Windchill Development Best Practice

**Name:** *Attribute Wizard within Wizard* Best Practice

<INTERNAL>

By: [Chetan Saraf](mailto:csaraf@ptc.com)

Change History:

|  |  |  |
| --- | --- | --- |
| **Date** | **Author** | **Description** |
| 05/18/2010 | Chetan Saraf | Initial draft |
| 05/19/2010 | Chetan Saraf | Make changes after review |

</INTERNAL>

# Best Practice Name and Classification

## Name

## Attribute Wizard within Wizard Best Practice

## Classifications

Design - Attributes handling in create and edit wizard within Wizard.

# Objective

## Problem Statement

In a case of nested wizards, we do not want the child wizard to persist any information into the database until the parent wizard is completely finished. You have a need to create or edit another object from within another create or edit parent wizard while saving the object within the same transaction of the object created or modified in the parent wizard.

## Scope/Applicability/Assumptions

This document applies to windchill developers who need to make use of JSON libraries for data handling within their JCA wizards using Type Instance APIs and JSON to maintain the object attributes. An assumption made by this document is that developer is already familiar with JSON libraries and Configuring JCA wizards and tables.

Please note that the document is generic for windchill objects that use jca wizards. However, the object name is used for understanding the implementation. The term,

ChangeActionItem/GenericActionItem corresponds to the child wizard while as

Change Task corresponds to the parent wizard.

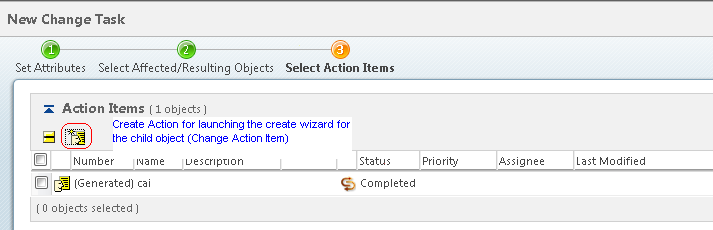
## Intended Outcome

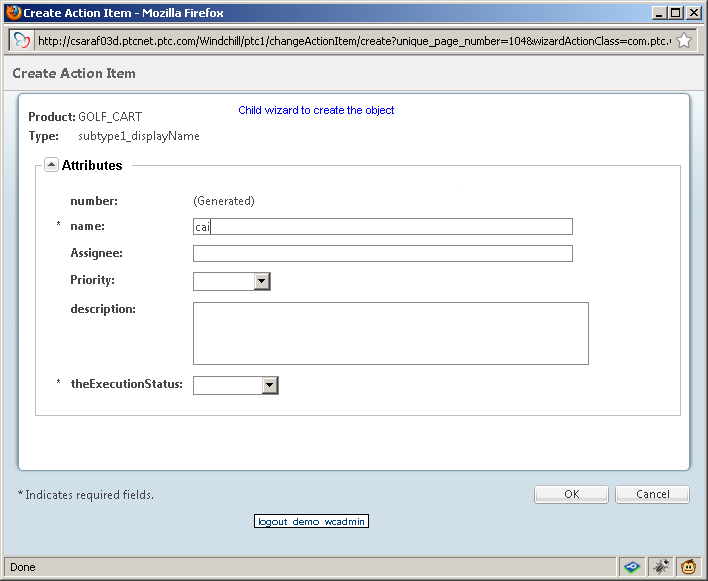
To understand how to create or edit an object within another wizard, saving both objects from each wizard within the same transaction. As part of understanding this approach developer will learn about data handling for creating and editing objects using JCA wizards, using JSON and object type instance APIs.

## UI for the wizards

The following is an example of a child wizard launched from within another JCA wizard.

Parent Wizard (Change Task Wizard 🡪 Select Action Items step 🡪 Create icon from the table level action)



**Child Wizard (Change Action Item)** 

# Solution

Create a Form Processor and delegate that will define the attributes in JSON representation to save and update the object attributes within wizards. The child objects are created within create wizard of the parent wizard and data of the child wizard can be edited without the child objects being saved. The following diagram is the flow of the design approach.

After entering the attribute values on the child wizard and clicking “Finish”, wizard calls the registered create form processor.

Create and register a new step for holding the row for the child objