17 * MVC builder class for Product List table
18 */
19
20 @ComponentBuilder("csc.mvc.sampleSearch01") 2
21 public class CSCSampleSearch01 extends AbstractComponentBuilder { 3
22
23     private static final String RESOURCE = "com.ptc.carambola.carambolaResource";
24
25     /* (non-Javadoc)
26      * @see com.ptc.mvc.components.ComponentDataBuilder#buildComponentData(com.ptc.mvc.components.ComponentConfig,
27      */
28
29     @Override
30     public Object buildComponentData(ComponentConfig config, ComponentParams params) throws Exception { 4
31         String number = (String) params.getParameter("number");
32         String name = (String) params.getParameter("name");
33         String partType = (String) params.getParameter("partType");
34
35         if ( number != null && number.length() > 0 ) {
36             number = number.trim();
37         }
38
39         return null;
40     }
41 }
```

1. Set package which is registered in mvc.xml
2. Create annotation for MVC component ID
3. Set Java Class name which extends to “AbstractComponentBuilder”
4. Override “buildComponentData” whose function will be programmed to get parameters by data input.
5. Override “buildComponentConfig” function which will work to design UI.

```
7  
8 @Override  
9 public ComponentConfig buildComponentConfig(ComponentParams params) throws WTEException {  
10     ComponentConfigFactory factory = getComponentConfigFactory();  
11     ClientMessageSource messageSource = getMessageSource(RESOURCE);  
12  
13     TableConfig table = factory.newTableConfig();  
14     table.setLabel(messageSource.getMessage("PART_TABLE_LABEL"));  
15 }
```

## Other Table Features

- **Paging**

- Paging concept removed in X20
- DataSource enabled table: client will load till the “size limit.”
- Non-DataSource table: client will load all the data.

- **Table Size Limit**

- Preference driven (visible in all Organization and Site Utilities Preference Managers )
- Value range: 1 to 100,000
- OOTB value is 2000

Tables		
Column descriptions in tooltips	No	Adds column descriptions to the column name in the tooltip for the table column header.
Size Limit	2000	Specifies the maximum number of rows to be displayed in a table or tree.

- DataBuilder to extend AbstractJcaDataSourceComponentDataBuilder
  - Override getComponentLimit(ComponentConfig config, ComponentParams params)

## 2. JCA MVC Table Features — Sorting

- **Sorting**

- Non-DataSource and DataSource.SYNCHRONOUS table
  - Table view sort criteria is applied in server.
  - User sort applied in data that is available in the client only.
- DataSource.ASYNCHRONOUS table
  - By default, Table view sort criteria is applied in client.
  - DataBuilder can send sorted data from the server and inform the Infrastructure
    - ComponentResultProcessor. [setPresorted\(boolean preSorted\)](#)
- How to specify client sort function for a column?
  - ColumnConfig. [setCompareJsFunction\(String compareJsFunction\)](#)

```
//Define the column
ColumnConfig col = factory.newColumnConfig("Quantity", columnLabel, true);
col.setNeed("quantityWithUnits");
col.setDataUtilityId("part.report.quantity");
//set the compareJsFunction
col.setCompareJsFunction("prc.quantityComparator");

// javascript function available in the view
quantityComparator : function(cellA, cellB, rowA, rowB) {
var comparableA = jsca.getComparable(cellA);
var comparableB = jsca.getComparable(cellB);
-----
}
```

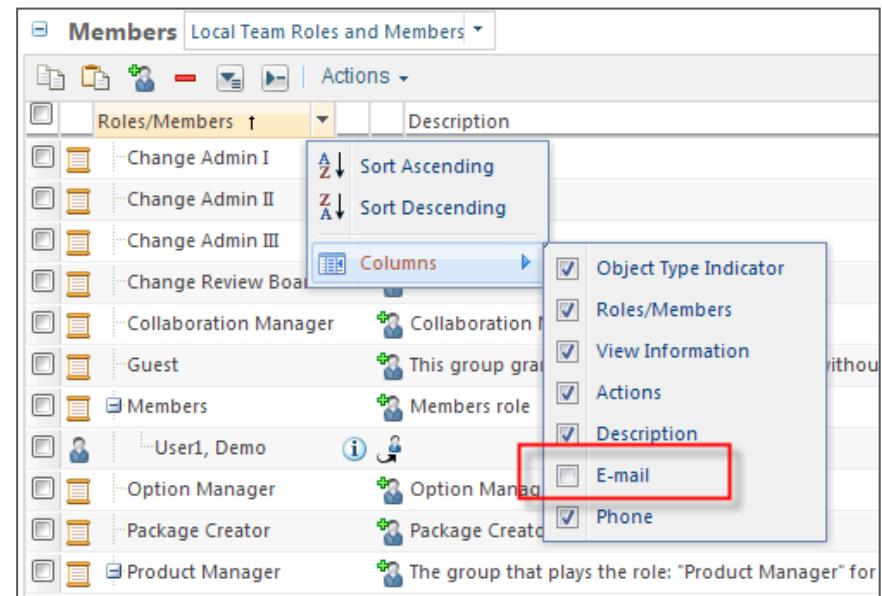
### 3. JCA MVC Table Features — Columns

- **Hidden Column**

- In X12, users cannot make the hidden column visible in the client; whereas in X20, it's possible to view the hidden column.
- Hidden columns are not displayed in the table OOTB. They can be made visible from the menu available in the column header.
- `ColumnConfig.setHidden(boolean hidden)`

- **Data Store Column**

- Data Store columns cannot be displayed in the table
- They are available in the client (store).
- Mainly used to send some custom values for the row.
- `ColumnConfig.setDataStoreOnly(boolean forDataStoreOnly)`



## 4. JCA MVC Table Features — Strikethrough

### Strikethrough Rows

<input type="checkbox"/>	WHEEL_ASSEMBLY	C000002		A.1 (Design)	2010-11-26 06:47 CST	Bicycle2
<input checked="" type="checkbox"/>	BICYCLE2	BICYCLE2		A.1 (Design)	2010-11-26 06:47 CST	Bicycle2
<input type="checkbox"/>	Worldcar	PV000003		A.1 (Design)	2010-11-26 06:46 CST	ProductView Demo
<input type="checkbox"/>	ProEngine	PV000002		A.1 (Design)	2010-11-26 06:46 CST	ProductView Demo
<input type="checkbox"/>	ProductView Demo	PV000001		A.1 (Design)	2010-11-26 06:46 CST	ProductView Demo

- `JcaTableConfig.setStrikeThroughColumn(ColumnConfig column)`

```
// strikeThrough column handling : "strikeThroughRow" should be the ID to which the default DataUtility is
// mapped to
ColumnConfig strikeThroughColumn = factory.newColumnConfig("strikeThroughRow", false);
strikeThroughColumn.setDataStoreOnly(true);
// map it to endItem attribute
strikeThroughColumn.setNeed("endItem");
// add the column
table.addComponent(strikeThroughColumn);
//mark the column as StrikeThroughColumn
table.setStrikeThroughColumn(strikeThroughColumn);
```

## 5. JCA MVC Table Features — Non-Selectable

### Non-Selectable Rows

	WHEEL_ASSEMBLY	C000002		A.1 (Design)	2010-11-26 06:47 CST	Bicycle2
	BICYCLE2	BICYCLE2		A.1 (Design)	2010-11-26 06:47 CST	Bicycle2
	Worldcar	PV000003		A.1 (Design)	2010-11-26 06:46 CST	ProductView Demo
	ProEngine	PV000002		A.1 (Design)	2010-11-26 06:46 CST	ProductView Demo
	ProductView Demo	PV000001		A.1 (Design)	2010-11-26 06:46 CST	ProductView Demo

- `JcaTableConfig.setNonSelectableColumn(ColumnConfig column)`

```
// nonSelectable column handling : "nonSelectableColumn" should be the ID to which the default DataUtility  
// is mapped to  
ColumnConfig nonSelectableColumn = factory.newColumnConfig  
        (DescriptorConstants.ColumnIdentifiers.NON_SELECTABLE_COLUMN, false);  
nonSelectableColumn.setDataStoreOnly(true);  
// map it to endItem attribute  
nonSelectableColumn.setNeed("endItem");  
// add the column  
table.addComponent(nonSelectableColumn);  
//mark the column as NonSelectableColumn  
table.setNonSelectableColumn(nonSelectableColumn);
```

### Find in Table/Search in Table

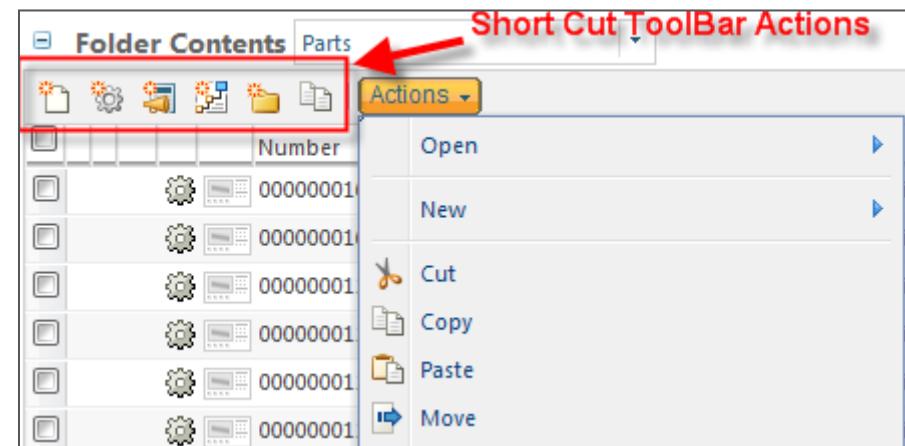


- Configured to disable or find in client or find in client and server
- `JcaTableConfig.setFindInTableMode(FindInTableMode findInTableMode)`
- Default is `FindInTableMode.CLIENT_AND_SERVER`

## 7. JCA MVC Table Features — ToolBar Actions

### Table ToolBar Actions

```
<model name="folderbrowser_toolbar_actions">
<submodel name="folderbrowser_toolbar_open_submenu"/> <action
name="separator" type="separator"/>
<submodel name="folderbrowser_toolbar_new_submenu"/> <action
name="separator" type="separator"/>
//following actions are not shortcut
<action name="list_cut" type="object"/>
<action name="pasteAsCopy" type="saveas"/>
---
</model>
```



```
<model name="folderbrowser_toolbar_new_submenu"
resourceBundle="com.ptc.windchill.enterprise.folder.FolderActionResource
">
//mark the following actions as shortcut
<action name="create" type="document" shortcut="true"/>
<action name="createPartWizard" type="part" shortcut="true"/>
---
</model>
```

## 8. JCA MVC Table Client-Side Events

- Some major client events fired during table render/data processing.

Event name	Target object	Event scenario
dataChanged	Ext.data.store	Fired when a data chunk is received at client.
dataSourceComplete	Ext.data.store	Fired when the last data chunk is received at client.
refresh	Ext.grid.GridView	Fired when the grid view refreshes to display the additional row data.
render	Ext.grid.GridPanel	Fired when Ext-JS finishes the grid rendering.
renderTableStart	Ext.grid.GridPanel	Fired when the infrastructure starts rendering of the table.
renderTableEnd	Ext.grid.GridPanel	Fired when the infrastructure ends the rendering of the table.
DSLoadingInterupted	Ext.data.store	Fired when table has partial data.

## 9. Registering Listeners

### In your view JSP

```
<%-- register the listeners --%>
<script type="text/javascript">
    PTC.onReady(function() {
        Ext.ComponentMgr.onAvailable(<table id>, function(){
            //register on renderTableEnd
            this.on("renderTableEnd", <function>);
            //register on dataSourceComplete
            this.getStore().on("dataSourceComplete", <function>);
        });
    });
</script>
<%-- render the table --%>
```

# Lab Session — Exercise 1

Create a page with one MVC JCA Table which contains the list of parts

The screenshot shows the Windchill 10.0 interface. On the left, the 'Customization' pane is open, displaying a list of available customizations. The 'Exercise 1' customization is highlighted with a red box and a red arrow points from it towards the main content area. The main content area displays the 'Part Table' with a list of parts. The table has columns for Name, Version, Last Modified, and Context. A row for 'Module Subassembly, Power Generation' is selected, indicated by a yellow background.

Name	Version	Last Modified	Context
Wire Harness, Communications	A.1 (Design)	2010-11-10 12:39 IST	Power System
Module Subassembly, Power Generation	A.1 (Design)	2010-11-10 12:39 IST	Power System
Wiring Harness, Battery Interconnect	A.1 (Design)	2010-11-10 12:39 IST	Power System
Enclosure Weldment Assembly	A.1 (Design)	2010-11-10 12:39 IST	Power System
Kit, Energy Delivery, 5U120FG w/Batteries	A.1 (Design)	2010-11-10 12:39 IST	Power System
Ultra Cap, 16V	A.1 (Design)	2010-11-10 12:39 IST	Power System
Template, Installation	A.1 (Design)	2010-11-10 12:39 IST	Power System
Wiring Harness, Thermal Management	A.1 (Design)	2010-11-10 12:39 IST	Power System
Switch, Mushroom, Weatherproof	A.1 (Design)	2010-11-10 12:39 IST	Power System
Kit, Energy Delivery, 5U48FG W/ Batteries	A.1 (Design)	2010-11-10 12:39 IST	Power System
Kit, Energy Delivery, 5U108FG w/Batteries	A.1 (Design)	2010-11-10 12:39 IST	Power System
Enclosure Base Assembly	A.1 (Design)	2010-11-10 12:39 IST	Power System
Converter, DC Voltage	A.1 (Design)	2010-11-10 12:39 IST	Power System
Kit, Energy Delivery, 5T48 w/Batteries	A.1 (Design)	2010-11-10 12:39 IST	Power System
Cable Assembly, Ultracap Interconnect	A.1 (Design)	2010-11-10 12:39 IST	Power System
Front Door	A.1 (Design)	2010-11-10 12:39 IST	Power System
Module Subassembly, Water Management	A.1 (Design)	2010-11-10 12:39 IST	Power System
01-52107.prt	A.1 (Design)	2010-11-10 12:21 IST	Drive System
01-2_cam_exhaust.prt	A.1 (Design)	2010-11-10 12:21 IST	Drive System
01-51368a.prt	A.1 (Design)	2010-11-10 12:21 IST	Drive System
01-72530.prt	A.1 (Design)	2010-11-10 12:21 IST	Drive System
01-51294.prt	A.1 (Design)	2010-11-10 12:21 IST	Drive System
valve_exhaust.prt	A.1 (Design)	2010-11-10 12:21 IST	Drive System

Configure an action that generates an MVC URL that builds a table with componentId “exercise1”

- Define the new action in WT\_HOME/codebase/config/actions/custom-actions.xml
- Find the definition of “Customization” action model
- Override it in WT\_HOME/codebase/config/actions/custom-actionModels.xml
- Load the updated action configurations.

## Create a Table builder

- Base package com.ptc.mvc.builders
- Extend AbstractComponentBuilder
- Annotate it with @ComponentBuilder
- Provide the ComponentConfig
- Provide the Data
- Compile the class and restart the Method Server

# Exercise 1: Code Snippets

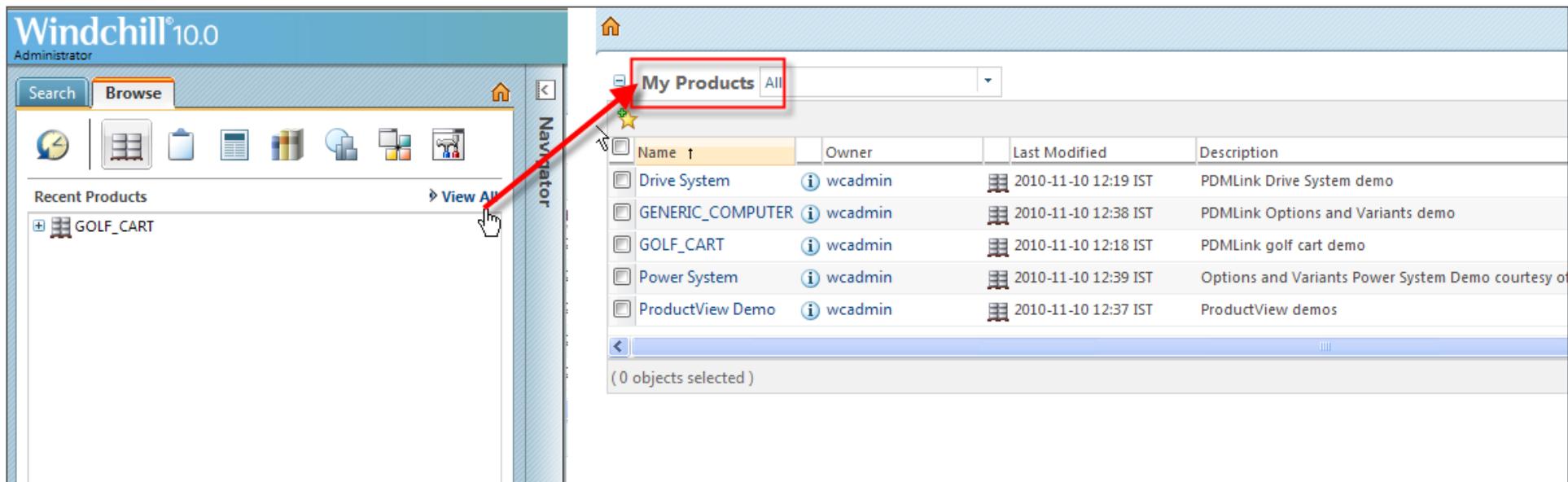
```
private static final String RESOURCE = "com.ptc.carambola.carambolaResource";  
 @Override  
 public ComponentConfig buildComponentConfig(ComponentParams params) throws WTException {  
     ComponentConfigFactory factory = getComponentConfigFactory();  
     ClientMessageSource messageSource = getMessageSource(RESOURCE);  
     TableConfig table = factory.newTableConfig();  
     table.setLabel(messageSource.getMessage("PART_TABLE_LABEL"));  
     table.setSelectable(true);  
     // set the actionModel that comes in the TableToolBar  
     table.setActionModel("mvc_tables_toolbar");  
     // add columns  
     table.addComponent(factory.newColumnConfig(ICON, true));  
     table.addComponent(factory.newColumnConfig(NAME, true));  
     table.addComponent(factory.newColumnConfig(FORMAT_ICON, false));  
     table.addComponent(factory.newColumnConfig(ORG_ID, false));  
     table.addComponent(factory.newColumnConfig(INFO_ACTION, false));  
     ColumnConfig nmActionsCol = factory.newColumnConfig(NM_ACTIONS, false);  
     // specify the actionModel for the action column  
     ((JcaColumnConfig) nmActionsCol).setActionModel("CustEx_table_row_actions");  
     table.addComponent(nmActionsCol);  
     return table;  
 }
```

# Exercise 1: Code Snippets

@Override

```
public QueryResult buildComponentData(ComponentConfig config, ComponentParams params) throws WTException {  
    QuerySpec qs = new QuerySpec(WTPart.class);  
    qs.setQueryLimit(10000);  
    return PersistenceHelper.manager.find((StatementSpec) qs);  
}
```

Override the OOTB builder for Product List table to show its label as “My Products”



- Find the builder for the current Product List Table
- Override the OOTB Builder
  - Base package com.ptc.mvc.builders
  - Annotate it with `@OverrideComponentBuilder`
  - Compile the class and restart the Method Server

# Lab Session: Exercise 3

- Create a page with multiple MVC JCA Components (Property Panel and Table)

The screenshot shows the Windchill 10.0 interface with a custom page. On the left, the 'Customization' sidebar lists various components: Table, Tree, Picker, Property Panel, Attribute Panel, Two-Pane Panels, Tools, MVC, MVC Search Demo, Example Information Page, Test Clients, Other Examples, Exercise 1, and Exercise 3. A red arrow points from the 'Exercise 3' link to the right pane. The right pane displays a 'Property Panel' section with company information and a 'Table' section titled 'Part Table' containing a list of parts with columns for Name, Version, Last Modified, and Context.

**Property Panel**

Name: Parametric Technology Corp.  
Address: 3785 Pheasant Ridge Drive NE  
Blaine MN 55449  
Country: USA  
Phone: 763-957-8000   
Fax: 763-957-8001

**Table**

**Part Table**

	Name	Version	Last Modified	Context
<input type="checkbox"/>	Wire Harness, Communications	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Module Subassembly, Power Generation	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Wiring Harness, Battery Interconnect	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Enclosure Weldment Assembly	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Kit, Energy Delivery, 5U120FG w/Batteries	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Ultra Cap, 16V	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Template, Installation	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Wiring Harness, Thermal Management	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Switch, Mushroom, Weatherproof	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Kit, Energy Delivery, 5U48FG W/ Batteries	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Kit, Energy Delivery, 5U108FG w/Batteries	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Enclosure Base Assembly	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Converter, DC Voltage	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Kit, Energy Delivery, 5T48 w/Batteries	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Cable Assembly, Ultracap Interconnect	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Front Door	A.1 (Design)	2010-11-10 12:39 IST	Power System
<input type="checkbox"/>	Module Subassembly, Water Management	A.1 (Design)	2010-11-10 12:39 IST	Power System

(0 objects selected) Next

- Configure an action that generates an MVC URL that builds a table with componentId “exercise3”
  - Define the new action in WT\_HOME/codebase/config/actions/custom-actions.xml
  - Find the definition of “Customization” action model
  - Override it in WT\_HOME/codebase/config/actions/custom-actionModels.xml
  - Load the updated action configurations
- Create the builder
  - Base package com.ptc.mvc.builders
  - Extend AbstractComponentBuilder
  - Annotate it with @ComponentBuilder
  - Provide the MultiComponentConfig
    - Add Property Panel
    - Add the Table created in exercise1
    - Set the custom view (/custom/exercise3.jsp)
  - Provide the Data for Property Panel
  - Compile the class and restart the Method Server

# Exercise 3: Code Snippets

```
@Override  
public ComponentConfig buildComponentConfig(ComponentParams params) throws WTEException {  
    ComponentConfigFactory factory = getComponentConfigFactory();  
    PropertyPanelConfig propertyPanelConfig = factory.newPropertyPanelConfig();  
    propertyPanelConfig.setId("exercise3PropertyPanel");  
    // add properties  
    PropertyConfig nameConfig = factory.newPropertyConfig();  
    nameConfig.setId("name");  
    propertyPanelConfig.addComponent(nameConfig);  
    PropertyConfig address1Config = factory.newPropertyConfig();  
    address1Config.setId("address1");  
    address1Config.setLabel("Address");  
    propertyPanelConfig.addComponent(address1Config);  
    MultiComponentConfig mconfig = new MultiComponentConfig();  
    // add the PropertyConfig  
    mconfig.addComponent(propertyPanelConfig);  
    // add a nested component  
    mconfig.addNestedComponent("exercise1");  
    // set the view  
    mconfig.setView("/custom/exercise3.jsp");  
    return mconfig;  
}
```

# Exercise 3: Code Snippets

@Override

```
public Object buildComponentData(ComponentConfig config, ComponentParams params) throws WTException {  
    // data for the PropertyConfig  
    Map<String, String> address = new HashMap<String, String>();  
    address.put("name", "Parametric Technology Corp.");  
    address.put("address1", "3785 Pheasant Ridge Drive NE");  
    address.put("address2", "Blaine MN 55449");  
    address.put("country", "USA");  
    address.put("phone", "763-957-8000");  
    address.put("fax", "763-957-8001");  
    return address;  
}
```

## Exercise 3: View

```
// content of lab/exercise3.jsp
<%@ taglib uri="http://www.ptc.com/windchill/taglib/components" prefix="jca"%>
<%@ taglib uri="http://www.ptc.com/windchill/taglib/jcaMvc" prefix="mvc"%>
```

- %@include file="/netmarkets/jsp/util/begin\_comp.jspf%"  
<%-- render the property panel --%>  
<jca:renderPropertyPanel componentDef="\${exercise3PropertyPanel}" />  
<%-- register the listeners --%>  
<script type="text/javascript">  
 PTC.onReady(function() {  
 Ext.ComponentMgr.onAvailable('exercise1', function(){  
 //register on renderTableEnd  
 this.on("renderTableEnd", function (){  
 alert("Data after renderTableEnd : " + this.getStore().getCount());  
 });  
 //register on dataSourceComplete  
 this.getStore().on("dataSourceComplete", function (){  
 alert("Data after dataSourceComplete : " + this.getCount());  
 });  
 });  
 });  
</script>

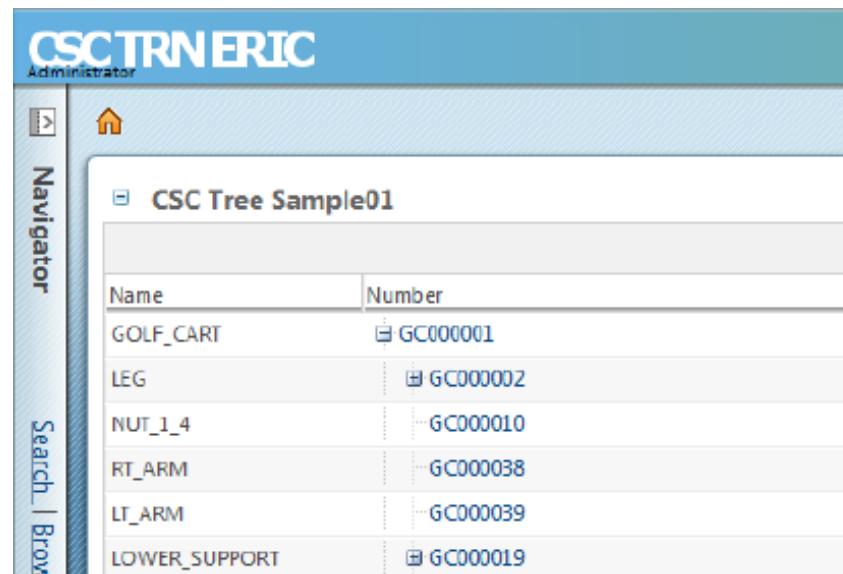
```
<%-- render the table --%>
<mvc:table complId="exercise1"/>
<%@ include file="/netmarkets/jsp/util/end_comp.jspf"%>
```

Tree

## Customization Process

Tree customization needs two-way implementation.

- Create Tree Builder similar to Table builder
- Table builder needs to implement some interface for tree synchronization
- Create Tree Handler – For creating roots and child nodes
- Optional: Register custom MVC library (if you have already set it then you need not do it again.)

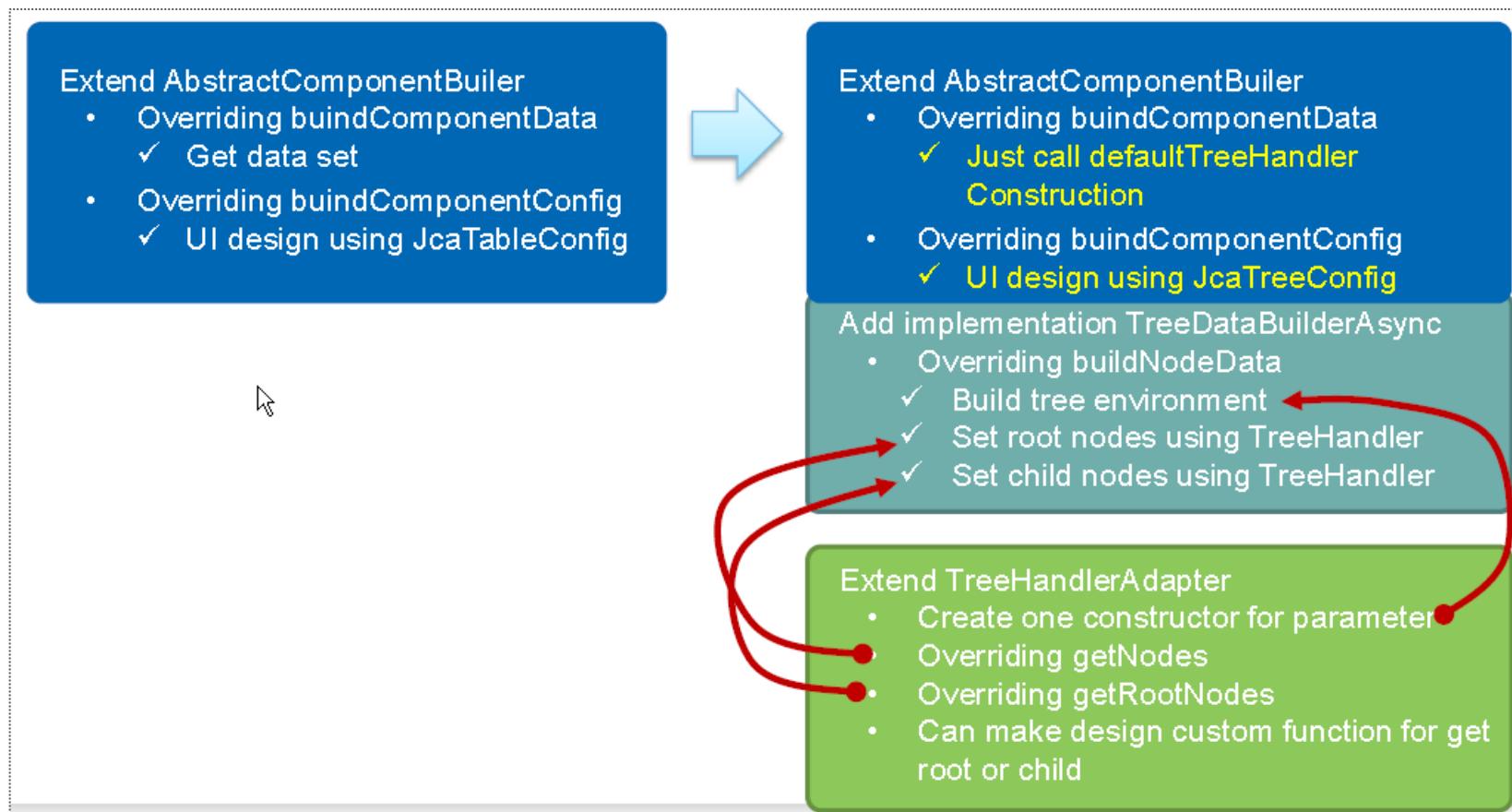


The screenshot shows a software interface titled "CSC TRN ERIC" with a user name "Administrator". On the left, there's a vertical toolbar with icons for Home, Navigator, and Search/Browse. The main area has a title bar "CSC Tree Sample01". Below it is a table with columns "Name" and "Number". The data rows are:

Name	Number
GOLF_CART	GC000001
LEG	GC000002
NUT_1_4	GC000010
RT_ARM	GC000038
LT_ARM	GC000039
LOWER_SUPPORT	GC000019

## Component Architecture

Creating component builder is similar to that of table component. In addition, tree component needs Tree handler class. The following is the comparison of architecture of class.



# 1. Create MVC Component for Tree Table

## Build MVC Tree Table

### 1. Create MVC Class

Before creating an MVC class, you must register the MVC directory. If you do not want to register the directory, you can directly register the new MVC class.

1. Set package which is registered in mvc.xml
2. Create annotation for MVC component ID
3. Set Java Class name which extends to “AbstractComponentBuilder”
4. Implement “TreeDataBuilderAsync” interface
5. Override “buildComponentData” whose function will be programmed to get parameters by data input.
6. Just return TreeHandler constructor function
7. Override “buildComponentConfig” function which will work to design UI.

The diagram illustrates the steps for creating an MVC component for a tree table. It shows a flow from the 'Extend AbstractComponentBuilder' step to the 'Add Implementation: TreeDataBuilderAsync' step, and finally to the 'Extend TreeHandlerAdapter' step. Numbered circles (1-7) correspond to the steps listed in the text above. A callout points to the 'Using JcaTreeConfig for tree UI design' code in the buildComponentData method, and another callout points to the 'If using TreeDataBuilderAdaptor, you must set DataSourceMode' code in the buildComponentConfig method.

```
1 package com.csc.mvc; 2 3 import java.util.ArrayList; 4 5 @ComponentBuilder("csc.mvc.sampleTree01") 6 public class CSCSampleTree01 extends AbstractComponentBuilder implements TreeDataBuilderAsync { 7     CSCSampleTreeHandler01 sampleHandler; 8 9     @Override 10    public Object buildComponentData(ComponentConfig arg0, ComponentParams arg1) throws Exception { 11        // TODO Auto-generated method stub 12        return new CSCSampleTreeHandler01(); 13    } 14 15    @Override 16    public ComponentConfig buildComponentConfig(ComponentParams arg0) throws WTEException { 17        // TODO Auto-generated method stub 18        //Create TreeConfig 19        ComponentConfigFactory factory = getComponentConfigFactory(); 20        JcaTreeConfig tree = (JcaTreeConfig) factory.newTreeConfig(); 21 22        // Need to set DataSourceMode explicitly to DataSourceMode.ASYNCHRONOUS 23        tree.setDataSourceMode(DataSourceMode.ASYNCHRONOUS); 24    } 25 }
```

Using JcaTreeConfig for tree UI design

If using TreeDataBuilderAdaptor, you must set DataSourceMode.

# 1. Create MVC Component for Tree Table

## Build MVC Tree Table

### 1. Create MVC Class

Before creating an MVC class, you must register the MVC directory. If you do not want to register the directory, you can directly register new a MVC class.

The diagram illustrates the inheritance path for the MVC component. It shows a vertical hierarchy of classes: AbstractComponentBuilder at the top, followed by buildComponentData, buildComponentData, buildComponentConfig, buildComponentConfig, and finally TreeTableComponentBuilder at the bottom. A red box highlights the 'Extend AbstractComponentBuilder' section, which includes instructions to override buildComponentData and buildComponentConfig. A green box highlights the 'Extend TreeTableComponentBuilder' section, which includes instructions to add implementation for buildDatabaseElements, overriding buildHandler, and setting nodes using TreeHandler. A blue box highlights the 'Add implementation' section, which includes instructions to implement TreeTableComponentBuilder, overriding buildDatabaseElements, overriding buildHandler, and setting nodes using TreeHandler. A yellow box highlights the 'Create Handler' section, which includes instructions to create a new constructor for parameters, overriding getNodes, overriding getParameters, and can handle single column function to get root or child.

```
55 @Override
56 public void buildNodeData(Object node, ComponentResultProcessor resultProcessor) throws Exception {
57     if (node == TreeNode.RootNode) { // special case for root nodes
58         sampleHandler = new CSCSampleTreeHandler01(resultProcessor.getParams());
59         resultProcessor.addElements(sampleHandler.getRootNodes()); // should fetch rootNodes and to
60     } else { // case for any other node
61         List nodeList = new ArrayList();
62         nodeList.add(node);
63         Map<Object, List> map = sampleHandler.getNodes(nodeList);
64         Set keySet = map.keySet();
65         for (Object key : keySet) {
66             resultProcessor.addElements(map.get(key));
67         }
68     }
69 }
```

1. **Overriding “buindNodeData”:** ComponentResultProcessor parameter can get opener’s parameters.
2. **Constructor Custom tree handler:** This sample’s constructor needs to opener parameters, so sample tree handler class has a constructor for initializing and getting parameters)
3. **Other source line is standard source code for get node data, so you just copy sample source codes**

# 1. Create MVC Component for Tree Table

PTC®

## Build MVC Tree Table

### 2. Create Tree Handler

1. Create custom TreeHandler extended from “TreeHandlerAdapter”
2. Create default constructor (Empty constructor)
3. Create custom constructor (Sample is for get parameter from opener)

The diagram illustrates the inheritance path from `AbstractComponentData` to `TreeHandlerAdapter`. It shows three levels of inheritance:

- `AbstractComponentData`:
  - Overriding `bindComponentData`
  - Get data
  - Override `NmCommandConfig`
  - UI design using `availableCells`
- `TreeHandlerAdapter`:
  - Override `bindComponentData`
  - Just return `TreeHandler` constructor
  - Override `bindAvailableCells`
  - Get root nodes using `TreeHandler`
  - Get child nodes using `TreeHandler`
- `CSCSampleTreeHandler01`:
  - Add implementation: `TreeHandlerAdapter`
  - Override `bindAvailableCells`
  - Get root nodes using `TreeHandler`
  - Get child nodes using `TreeHandler`

```
1 package com.csc.mvc;
2
3 import java.util.ArrayList;
4
5 public class CSCSampleTreeHandler01 extends TreeHandlerAdapter { 1
6     private NmCommandBean cb = null;
7     String number = null;
8     String name = null;
9     String partType = null;
10
11     public CSCSampleTreeHandler01() { 2
12     }
13
14     *****
15     public CSCSampleTreeHandler01(ComponentParams params) throws WTEException { 3
16         cb = ((JcaComponentParams) params).getHelperBean().getNmCommandBean();
17     }
18     *****
19
20     public CSCSampleTreeHandler01(ComponentParams params) throws WTEException {
21         //String number = (String) params.getParameter("number");
22         //String name = (String) params.getParameter("name");
23         //String partType = (String) params.getParameter("partType");
24         number = "GC000001";
25     }
26 }
```

It is other sample when the parameter is  
“NmCommandBean”

# 1. Create MVC Component for Tree Table

PTC®

## Build MVC Tree Table

### 2. Create Tree Handler

1. Overriding for get child nodes
2. Overriding for get root nodes

The diagram illustrates the process of creating an MVC component for a tree table. It shows a flow from 'Create MVC Component' through 'Override buildComponentData' and 'Override buildComponentConfig' to 'Add Implementation'. A red box highlights the 'Override buildComponentData' step, which includes 'Overriding buildComponentData' and 'Overriding buildComponentConfig' with 'UI design using JavaTableConfig'. A yellow circle labeled '1' points to the 'getNodes' method in the code. Another yellow circle labeled '2' points to the 'getRootNodes' method.

```
51 @Override
52 public Map<Object, List> getNode(List parents) throws WTEException { 1
53     // TODO Auto-generated method stub
54     WTPartStandardConfigSpec standardConfigSpec = WTPartStandardConfigSpec.newWTPartSta
55     WTPartConfigSpec configSpec = WTPartConfigSpec.newWTPartConfigSpec(standardConfigSp
56
57     Map<Object, List> result = new HashMap<Object, List>();
58
59     Persistable[][][] all_children = WTPartHelper.service.getUsesWTParts(new WTA
60
61     for ( ListIterator i = parents.listIterator(); i.hasNext(); ) {
62         WTPart parent = (WTPart)i.next();
63
64         Persistable[] oneParentNode = all_children[i.previousIndex()];
65         if ( oneParentNode == null ) {
66             continue;
67         } else {
68             List children = new ArrayList(oneParentNode.length);
69             for( int j=0; j < oneParentNode.length; j++ ) {
70                 children.add(oneParentNode[j][1]);
71             }
72             result.put(parent, children);
73         }
74     }
75     return result;
76 }
77
78 @Override
79 public List<Object> getRootNodes() throws WTEException { 2
80     // TODO Auto-generated method stub
81     List<Object> resultList = new ArrayList<Object>();
82
83     if ( number != null && number.length() > 0 ) {
84         number = number.trim();
85     } else {
86         return resultList;
87     }
88 }
```

## 2. Exercise: Search and Tree result

### Scenario for Search and Tree Table

Before we understand how to create search and tree table, we must have the scenario shown here.

- Search target will be all WTPart object.
- Search criteria need to be numbers, names and Part Types.
- Number field must be required field
- Part Type must be designed ComboBox
- Result tree table must be show like Part list.
  - Number, Name, Small thumbnail, and so on.

The screenshot shows the CSC TRN ERIC software interface. At the top, there is a navigation bar with links for Home, Administer, and Help. Below the navigation bar is a search bar with fields for 'Number' (containing 'GC000001'), 'Name', 'Part Type', and a 'Search' button. To the right of the search bar is a 'Recently Accessed' dropdown menu. The main area is titled 'CSC Tree Sample01' and displays a tree view of objects. A message '(83 objects)' is shown above the tree. The tree structure includes nodes for 'GC000001' (GOLF\_CART), 'GC000002' (LEG), 'GC000016' (ACTUATOR\_LOCK), 'GC000017' (BOLT\_1\_8), 'GC000003' (LEFT\_ACTUATOR), 'GC000008' (AXLE\_FASTENER), 'GC000007' (AXLE\_LATCH), 'GC000009' (BOLT\_1\_4), 'GC000006' (LOWER\_LEFT\_ACTUATOR), and 'GC000004' (LOWER\_LEFT\_ARM). Below the tree, a table lists detailed information for each object, including 'Number', 'Name', 'Version', 'Last Modified', and 'Context'. A 'Find in tree' button is located at the top right of the table area.

Number	Name	Version	Last Modified	Context
GC000001	GOLF_CART	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000002	LEG	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000016	ACTUATOR_LOCK	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000017	BOLT_1_8	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000003	LEFT_ACTUATOR	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000008	AXLE_FASTENER	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000007	AXLE_LATCH	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000009	BOLT_1_4	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000006	LOWER_LEFT_ACTUATOR	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART
GC000004	LOWER_LEFT_ARM	A1 (Design)	2011-07-04 21:27 CST	GOLF_CART

## 2. Create Search and Table

### Build JSP for Criteria

#### 1. Create JSP and set basic environment

- sampleSearch01\_tree.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ include file="/netmarkets/jsp/util/begin.jspf"%>
<%@ taglib prefix="mvc" uri="http://www.ptc.com/windchill/taglib/jcaMvc"%>
<%@ taglib prefix="wrap" uri="http://www.ptc.com/windchill/taglib/wrappers"%>
<%@ taglib prefix="csc" uri="http://www.ptc.com/windchill/taglib/csc"%>
<%@ taglib tagdir="/WEB-INF/tags" prefix="wctags"%>

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

#### 2. Add populator tag for get criteria data set

- sampleSearch01\_tree.jsp

```
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ include file="/netmarkets/jsp/util/begin.jspf"%>
<%@ taglib prefix="mvc" uri="http://www.ptc.com/windchill/taglib/jcaMvc"%>
<%@ taglib prefix="wrap" uri="http://www.ptc.com/windchill/taglib/wrappers"%>
<%@ taglib prefix="csc" uri="http://www.ptc.com/windchill/taglib/csc"%>
<%@ taglib tagdir="/WEB-INF/tags" prefix="wctags"%>

<csc:searchPopulator name="com.csc.search.SampleSearchDataPopulator01" />

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

## 2. Create Search and Table

### Build JSP for Criteria

Part Type:	Component
------------	-----------

#### 3. Create populator class for sample criteria

- sampleSearchDataPopulator01.class

```
package com.csc.search;
import java.util.ArrayList;
import java.util.List;
import javax.servlet.jsp.JspContext;
import com.csc.common.tags.SearchDataPopulator;
public class SampleSearchDataPopulator01 implements SearchDataPopulator {
    public static final String VERSION = "$Id: $";
    @Override
    public void populateSearchCriteria(JspContext jspContext) {
        // TODO Auto-generated method stub
        try {
            // Set all of key in here
            //Sample01 - Part Type Population
            List<String> partTypeKey = new ArrayList<String>();
            List<String> partTypeDisplay = new ArrayList<String>();
            partTypeKey.add("");
            partTypeDisplay.add("");
            PartType[] partSet = PartType.getPartTypeSet();
            for( int i=0; i < partSet.length; i++) {
                partTypeKey.add(partSet[i].getStringValue().substring(partSet[i].getStringValue().lastIndexOf(".") + 1));
                partTypeDisplay.add(partSet[i].getDisplay());
            }

            jspContext.setAttribute("partType_internal_vals", partTypeKey);
            jspContext.setAttribute("partType_display_vals", partTypeDisplay);
        } catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

## 2. Create Search and Table

### Build JSP for Criteria

#### 4. Build criteria UI in JSP page

- Add the following source in “sampleSearch01\_tree.jsp”

```
<b>Search Sample01</b>
<fieldset class="x-fieldset x-form-label-left" id="Visualization_and_Attributes" style="width: 1024px;">
    <legend>SEARCH CONDITION</legend>
    <table>
        <tr>
            <td scope="row" width="50" class="tableColumnHeaderfont" align="left">*Number:</td>
            <td class="tabledatafont" align="left">&nbsp;
                <wrap:textBox name="number" id="number" 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">&nbsp;
                <wrap:textBox name="name" id="name" maxLength="100" size="20"/>
            </td>
            <td scope="row" width="100" class="tableColumnHeaderfont" >Part Type:</td>
            <td class="tabledatafont" align="left">
                <wrap:comboBox id="partType" name="partType" multiSelect="false" size="1"
                    displayValues="${partType_display_vals}"
                    internalValues="${partType_internal_vals}" />
            </td>
        </tr>
        <tr>
            <td colspan="6" scope="row" class="tableColumnHeaderfont" align="right">
                <wrap:button name="search" value='Search' onclick="submitDocSearch();"/>
            </td>
        </tr>
    </table>
</fieldset>
```

## 2. Create Search and Table

### Build JSP for Criteria

#### 5. Add MVC tree table and JavaScript function in JSP page

- Add the following source in “sampleSearch01\_tree.jsp”

```
<mvc:tableContainer compId="csc.mvc.sampleSearch01" height="500" />
<%@ include file="/netmarkets/jsp/util/end.jspf"%>

<script>
    function submitDocSearch() {
        if (document.getElementById('number').value=="") {
            JCAAlert("Number must be inputted");
        } else {
            submitParameters();
        }
    }
    function submitParameters() {
        var number = document.getElementById('number').value;
        var name = document.getElementById('name').value;
        var partType = document.getElementById('partType').value;
        var params = {
            number : number,
            name : name,
            partType : partType
        };
        PTC.jca.table.Utils.reload('csc.mvc.sampleSearch01', params, true);
    }
</script>
```

Call MVC tree table

Recall MVC tree  
table using  
parameters

## 2. Exercise: Search and Tree Result

### Build MVC Tree Table

#### 5. Create MVC Class

- Before creating MVC class, you must register MVC directory.  
If you do not want to register directory, you can directly register new MVC class.

1. Set package which is registered in mvc.xml
2. Create annotation for MVC component ID
3. Set Java Class and extend “AbstractComponentBuilder”
4. Implement “TreeDataBuilderAsync” interface
5. Override “buildComponentData” which will be programmed to get data by input parameters.
6. Set return for TreeHandler constructor
7. Override “buildComponentConfig” function which will work to design UI.

```
1 package com.csc.mvc; 1
2
3@import static com.ptc.core.components.descriptor.DescriptorConstants.ColumnIdentifiers.CONTAINER_NAM
32
33 @ComponentBuilder("csc.mvc.sampleTree01") 2
34 public class CSCSampleTree01 extends AbstractComponentBuilder implements TreeDataBuilderAsync {
35     CSCSampleTreeHandler01 sampleHandler; 3
36
37     @Override 4
38     public Object buildComponentData(ComponentConfig arg0, ComponentParams arg1) throws Exception { 5
39         // TODO Auto-generated method stub
40         return new CSCSampleTreeHandler01(); 6
41     }
42
43     @Override 7
44     public ComponentConfig buildComponentConfig(ComponentParams arg0) throws WTEException {
45         // TODO Auto-generated method stub
46         //Create TreeConfig
47         ComponentConfigFactory factory = getComponentConfigFactory();
48         JcaTreeConfig tree = (JcaTreeConfig) factory.newTreeConfig();
49 }
```

Using JcaTreeConfig for tree UI design

## 2. Exercise: Search and Tree Result

### Build MVC Tree Table

#### 5. Create MVC Class

- Before creating an MVC class, you must register the MVC directory. If you do not want to register the directory, you can directly register the new MVC class.

```
33 @ComponentBuilder("csc.mvc.sampleTree01")
34 public class CSCSampleTree01 extends AbstractComponentBuilder implements TreeDataBuilderAsync {
35     CSCSampleTreeHandler01 sampleHandler;
36
37     public Object buildComponentData(ComponentConfig arg0, ComponentParams arg1) throws Exception {}
38
39     public ComponentConfig buildComponentConfig(ComponentParams arg0) throws WTEException {}
40
41     @Override
42     public void buildNodeData(Object node, ComponentResultProcessor resultProcessor) throws Exception {
43         if (node == TreeNode.RootNode) { // special case for root nodes
44             sampleHandler = new CSCSampleTreeHandler01(resultProcessor.getParams());
45             resultProcessor.addElements(sampleHandler.getRootNodes()); // should fetch rootNodes and to
46         } else { // case for any other node
47             List nodeList = new ArrayList();
48             nodeList.add(node);
49             Map<Object, List> map = sampleHandler.getNodes(nodeList);
50             Set keySet = map.keySet();
51             for (Object key : keySet) {
52                 resultProcessor.addElements(map.get(key));
53             }
54         }
55     }
56 }
```

1

1. Overriding “buildNodeData” function. You just copy sample source code because the source code block is the most general implementation for tree. For getting Root and Child, call to TreeHandler.
2. TreeHandler is custom handler, so you have to create your custom handler.

## 2. Exercise: Search and Tree Result

### Build MVC Tree Table

#### 6. Create Tree Handler Class

- Handler class is custom class for get Root Node and Child nodes. Before creating MVC class, you must register MVC directory. If you do not want to register directory, you can directly register new MVC class.

- Create custom TreeHandler.** Extended from “TreeHandlerAdapter” If you have parameter, you set global parameter for keeping parameter values.
- Create Default constructor**
- Create custom constructor.** If you have parameters, use this constructor
- Overriding “getNodes” function.** Find child nodes using “parents” input return must be sent Map<parent, children>
- Overring “getRootNodes” function.** Find root nodes

```
28 public class CSCSampleTreeHandler01 extends TreeHandlerAdapter { 1
29     private NmCommandBean cb = null;
30     String number = null;
31     String name = null;
32     String partType = null;
33
34     public CSCSampleTreeHandler01() { 2
35
36     }
37
38
39     public CSCSampleTreeHandler01(ComponentParams params) throws WTEException { 4
40
41     }
42
43     public CSCSampleTreeHandler01(ComponentParams params) throws WTEException { 3
44         number = (String) params.getParameter("number");
45         name = (String) params.getParameter("name");
46         partType = (String) params.getParameter("partType");
47         //number = "GC000001";
48     }
49
50
51     public Map<Object, List> getNodes(List parents) throws WTEException { 4
52
53     }
54
55     @Override
56     public List<Object> getRootNodes() throws WTEException { 5
57         // TODO Auto-generated method stub
58         List<Object> resultlist = new ArrayList<Object>();
59     }
56
57 }
```

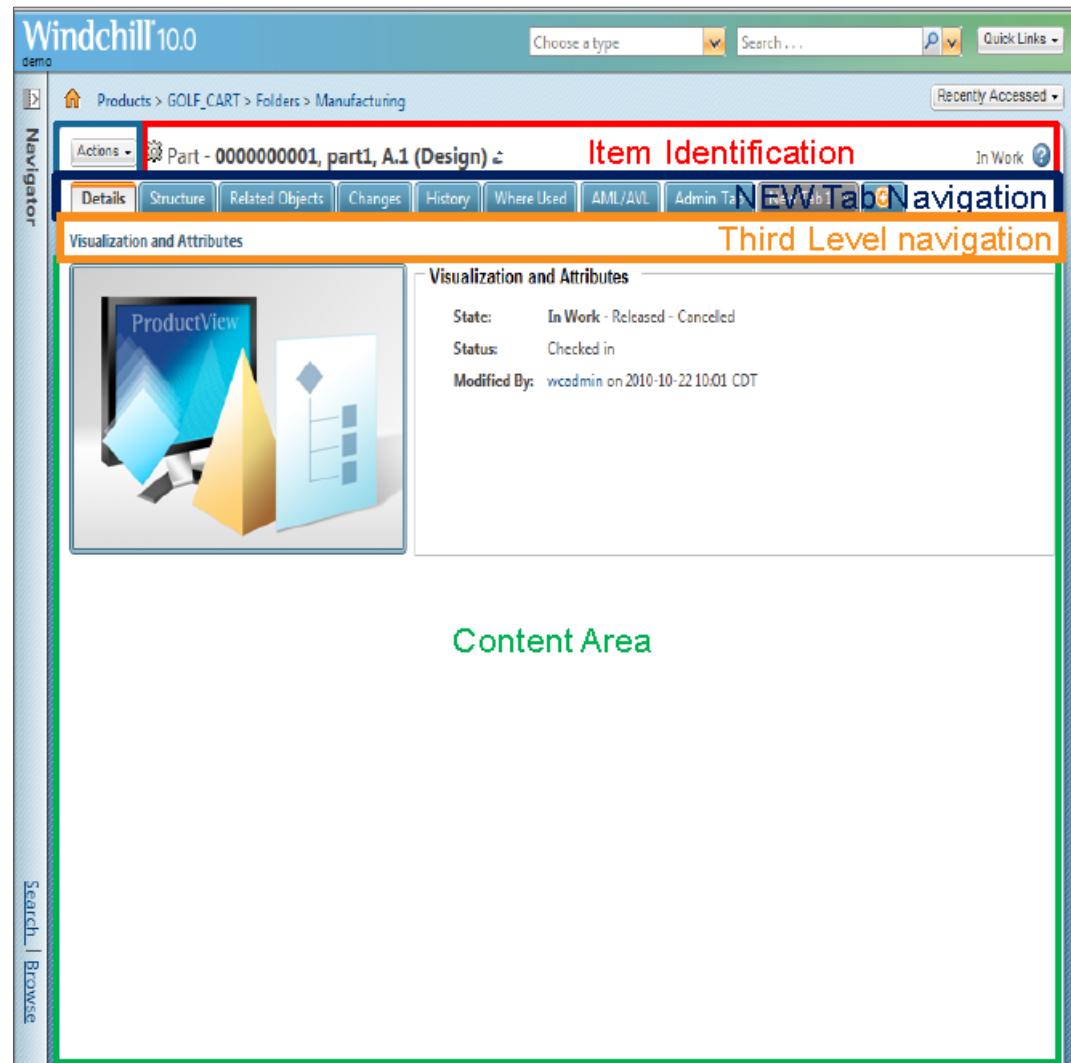
# Info Page Customization

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

# 1.1 Understanding Info Page

When implementing info pages, the following sections must be designed to describe the UI.

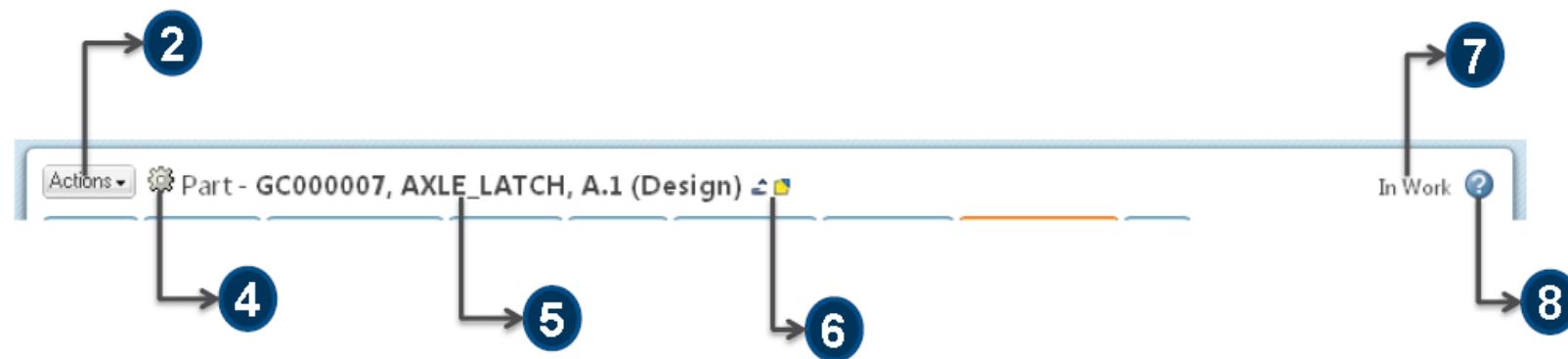
- Action Menu
- Item Identification
- New Tab navigation
  - New tab navigation includes the third-level navigation menu.
- Third-level navigation
  - Third-level navigation includes contents for displaying on content area.
- Content Area
  - Attributes and Visualization are now content just like any 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 consists of several configurations such as the following:

1. Go to latest link (only appears on the 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, and so on)
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 of tabs
- Command line tool for transferring tab configures from test to production

## How to Control for Each Part of Information Page

Before creating the Info page, configuration should be ready because many parts of the UI design because the Info pages are separated by setting up action and action model.

### – Action and Action Model

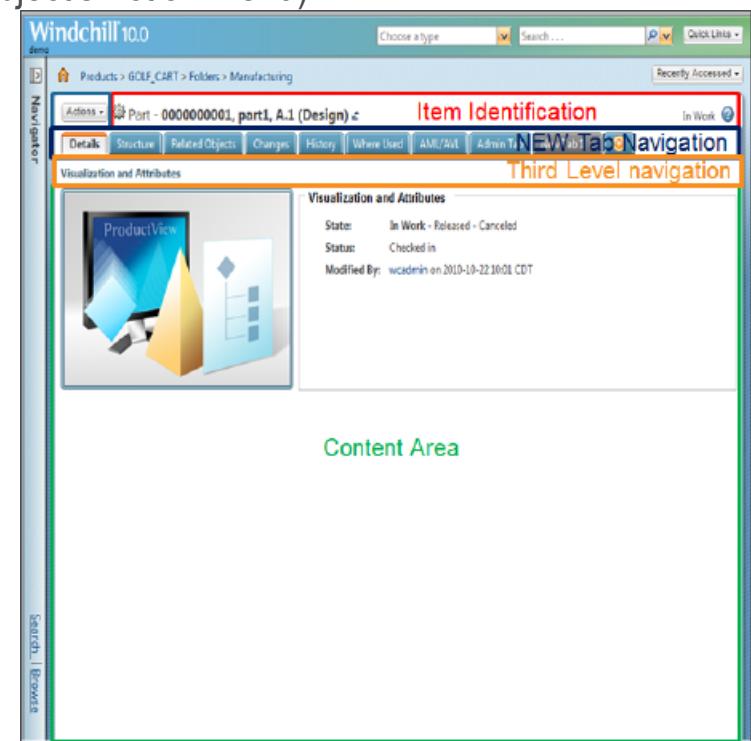
- Action Menu (if it is not set, it will automatically set the parent object's Action Menu)
- New tab navigation
- Third-level navigation

### – MVC component or JSP

- Content area

### – Information Page component

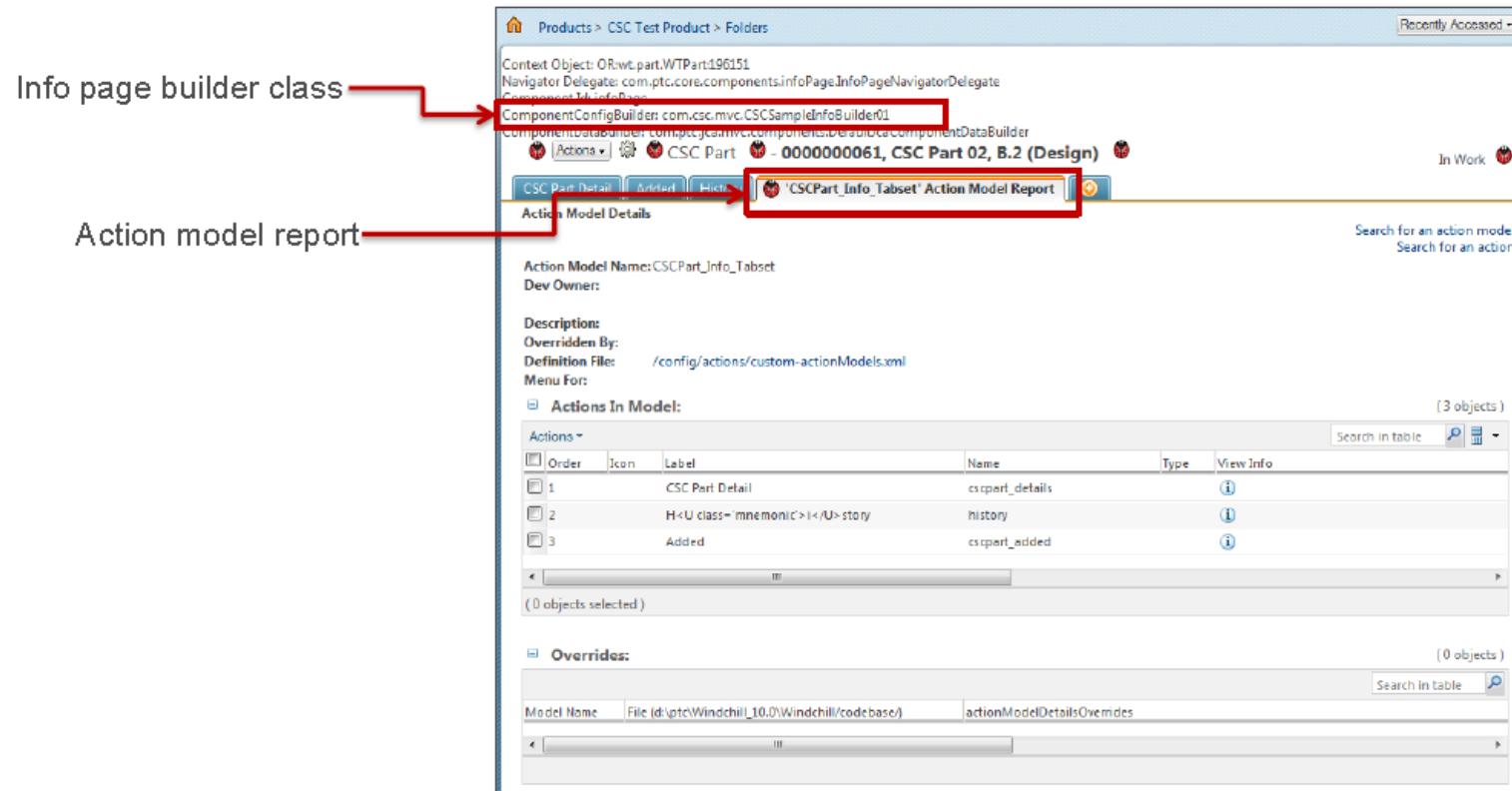
- Item Identification



## Look Action and Action Model Report

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

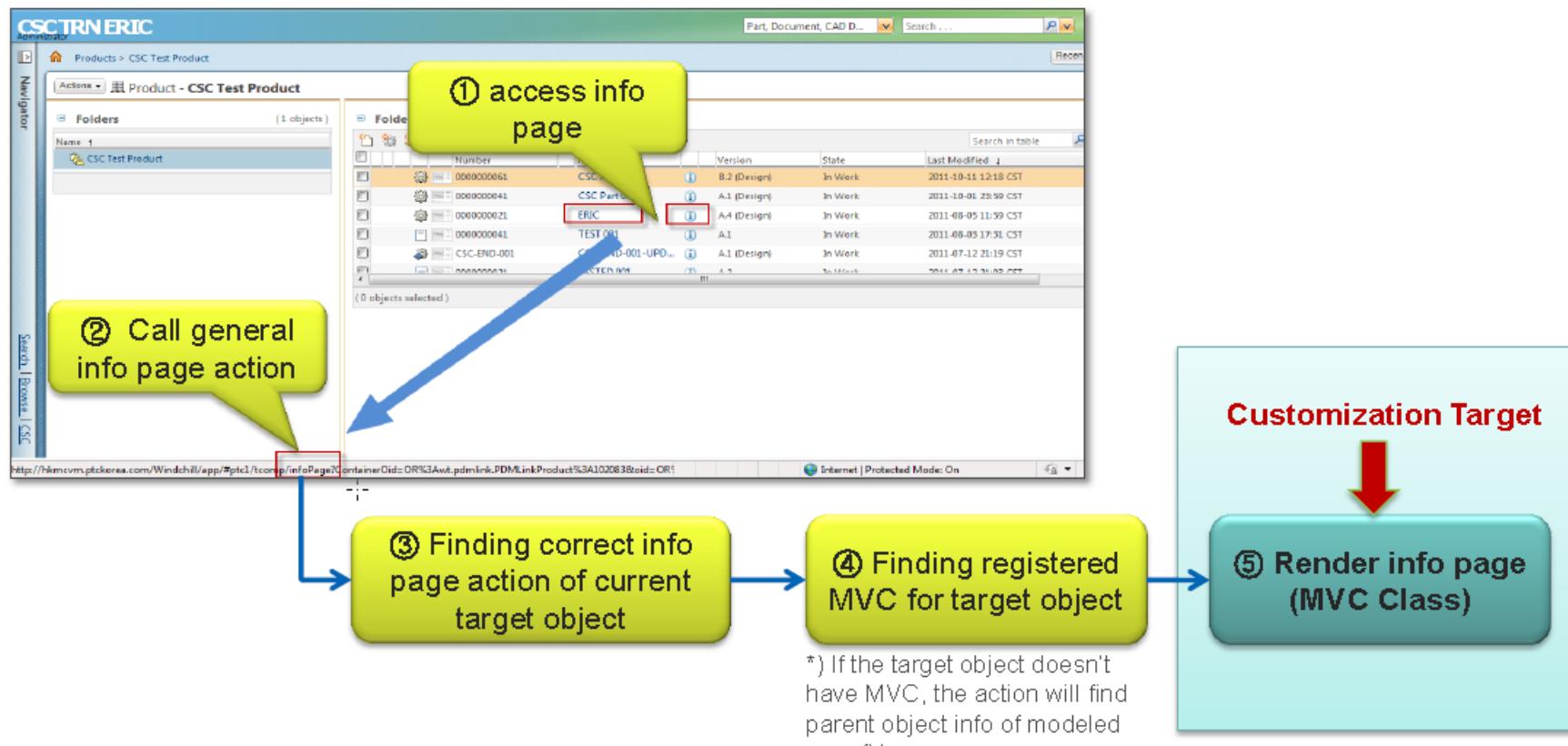
- Can know Info page builder class
- Can know action model report for Tab and third-level content configuration



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

## 2.1 Progress of Calling Info Page

Generally, Info page will be called from info page action in UI (OOTB). This time, the system progresses to find the correct info page action. Finally, Windchill 10 architecture starts finding registered MVC class for targeted object which displays info page.



## 2.2 Information Page Process Flow

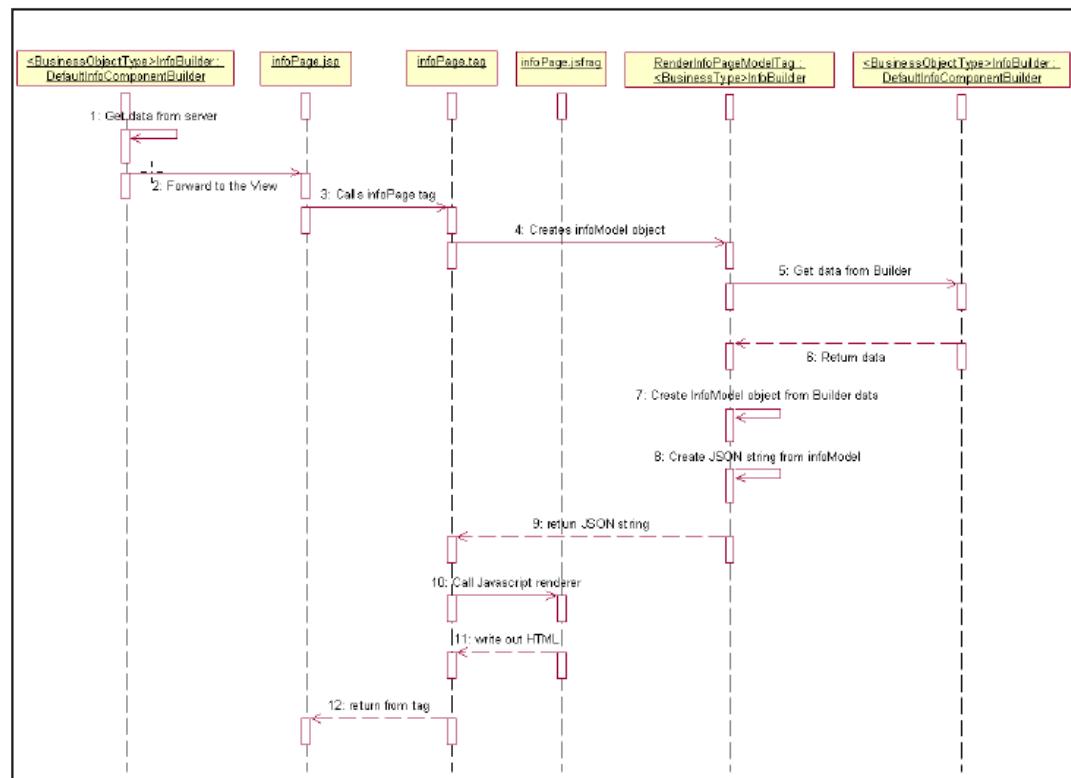
Following is the process sequence for rendering the info page.

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

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

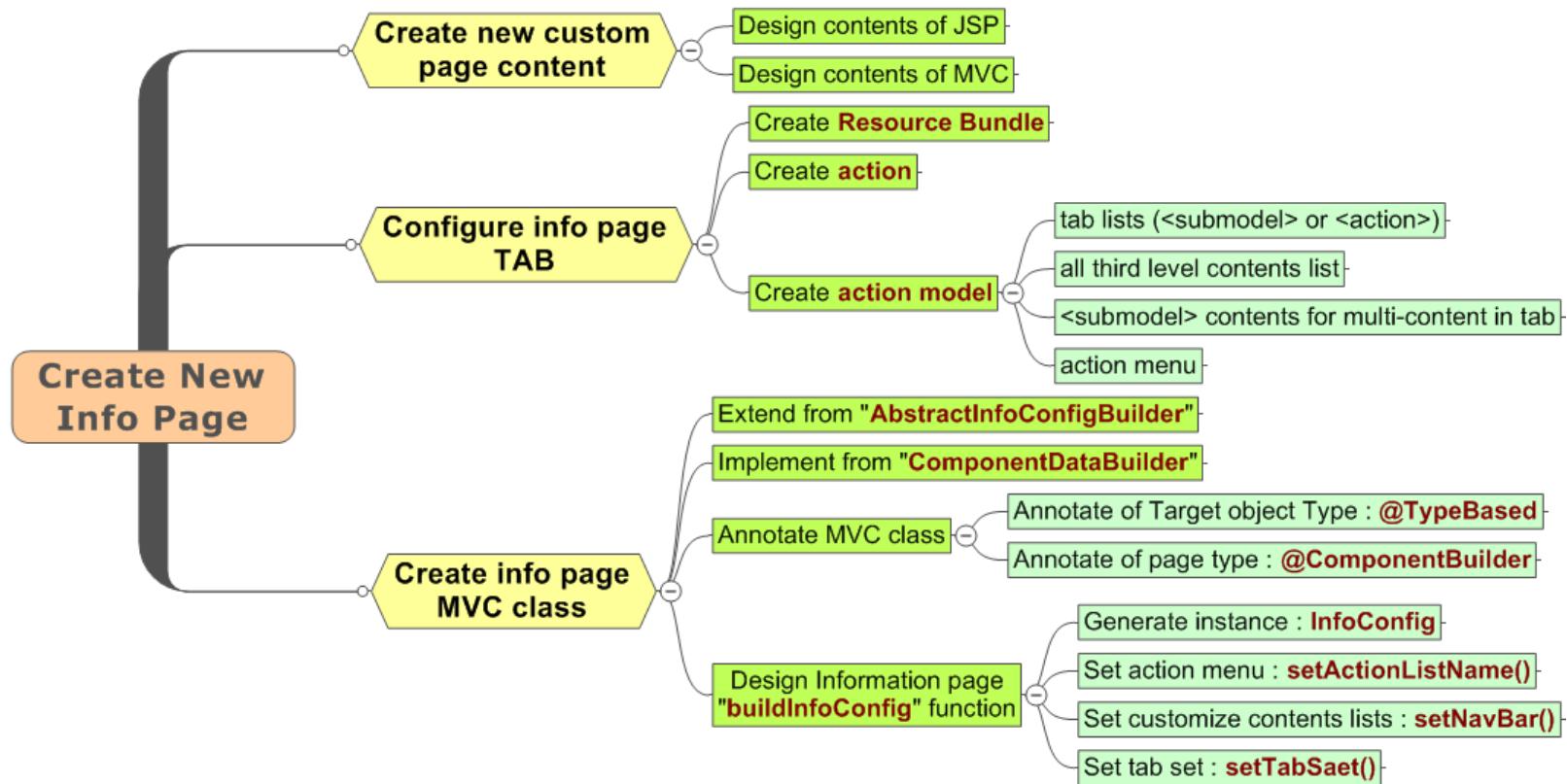
The tag creates a JSON object, known as the infoModel object, from the data it collected from the builder.

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

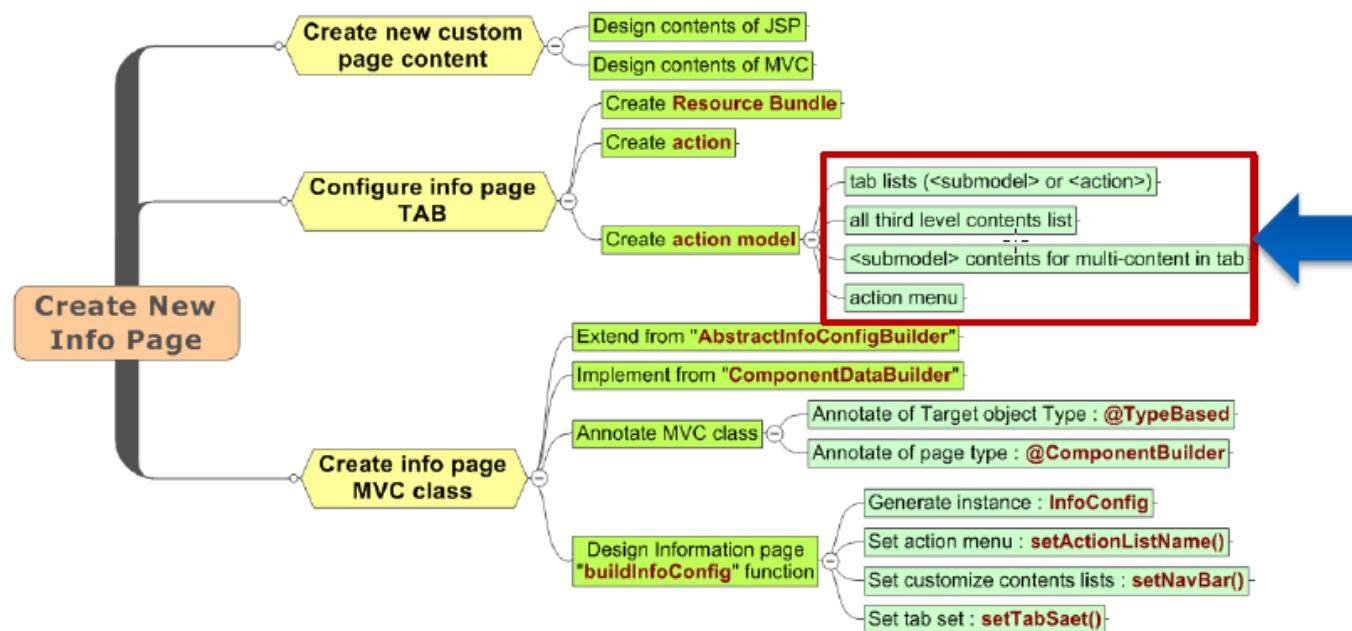
Information page is needed to be ready for many kinds of configuration for creating new pages. First of all, you must understand all related configuration.



## 2.4 Introduction of Action Model for Information Page

You have to set three group of action models for the info page.

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



## 2.5 Description of Action Model

```
<!-- CUSTOM MENU LIST -->
<model name="third_level_nav_part">
  <submodel name="general"/>
  <submodel name="relatedItems"/>
  <submodel name="changes"/>
  <submodel name="history"/>
  <submodel name="collaboration"/>
  <submodel name="prodAnalytics"/>
</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

## 2.5 Description of Action Model (Custom Menu List)

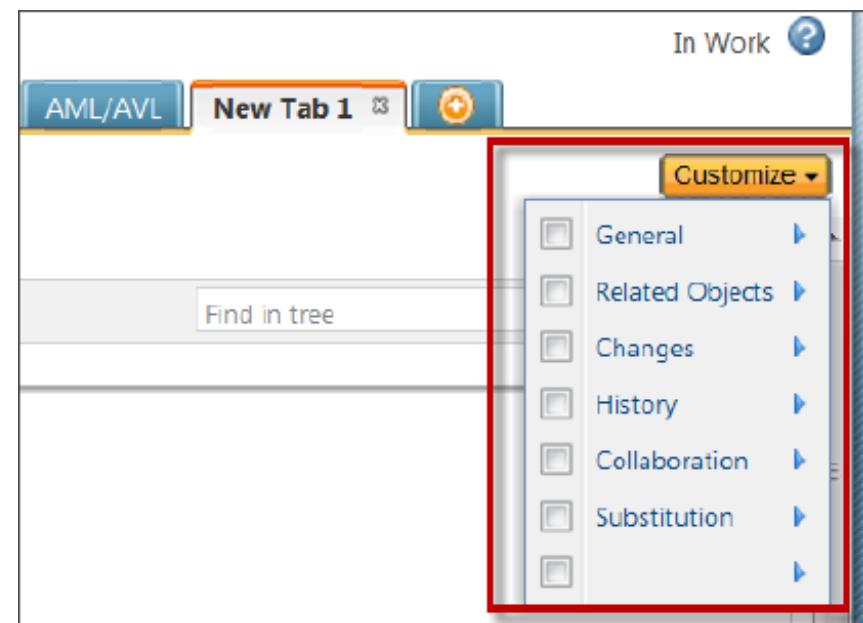
```
<!----- CUSTOM MENU LIST ----->
<model name="third_level_nav_part">
  <submodel name="general"/>
  <submodel name="relatedItems"/>
  <submodel name="changes"/>
  <submodel name="history"/>
  <submodel name="collaboration"/>
  <submodel name="prodAnalytics"/>
</model>
```

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)

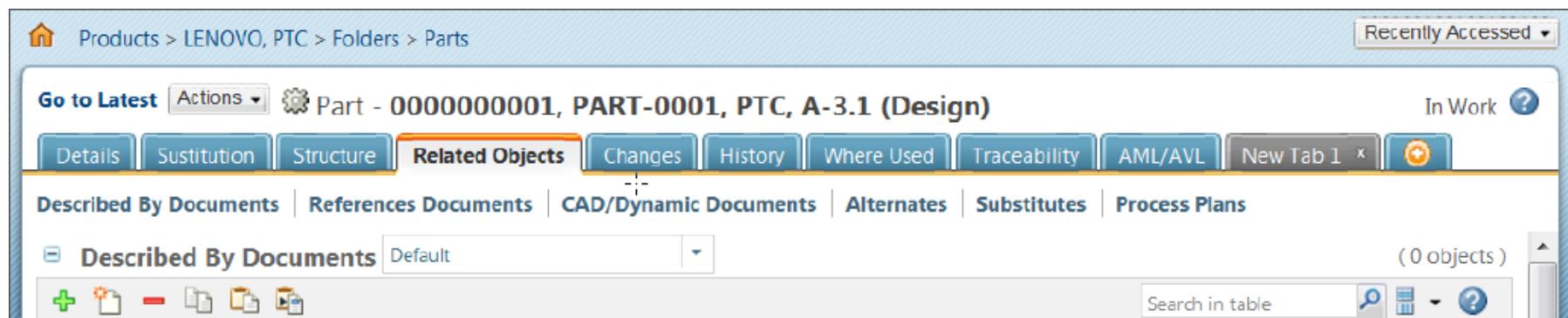
If you used <submodel> for tab, all submodel action must be existed on custom list.

If you used <action> directly in tab configuration, it doesn't need to add custom menu list.



## 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

## 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)

The screenshot shows a Java code editor with the following code:

```
1 package com.csc.mvc;
2
3 import wt.part.build.WTPartBuildHelper;
4
5 @TypeBased(value = "wt.part.WTPart|com.ptckorea.hkmcm.CSCPart")
6 @ComponentBuilder(value = ComponentId.INFOPAGE_ID)
7 public class CSCSampleInfoBuilder01 extends AbstractInfoConfigBuilder implements ComponentDataBuilder {
8
9     @Override
10    public Object buildComponentData(ComponentConfig arg0, ComponentParams arg1) throws Exception {
11        return arg1.getContextObject();
12    }
13
14    @Override
15    protected InfoConfig buildInfoConfig(ComponentParams arg0) throws WTEexception {
16        InfoComponentConfigFactory factory = getComponentConfigFactory();
17        InfoConfig result = factory.newInfoConfig();
18
19        result.setNavBarName("cscpart_third_level_nav");
20        result.setTabSet("CSCPart_Info_Tabset");
21
22        return result;
23    }
24}
```

Annotations are placed on specific code lines:

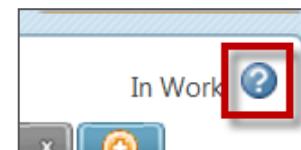
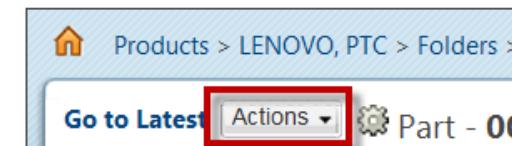
- Annotation 1: On the line `implements ComponentDataBuilder`.
- Annotation 2: On the line `ComponentId.INFOPAGE_ID`.
- Annotation 3: On the line `value = "wt.part.WTPart|com.ptckorea.hkmcm.CSCPart"`.
- Annotation 4: On the line `ComponentBuilder`.
- Annotation 5: On the line `buildComponentData`.
- Annotation 6: On the line `buildInfoConfig`.

## 2.7 MVC Function

The 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.

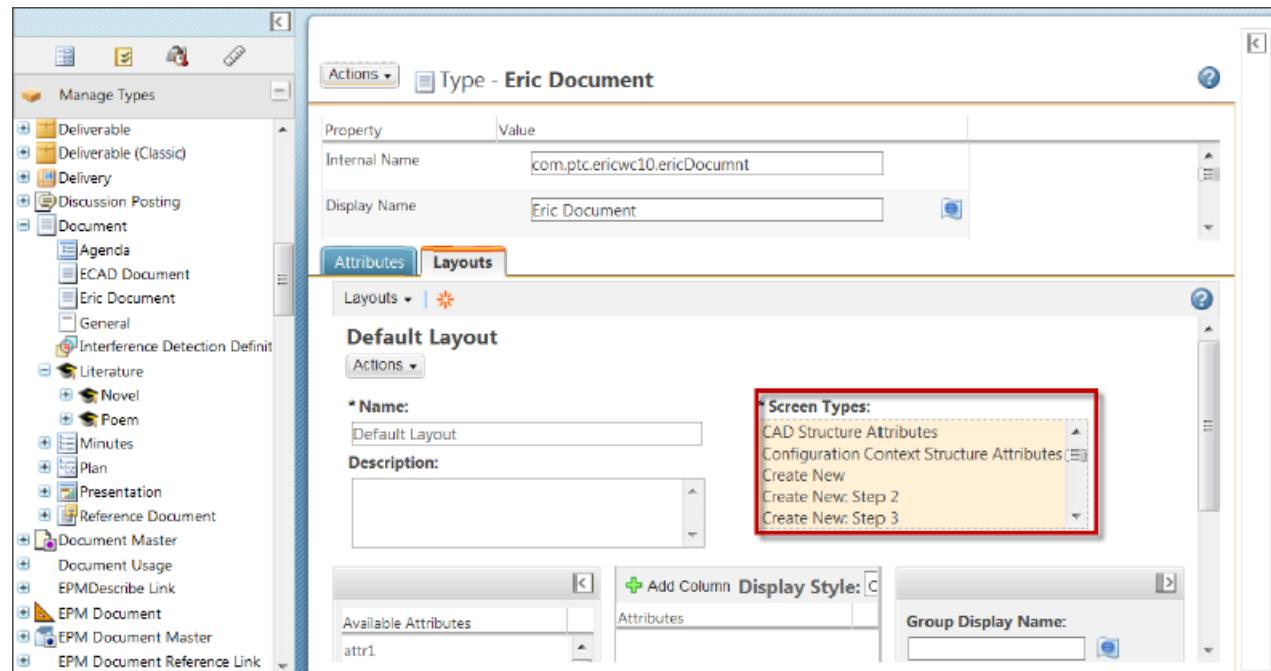


1. Introduction Info Page
2. Architecture of custom information page
3. Add custom third navigation on Info Page
4. Exercise

## 3.1 Understanding the layout

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

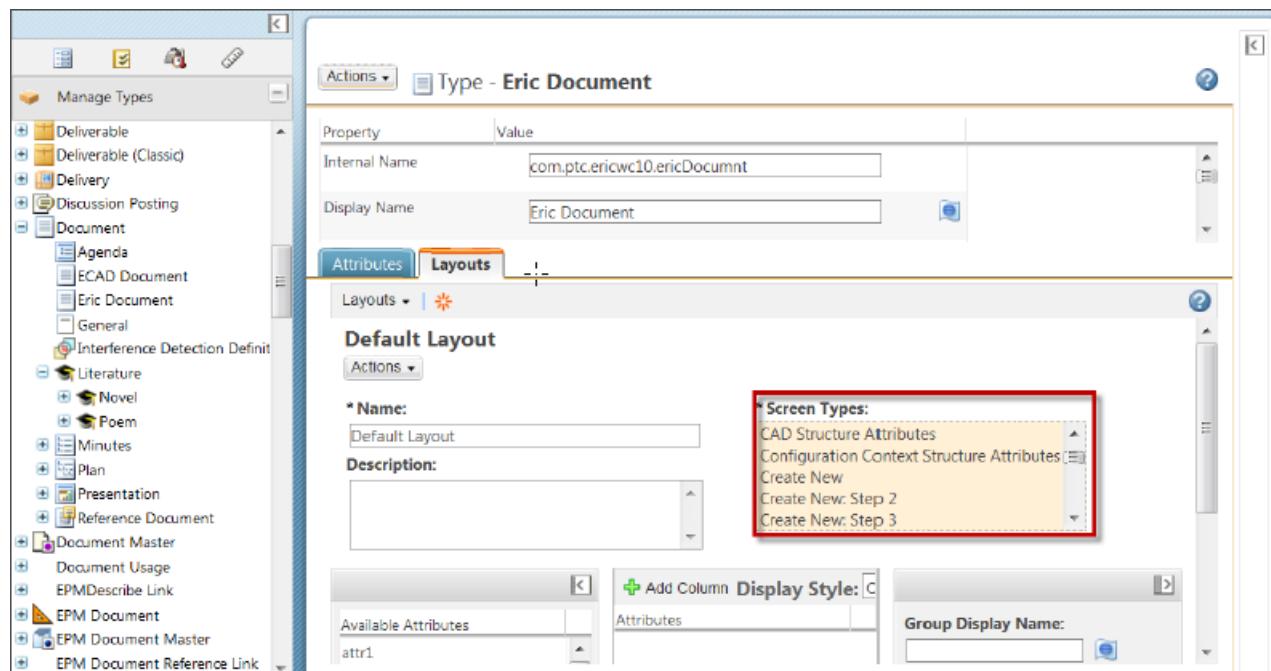
- Show Primary/More attributes (Detail tab of info page)
- Input attributes set (Create/Edit wizard page)
- Etc (Several UI pages are using layout)



## 3.1 Understanding the Layout

- **Screen Type of Layout**

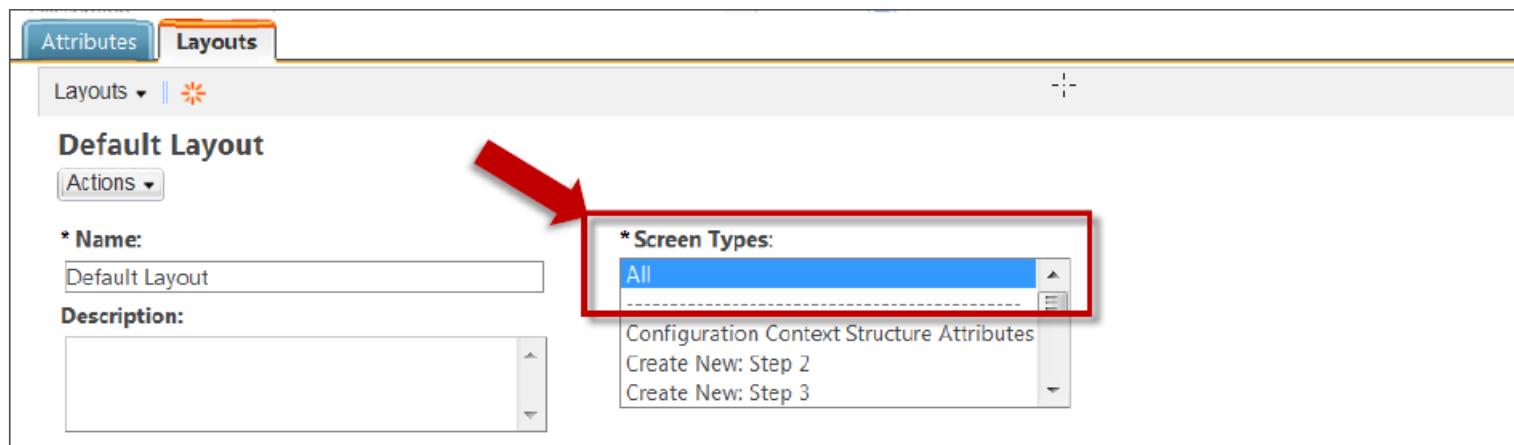
- Screen type was already fixed by Windchill core, so the layout is just supported on the fixed screen type.
- The screen type can be selected just one time by layout (if one layout selects a screen type, the type cannot be displayed on the other layout – except the default layout).
- If you want to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



## 3.1 Understanding the Layout

### Default Layout

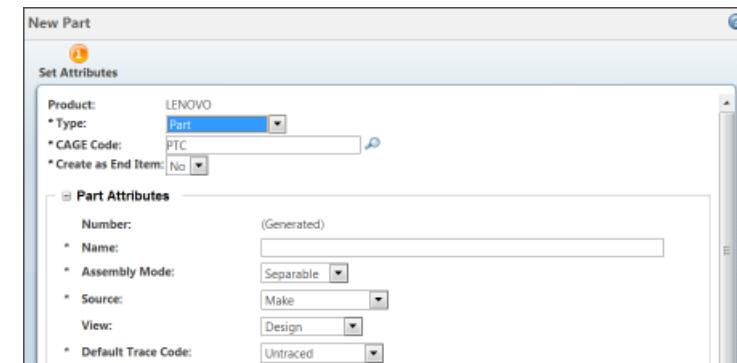
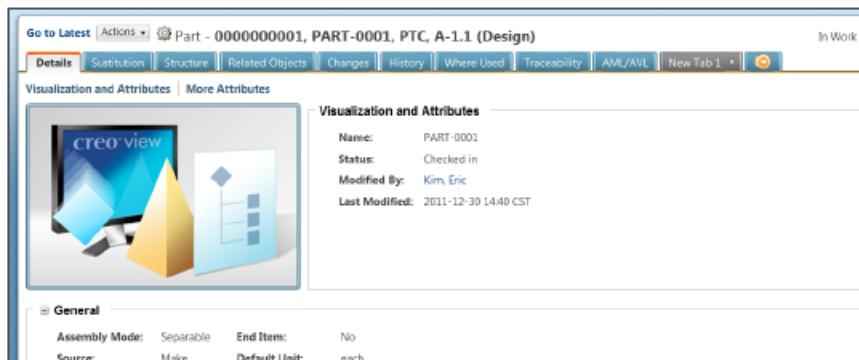
- Default layout is assigned for all screen types. So if the screen type doesn't have a layout system, it will show the default layout.
- Be careful that each type must have one default layout which uses all screen types.
- The layout name doesn't affect the system. Screen type has a real effect on the system.



## 3.2 OOTB Action Set for Using Layout

The following is a 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 a special layout which is EPMDocument, you have to use the layout action.

## 3.3 Example of Layout in Info Page

Info page uses layout especially 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)

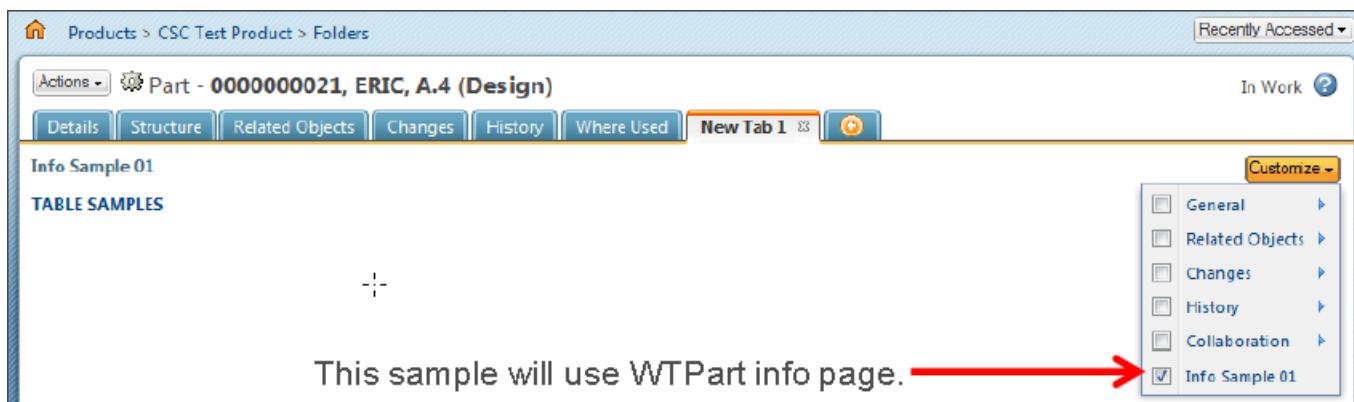
This document will not describe how to use and create a layout. You can find several documents on the PTC support sites.

1. Introduction Info Page
2. Architecture of Custom Information Page
3. Add Custom Third Navigation on Info Page
4. Exercise

### 3.1 Exercise 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 the new information page.
2. How to add customized contents (MVC or JSP) without non-modified info page source code (just changed configuration).
3. Understanding to create a “custom new tab.”
4. Understanding which configuration file is needed to modify the info page and how to do it.
5. Design rule in the customize button:
  - **TAB design for using <submodel>**: This tab will show several contents in the area.  
If page contents included in <submodel> do not exist on the 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>**: This tab just shows one content. Although the content does not exist in the Customize menu lists, it can be shown on the info page.



### 3.1 Exercise 1 — Add Custom Action on Customize Menu

Find OOTB Action model and Builder source code

#### 1. Open OOTB info page and look for 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.

The screenshot shows the 'Action Model Details' dialog for a WTPart info page. The 'ComponentConfigBuilder' field is highlighted with a yellow circle labeled '1'. The 'Definition File' field is highlighted with a yellow circle labeled '2'. The dialog also shows other details like 'Action Model Name: partInfoPageTabSet', 'Dev Owner:', 'Description:', 'Overridden By:', and 'Menu For: /config/actions/PartClient-actionmodels.xml'.

Order	Icon	Label	Name	Type	View Info
1		Details	partInfoDetailsTab	psb	(1)
2		Structure	productStructureGWT	psb	(1)
3		Related Objects	partInfoRelatedItemsTab	psb	(1)
4		Changes	changesTab	psb	(1)

## Find Correct Action Model Name

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

OpenGrok has native source codes of Windchill; you can find OOTB info builder class and check which kind of configure name is used for info page. Refer to the <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 WTEexception {
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 Customize menu contents are used on <submodel>, they must be registered in this area, although it didn't set on the tab.

### 2. The configuration name for tab set

- This is configured for tab design.

# 3.1 Exercise 1 — Add Custom Action on Customize Menu

## Update Action Model and Action

3. Open “PartClient-actionmodels.xml” and copy the “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">
    <!-- General -->
    <!-- Related Objects -->
    <!-- Change -->
    <!-- History -->
    <!-- Collaboration -->
    <!-- Product Analytics -->
    <action name="infoSample01" type="csc"/>
</model>
```

4. Create a 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 you don't have a resource bundle, you must create resource bundle for action.

action.properties, action\_[locale].properties

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

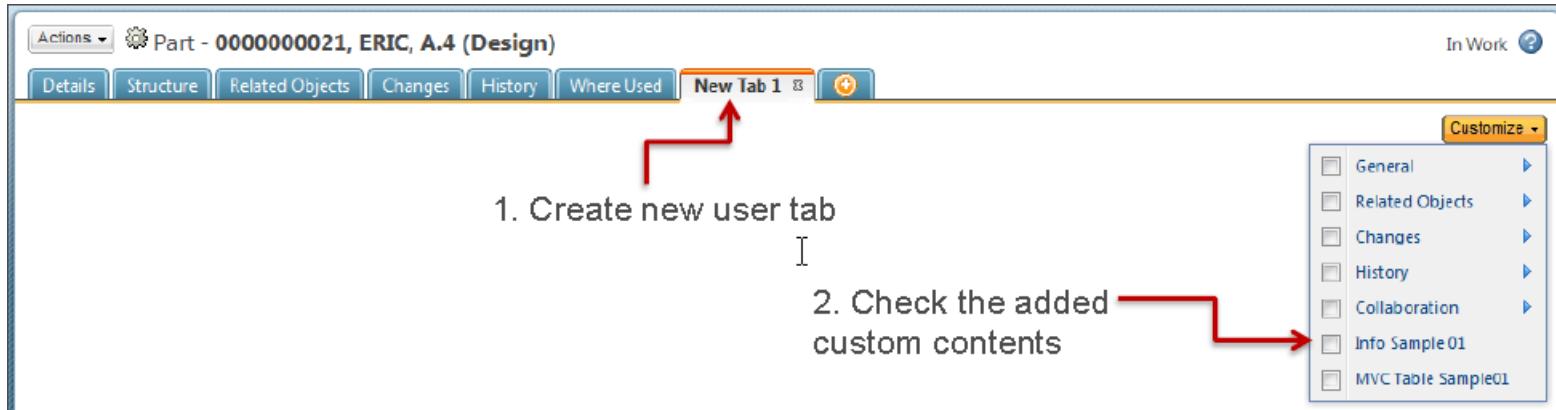
6. Restart the Method Server service or Reload actions.

### 3.1 Exercise 1 — Add Custom Action on Customize Menu

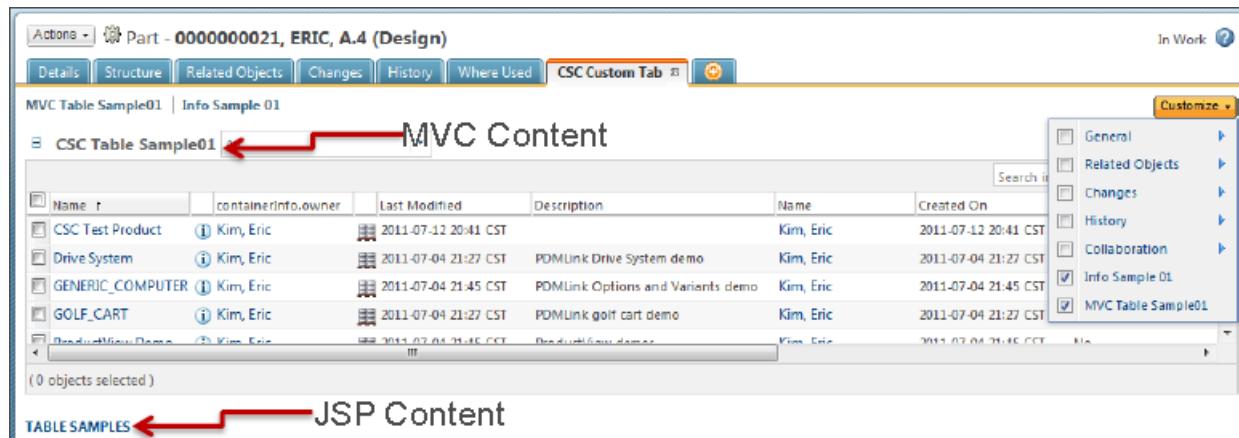
PTC®

#### Result

- Check the Customize button (add more actions for testing MVC).



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



## 3.2 Exercise 2 — Show Other Web Pages on the Info Page

PTC®

### Purpose

Some customers want to see additional Web pages on the info page. For example, they should be able to check cost of parts when they see part information.

The screenshot shows the Windchill 10.0 interface. The top navigation bar includes 'Part, Document, CAD D...', 'Search...', and 'Quick Links'. Below the navigation bar, the breadcrumb path shows 'Products > LENOVO, PTC > Folders > Parts'. The main title is 'Actions - Part - 0000000041, PART-0004, PTC, A.1 (Design)'. The toolbar below the title contains several tabs: Details, Sustitution, Structure, Related Objects, Changes, History, Where Used, Traceability, AML/AVL, PDSAdvisor, OpenGrok, and GS Wiki. The 'GS Wiki' tab is highlighted with a red arrow. The right side of the interface features a sidebar with 'Navigator' and 'Search/Browse' buttons. The main content area displays a 'Home' page for the GS Wiki, which includes a search bar, edit/share tools, and a list of communities.

## 3.2 Exercise 2 — Show Other Web Pages on the Info Page

### Update Action Model and Action

1. Open “PartClient-actionmodels.xml” and copy the “third\_level\_nav\_part” block to custom action model. And add the custom action.

csc-actionmodels.xml

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

2. Create a 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 you do not have a resource bundle, you must create resource bundle for action.

action.properties, action\_[locale].properties

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

## 3.2 Exercise 2 — Show Other Web Pages on the Info Page

### Create JSP and Check Result

#### 4. Create the JSP page in “\$WT\_HOME/codebase/wtcore/jsp/csc”

wikiPageShow.jsp

```
<%@ taglib uri="http://www.ptc.com/windchill/taglib/components" prefix="jca"%>
<jca:tabToHighlight actionPerformed="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 the Method Server service or Reload actions.



### 3.3 Exercise 3 — Create a New Info Page

#### Scenario

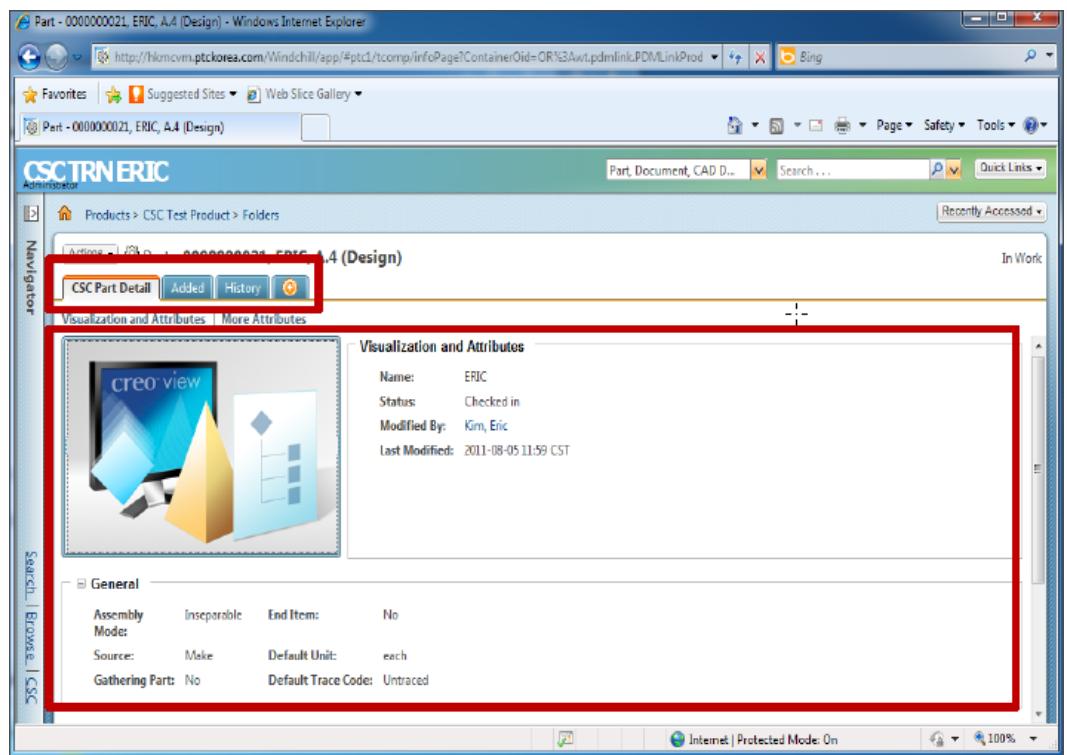
Customer created one new soft type object which is extended from WTPart. But the soft type info page show like part, so customer want to change tab list and content.

#### 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)



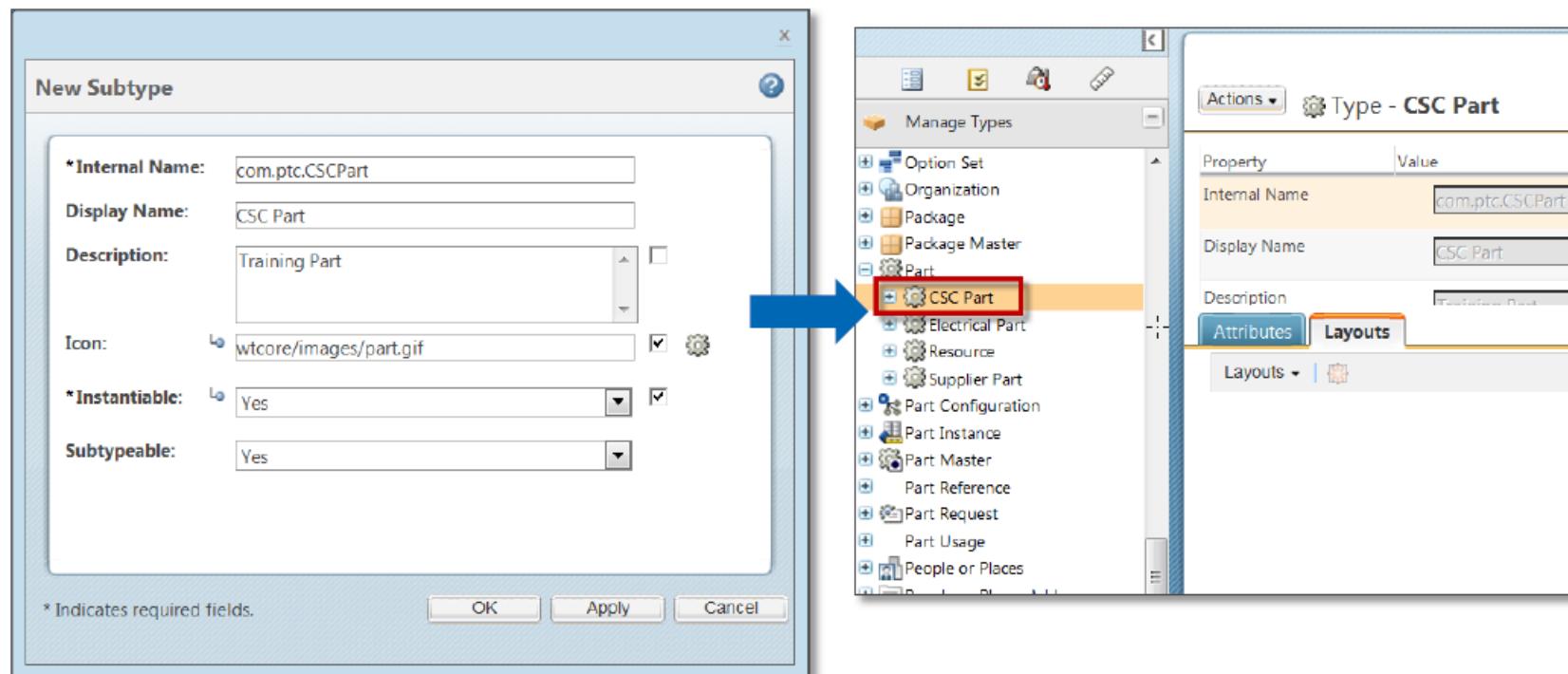
### 3.3 Exercise 3 — Create a New Info Page

#### Create Soft Type of Part

##### 1. Create Soft type object

Create a subtype object below a part like the following attributes:

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



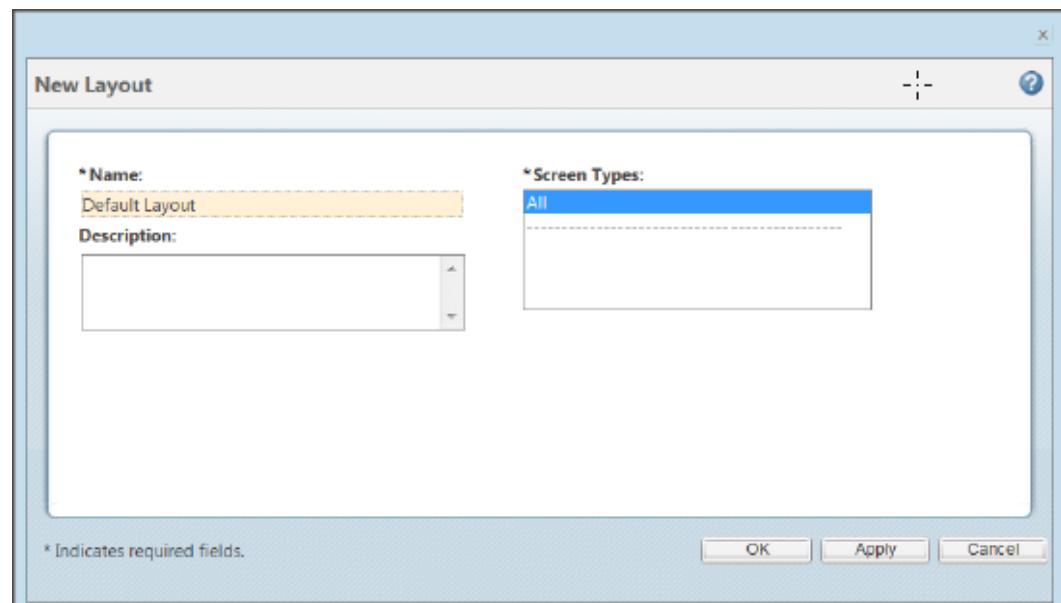
### 3.3 Exercise 3 — Create a New Info Page

#### Create a Default Layout

##### 2. Create a Default Layout

Create a default layout for a new subtype object. If you want to know why we need to create a default layout, look at [“3.1 Understanding of layout.”](#) When you create a new layout of subtype, all of the 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 the **Layout tab** and **click create new layout icon**.
4. Enter information:
  - Layout name: Default Layout
  - Screen Type: All
5. Edit the default attributes on default layout (Optional).



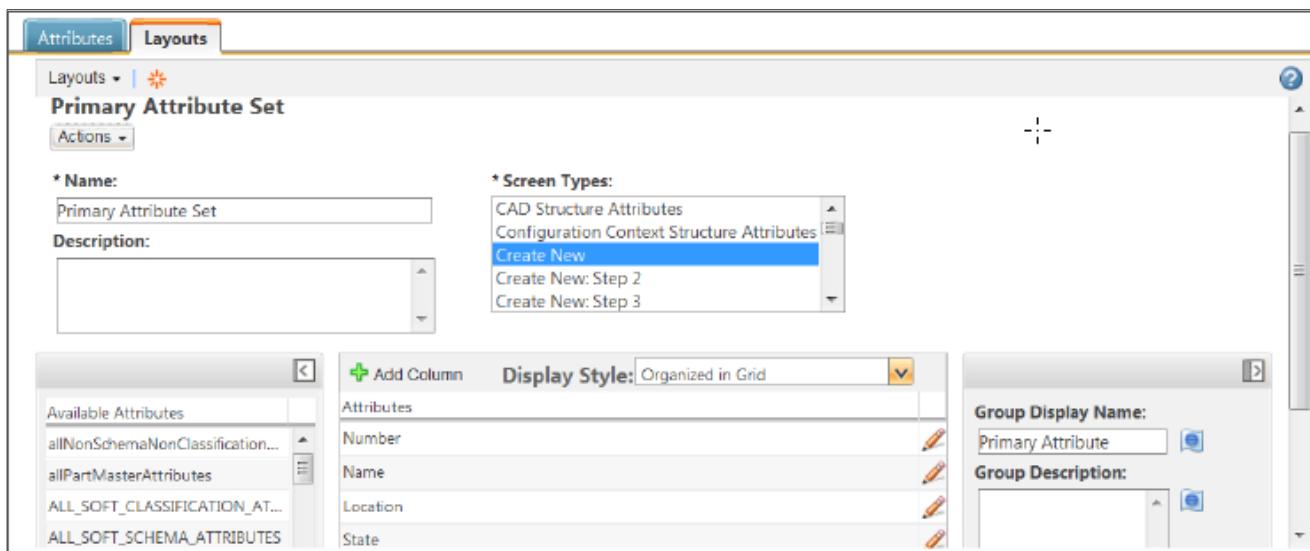
### 3.3 Exercise 3 — Create a New Info Page

#### Create/Modify the Screen Layout (Primary Attributes Set)

##### 3. Create a Primary attribute set for create/modify

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

1. Click the 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 on primary attributes:
  - Group name: Primary Attribute
  - Add Attribute: Number/Name/Location/State



### 3.3 Exercise 3 — Create a New Info Page

Modify custom-actionModels.xml

#### 4. Create the Customize menu.

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

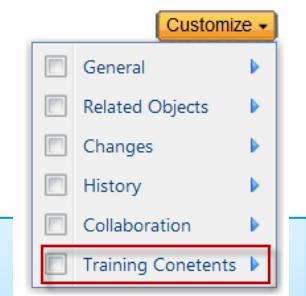
1. Open “partClient-actionmodel.xml” and copy block of "third\_level\_nav\_part."
2. Open “custom-actionmdoel.xml” and paste copied block on the file.
3. Add all of the custom actions in copied block freely.

custom-actionmodels.xml

```
<model name="CSCPart_customize_menu">
    <submodel name="general"/>
    <submodel name="relatedItems"/>
    <submodel name="changes"/>
    <submodel name="history"/>
    <submodel name="collaboration"/>
    <submodel name="prodAnalytics"/>
    <submodel name="CSCPartContents"/>

!-- General -->  
!-- Related Objects -->  
!-- Change -->  
!-- History -->  
!-- Collaboration -->  
!-- Product Analytics -->  
<!-- CUSTOM SET -->


```



#### Design rule in the Customize button:

- TAB design for using <submodel>: This tab shows several contents in the area. If the page contents included in <submodel> do not exist in the 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>: This tab just shows one content. Although the content does not exist in the customized menu lists, it can be shown on the info page.

### 3.3 Exercise 3 — Create a New Info Page

Modify custom-actionModels.xml

#### 5. Create a Tab list and submodel

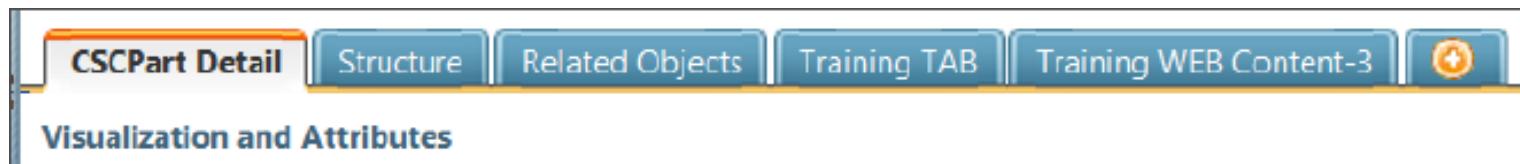
It must be configured for info page. It will also be referred from WTPart tab configuration.

1. Open “partClient-actionmodel.xml” and copy block of “partInfoPageTabSet.”
2. Open “custom-actionmdoel.xml” and paste copied block on the file.
3. Edit for custom style as follows:

custom-actionmodels.xml

```
<model name="CSCPartInfoPageTabSet">
    <submodel      name="CSCPartInfoDetailsTab"/>
    <action        name="productStructureGWT" type="psb"/>
    <submodel      name="partInfoRelatedItemsTab"/>
    <submodel      name="customTab1"/>
    <action        name="CSCPartContent3" type="info_trn"/>
</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>
```

**<!-- TRAINING INFO PAGE -->**  
**<!-- OOTB -->**  
**<!-- OOTB -->**  
**<!-- CUSTOM TAB (SUBMODEL) -->**  
**<!-- CUSTOM TAB (ACTION) -->**



# 3.3 Exercise 3 — Create a New Info Page

## Modify custom-actions.xml

### 6. Create a new action

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

1. Open “custom-action.xml” and edit the custom action as follows:

#### custom-actionmodels.xml

```
<model name="CSCPartInfoPageTabSet">
    <submodel      name="CSCPartInfoDetailsTab"/>
    <action        name="productStructureGWT" type="psb"/>
    <submodel      name="partInfoRelatedItemsTab"/>
    <submodel      name="customTab1"/>
    1 <action        name="CSCPartContent3" type="info_trn"/>
</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"
    3 <action name="CSCPartContent2"
</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>
```

### 3.3 Exercise 3 — Create a New Info Page

#### Create a Resource Bundle File

##### 7. Create a resource bundle file

If you add a new action and a tab while configuration, you should create resource bundle for display it. Open eclipse:

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.

### 3.3 Exercise 3 — Create a New Info Page

#### Create MVC Class of Info Builder

##### 8. Create info builder class

Finally we have to make info builder for a special subtype – “CSCPart”

1. Open eclipse and create a new package “ext.csc.training.mvc”
  - The package must be registered in MVCDISPATCHER
2. Create a 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 WTEException {
        // TODO Auto-generated method stub
        InfoComponentConfigFactory factory = getComponentConfigFactory();
        InfoConfig result = factory.newInfoConfig();

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

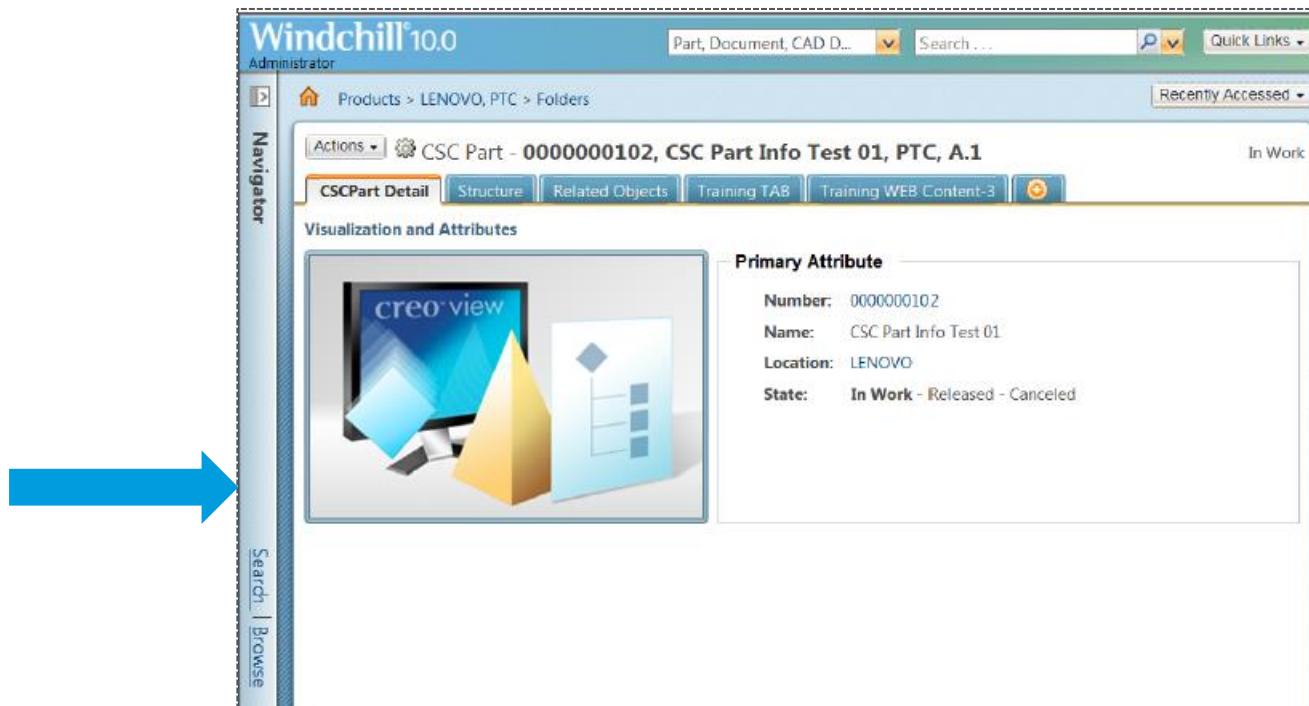
### 3.3 Exercise 3 — Create a New Info Page

Restart Method Server and Check Result

#### 9. Restart Method Server

#### 10. Check result

1. Create CSCPart in any container
2. Open the info page



# 3.4 Exercise 4 — Validate Tab & Action

## Scenario

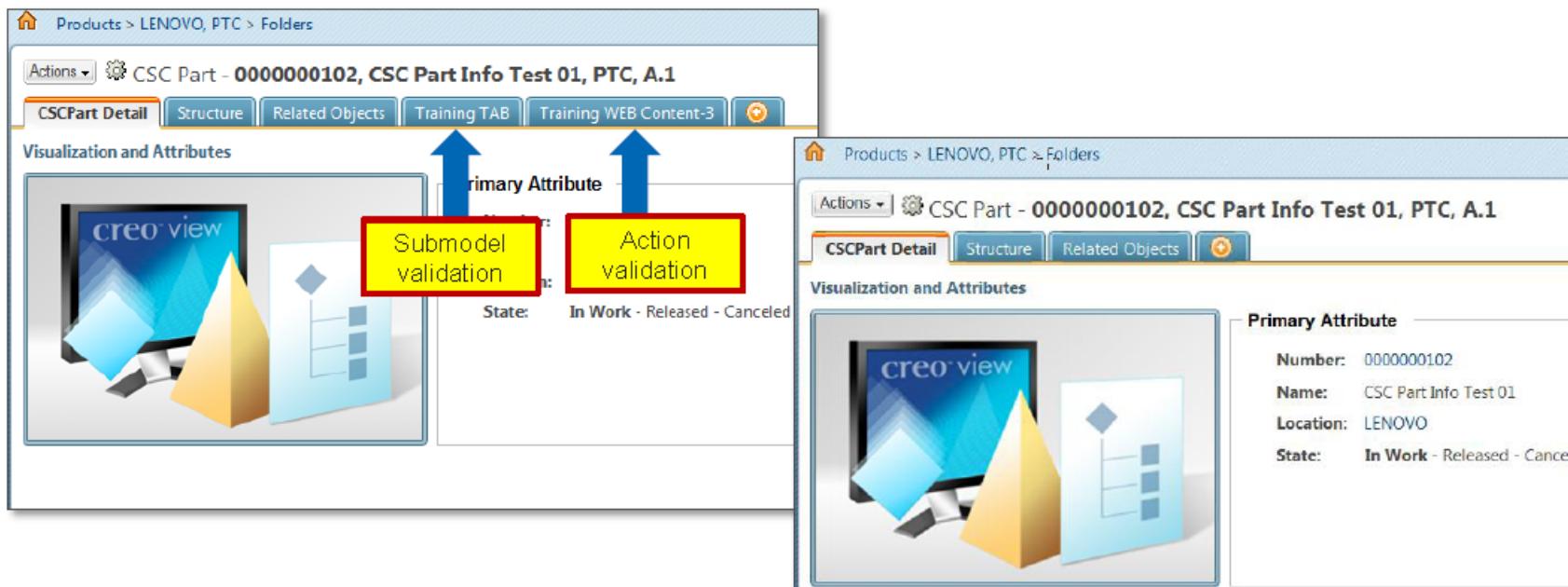
### 1. Scenario

User wants to show or hide the target tab or action (content/menu) inside the correct condition. Now we will try to make validation using the Exercise 3 result on the following condition.

- The “Training TAB” and the “Training WEB content 3” tabs show just the “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 Exercise 4 — Validate Tab & Action

### Set Custom Service Properties

#### 1. Register custom service properties

If possible, we have to try not to touch OOTB configuration file, so we will try to make a new custom service property file and will set to indicate by Method Server

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

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

Registering for X-configuration

```
<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`”

## 3.4 Exercise 4 — Validate Tab & Action

### Understand the <model> Tag Validation

#### 2. Register services key for validator of the <submodel> tag name

We have pre-set the validator of the <submodel> tag which is “customTab1,” so now we need to try to set the key on Windchill services like the following (using the <submodel> tag name):

\$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>
```

#### The following is the detailed description of service parameter:

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

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

When you finish updating x-configuration file, try to generate properties file using “xconfmanager -p.”

## 3.4 Exercise 4 — Validate Tab & Action

### Understand the <model> Tag Validation

#### 3. Create a Validator Class

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

```
package ext.csc.training.validator;
public class CSCPartSubmodelValidator extends DefaultUIComponentValidator {
    public UIValidationResultSet performFullPreValidation(UIValidationKey validationKey, UIValidationCriteria validationCriteria, Locale locale)
        throws WTEException {
        UIValidationResult result = null;
        UIValidationResultSet resultSet = UIValidationResultSet.newInstance(); Validation result set
        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

## 3.4 Exercise 4 — Validate Tab & Action

Understand the <model> Tag Validation

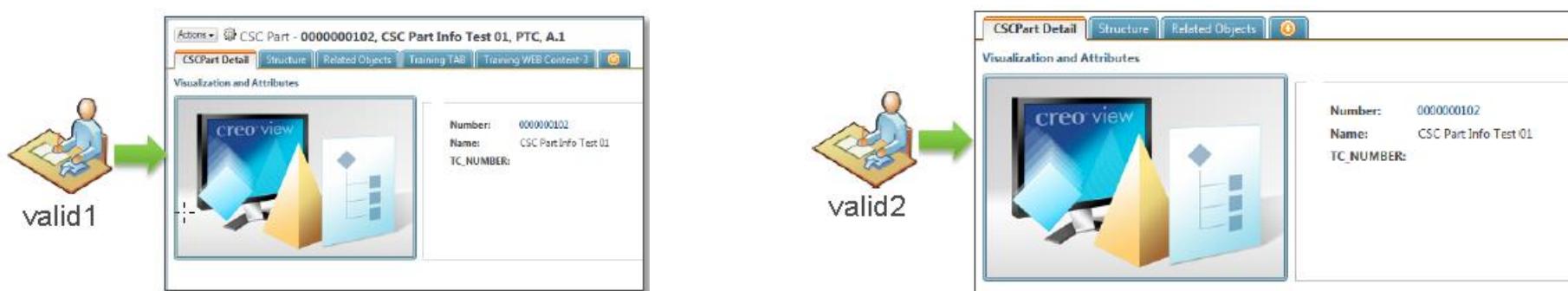
### 4. Restart the server and check your validation for tab

Restart the server; because you changed configuration files, so it needs to be restarted.

### 5. Check the <submodel> tag validation

The following is the test system configuration.

- User: valid1, valid2
- Group: CSC(member=valid1)
- Product: CSC\_TEST / members = valid1, valid2



## 3.4 Exercise 4 — Validate Tab & Action

### Understand Action Validation

#### 6. Check the action key name for validation

Action validator doesn't need any configuration on the "actions.xml." You just need to remember the 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" name="cscPartContent1" />
    </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="page">
        <command windowType="page" url="wtcore/jsp/csc/info_content3.jsp"/>
    </action>
</objecttype>
```

Remember class name

Remember key name

#### 7. 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>
```

## Understand Action Validation

### 8. Details of service configuration

The following is the description for action validator on service property:

1. **name**: Super class for validator. Generally we use “UIComponentValidator”
2. **serviceClass**: Custom validator class
3. **Selector**: Action name (we remember 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="CSCPContent3"
            requestor="wt.part.WTPart" />
    </Service>
</Configuration>
```

### 9. Restart the server and check your validation for tab

The checking is same as the <submodel> tag validation.

# Wizard Customization

## Wizard

- Customization of wizard was not changed basically. It is the same methodology with the previous version.
- However, WC10 is using ExtJS, so partially some JavaScript functions were changed to ExtJS functions. For example, button action.
- All wizard JSP pages must be located at “\$WT\_HOME/codebase/netmarkets/jsp” because wizard is controlled and calls several EXT-JS functions and the JavaScript located on the netmarkets directory. (IMPORTANT)

## 2. Architecture of Wizard

### Base Wizard

- Wizard body will be called from action (button, link, menu, navigation). Action call will open the body JSP of wizard page. The body includes all step JSP pages. Wizard is especially using the 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"%>**  
Tag declaration is required for using the JCA component. (Required)
- **<%@ include file="/netmarkets/jsp/components/beginWizard.jspf"%>**
- **<%@ include file="/netmarkets/jsp/components/includeWizBean.jspf"%>**  
It is similar to “begin.jspf” of the general JCA page, but you have to use the above JSPF for wizard.
- **%@include file="/netmarkets/jsp/util/end.jspf%"**  
If it includes the latest file, it is the same JSPF with a general JCA declaration.

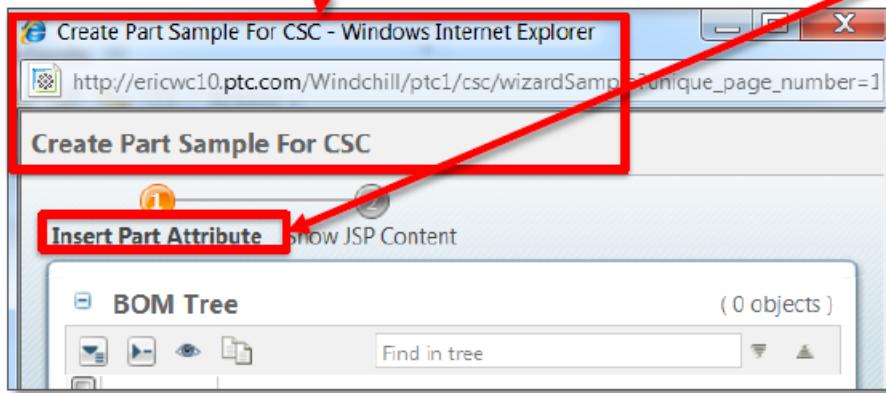
## 2. Architecture of Wizard

### JCA Tag for Wizard

The following is the description of the wizard tag.

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

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



- “**<jca:wizardStep>**” is for each step action. All step Action names must exist on “action.xml”.

## 2. Architecture of Wizard

### Definition of action.xml

The diagram illustrates the architecture of a wizard. On the left, a screenshot of the PTC Integrity interface shows a folder named 'GOLF\_CART' selected in the 'Folders' view. A red arrow points from this selection to the 'Create Part Sample For CSC' wizard window on the right. The wizard window displays a table with one row, labeled '(1 objects)'. Below the table, a large blue box contains the 'action.xml' configuration file. Red arrows point from specific XML elements to their corresponding components in the interface: the 'wizardSamples' action points to the selected folder in the Integrity interface; the 'setProcessStepAttributes' and 'checkAttribute' actions point to the table in the wizard window; and the 'wizard\_step' command points to the table in the wizard window.

```

<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="cscWizard">
    <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>

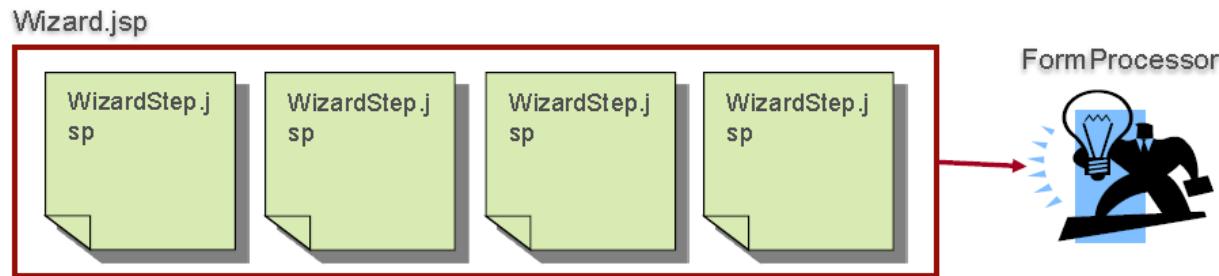
```

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

## 2. Architecture of Wizard

### Wizard Architecture

After completing basic environment configuration, you should design a detailed wizard page which includes wizard page design and step design. The following are details of basic understanding of wizard .

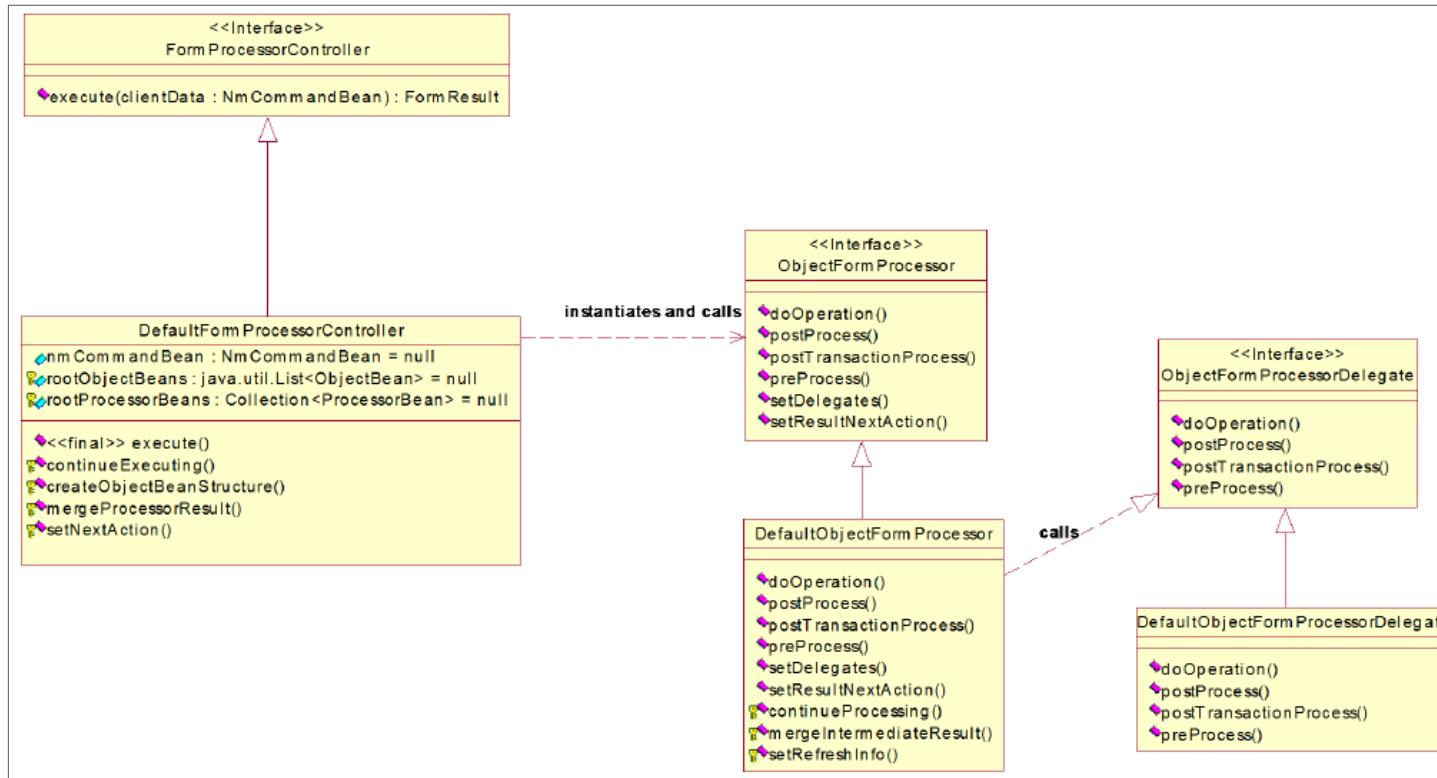


1. First of all, you have to design all steps on the base wizard and decide which wizard steps should be called JSPs.
2. **Basically all step pages will be loaded on the session when loading basic wizard page.** It seems like all step pages include pages of basic wizard, although all step pages have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

## 2. Architecture of Wizard

### Modeling of Form Process

Generally form process includes PRE/DO/POST function. With this function, you can control the form process based on time period. When finishing the wizard process, the processor class will return form result. Using this result, the wizard will work to complete/close and show warning or stopping.



## 2. Architecture of Wizard

### Sample of Form Process

Generally wizard form process will be extended from the “**DefaultObjectFormProcess**” class.

- 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.

```
public class CSCWizardSampleProcess extends CreateObjectFormProcessor {  
  
    /**  
     * @param args  
     */  
    public FormResult doOperation(NmCommandBean nmcommandbean, List list) throws WTEexception {  
        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 WTEexception {  
        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  
    }  
}
```

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



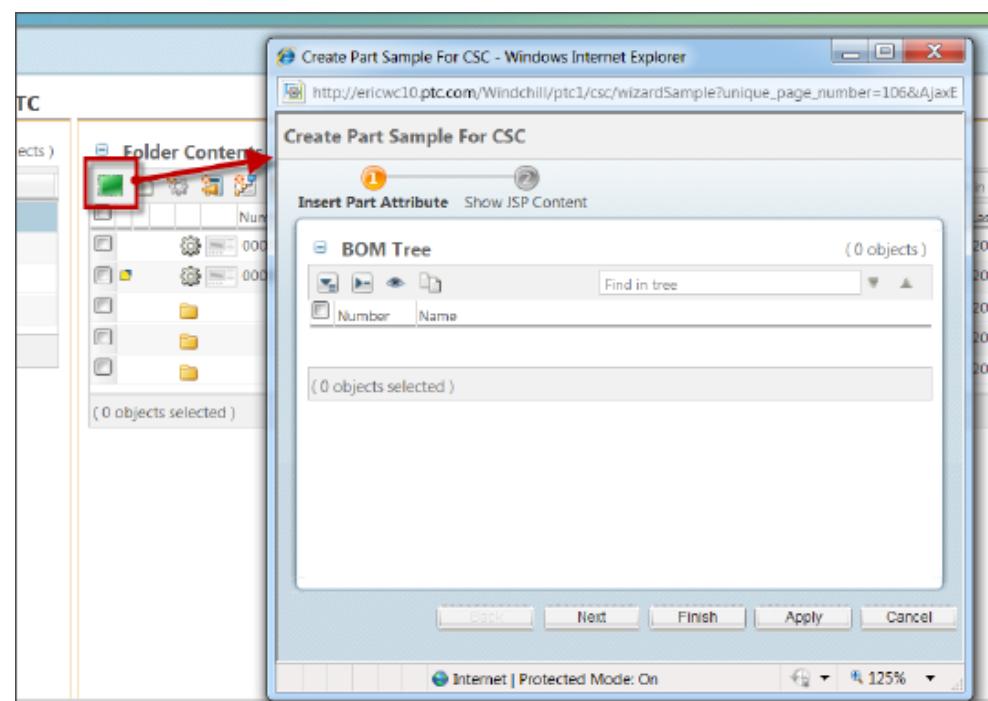
### 3. Exercise 1 — Creating a Simple Wizard

#### Design of Wizard Customization

- Add a new toolbar button for calling the wizard. The button also includes FormProcessor. (Add one action and modify the toolbar action.)
- Create one wizard page (JSP page).
- Create two step pages (JSP or MVC page).
- Add two actions for step page (add actions) – because wizard page calls 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.



### 3. Exercise 1 — Creating a Simple Wizard

#### Configure Step and Base Wizard

##### 1. Design actions for step and button

- On the wizard configuration, you have to set the execution class. And you also have to take an action for each 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 for calling back form processor.
- **<action name="show\_mvc\_content"> <action name="show\_jsp\_content">**
  - Action configuration for step page

### 3. Exercise 1 — Creating a Simple Wizard

#### Add Button on Folder Toolbar

##### 2. Add one button on Folder toolbar

- For starting the wizard, we have to use one button or menu link. This button or menu will include wizard executing process when all steps of wizard are completed.

\$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 the whole block named “**folderbrowser\_toolbar\_actions**” from “**FolderManagement-actionModels.xml**”
- Add custom action on the folder browser toolbar for calling the wizard.  
`<action name="wizardSamples" type="cscwizard" shortcut="true"/>`  
`<action name="separator" type="separator"/>`
- Don't touch other existing actions.

### 3. Exercise 1 — Creating a Simple Wizard

#### Create Wizard Page

##### 3. Create wizard page

The wizard page includes step page configuration, and also the wizard page and the step page will be merged on a same one page internally. So if you have constants or whole environment, you can set those on wizard page design. The following is just a 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 a 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.

### 3. Exercise 1 — Creating a Simple Wizard

#### Create a Wizard Step

##### 4. Create a wizard step

- This sample has two wizard steps. One will call MVC. Another will call a JSP page. In action configuration, we already set “netmarkets.project.list.table.” It is OOTB MVC function, so it doesn’t need to create a 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-actions.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 a step page is included in the wizard body, the step also has same architecture like a wizard body.

### 3. Exercise 1 — Creating a Simple Wizard

#### Create a Resource Bundle

##### 5. Create a Resource Bundle

We set the custom resource bundle for displaying action name, so we have to make a 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;

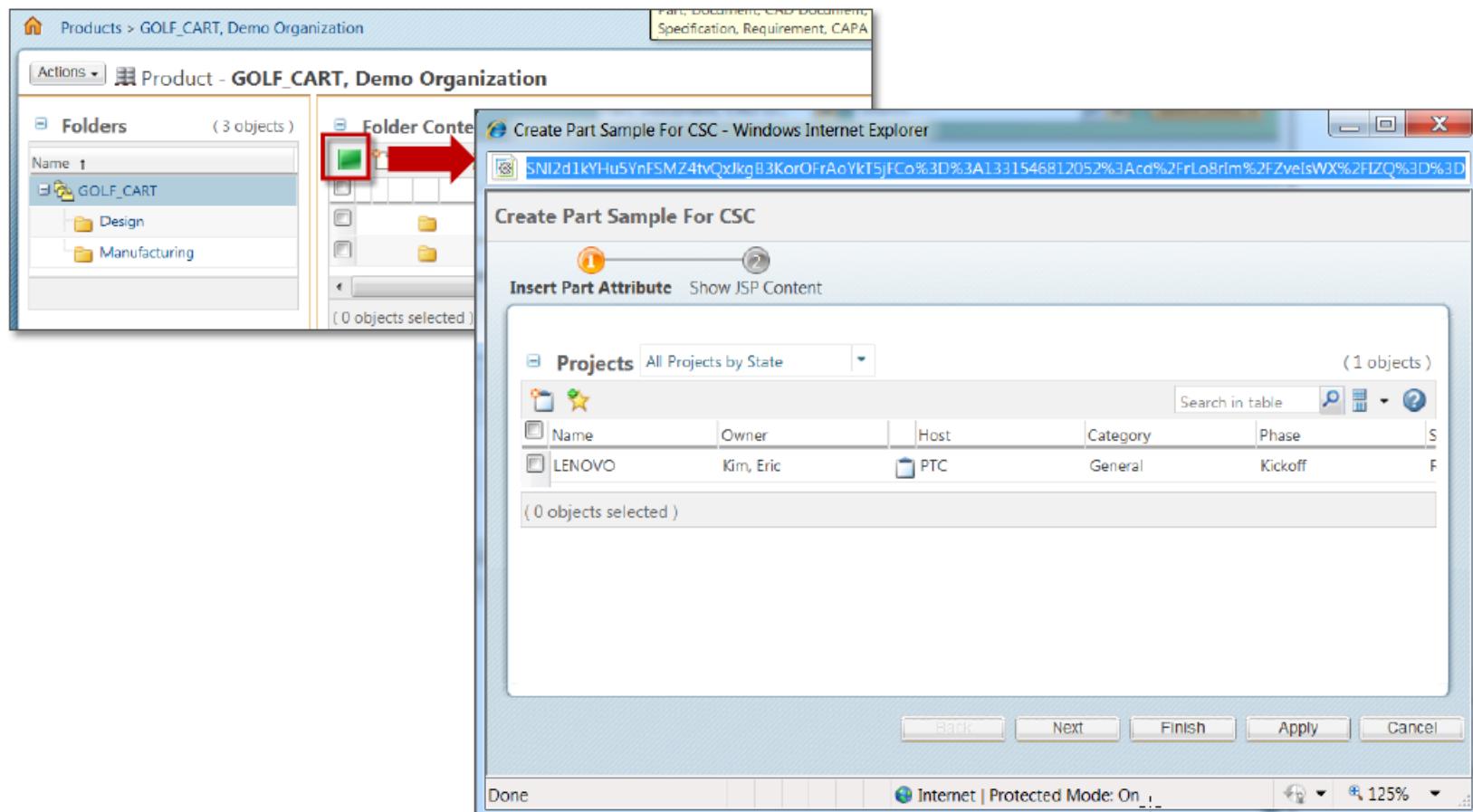
import wt.util.resource.RBComment;
import wt.util.resource.RBEntry;
import wt.util.resource.RBUUID;
import wt.util.resource.WTListResourceBundle;

@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. Exercise 1 — Creating a Simple Wizard

Create a Wizard Step

#### 6. Restart the Service and check the result.



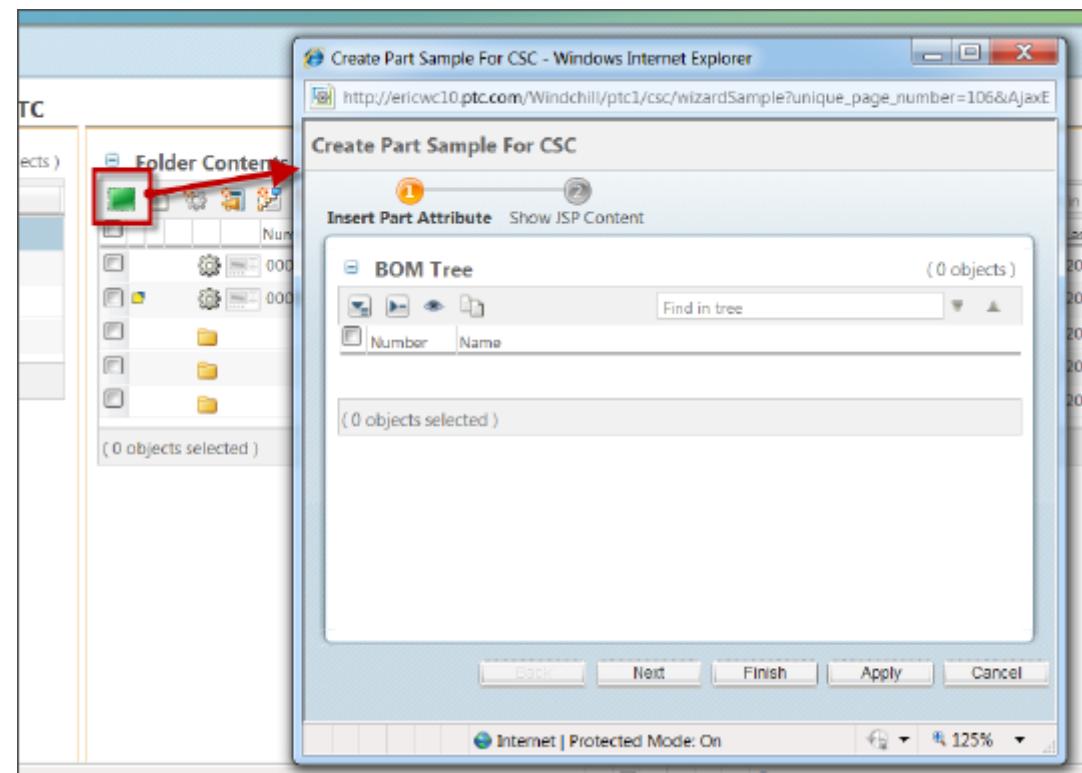
### 3. Exercise 2 — Simple Wizard for Creating a Part

#### Design of Wizard Customization

- When creating a part, you simply need **number, name, and target container**.
- So we will **add one more icon in the folder** because the folder is managed by container. And also when opening the wizard, the container information is transferred from folder to wizard (present 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. Exercise 2 — Simple Wizard for Creating a 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.

### 3. Exercise 2 — Simple Wizard for Creating a Part

#### Add Button on the Folder Toolbar

##### 2. Add one button on the Folder toolbar.

Add button on the 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 the whole block named “**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 existing actions.

### 3. Exercise 2 — Simple Wizard for Creating a Part

#### Create a Wizard Page

##### 3. Create a wizard page.

Create a 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 is just a design for wizard.
- Add step for input attributes.

### 3. Exercise 2 — Simple Wizard for Creating a Part

#### Create a Wizard Step

##### 4. Create a wizard step (Using HTML Tags).

Create a 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" %>



|                                                                                                                                                                                 |                                                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|
| *Number:</td> <td align="left" class="tabledatafont">&amp;nbsp; <input &gt;="" <="" id="number" maxlength="30" name="null__number__textbox" size="10" td="" type="text"/> </td> | &nbsp; <input &gt;="" <="" id="number" maxlength="30" name="null__number__textbox" size="10" td="" type="text"/> |
| Name:</td> <td align="left" class="tabledatafont">&amp;nbsp; <input &gt;="" <="" id="name" maxlength="100" name="null__name__textbox" size="20" td="" type="text"/> </td>       | &nbsp; <input &gt;="" <="" id="name" maxlength="100" name="null__name__textbox" size="20" td="" type="text"/>    |


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

If using HTML tag for input, the tag name must be the following style for being read at NmCommandBean.  
“**null\_\_[attribute name]\_\_textbox**”

If you want to use a combo box, you must set a name like the following for NmCommandBean.

“**null\_\_[attribute name]\_\_combobox**”

### 3. Exercise 2 — Simple Wizard for Creating a Part

#### Create a Wizard Page

##### 4. Create a wizard step (Using Wrapper Tag).

Wrapper tag is already explained in Customization-1 document. Look at 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" %>


```

### 3. Exercise 2 — Simple Wizard for Creating a Part

#### Create a Resource Bundle

##### 5. Create a 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 Exercise 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. Exercise 2 — Simple Wizard for Creating a Part

#### Create a Form Process

##### 6. Create a form process for creating a part.

\$WT\_HOME/codebase/ext/csc/jca/wizard/CSCCreateWizardSampleProcess.java

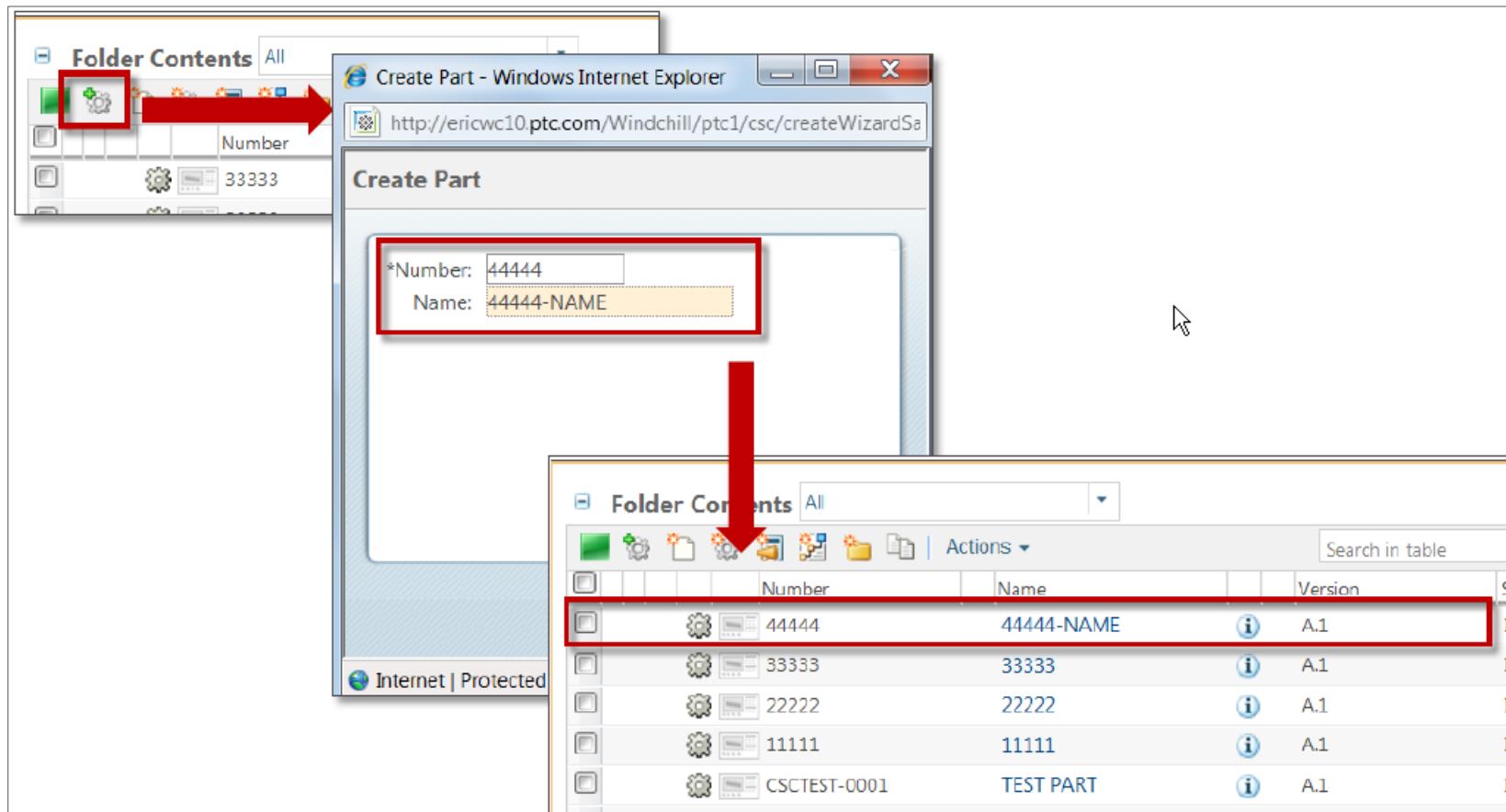
```
package ext.csc.jca.wizard;
import java.util.HashMap;
...
public class CSCCreateWizardSampleProcess extends DefaultObjectFormProcessor {
    @Override
    public FormResult doOperation(NmCommandBean bean, List list) throws WTEException {
        FormResult result = new FormResult();
        try {
            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(WTEException wte) {
            result.setStatus(FormProcessingStatus.FAILURE);
            return result;
        }
        result.setStatus(FormProcessingStatus.SUCCESS);
        return result;
    }
}
```

### 3. Exercise 2 — Simple Wizard for Creating a Part

PTC®

Create a Wizard Step

7. Restart the Service and check the result.

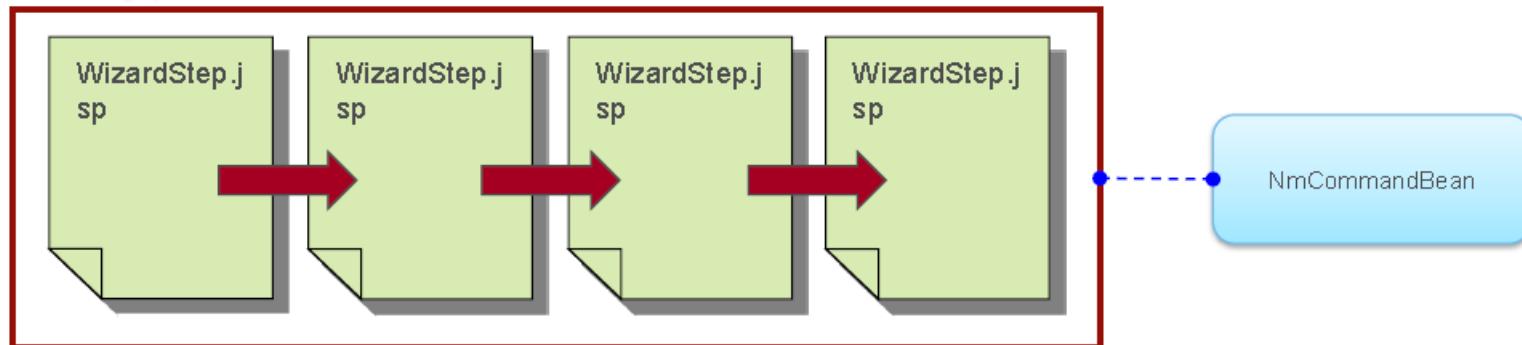


### 3. Exercise 3 — Receiving Previous Step Info

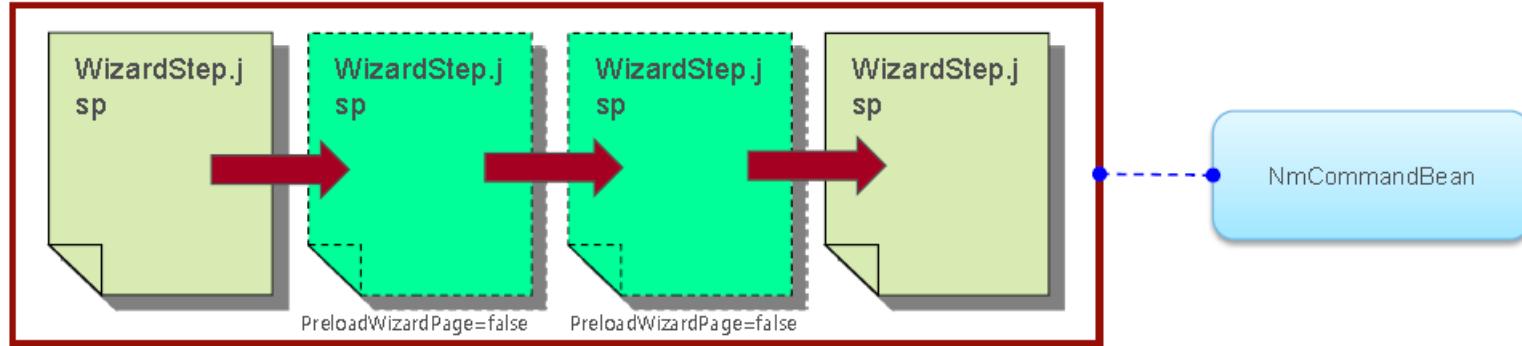
#### Understand the Preload Option

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

Wizard.jsp



Wizard.jsp



### 3. Exercise 3 — Receiving Previous Step Info

#### Understand the Preload Option

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

The diagram illustrates the process of receiving previous step info. It starts with a 'WizardStep.jsp' page, which has a red arrow pointing to a code snippet. This code snippet shows Java code that prints the command bean and its parameter map. A yellow callout box notes that you must pick up the correct function based on previous pages. Below the code is a screenshot of a 'Create Part Sample For CSC' browser window. The browser shows the same page as the JSP, with a red box highlighting the input field 'name=aaaa'. A red arrow points from this input field to the 'elementName' variable in the code snippet. Another yellow callout box explains that Windchill does not use the input name by the component engine, so you must use the 'indexOf' function to find it.

```
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];  
    }  
    out.println("elementName=" + elementName);  
}
```

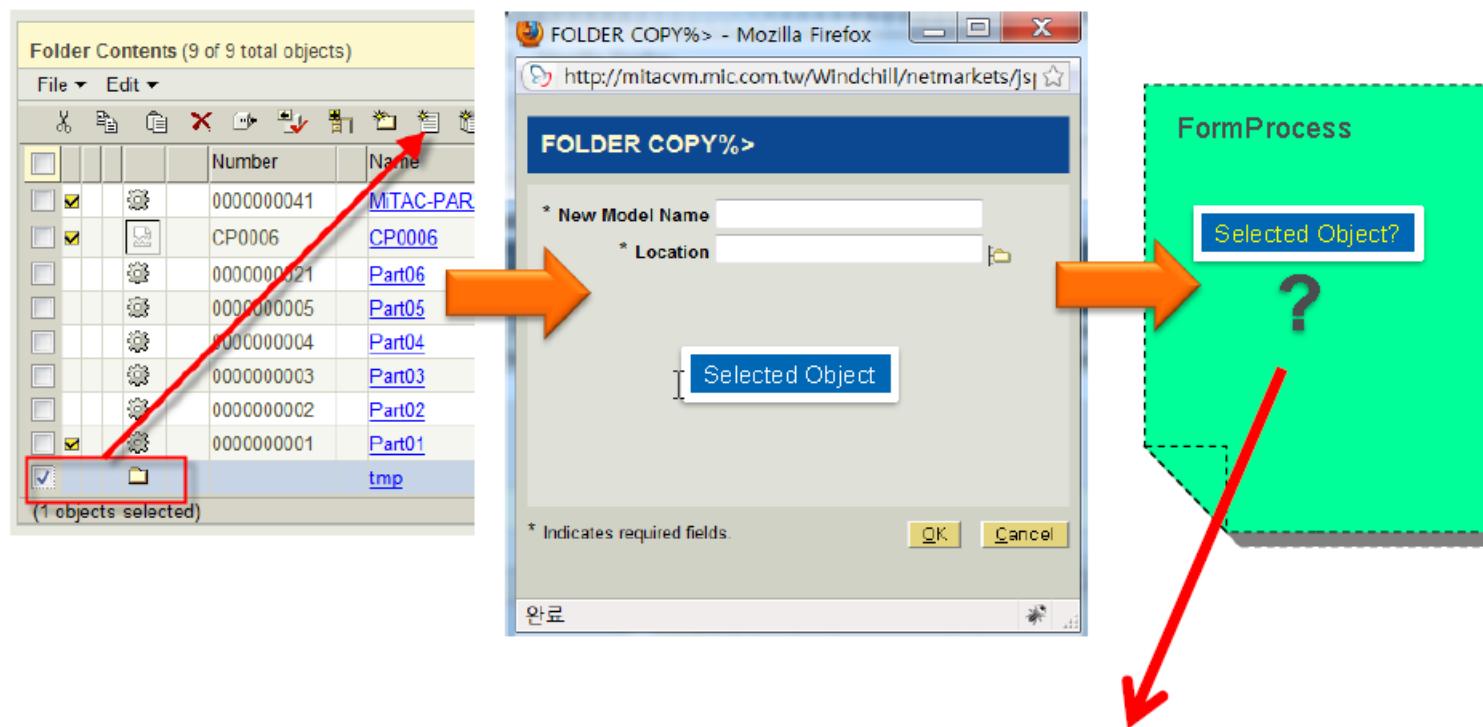
You have to pick up correct function  
Base on previous pages, you have to change  
the function for each of components.

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. Exercise 4 — Get Object Instance on form 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 the form processor if we need to get selected object instance on form processor. Look at the following sample.



`ArrayList aList = nmcommandBean.getNmOidSelectedInOpener();`

# Picker Customization

## Understanding WC10 Picker

1. Customization of picker methodology wasn't changed for WC10.
2. Picker is not exactly customization. It is mostly about configuration.
3. Windchill core has several picker components. You just choose a usable picker for correct requirement. (If you want to make a special picker component, it is a very complex customization. If possible, avoid that customization.)
4. The following is the readied system component generally used.
  - User Picker
  - Organization Picker
  - Context Picker
  - Item Picker
  - Type Picker
  - Participant Picker
5. For the above picker, WC10 uses 3 kinds of tags as follows.
  - <%@ 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"%>

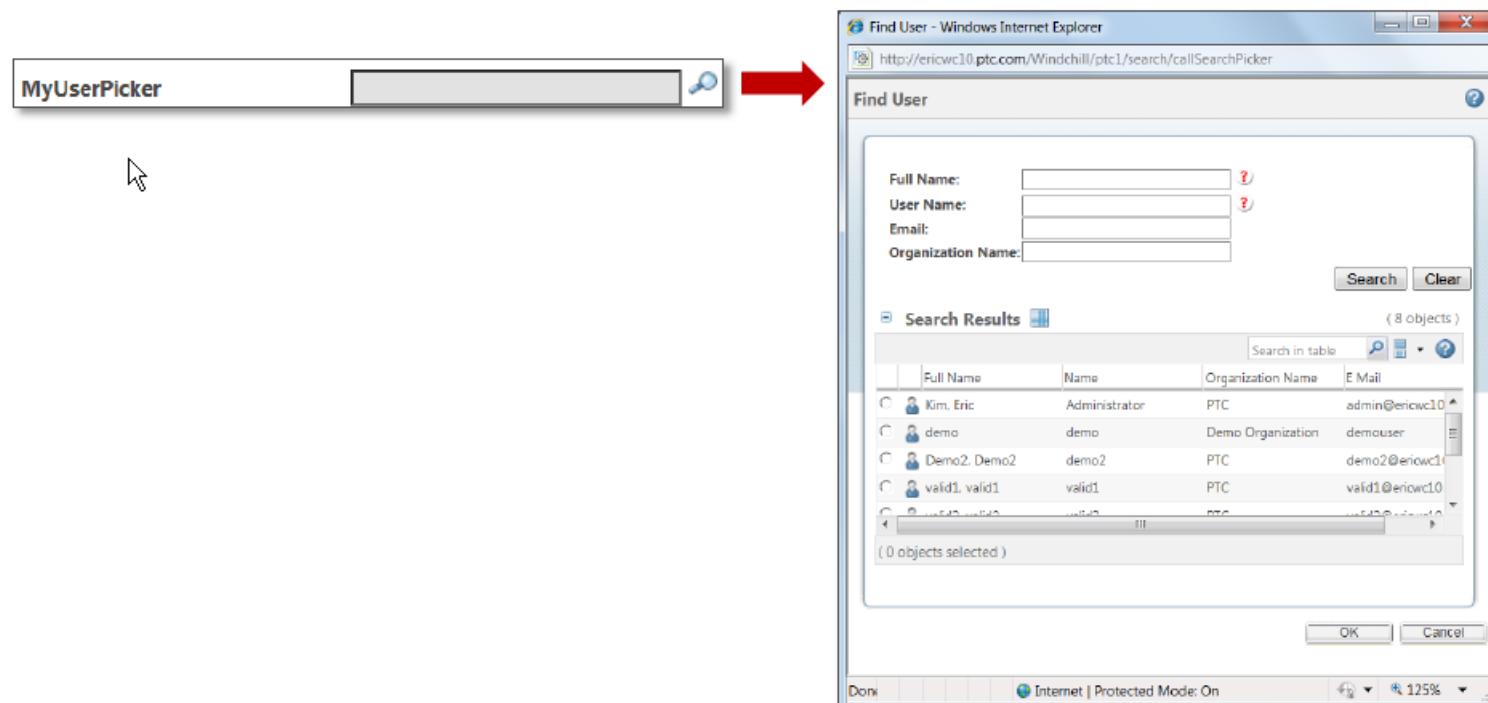
## 2. Review Picker Components

### 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 a certain user. In this case, you can have a user picker and then through this you can select the user and pass it onto the search criteria to perform search for parts.

- **User Picker**

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



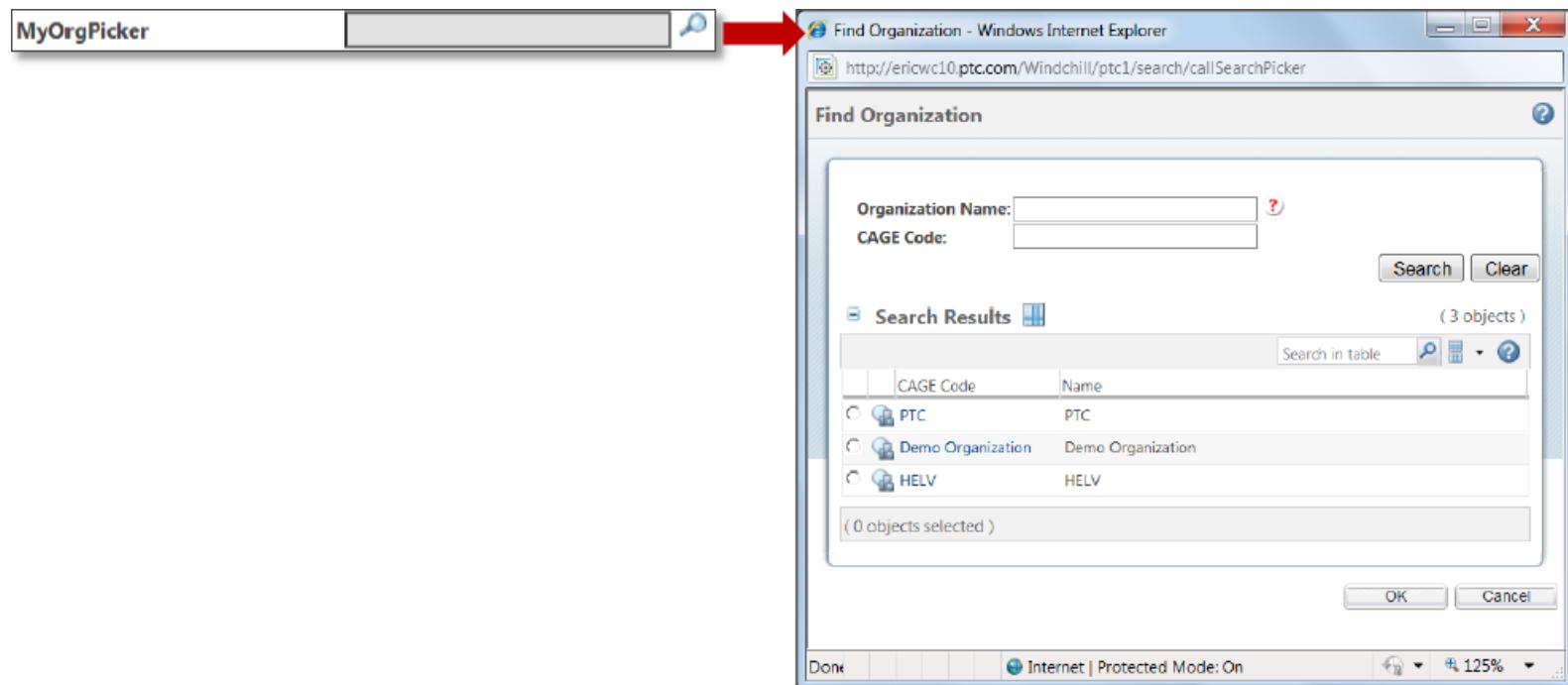
## 2. Review Picker Components

### 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 a particular organization. In this case, you can have an organization picker and then through this you can select the organization and pass it onto the search criteria to perform search for parts.

- **Organization Picker**

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



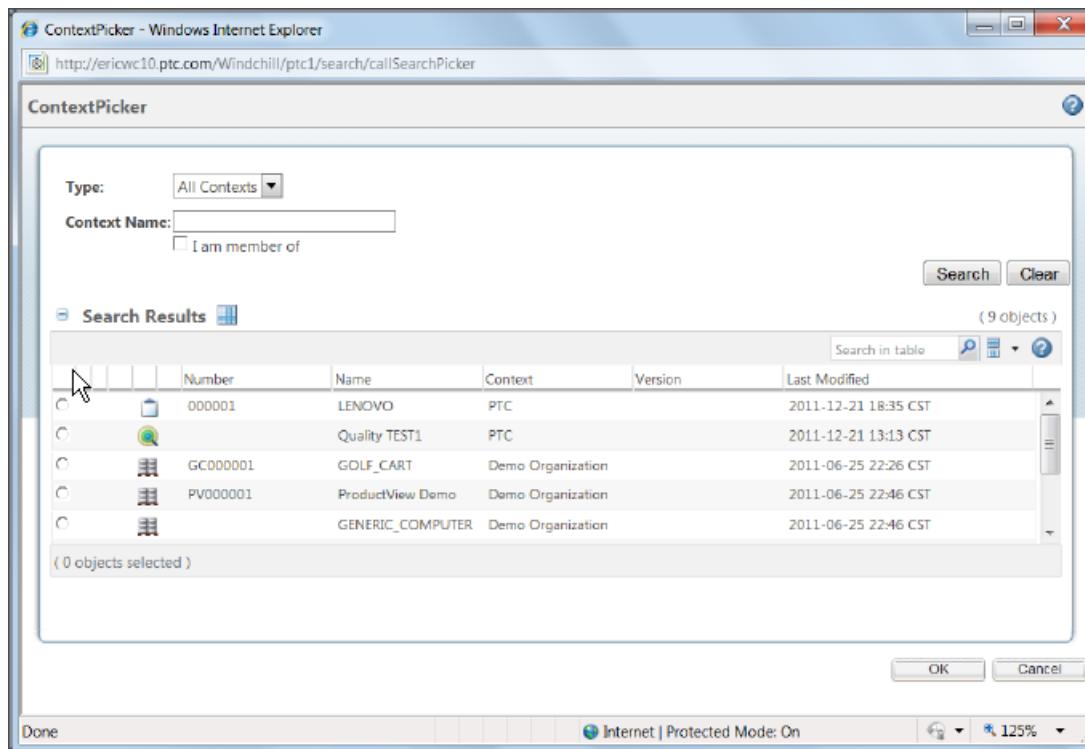
## 2. Review Picker Components

### Context Picker

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

- **Context Picker**

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



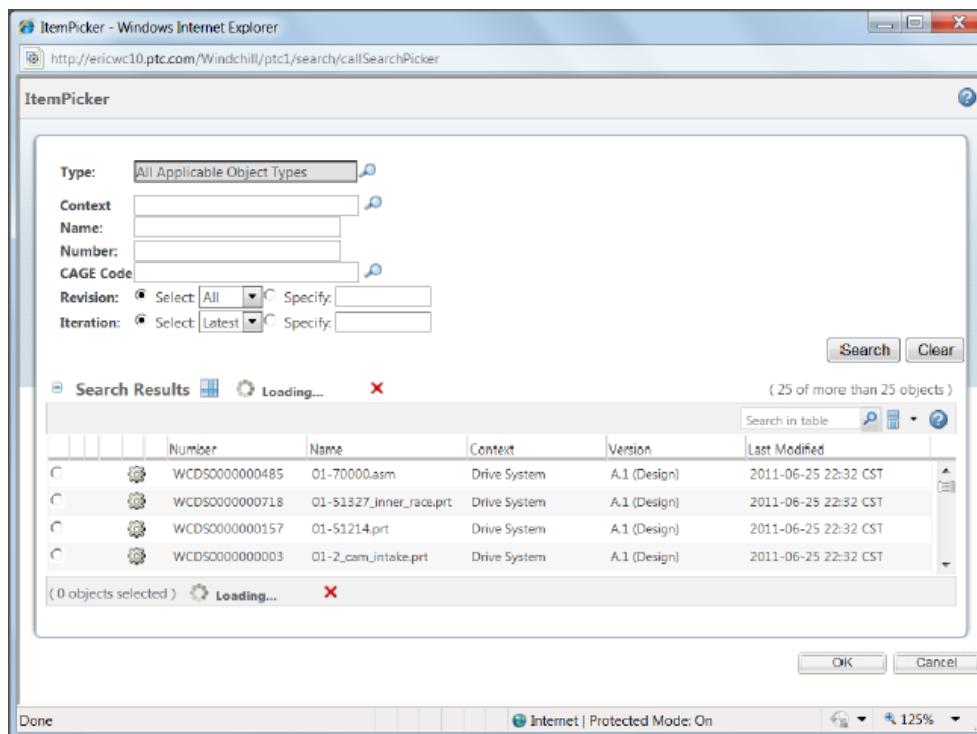
## 2. Review Picker Components

### 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

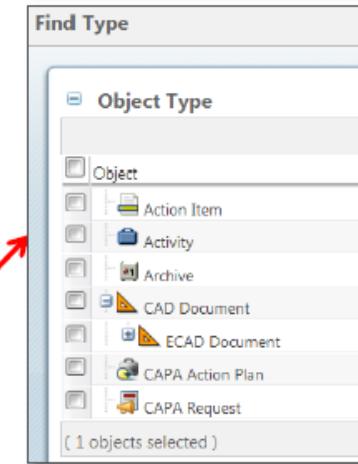
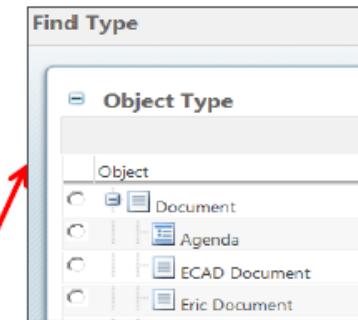
Type Picker Common Component is to be used either for display or for assignment of type-able items. For example, in a search application to select the type of objects that you are interested to do a search on or in a create application to create an item of a specific type. It can be used to display the 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 -->
<tr>
  <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>

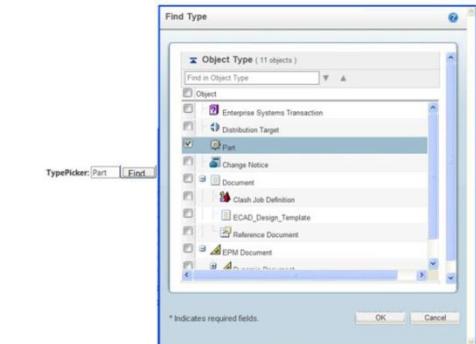
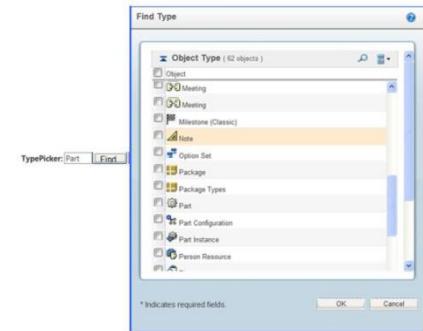
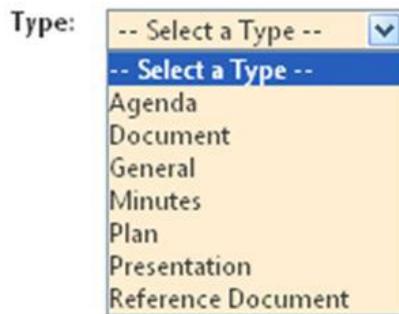
```



## 2. Review Picker Components

### Format of Type Picker

Type Picker has several types of format like the following: Tree Type, Table Type, and Drop-Down



- Type Picker in a “Drop-Down” format**

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

- Type Picker in a “table” format**

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

- Type Picker in a “tree” format**

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

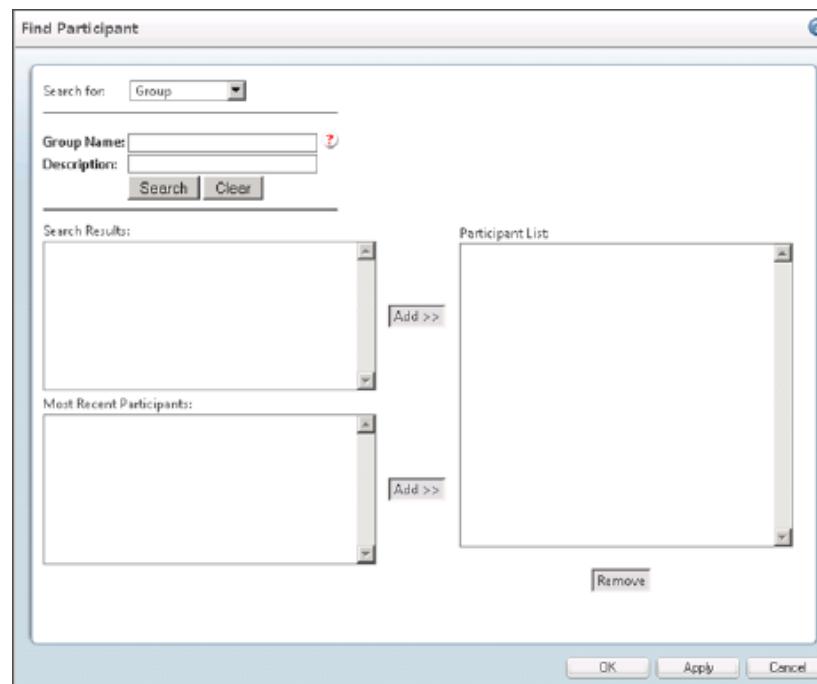
## 2. Review Picker Components

### Participant Picker

The current way of picking of participants is limited to Users and Groups and is not consistent across Windchill. The new Participant Picker Common Component gives a 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 which can be used to narrow down the search.

- **Participant Picker**

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



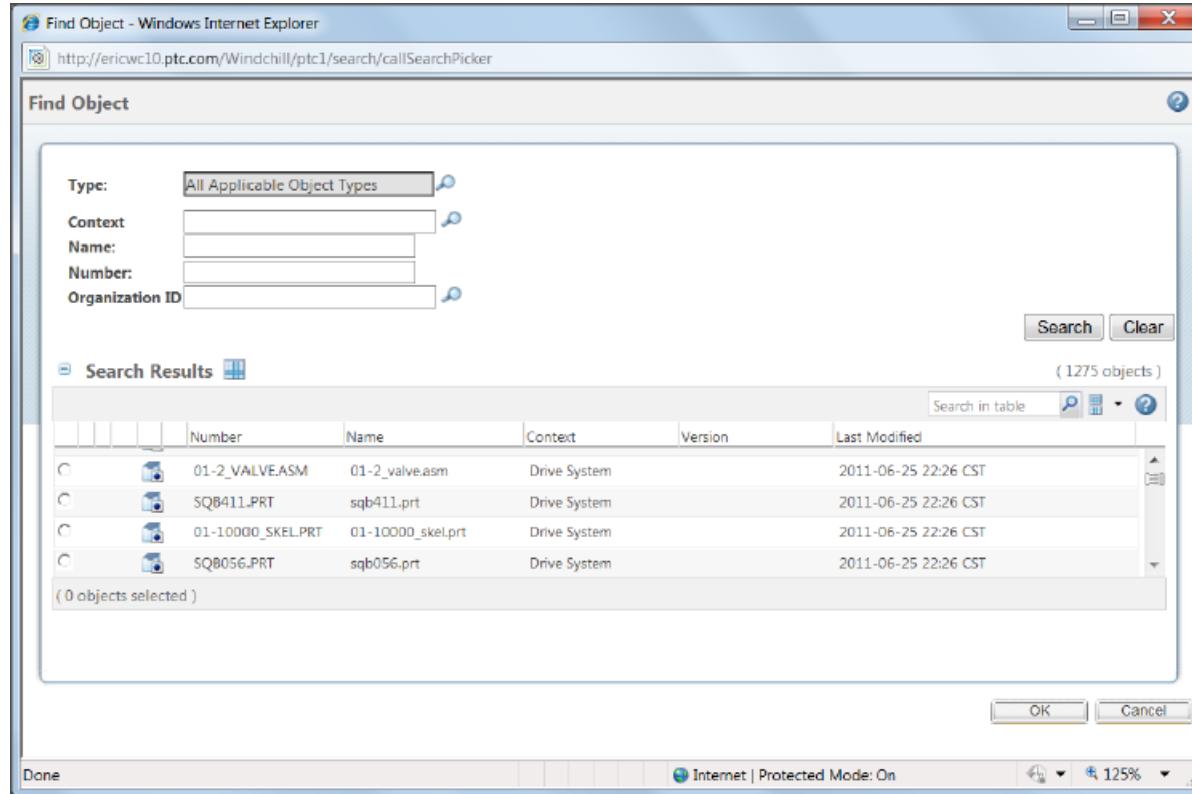
## 2. Review Picker Components

### Item Master Picker

Item Master picker just tries to find the mastered object.

- **Item Master Picker**

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

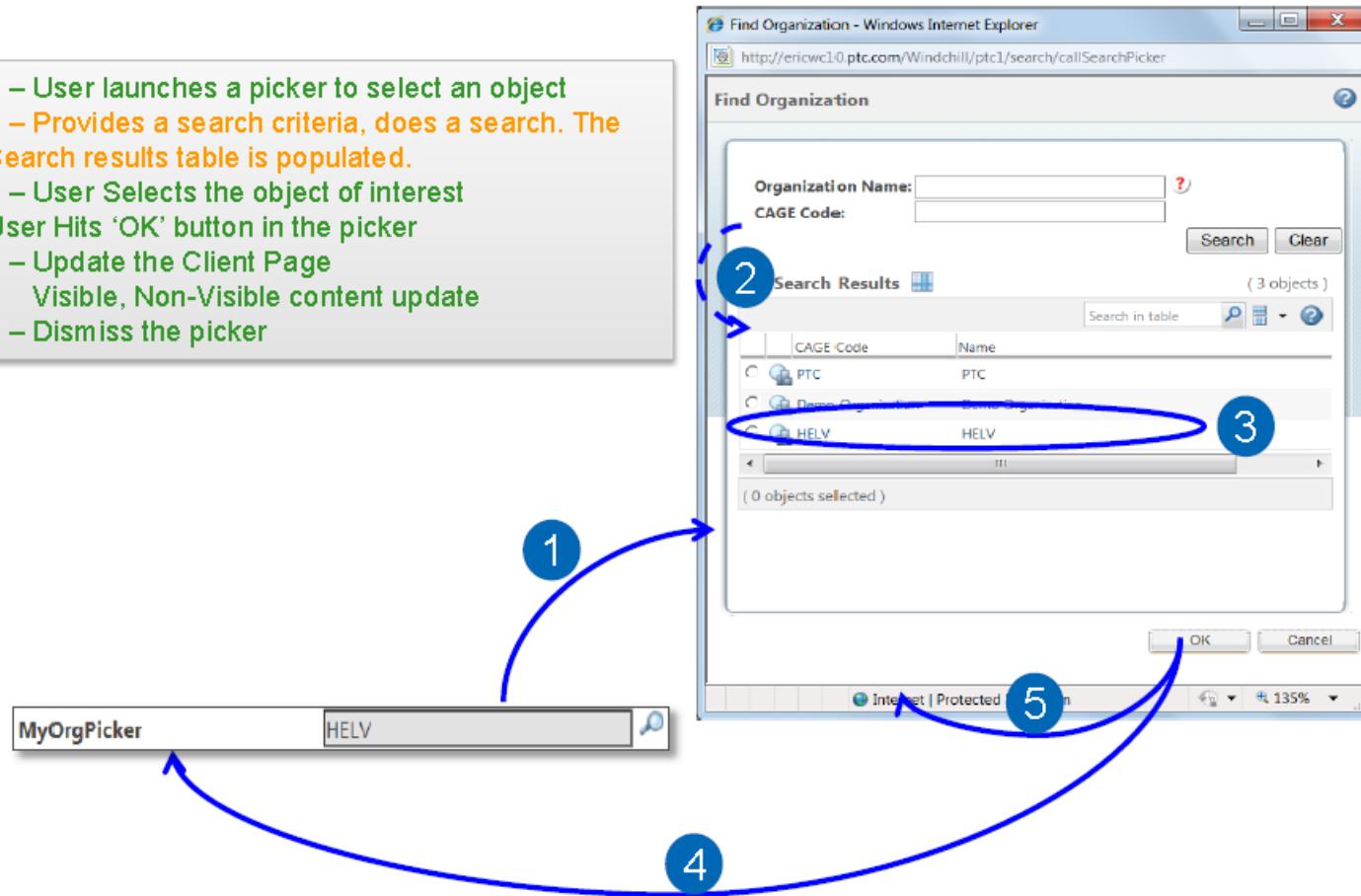


### 3. Understand the Picker Process

#### Picker Process

The following is the process of basic pickers. All of JCA picker uses the same architecture.

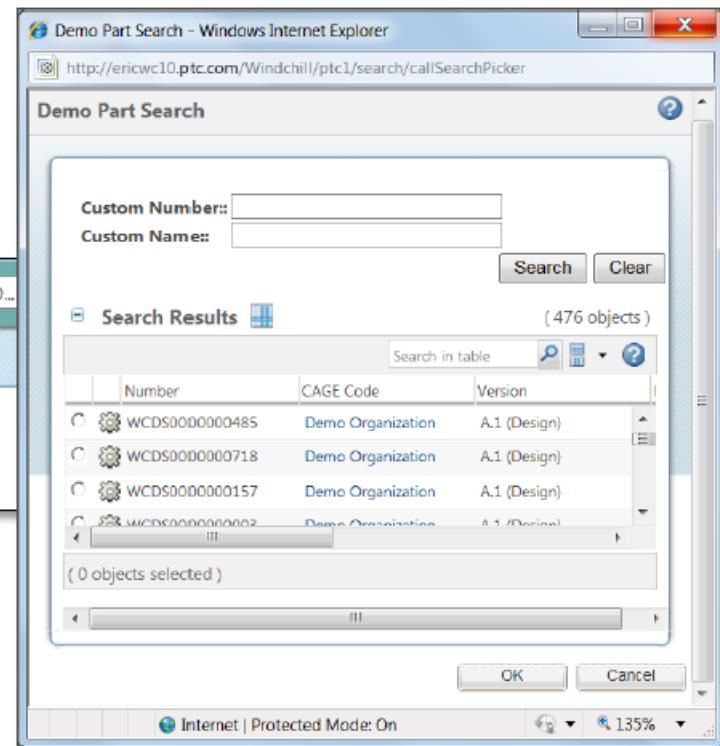
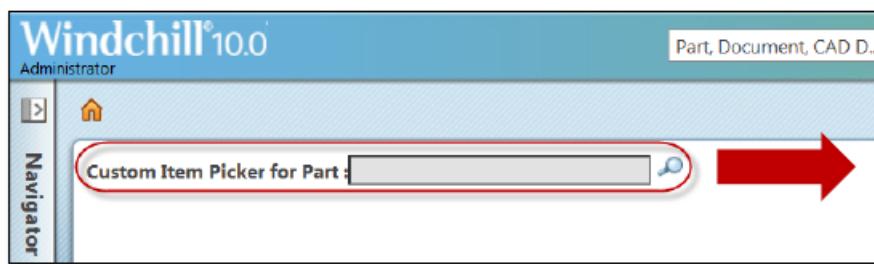
- 1 – User launches a picker to select an object
- 2 – Provides a search criteria, does a search. The Search results table is populated.
- 3 – User Selects the object of interest
- 4 – Update the Client Page  
Visible, Non-Visible content update
- 5 – Dismiss the picker



## 4. Exercise 1 — Item Picker

### Design of Item Picker

- Item Picker can be designed by the customer in cases such as:
  - Search Criteria
  - Target Search Object
- Other pickers are designed by the system, so you just call target picker using tags.



# 4. Exercise 1 — Item Picker

## 1. Create a Search Criteria Set – Item Picker

Before you create a picker pop-up window, you should set search attribute set.

- The Set file is located in “**\$WT\_HOME/codebase**” and the file name is “**pickerAttributes.xml**.”
- We will use the following sample for searching WTPart, so the objectType is set as “**wt.part.WTPart**.”
- The attribute name should be used for object’s model or IBA logical name. The name can be checked in the Property Report.
- Each Display name is registered in ResourceBundle, so the file name is the bundle’s key.

The diagram illustrates the configuration of an Item Picker. On the left, a code snippet from `pickerAttributes.xml` is shown:

```
<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>
```

A red box highlights the `<ComponentID id="cscPartPicker">` tag, which is mapped to the `componentId=cscPartPicker` attribute in the `SearchCriteriaAttributes` section. A red box also highlights the `<Attributes>` sections, which map to the input fields in the search dialog. A callout box points to the `componentId` attribute with the text: “This id will use in tag ‘componentId=cscPartPicker’”. On the right, a screenshot of a Windows Internet Explorer window titled “Demo Part Search” shows a search dialog with two text input fields: “Custom Number:” and “Custom Name:”, both highlighted with a red box. Below the input fields are “Search” and “Clear” buttons, and at the bottom are “OK” and “Cancel” buttons. The status bar at the bottom of the browser window indicates “Internet | Protected Mode: On” and “135%”.

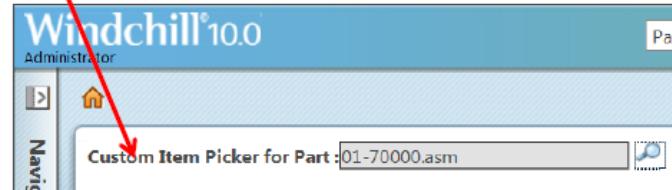
## 4. Exercise 1 — Item Picker

### 2. Create a Picker Page (1)

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

- Creating a Picker base (parent page) is same as creating a JCA page, so you should set and include all taglib.
- 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"%>  
This is the component in  
"pickerAttributes.xml" 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"%>
```



# 4. Exercise 1 — Item Picker

## 2. Create a 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 of same architecture as a JCA page, so you should set and include all taglib.
- 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"%>

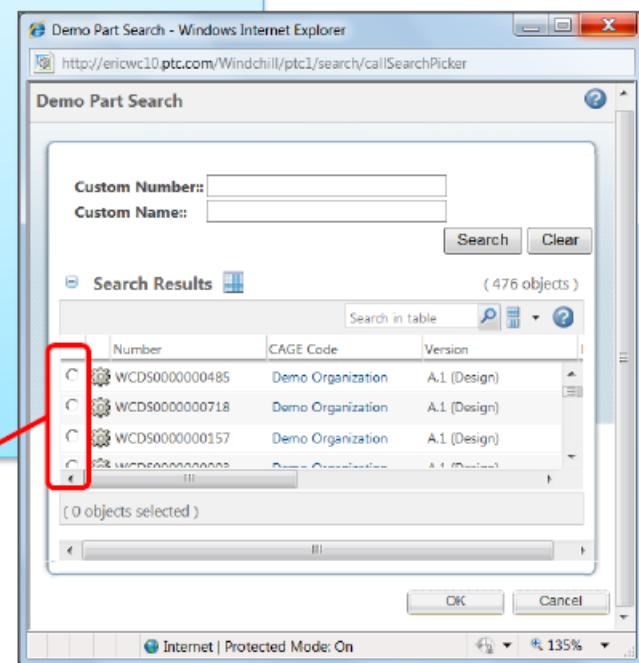
<%@ include file="/netmarkets/jsp/util/begin.jspf"%>

<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.jspf"%>
```

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.



### 3. Picker Resource Bundle

The resource bundle is located at “\$WT\_HOME/src/com/ptc/windchill/enterprise/search/client.” Resource Bundle file name is ‘searchClientResource.java.’

- Copy all java codes to eclipse compiled directory.
- If you are not using the existing locale, add the 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. Exercise 1 — Item Picker

### 4. Picker Callback Function

The previous JSP page should include the JavaScript function for a picker callback. The picker pop-up page will return to parent page's JS function about selected object information. The picker will call the function when you create the picker tag within pickerCallBack's name.

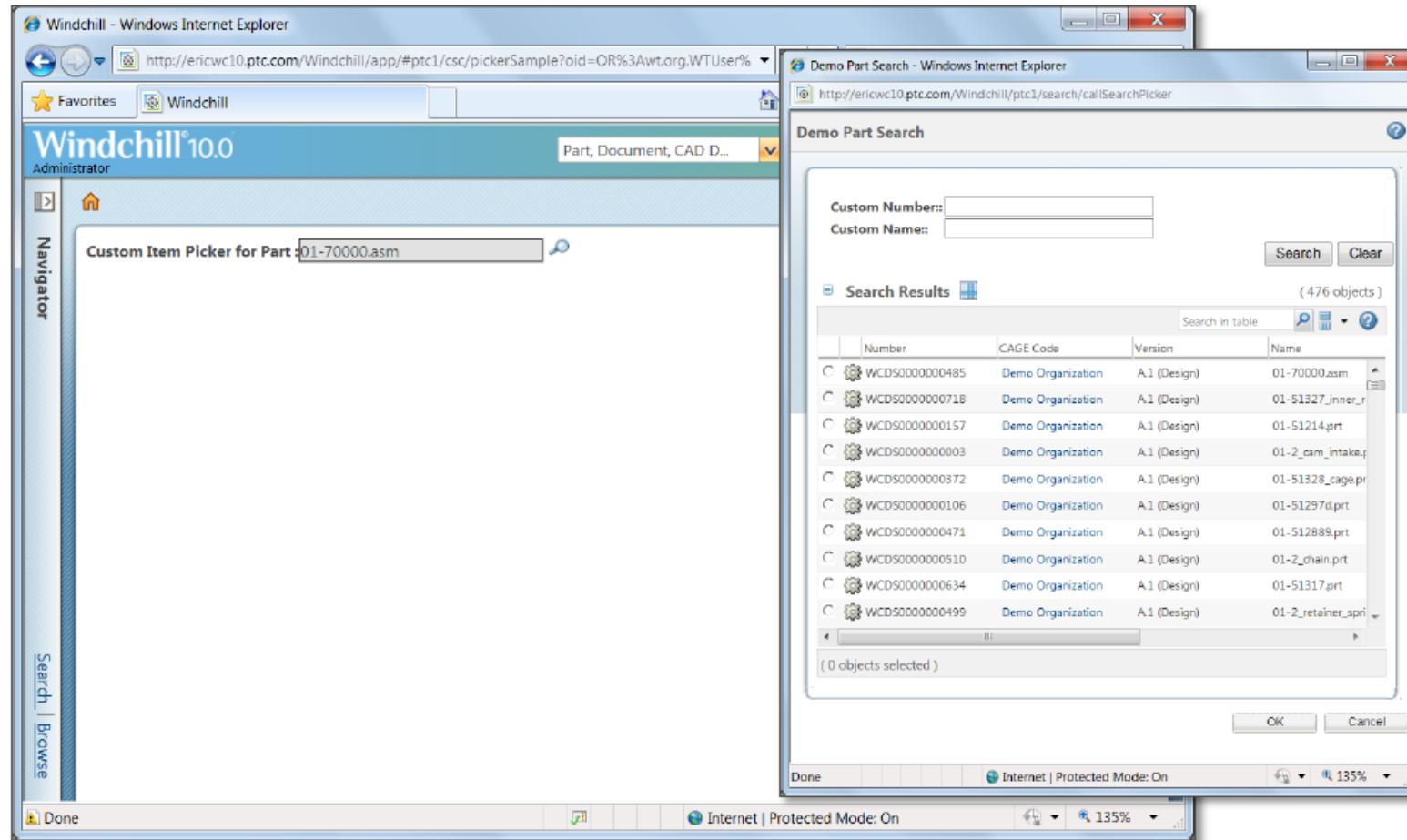
```
<script>
function partPickerCallback (objects, pickerID)
{
    document.getElementById(pickerID
+"$label$").setAttribute("value",objects.pickedObject[0].name);
    document.getElementById(pickerID
+"$label$__old").setAttribute("value",objects.pickedObject[0].name);
}
</script>
<wctags:itemPicker id="cscPickerForPart" label="Custom Part Picker"
    pickerTitle="Demo Part Search" showVersion="false"
    objectType="wt.part.WTPart" showTypePicker="false"
    componentId="cscPartPicker" pickerCallback="partPickerCallback"
/>
```

## 4. Exercise 1 — Item Picker

PTC®

## 5. Result

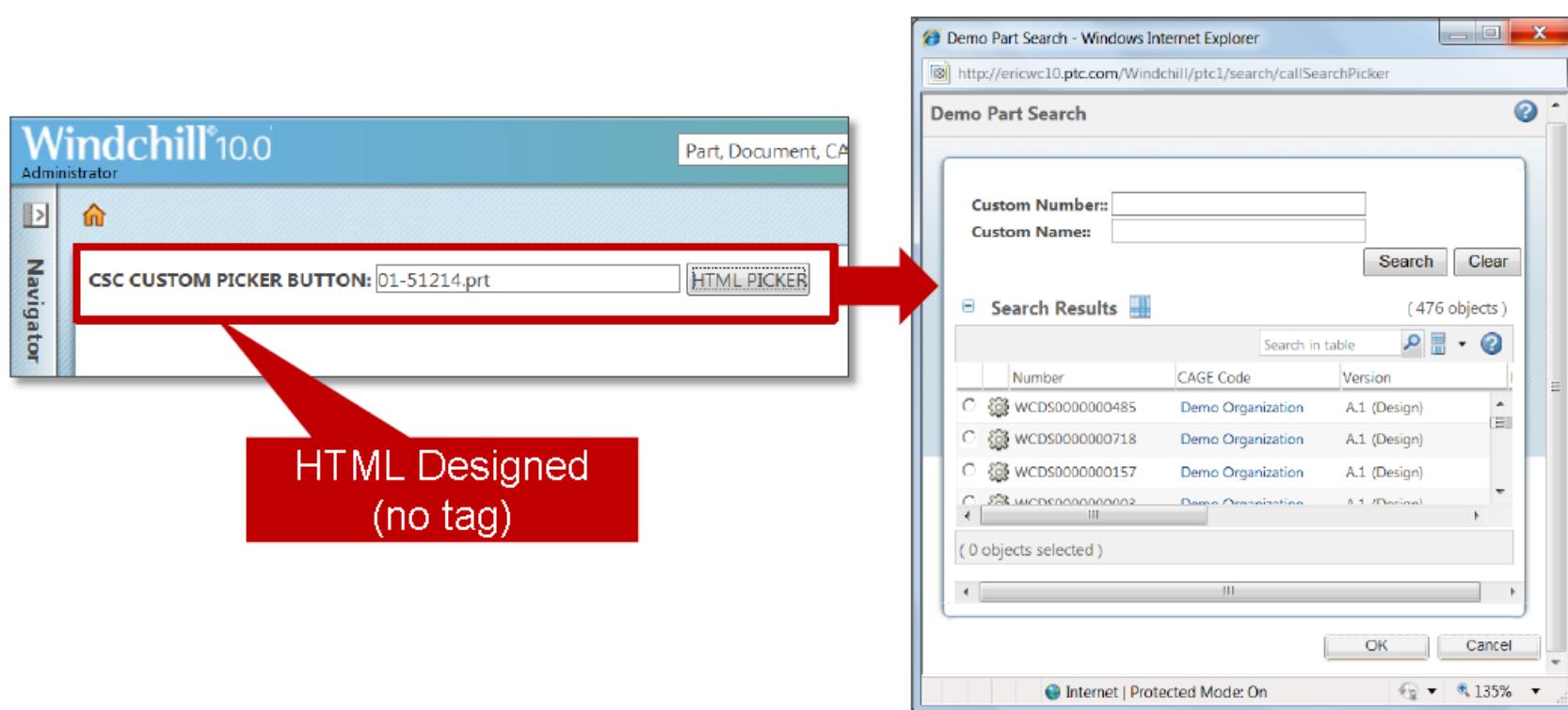
The following is the result.



## 4. Exercise 2 — Custom Picker Button

### Design of Custom Picker Button

- Sometimes customer wants to change the picker button and the input field for customer standard.
- In this case, we cannot use Windchill tag rendering.
- This exercise will explain how custom button calls picker window.



## 4. Exercise 2 — Custom Picker Button

### 1. Create the Main Page

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



The screenshot shows the Windchill interface with a custom picker button. The button is labeled "HTML PICKER" and is located next to a text input field. A red box highlights this area. To the right, a callout box contains the following text:

- 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.

Below the screenshot is the corresponding HTML and JavaScript code:

```
<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?<b>portlet=poppedup</b>',"",width=500 height=400);
}
</script>
```

A red box highlights the URL in the window.open() function: "/Windchill/ptc1/csc/htmlPicker?<b>portlet=poppedup</b>". A red arrow points from this highlighted text to the callout box containing the parameter explanation.

## 4. Exercise 2 — Custom Picker Button

PTC®

### 2. Create a Picker Page

Create your picker JSP page using “wctags.” This is almost same as using the previous material. But the tags do not display input and button. It just display the picker main page directly.

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

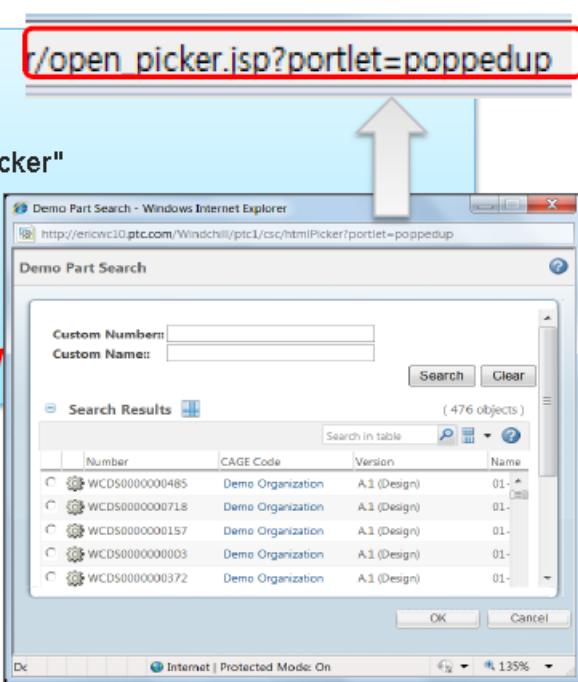
```
<%@ taglib prefix="wctags" tagdir="/WEB-INF/tags" %>
<%@ include file="/netmarkets/jsp/components	beginWizard.jspf"%>

<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"%>
```

CSC CUSTOM PICKER BUTTON:

HTML PICKER



## 4. Exercise 2 — Custom Picker Button

### 3. Callback Function on the Main Page

The Callback JavaScript is present on the parent page, and the function's name will be used in the pickerCallback parameter in "wctags."

- The callback function is the 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

# 1. Introduction

## 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

<b>Name:</b>	PTC Corporate Headquarters
<b>Address:</b>	140 Kendrick Street Needham MA 02494
<b>Phone:</b>	(781) 370-5000
<b>Fax:</b>	(781) 370-6000

Read-Only attribute Panel

<b>Number:</b>	(Generated)
<b>Name:</b>	<input type="text"/>
<b>Assembly Mode:</b>	Separable
<b>Source:</b>	Make
<b>View:</b>	Design
<b>Default Trace Code:</b>	Untraced
<b>Default Unit:</b>	each
<b>Gathering Part:</b>	No
<b>Phantom:</b>	No
<b>Life Cycle Template:</b>	(Generated)
<b>Team Template:</b>	(Generated)
<b>Location:</b>	<input checked="" type="radio"/> Autoselect Folder I/GOLF_CART3 <input type="radio"/> Select Folder <input type="text"/> GOLF_CART3/Design

Editable advanced attribute panel

* Organization ID:	<input type="text"/> Demo Organization
* Configurable:	<input type="button" value="No"/>
* Create as End Item:	<input type="button" value="No"/>

Editable attribute panel

<b>System</b>				
State:	In Work - Released - Canceled			
Context:	GOLF_CART			
Life Cycle Template:	Basic			
Created By:	wcadmin	Modified By:	demo	
Created On:	2011-01-19 09:47 CST		Last Modified:	2011-01-19 13:08 CST

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 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
primaryAttributes	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 to create a manually configured panel, you can extend an existing Java builder class and make new MVC class for attribute display. (Next Page)

## 2. Implementation of Attribute Panel

### 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:

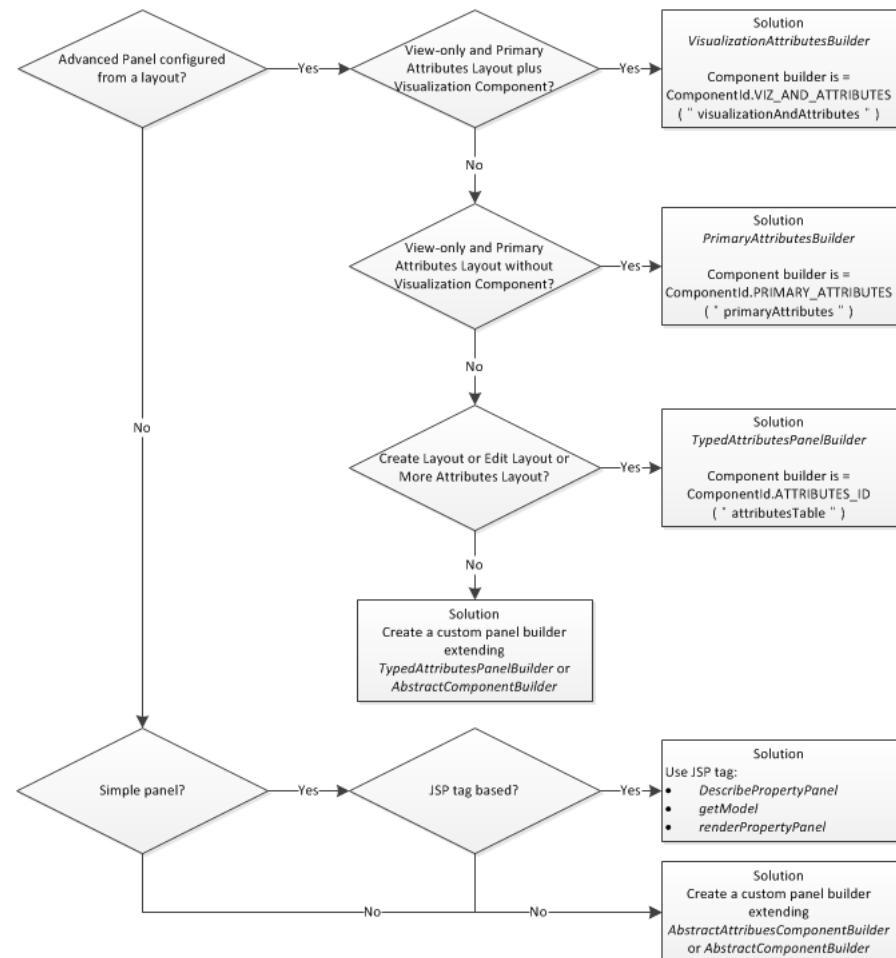
VisualizationAttributesBuilder

PrimaryAttributesBuilder

TypedAttributesPanelBuilder

AbstractAttributesComponentBuilder

AbstractComponentBuilder



### 3. Exercise 1 — Simple Attributes Panel

#### Design a Simple Attribute Panel

Simply we will try to show attributes like the following design.

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



### 3. Exercise 1 — Simple Attributes Panel

#### 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
}

}
```

### 3. Exercise 1 — Simple Attributes Panel

#### Design Attributes Panel

##### 2. Design attributes panel (buildComponentConfig)

- This function is a design area for attribute panel.

```
@Override
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTEException {
    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.

### 3. Exercise 1 — Simple Attributes Panel

#### 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 WTEException {
    // 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;
}
```

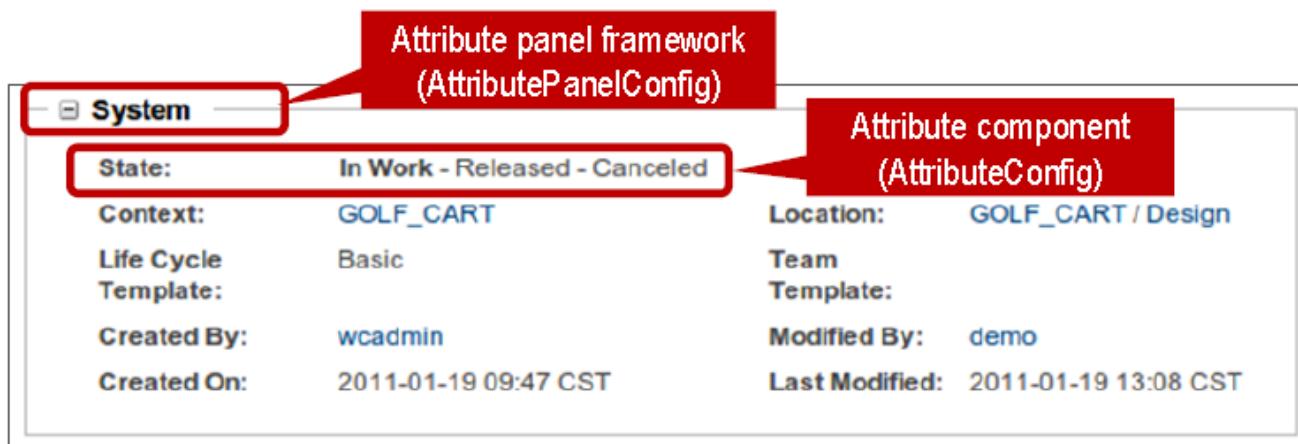
### 3. Exercise 1 — Simple Attributes Panel

#### 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;  
}
```

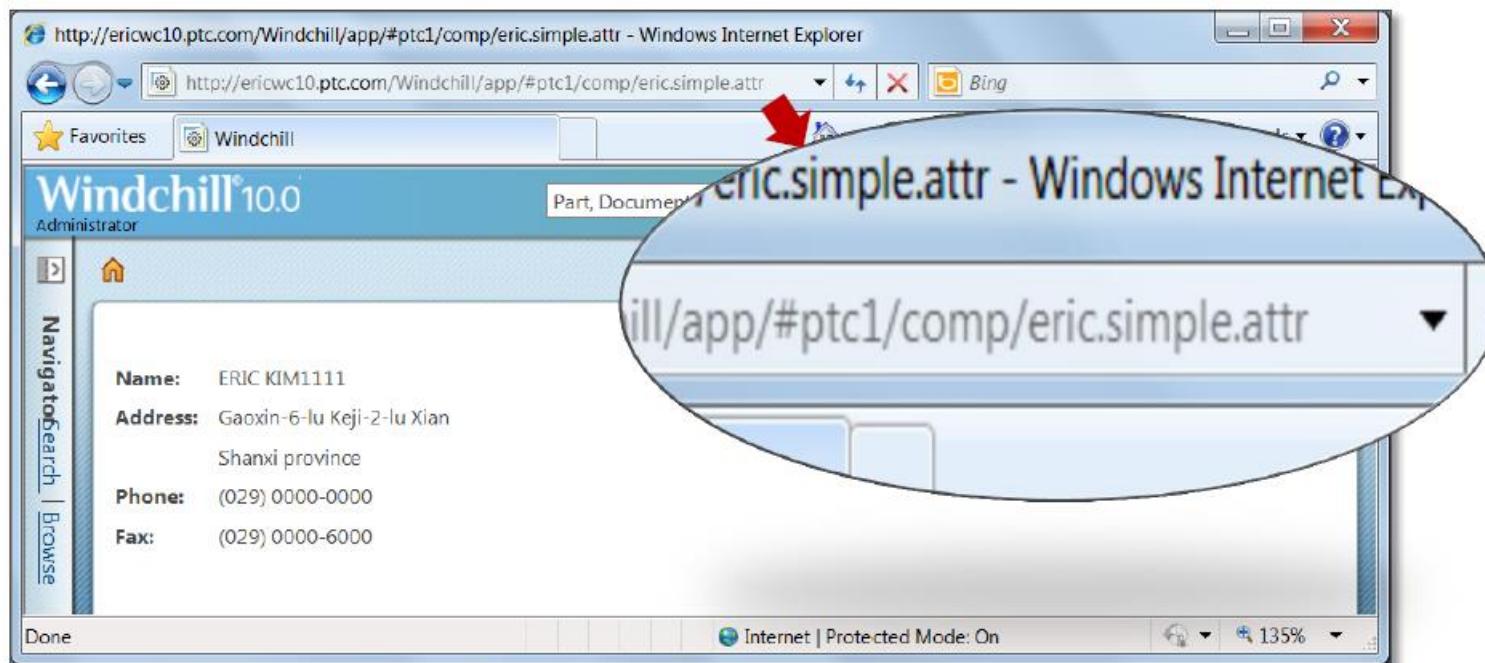


### 3. Exercise 1 — Simple Attributes Panel

#### Test Result

##### 5. Restart all services and tests

- We have created a new MVC class for attributes panel, so we can call MVC component directly on Web browser. If the MVC component does not work, you have to check the following:
  - Is the package of existing MVC component registered MVC Dispatcher?
  - Doesn't the MVC component have an error?



### 3. Exercise 2 — Group Framework

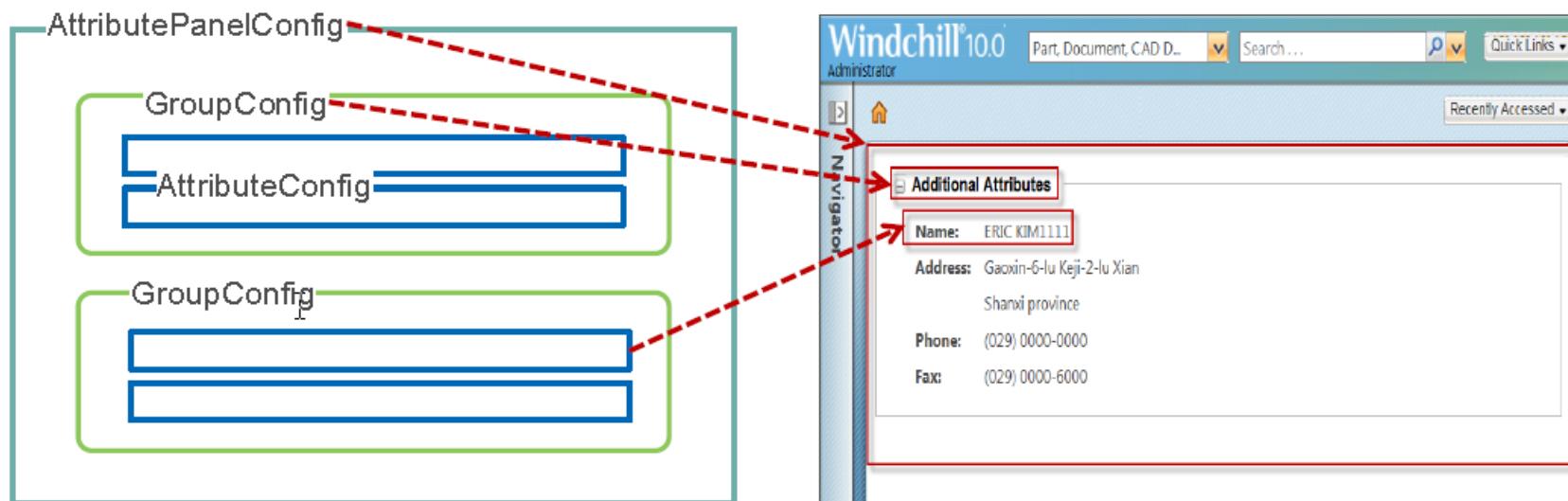
#### Design of Group Framework

In Exercise 1, we deleted the group framework, so Exercise 2 will explain how to add group framework.

- Show Group framework

When you want to add a group framework, it is very similar to the previous exercise. But we have to add one more component for group framework. Before adding a 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



### 3. Exercise 2 — Group Framework

#### Design an Attributes Panel

##### 1. Design an attributes panel (`buildComponentConfig`)

We will update Exercise 1 source code for adding group framework. Copy Exercise 1's class and modify new class name and MVC id. We do not need to change other areas except the “`buildComponentConfig`” function.

```
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTEException {
    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

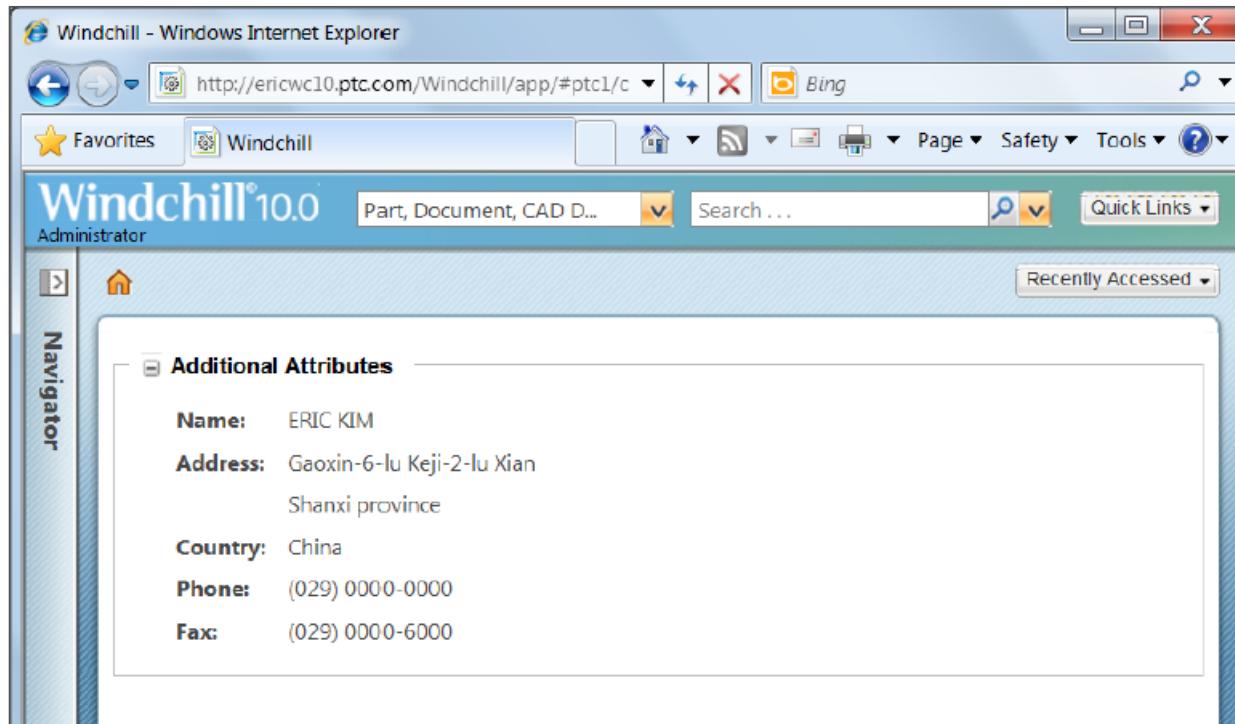
### 3. Exercise 2 — Group Framework

#### Test Result

##### 2. Restart all services and tests

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

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

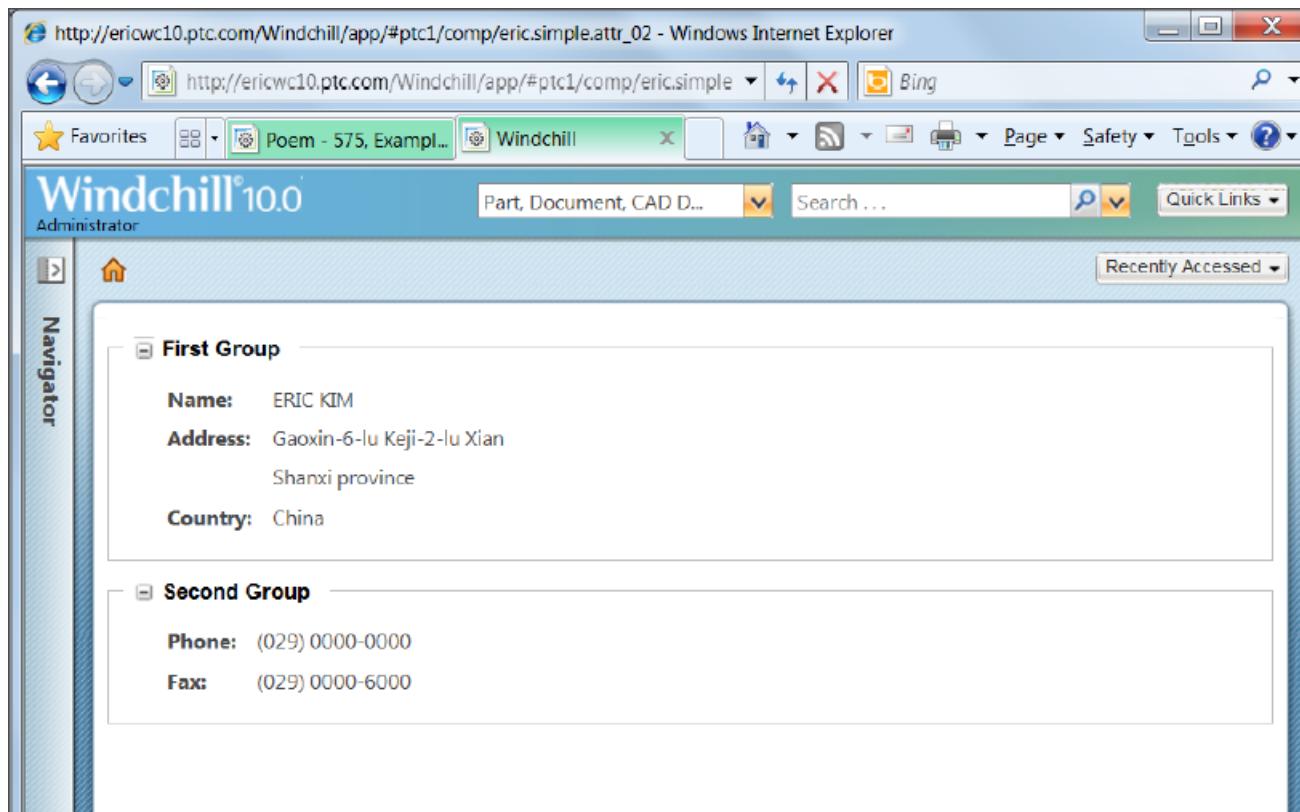


### 3. Exercise 3 — Multi-Group Attributes Panel

#### Design a Multi-attributes Panel

Using Exercise 2, we will make a multi-group attributes panel.

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



### 3. Exercise 3 — Multi-Group Attributes Panel

#### Design an Attributes Panel

##### 1. Design an attributes panel (buildComponentConfig)

We will update Exercise 2 source code for making multi-group framework. Copy Exercise 2's class and modify new class name and MVC id. We do not need to change other areas except the “**buildComponentConfig**” function.

```
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTEException {
    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;
}
```

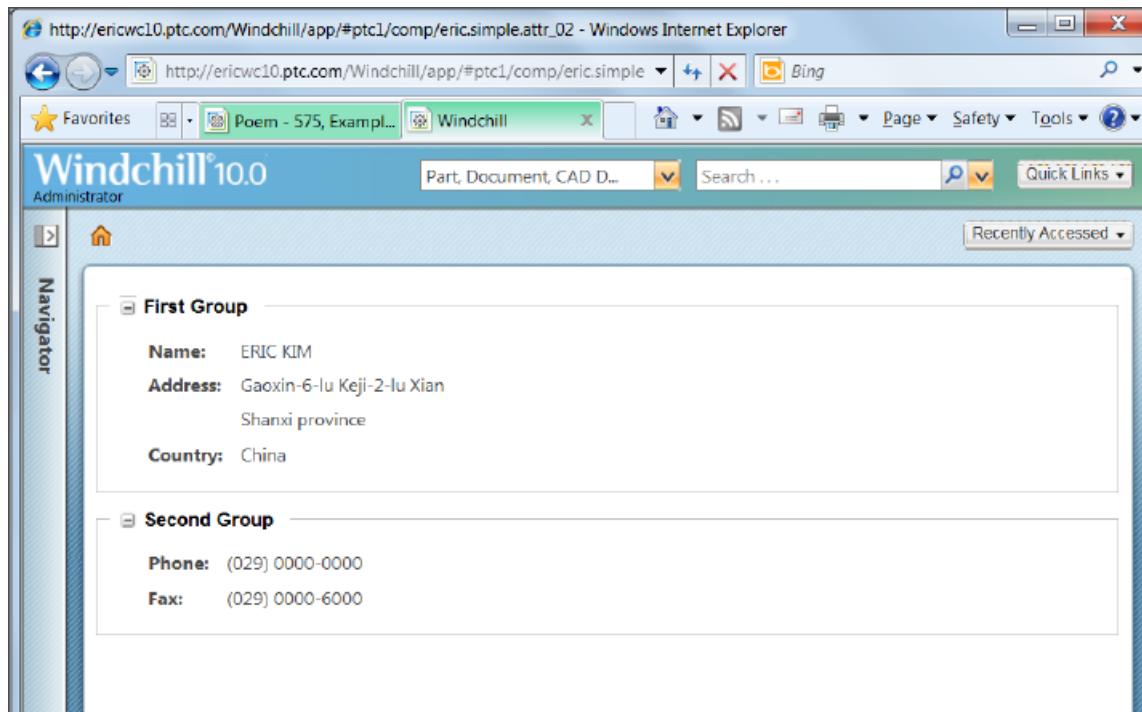
### 3. Exercise 3 — Multi-Group Attributes Panel

#### Test Result

##### 2. Restart all services and tests

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

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

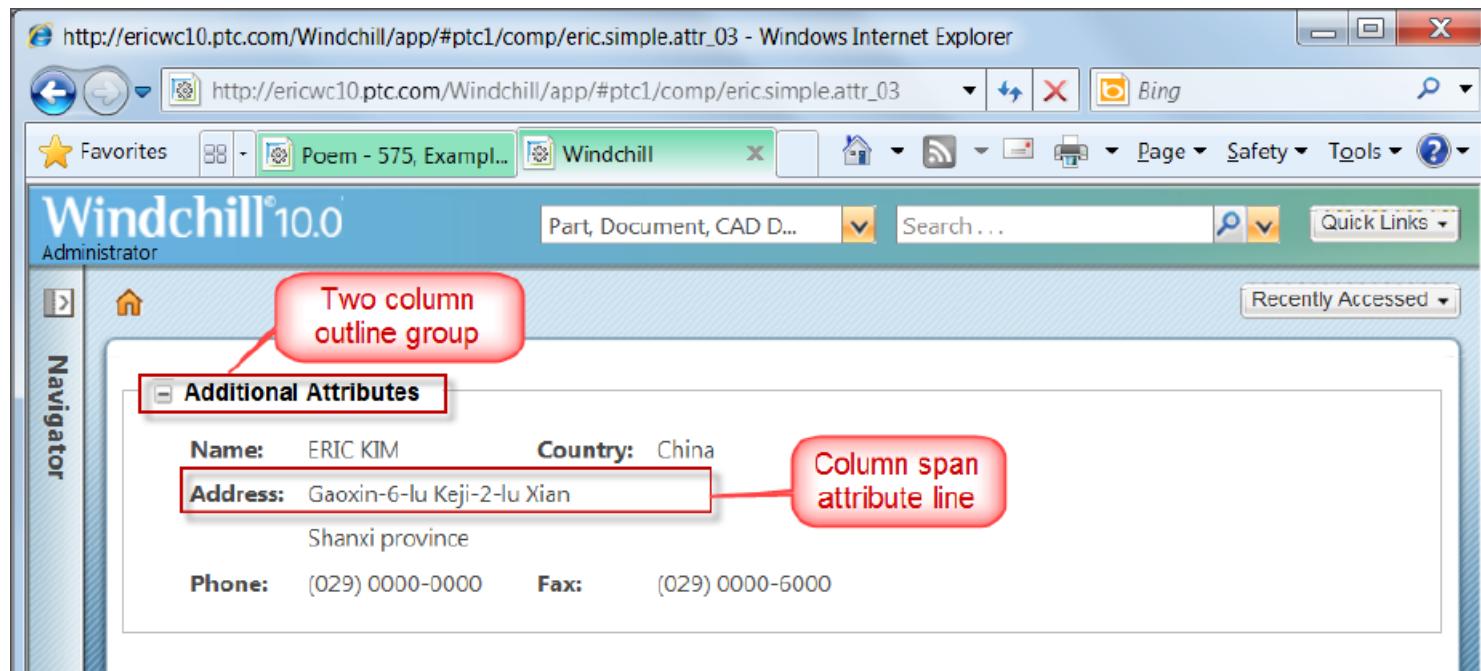


### 3. Exercise 4 — Multi-Column and Column Span

#### Design a Multi-Column and Column Span

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

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



### 3. Exercise 4 — Multi-Column and Column Span

#### Design an Attributes Panel

##### 1. Design an attributes panel (buildComponentConfig).

We will update Exercise 2 source code for making a multi-group framework. Copy Exercise 2's class and modify a new class name and MVC id. We do not need to change other areas except the “**buildComponentConfig**” function.

```
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTEException {
    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

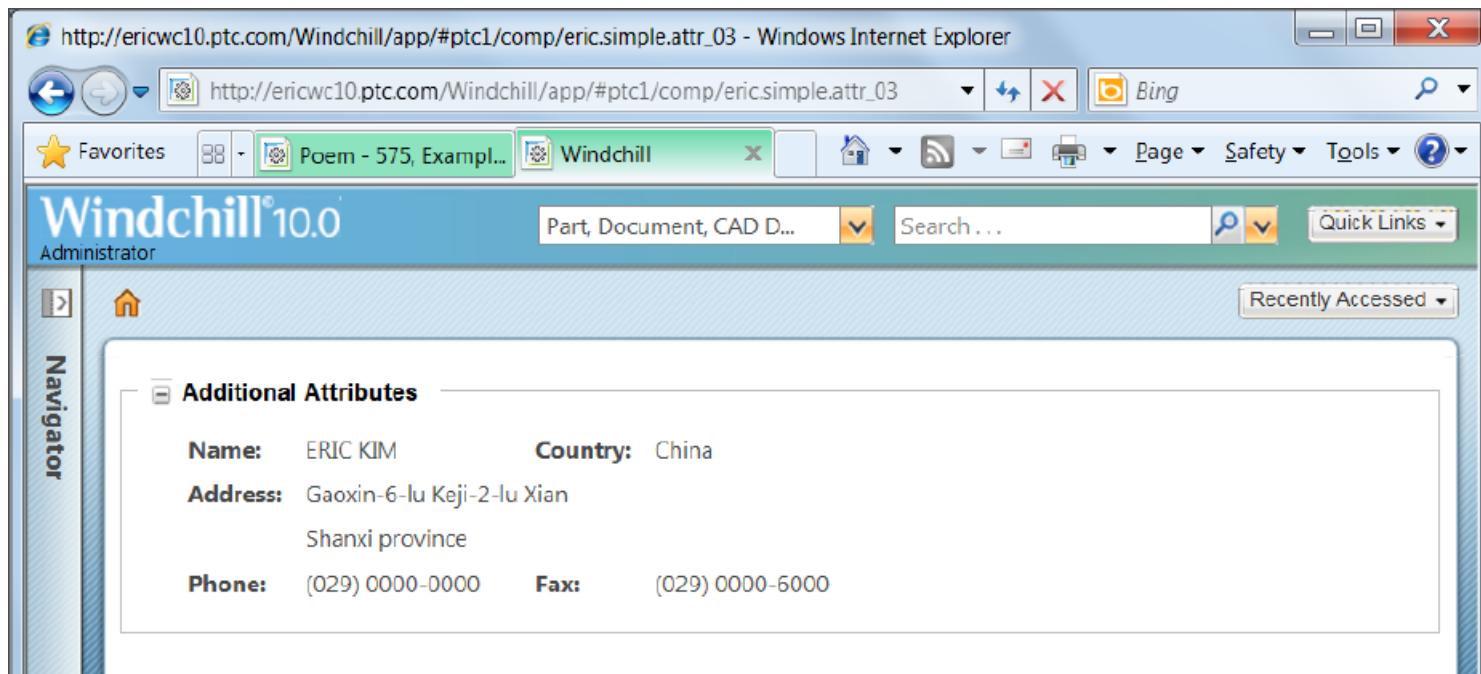
### 3. Exercise 4 — Multi-Column and Column Span

#### Test Result

##### 2. Restart all services and tests

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

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

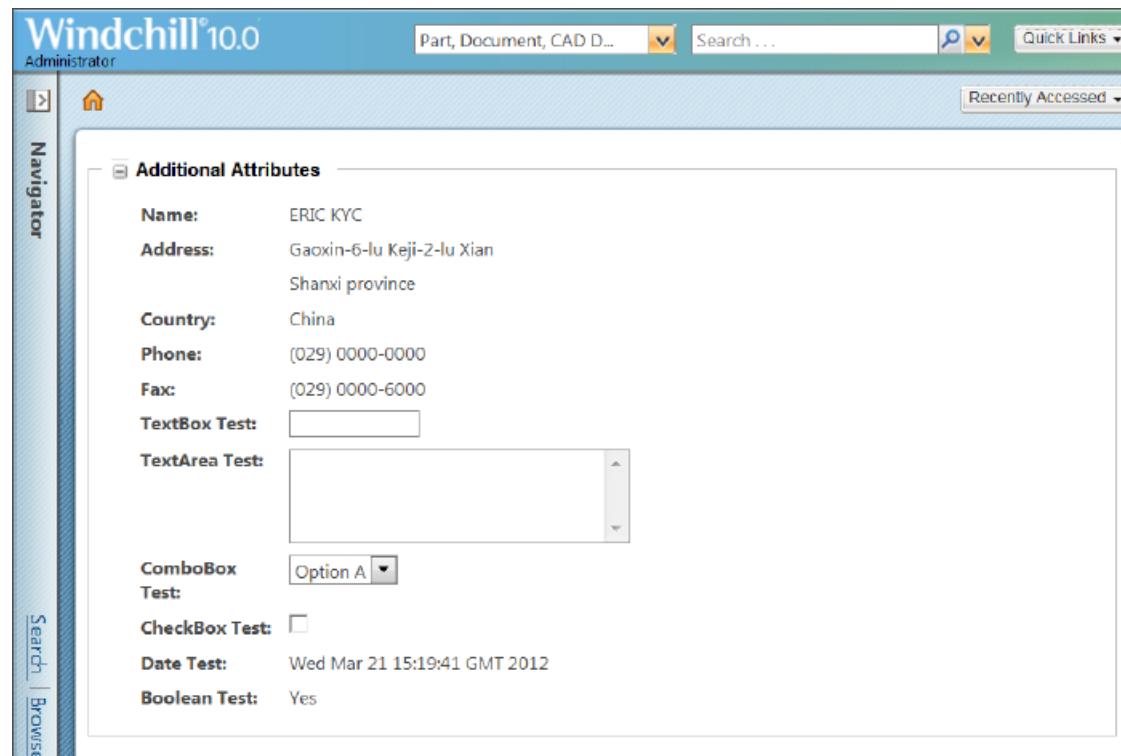


### 3. Exercise 5 — Add GUI Component Attributes

#### Design a Multi-Column and a Column Span

Using Exercise 2, we will try to add several external GUI Component attributes.

- Add external GUI component attributes data for each of types (**buildComponentData**)
  - Make data model using the UIComponent class.
  - UIComponent is like an attribute on “buildComponentData.”



### 3. Exercise 5 — Add GUI Component Attributes

#### Design Attributes Panel

##### 1. Design attributes panel (buildComponentConfig)

This function just needs to set attributes design for defined attributes including external editable attributes.

**The GUI component attributes are same as attributes. If you add editable GUI component attributes, editable attributes will be shown, although not in the EDIT mode.**

```
public AttributePanelConfig buildComponentConfig(ComponentParams params) throws WTEException {  
    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);  
    return panelConfig;  
}
```

Add attributes for testing  
each of UIComponents  
types

### 3. Exercise 5 — Add GUI Component Attributes

#### Query Data

##### 2. Query data for an attribute panel (buildComponentData)

The GuiComponent is same as an attribute. A special GUI component type will be shown. It is very useful to show abnormal cases when you implement an attribute panel.

```
public Object buildComponentData(ComponentConfig config, ComponentParam  
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. Exercise 5 — Add GUI Component Attributes

#### Test Result

##### 2. Restart all services and tests

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

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

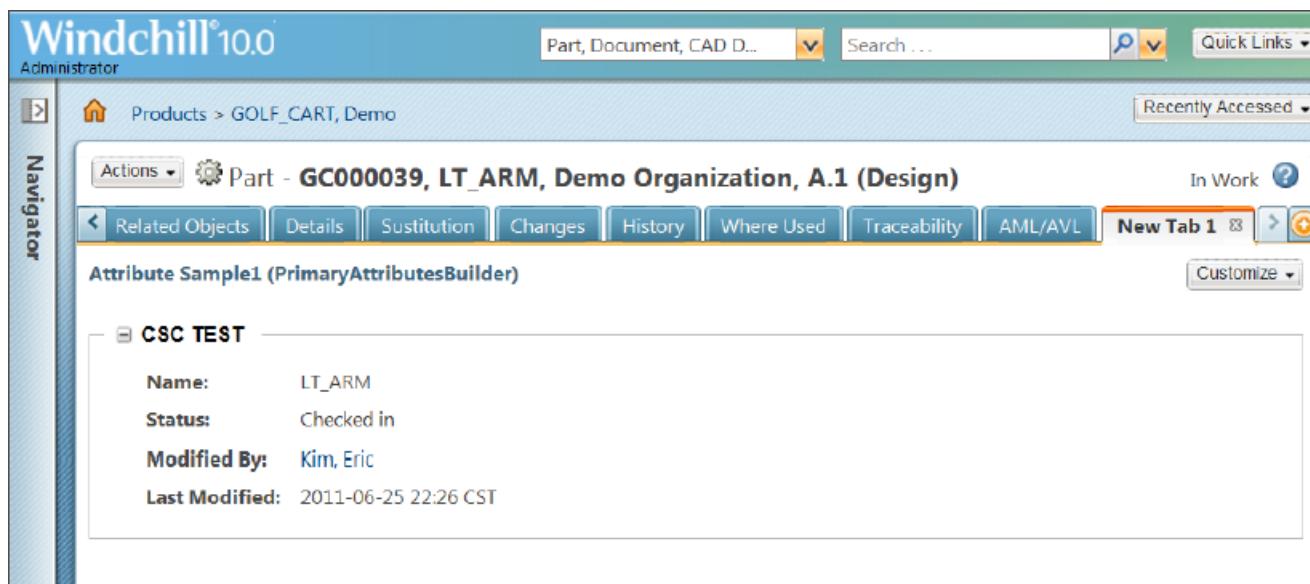


### 3. Exercise 6 — PrimaryAttributesBuilder

#### Design for PrimaryAttributesBuilder

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

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



### 3. Exercise 6 — PrimaryAttributesBuilder

#### Update Action Model and Action

1. Open “PartClient-actionmodels.xml” and copy the “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 a 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 you don't have a resource bundle, you must create a resource bundle for action action.properties, action\_[locale].properties

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

### 3. Exercise 6 — PrimaryAttributesBuilder

Create an MVC class

#### 4. Create an 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 WTEexception {}

    @Override
    public void setTypedAttrLayOutFactory(TypedAttrLayOutFactory factory) {}

}
```

### 3. Exercise 6 — PrimaryAttributesBuilder

#### Design an attributes panel

##### 1. Design an attributes panel (buildPrimaryAttributePanelConfig)

This function is a design area for attribute panel.

```
public AttributePanelConfig buildPrimaryAttributePanelConfig(ComponentParams params) throws WTEException {  
  
    // 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. Exercise 6 — PrimaryAttributesBuilder

Get a Layout object instance and test

#### 6. Get a 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 the Method Server and Test

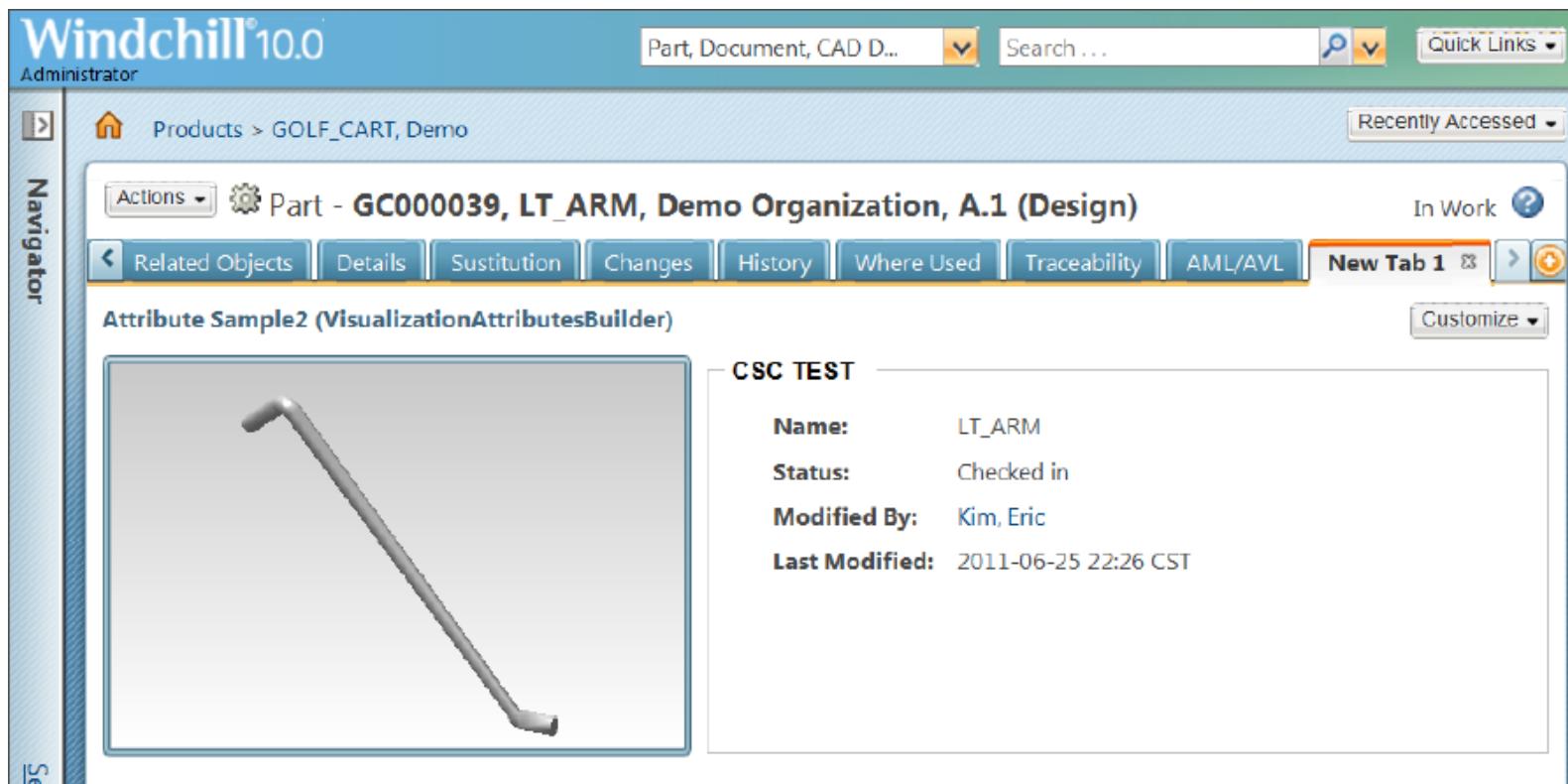
- The menu will be set on the info page customize button on part information. You have to create one new tab and select ‘add new item’ for PrimaryAttributesBuilder on the new tab page.



### 3. Exercise 7 — VisualizationAttributesBuilder

#### Design for VisualizationAttributesBuilder

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



### 3. Exercise 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 you have no resource bundle, you must create resource bundle for action  
action.properties, action\_[locale].properties

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

### 3. Exercise 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 WTEexception {}
}
```

### 3. Exercise 7 — VisualizationAttributesBuilder

#### Design Attributes Panel

##### 5. Design attributes panel (buildComponentConfig)

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

```
@Override
public ComponentConfig buildComponentConfig(ComponentParams params) throws WTEException {
    // 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;
}
```

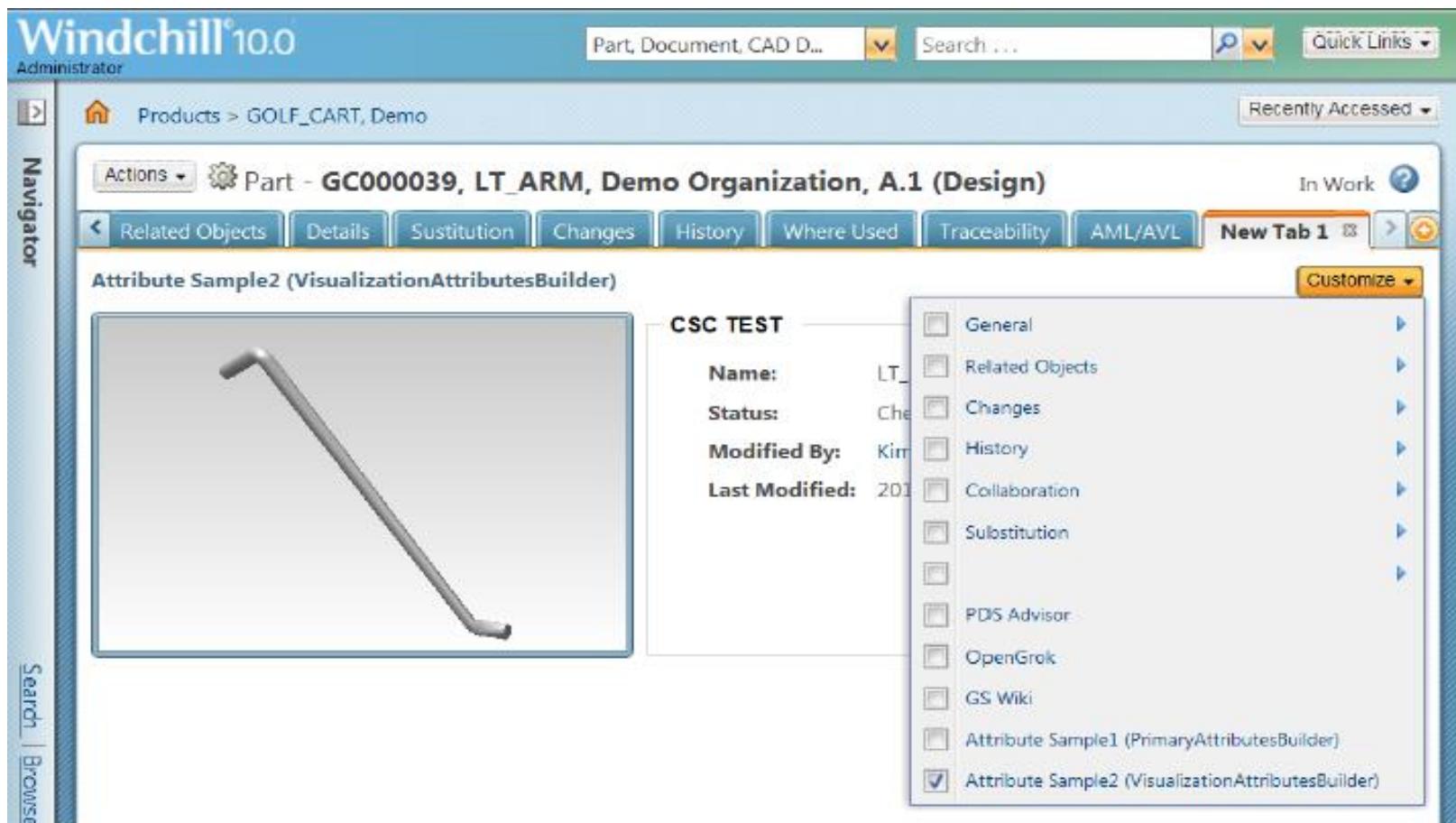
VisualizationAttributesBuilder doesn't need to be implemented “**setTypedAttrLayoutFactor**” function because it will be initialized on the super class.

### 3. Exercise 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.



### 3. Exercise 8 — AbstractAttributesComponentBuilder

PTC®

#### Design for AbstractAttributesComponentBuilder

AbstractAttributesComponentBuilder can use all layouts existing on Type Manager. So you can choose which kind of layout you want to use on customized pages.

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

The screenshot shows the Windchill 100 interface. The title bar says "Windchill 100 Administrator". The main area is titled "Part - GC000039, LT\_ARM, Demo Organization, A.1 (Design)". The "Attribute Sample3 (AbstractAttributesComponentBuilder)" tab is selected. The "Part Attributes" section displays the following data:

Number:	GC000039
Name:	LT_ARM
Assembly Mode:	Separable
Source:	Make
View:	Design
Default Trace Code:	Untraced
Default Unit:	each
Gathering Part:	No
Phantom Manufacturing Part:	No
Contract Number:	
Job Authorization Number:	
Phase:	
Life Cycle Template:	Basic
Team Template:	
Location:	GOLF_CART / Design

Info Page Using CREATE  
Layout

The screenshot shows a "Create Part" dialog box. The "Part Attributes" section contains the following fields:

Number:	(Generated)
Name:	Separable
Source:	Make
View:	Design
Default Trace Code:	Untraced
Default Unit:	each
Gathering Part:	No
Phantom Manufacturing Part:	No
Contract Number:	
Jobs Authorization Number:	
Phase:	
Life Cycle Template:	(Generated)
Team Template:	(Generated)
Location:	<input type="radio"/> Autoselect Folder (/GOLF_CART) <input checked="" type="radio"/> Select Folder GOLF_CART/Design
Classification:	<input type="checkbox"/> Do Not Classify <input type="checkbox"/> Choose Classifications

\* Indicates required fields.

Create Wizard Using CREATE Layout

### 3. Exercise 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. Exercise 8 — AbstractAttributesComponentBuilder

#### Create Resource Bundle and MVC Class

3. If you have no resource bundle, you must create one 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

#### 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 WTEException {
    // 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;
}
```

### 3. Exercise 8 — AbstractAttributesComponentBuilder

#### Update Wizard Step

##### 7. Update wizard step (setPartAttribute.jsp)

Previous wizard sample will be reused. 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. Exercise 8 — AbstractAttributesComponentBuilder

PTC®

#### Test Result

#### 8. Restart Method Server and Test

Check two types which are info page and wizard page.

The screenshot shows the Windchill 10.0 interface. On the left is the 'Info page' titled 'Attribute Sample3 (AbstractAttributesComponentBuilder)'. It displays various part attributes such as Number, Name, Assembly Mode, Source, View, Default Trace Code, Default Unit, Gathering Part, Phantom Manufacturing Part, Contract Number, Job Authorization Number, Phase, Life Cycle Template, Team Template, and Location. A sidebar on the right lists various attributes like General, Related Objects, Changes, History, Collaboration, Substitution, PDS Advisor, OpenGrok, GS Wiki, Attribute Sample1 (Primary), Attribute Sample2 (Visualizer), and Attribute Sample3 (Abstract). A red arrow points from the 'Info page' towards the 'Create Part' wizard on the right.

**Create Part**

**Part Attributes**

- Number: GC000039 (Generated)
- Name: LT\_ARM
- Assembly Mode: Separable
- Source: Make
- View: Design
- Default Trace Code: Untraced
- Default Unit: each
- Gathering Part: No
- Phantom Manufacturing Part: No
- Contract Number:
- Job Authorization Number:
- Phase:
- Life Cycle Template: Basic (Generated)
- Team Template: (Generated)
- Location: GOLF\_CART / Design

\* Indicates required fields.

OK Cancel

Info page used CREATE layout

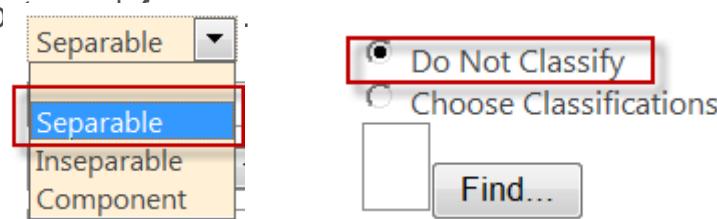
Create wizard used CREATE layout

# Data Utility

## Background

Most of the Windchill pages are constructed using GUI components which are tables, trees, attributes panels, and so on. Sometimes, we need to change part of UI Elements on GUI components as shown in the 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 attrib



## What is UI Element?

Any visual element displayed on a Page.

## 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 Components have been controlled by user interface renderer, so it is difficult to change partial user interface directly. This time, if you want to change UI Element, you 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 so 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 a 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 the actual data to be displayed in the component

## What is Data Utility (2)

### What is Component Bean?

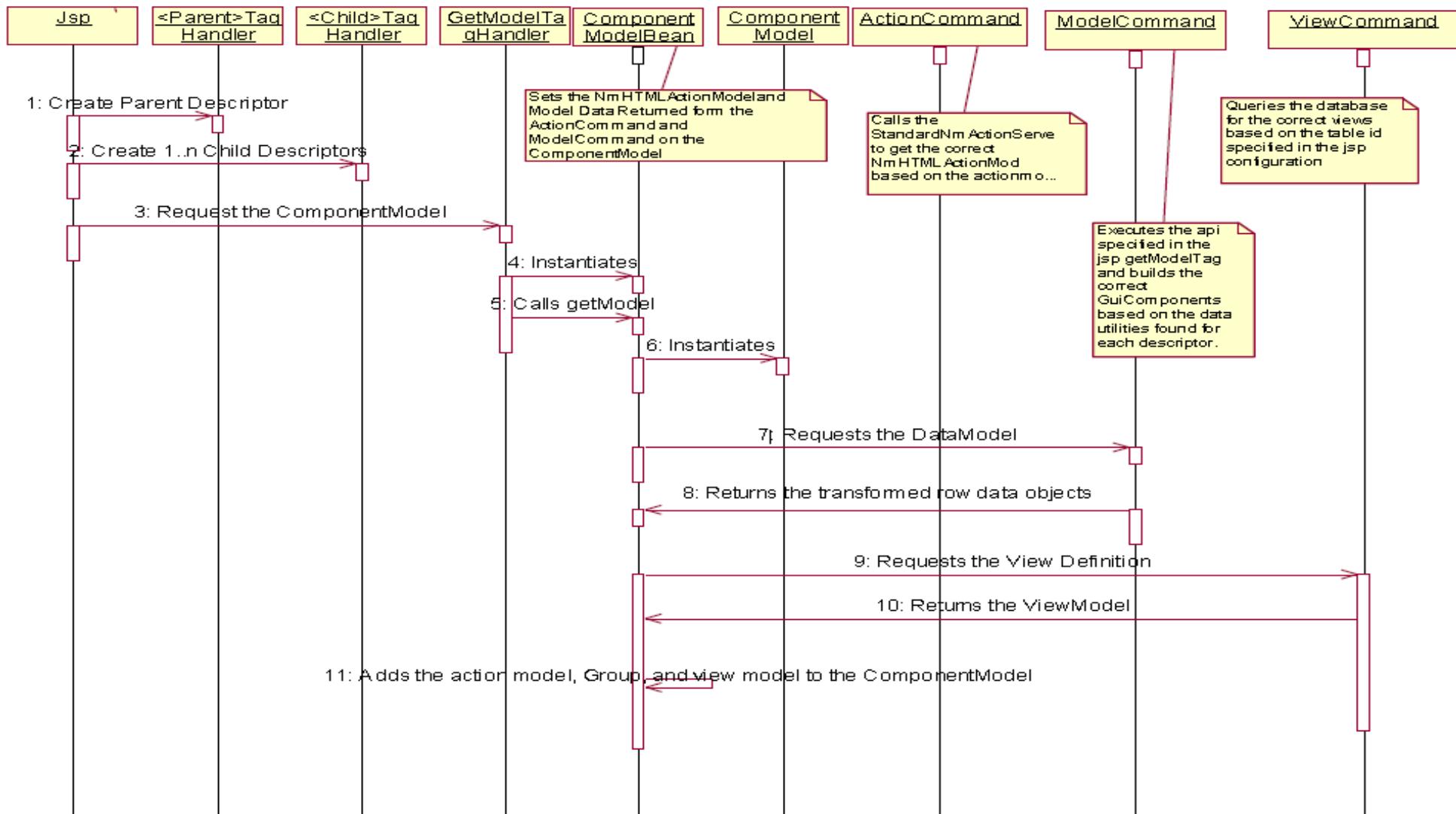
Actually it does the work of building a ComponentModel.

- It constructs a BatchCommand with three subcommands to get each of the ComponentModel's three 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.

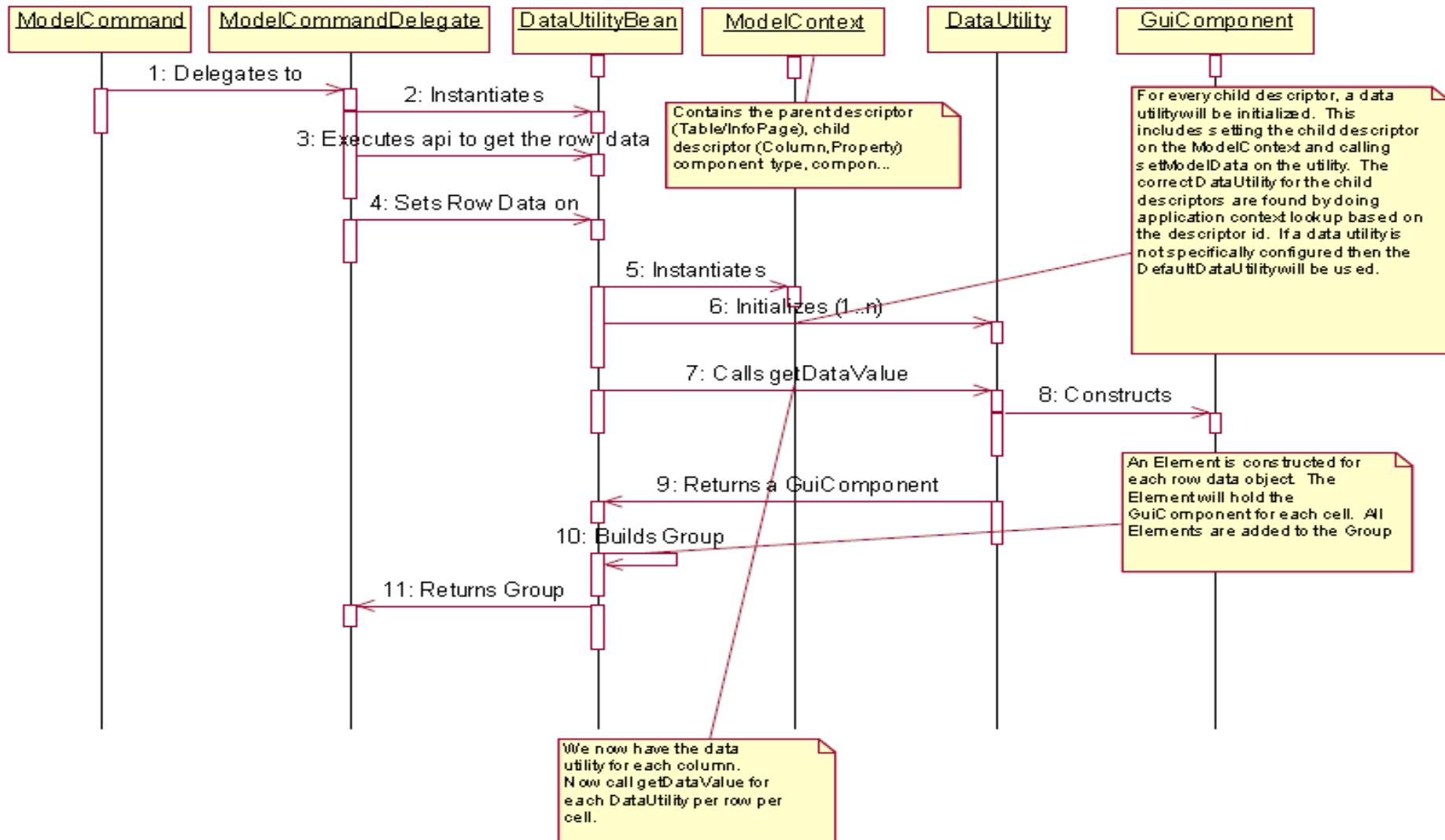
## 2. General Process for GUI Component

### Getting Data from Request



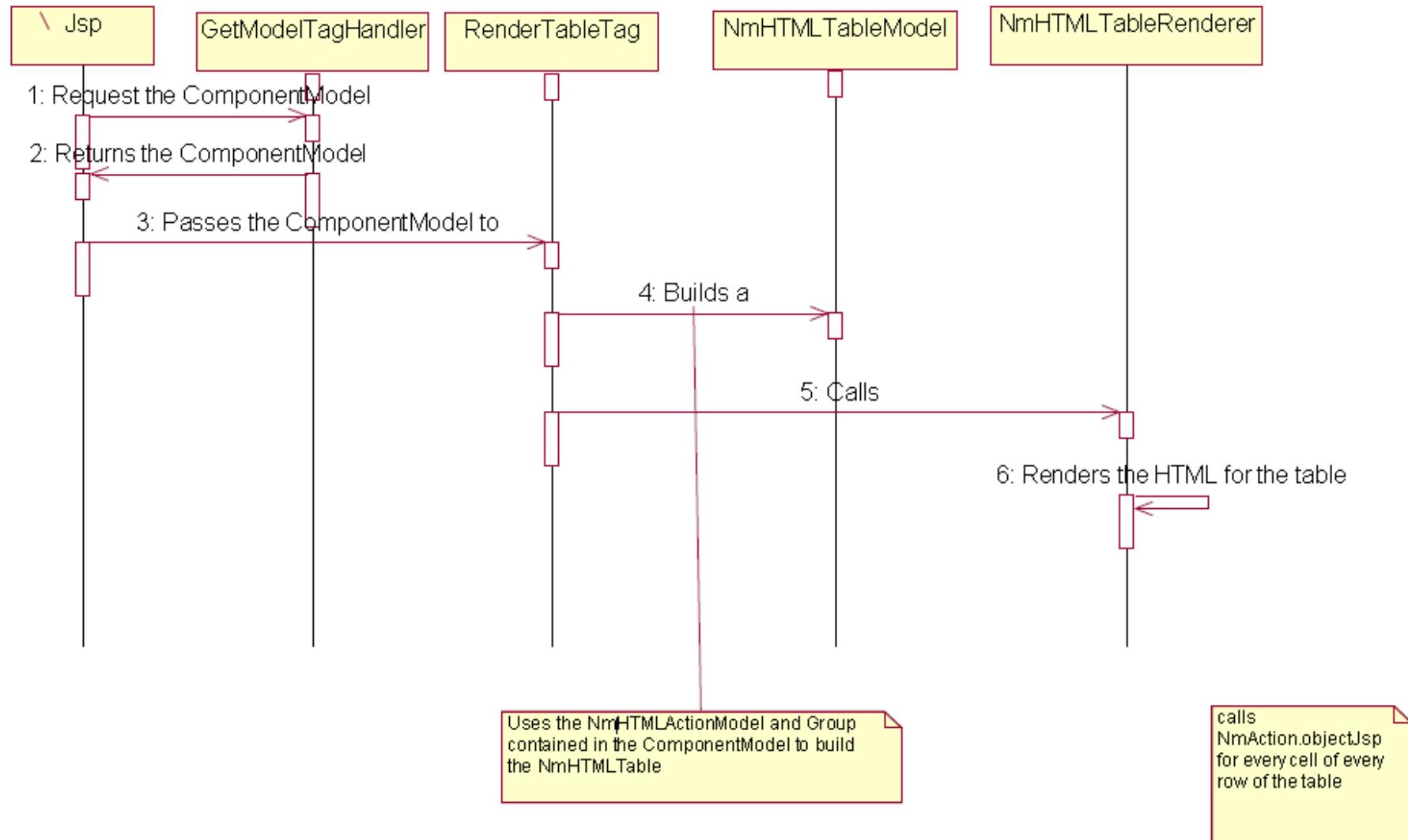
## 2. General Process for GUI Component

### 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 be understand, we will not touch OOTB function.*

The following are the OOTB lists of GUI Components

AttributeDisplayComponent	CheckBox	LocationInputComponent
ComboBox		AttributeGuiComponent
DateDisplayComponent		AttributeInputComponent
DateInputComponent		AttributeInputCompositeComponent
EnumInputComponent		BooleanInputComponent
UrlInputComponent		NumberInputComponent
IconComponent		NumericDisplayComponent
Label		NumericInputComponent
PushButton		TextArea
RadioButton		TextBox
RevisionInputComponent		TextDisplayComponent
StringInputComponent		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 with `NmString`, `NmDate`.

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

The **`setModelData`** method enables 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 with a List of all the objects that will be processed.

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

The **`getLabel`** method enables 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 WTEException;
```

When using `DataUtility`, we have to use an abstract super class named **`AbstractDataUtility`** generally. `AbstractDataUtility` is the child class of `DataUtility`.

## Sample DataUtility Class

```
public class CSCLifeCycleDataUtility extends AbstractDataUtility {
    public Object getDataValue(String component_id, Object datum, ModelContext mc) throws WTEException {
        ...
        // 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();
            requestValue.add(lcTemplate.getName());
            displayValue.add(lcTemplate.getName());
        }
        ...
        comboBox.setInternalValues(requestValue); //Key
        comboBox.setValues(displayValue); // display
        return comboBox;
    }
    private String getValue(String component_id, Object datum, ModelContext mc) throws WTEException {
        return UtilityHelper.getStringValue(component_id, datum, mc);
    }
}
```

### Register DataUtility

Customized data utility class cannot be used directly on GUI Component. All GUI Components can read service identification which is registered in Method Server 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)

# 4. Implementation

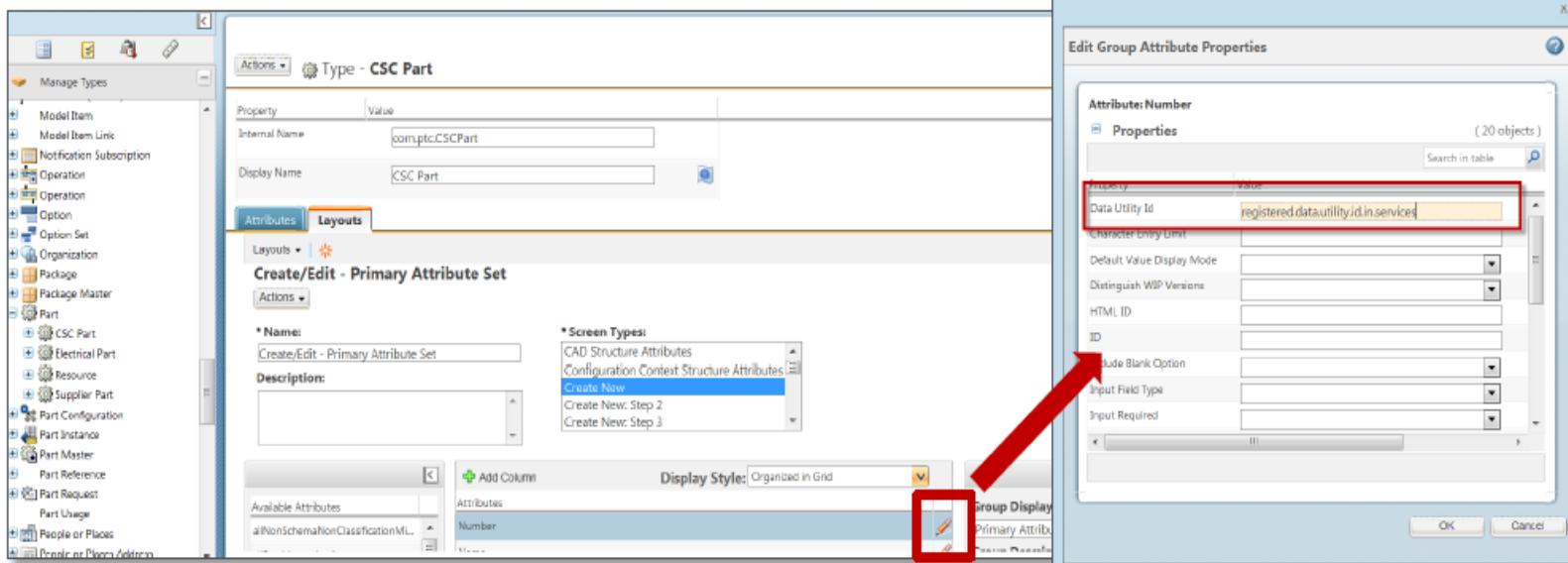
## How to Set Data Utility on the GUI Component

GUI Component will be rendered by using several methodologies, 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. Exercise 1 — Change Part Attribute Field (Layout)

PTC®

### Design Part Name Value of Info Page

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

The screenshot shows the Windchill 10.0 interface for a part named GC000039, LT\_ARM, Demo Organization, A.1 (Design). The main panel displays the part's visualization and basic information. A red box highlights the 'Name' field, which contains 'LT\_ARM'. A red arrow points from this field to a secondary panel titled 'Visualization & All Attributes', also containing a 'Name' field with the value 'LT\_ARM ( GC000039 )'. Both panels show other attributes like Status (Checked in), Modified By (Kim, Eric), and Last Modified (2011-06-25 22:26 CDT).

CSC TEST	
Name:	LT_ARM
Status:	Checked in
Modified By:	Kim, Eric
Last Modified:	2011-06-25 22:26 CDT

Visualization & All Attributes	
Name:	LT_ARM ( GC000039 )
Status:	Checked in
Modified By:	Kim, Eric

# 5. Exercise 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**

```
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 WTEException {
        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. Exercise 1 — Change Part Attribute Field (Layout)

## Register Data Utility on Method Server Service

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

Open “\$WT\_HOME/codebase/service.properties.xconf” and add the following block at the end of the file. Actually, it must be located in the front of the “</Configuration>” tag.

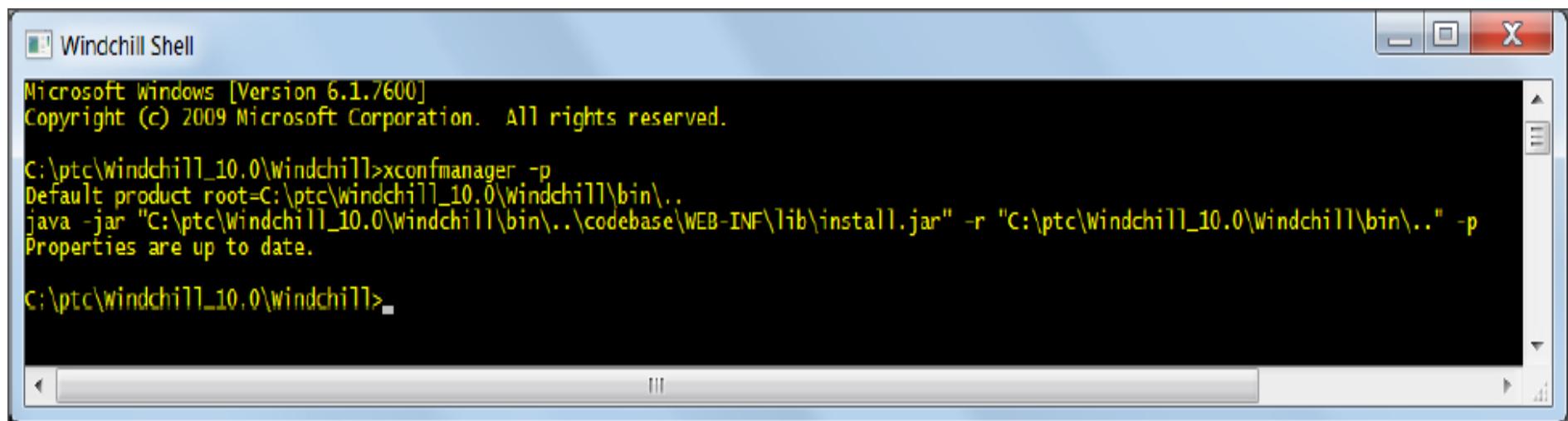
```
<Service context="default" name="com.ptc.core.components.descriptor.DataUtility">
    <Option serviceClass="ext.csc.training.datautility.CSCPartNameDataUtility"
        selector="cscPartNameDataUtility"
        requestor="java.lang.Object"
        cardinality="duplicate" />
</Service>
```

DataUtility Id

DataUtility class

### 3. Register service on system configuration

Open Windchill shell and execute “xconfmanger -p”

A screenshot of a Microsoft Windows command prompt window titled "Windchill Shell". The window shows the following text:

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\ptc\Windchill_10.0\Windchill>xconfmanger -p
Default product root=C:\ptc\Windchill_10.0\Windchill\bin\..
java -jar "C:\ptc\Windchill_10.0\Windchill\bin\..\codebase\WEB-INF\lib\install.jar" -r "C:\ptc\Windchill_10.0\Windchill\bin\.." -p
Properties are up to date.

C:\ptc\Windchill_10.0\Windchill>
```

The window has a standard Windows title bar and a scroll bar on the right side.

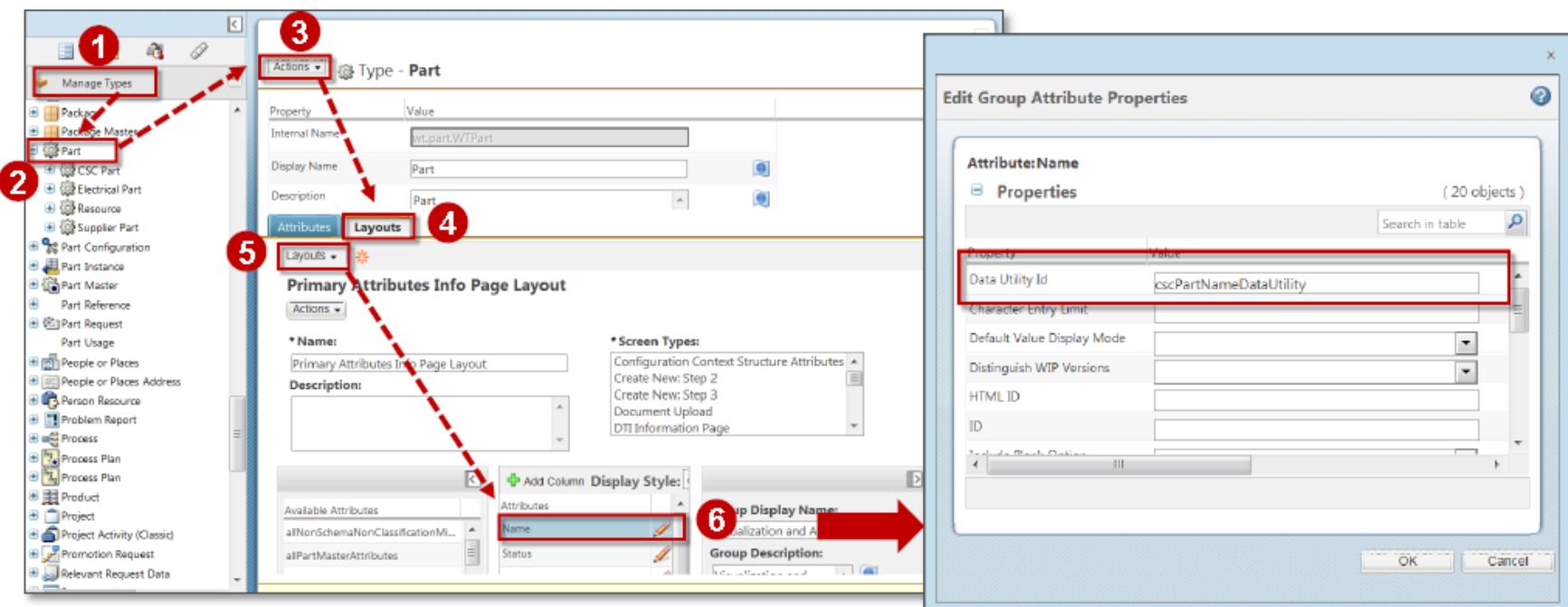
# 5. Exercise 1 — Change Part Attribute Field (Layout)

PTC®

## 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 named “**cscPartNameDataUtility**” on the editing window of name



## 5. Exercise 1 — Change Part Attribute Field (Layout)

PTC®

### Test Result

### 6. Restart Method Server and Test

Open part information and check name attribute

The screenshot shows the Windchill 10.0 interface. The title bar reads "Windchill 10.0 Administrator". The top navigation bar includes "Part, Document, CAD D...", "Search ...", and "Quick Links". The main area shows a product tree under "Products > GOLF\_CART, Demo". A part card is displayed for "Part - GC000039, LT\_ARM, Demo Organization, A.1 (Design)". The card has tabs for "Actions", "Structure", "Related Objects", "Details" (which is selected), "Substitution", "Changes", "History", "Where Used", "Traceability", "AML/AVL", and "New Tab". The "Visualization and Attributes" section shows a 3D model of a long, thin metal arm. To the right, the "Visualization and Attributes" panel lists:

Name:	LT_ARM ( GC000039 )
Status:	Checked in
Modified By:	Kim, Eric
Last Modified:	2011-06-25 22:26 CST

Below this, the "General" section contains the following table:

Assembly Mode:	Separable	End Item:	No
Source:	Make	Default Unit:	each
Gathering:	No	Default Trace:	Untraced

# 5. Exercise 2 — Change Table Column (MVC)

## Design Table

We will change the columns of table which was designed by MVC.

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

The screenshot shows two instances of the PTC Data Utility interface, labeled "Data Utility Sample1". Both instances display a table with 16 objects. The columns are: Name, CAGE Code, State, Version, Last Modified, and Context. The "State" column contains values like "Released", "Canceled", and various "In Work" states. In the second instance, the "State" column has been modified. The original "In Work" states now appear as green circles with yellow outlines and red dots, while the other states remain as text. The "Version" column also shows some changes, such as "A-1 (Design)" becoming "A-1.1 (Design)". A large red arrow points from the left table to the right table, highlighting the transformation of the "State" column.

Name	CAGE Code	State	Version	Last Modified	Context
TEST01	PTC	Released	A.1 (Design)	2011-12-15 15:37 CST	LENOVO
HELV_001	PTC	Canceled	A.3 (Design)	2012-02-17 17:24 CST	HELV
PART-0001	PTC	In Work	A.3 (Design)	2011-12-14 20:59 CST	LENOVO
CSC_PART_001	PTC	In Work	A.1	2012-02-29 11:36 CST	HELV
CSC_PART_002	PTC	In Work	A.1	2012-02-29 12:05 CST	HELV
CSC_PART_003	PTC	In Work	A.1	2012-02-29 12:23 CST	HELV
CSC_PART_004	PTC	In Work	A.1	2012-	
PART-0004	PTC	In Work	A.1 (Design)		
PART-0001	PTC	In Work	A-2.2 (Design) 2011		
CSC Part Info Test 01	PTC	In Work	A.1	2012-	
PART-0003	PTC	In Work	A.1 (Design)	2011-	
PART-0001	PTC	In Work	A-1.1 (Design) 2011		

Name	CAGE Code	State	Version	Last Modified	Context
TEST01	PTC	●	A.1 (Design)	2011-12-15 15:37 CST	LENOVO
HELV_001	PTC	●	A.3 (Design)	2012-02-17 17:24 CST	HELV
PART-0001	PTC	●	A.3 (Design)	2011-12-14 20:59 CST	LENOVO
CSC_PART_001	PTC	●	A.1	2012-02-29 11:36 CST	HELV
CSC_PART_002	PTC	●	A.1	2012-02-29 12:05 CST	HELV
CSC_PART_003	PTC	●	A.1	2012-02-29 12:23 CST	HELV
CSC_PART_004	PTC	●	A.1	2012-02-29 12:25 CST	HELV
PART-0004	PTC	●	A.1 (Design)	2011-12-16 14:34 CST	LENOVO
PART-0001	PTC	●	A-2.2 (Design) 2011-12-30 14:53 CST	LENOVO	
CSC Part Info Test 01	PTC	●	A.1	2012-02-02 13:45 CST	LENOVO
PART-0003	PTC	●	A.1 (Design)	2011-12-14 00:29 CST	LENOVO
PART-0001	PTC	●	A-1.1 (Design) 2011-12-30 14:40 CST	LENOVO	

# 5. Exercise 2 — Change Table Column (MVC)

## 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**

```
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 WTEException {
        ...
        ColumnConfig col = factory.newColumnConfig(STATE, true);
        col.setDataUtilityId("cscLifecycleIconDataUtility");
        table.addComponent(col);
        ...
    }
}
```

You must remember  
the data utility Id.

# 5. Exercise 2 — Change Table Column (MVC)

## 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**

```
package ext.csc.training.datautility;
...
public class CSCLifecycleIconDataUtility extends AbstractDataUtility {
    @Override
    public Object getDataValue(String component_id, Object datum, ModelContext mc) throws WTEException {
        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. Exercise 2 — Change Table Column (MVC)

## Register Data Utility on Method Server Service

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

Open “\$WT\_HOME/codebase/service.properties.xconf” and add the following block at the end of the file. Actually, it must be located in the front of the “</Configuration>” tag.

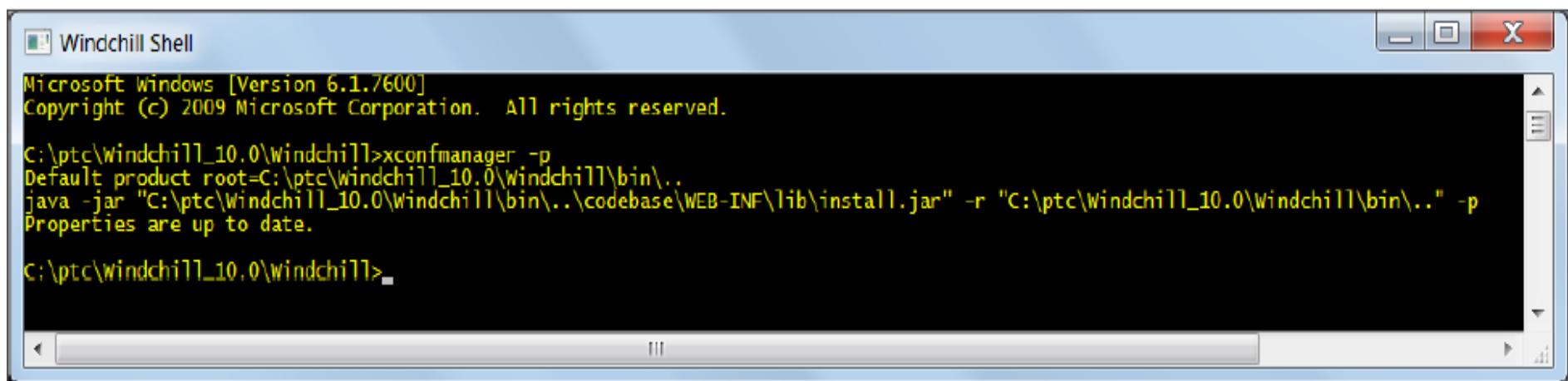
```
<Service context="default" name="com.ptc.core.components.descriptor.DataUtility">
    <Option serviceClass="ext.csc.training.datautility.CSCLifecycleIconDataUtility"
        selector="cscLifecycleIconDataUtility"
        requestor="java.lang.Object"
        cardinality="duplicate" />
</Service>
```

DataUtility Id

DataUtility class

### 4. Register service on system configuration

Open Windchill shell and execute “xconfmanger -p”



# 5. Exercise 2 — Change Table Column (MVC)

PTC®

## Test Result

### 5. Restart Method Server and Test

Open Internet explorer and directly type the following as URL

[http://localhost/Windchill/app/#ptc1/comp/eric.test.datautility\\_01](http://localhost/Windchill/app/#ptc1/comp/eric.test.datautility_01)

The screenshot shows a Windows Internet Explorer window displaying the Windchill 10.0 Data Utility Sample1 interface. The browser address bar shows the URL [http://ericwc10.ptc.com/Windchill/app/#ptc1/comp/eric.test.datautility\\_01](http://ericwc10.ptc.com/Windchill/app/#ptc1/comp/eric.test.datautility_01). The main content area is titled "Data Utility Sample1" and displays a table of objects. The columns are: Name, CAGE Code, State, Version, Last Modified, and Context. The "State" column is highlighted with a red box. The table contains 16 rows of data, each representing an object with its name, CAGE code, state, version, last modified date, and context.

Name	CAGE Code	State	Version	Last Modified	Context
PART-0001	PTC	●	A.3 (Design)	2011-12-14 20:59 CST	LENOVO
PART-0001	PTC	●	A-2.2 (Design)	2011-12-30 14:53 CST	LENOVO
PART-0004	PTC	●	A.1 (Design)	2011-12-16 14:34 CST	LENOVO
CSC Part Info Test 01	PTC	●	A.1	2012-02-02 13:45 CST	LENOVO
HELV_001	PTC	●	A.3 (Design)	2012-02-17 17:24 CST	HELV
PART-0001	PTC	●	A-1.1 (Design)	2011-12-30 14:40 CST	LENOVO
PART-0002	PTC	●	A.4 (Design)	2011-12-30 17:44 CST	LENOVO
PART-0003	PTC	●	A.1 (Design)	2011-12-14 00:29 CST	LENOVO
PART-0001	PTC	●	A-3.1 (Design)	2011-12-30 16:32 CST	LENOVO
aaaaaaaaaa	PTC	●	A.1 (Design)	2012-02-01 15:34 CST	LENOVO
aaaaaaaaaa	PTC	●	A.1 (Design)	2012-02-01 15:35 CST	LENOVO
TEST01	PTC	●	A.1 (Design)	2011-12-15 15:37 CST	LENOVO

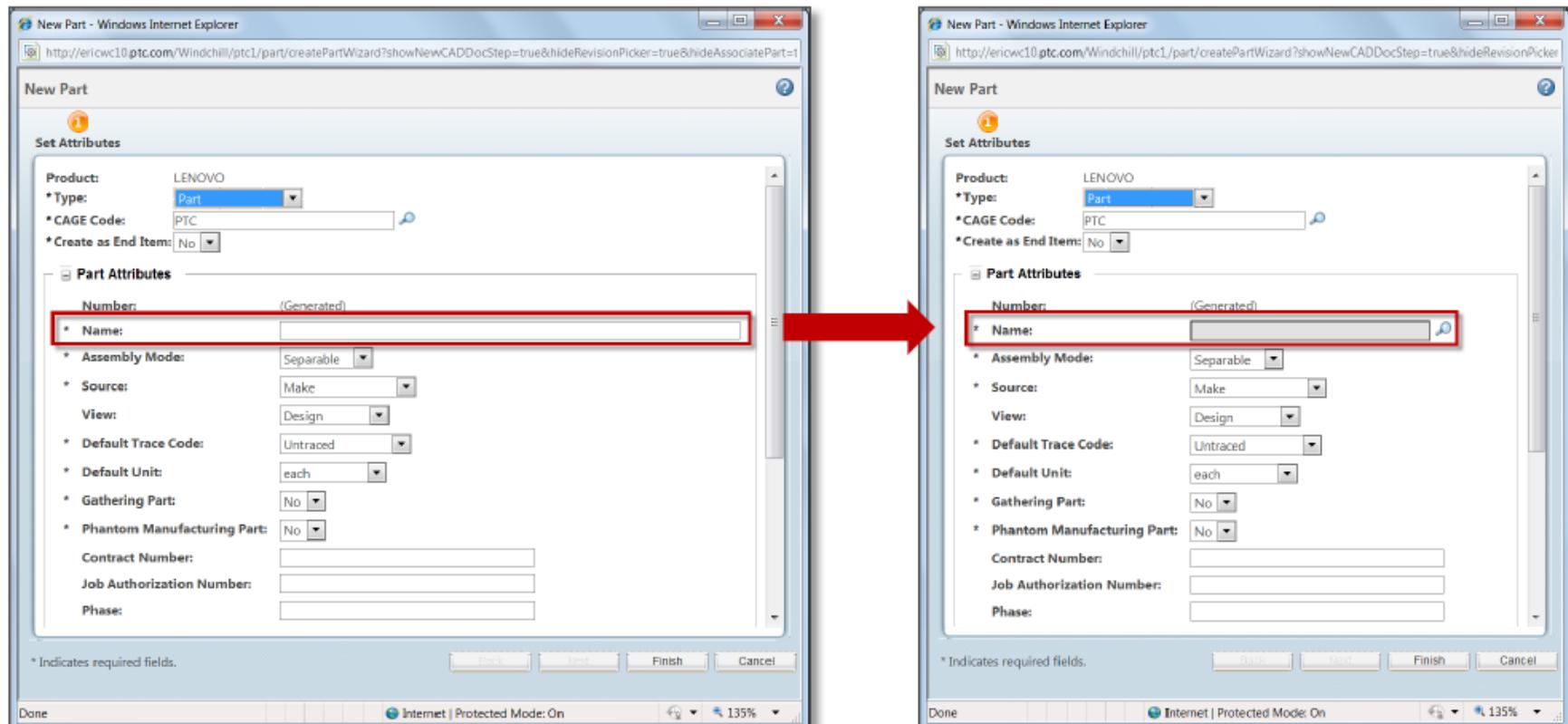
# 5. Exercise 3 – Picker Data Utility Component

## Design Table

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

We had created one custom wizard which is given in [previous exercise 8 of “Attributes Panel” section](#). We will reuse the sample and change the name field to use picker data utility.

The picker component will use item picker, so picker configuration also will reuse [exercise 1 of “Picker”](#)



# 5. Exercise 3 – Picker Data Utility Component

## Create Data Utility

### 1. Create DataUtility class.

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

```
package ext.csc.training.datautility;
...
public class CSCPickerSampleDataUtility extends AbstractDataUtility {
    public Object getDataValue(String component_id, Object datum, ModelContext modelcontext) throws WTEexception {
        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. Exercise 3 – Picker Data Utility Component

## Register Data Utility on Method Server Service

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

Open “\$WT\_HOME/codebase/service.properties.xconf” and add the following block at the end of the file. Actually, it must be located in the front of the “</Configuration>” tag.

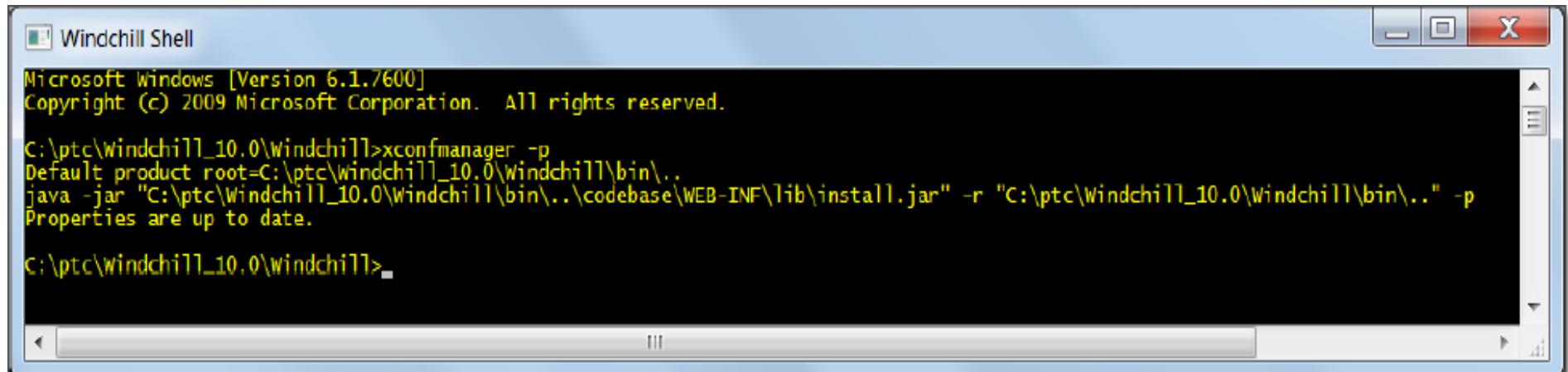
```
<Service context="default" name="com.ptc.core.components.descriptor.DataUtility">
    <Option serviceClass="ext.csc.training.datautility.CSCPickerSampleDataUtility"
        selector="cscPickerDataUtility"
        requestor="java.lang.Object"
        cardinality="duplicate" />
</Service>
```

DataUtility Id

DataUtility class

### 3. Register service on system configuration

Open Windchill shell and execute “xconfmanger -p”

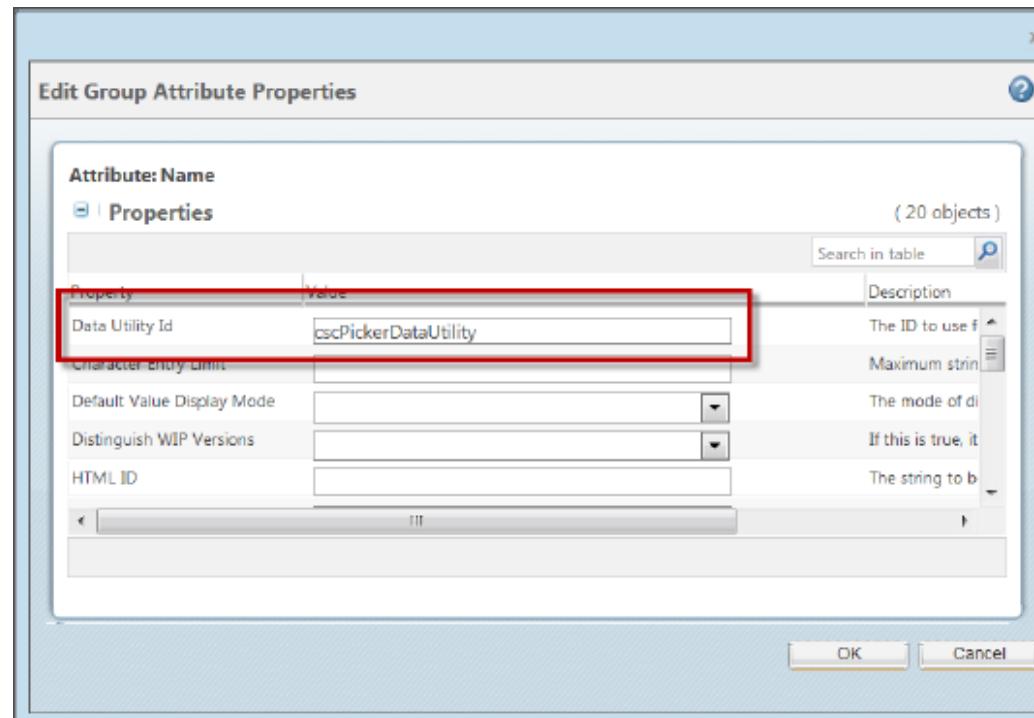


## 5. Exercise 3 – Picker Data Utility Component

### 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



## 5. Exercise 3 — Picker Data Utility Component

### 4. Picker Call Back Function

#### 5. Add call back JavaScript on wizard body page.

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

Add the 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>
```

The 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>
```

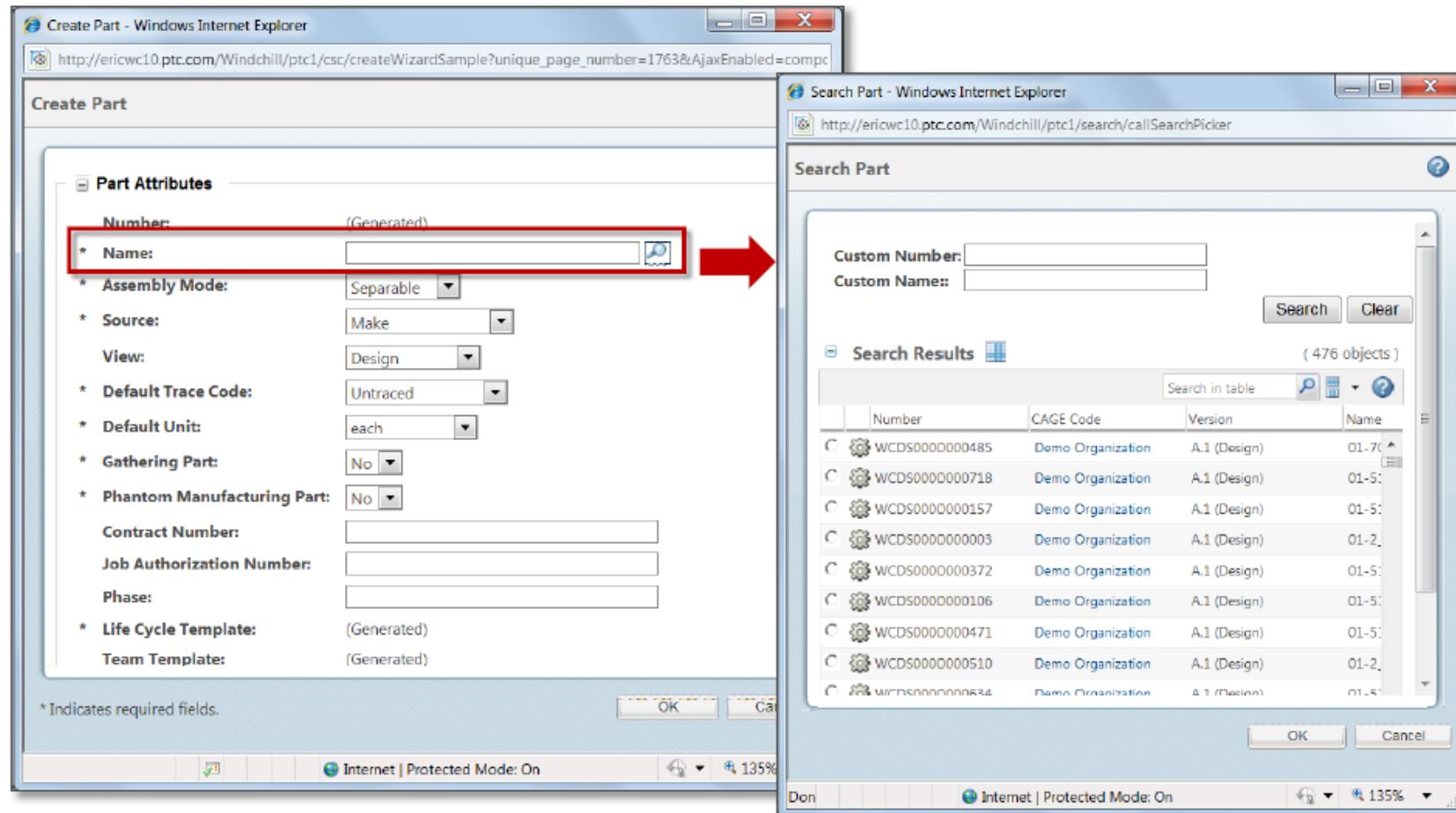
# 5. Exercise 3 — Picker Data Utility Component

PTC®

## Test Result

### 6. Restart Method Server and Test

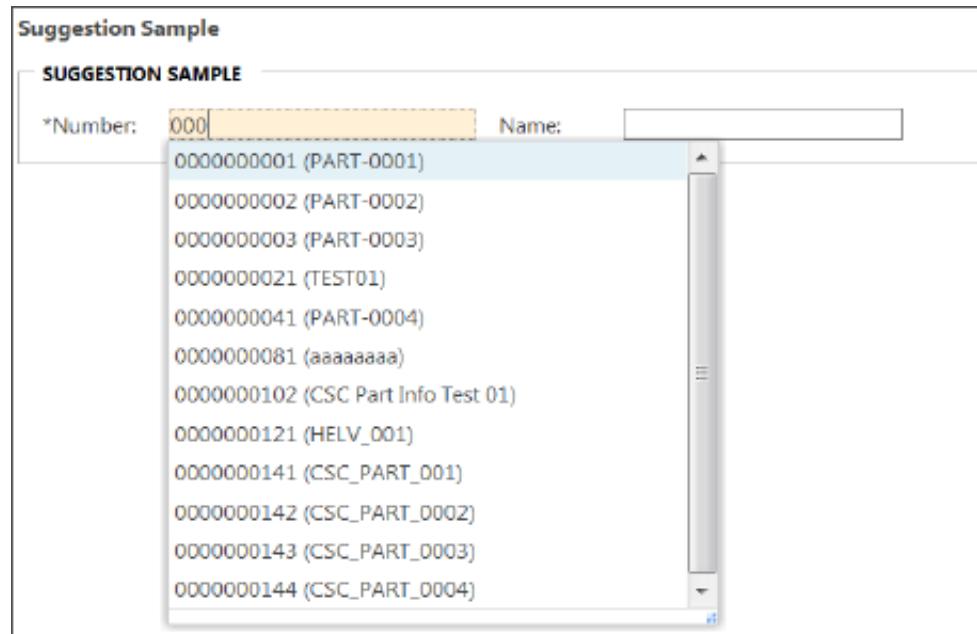
Open product folder and click custom button. (Exercise 8 of Attributes panel)



## Suggestion Text Box

## Suggestion Text Box on Wrapper Tag

Previously, we had studied about wrapper tag. Windchill 10 has one useful tag named "**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)

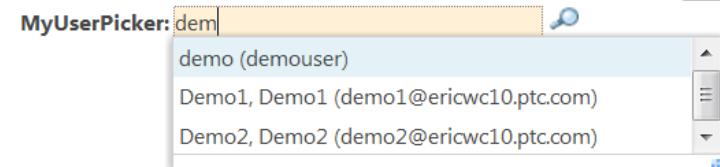
# 1. Introduction

## Suggestion Box on Picker

Some pickers also can use suggestion configuration, but unfortunately suggest configuration is not supported on all pickers. The following is a list of supported pickers.

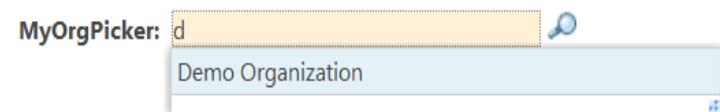
- User Picker

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



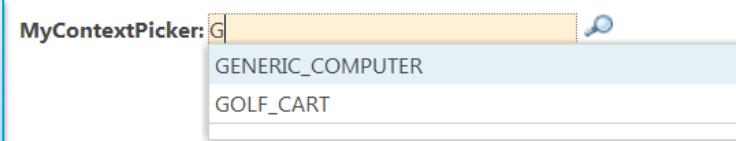
- Organization Picker

```
<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" />
```

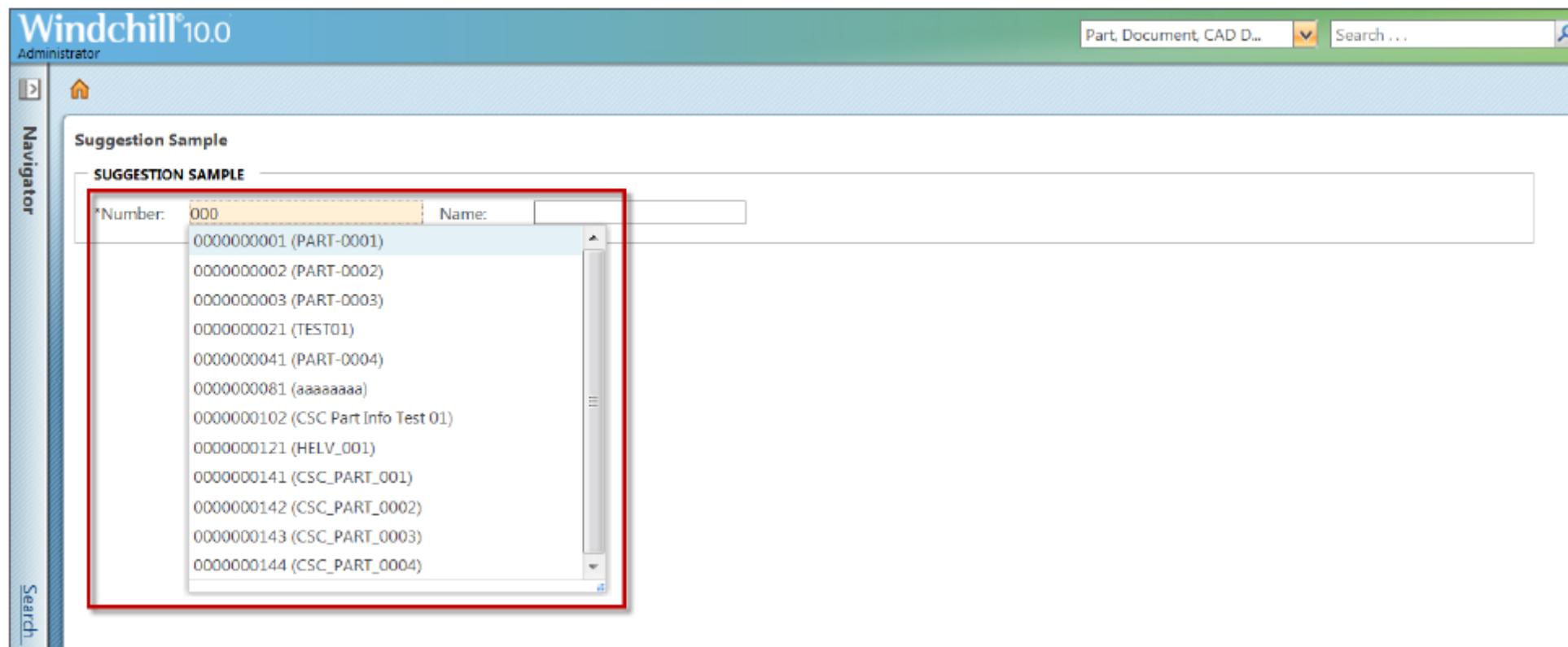


This picker also can use customized suggestion using “suggestServiceKey” which is registered suggest class.

## 2. Exercise 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. (Not by using Search action button.) And then, finally, we will set suggestion class on wrapper tag.



## 2. Exercise 1 – Simple Suggestion Text Box

### Suggestion Class

#### 1. Create Suggestion class

- When a user enters some value on “suggestTextBox,” this class will be called and create a result for showing data.
  - Class Name: **CSCNumberSuggest**
  - Implements: **Suggestable**
  - Override: **getSuggestions**

```
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(WTEexception wte) {}
    return list;
}
```

## 2. Exercise 1 — Simple Suggestion Text Box

### Register Suggestion Class on Method Server Service

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

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

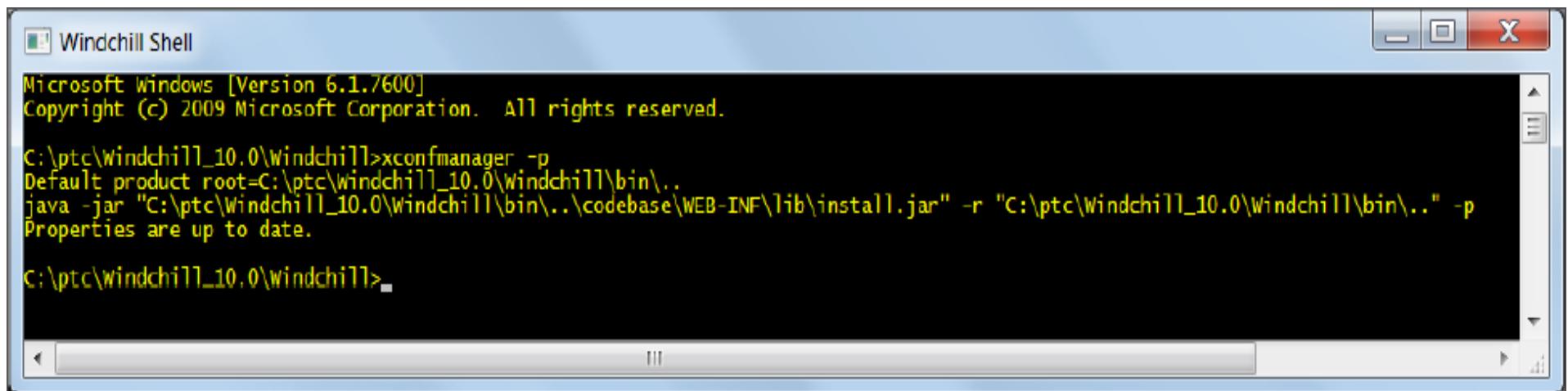
```
<Service context="default" name="com.ptc.core.components.suggest.Suggestable">
    <Option serviceClass="ext.csc.training.suggestion.CSCNumberSuggest"
        selector="cscNumberSuggest"
        requestor="null"
        cardinality="duplicate" />
</Service>
```

Suggestable Id

Suggestable class

#### 3. Register service on system configuration

Open Windchill shell and execute “xconfmanger -p”



## 2. Exercise 1 — Simple Suggestion Text Box

### Create JSP

#### 4. Create JSP page including wrapper tag

Add two input fields. One is the number field and the other is the 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">&nbsp;
                <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">&nbsp;
                <wrap:textBox name="name" id="name" maxlength="100" size="20"/>
            </td>
        </tr>
    </table>
</fieldset>
<%@ include file="/netmarkets/jsp/util/end.jspf"%>
```

## 2. Exercise 1 — Simple Suggestion Text Box

### Set Style on the JSP

#### 5. Update style configuration on the 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>
```

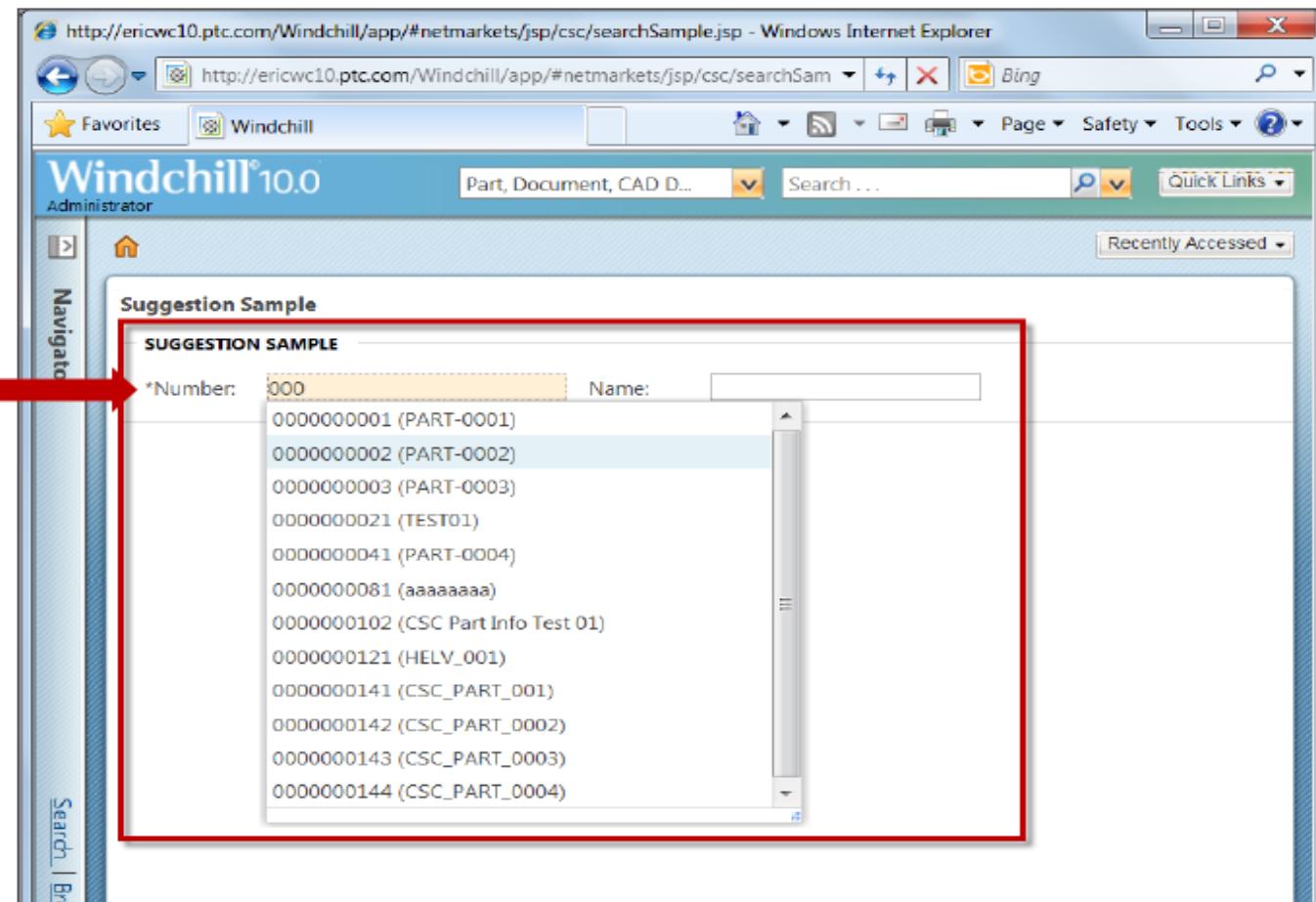
## 2. Excise 1 – Simple suggestion text box

### Test Result

#### 6. Restart Method Server and Test

Open explorer and enter the following URL:

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



**PTC®** PRODUCT & SERVICE  
ADVANTAGE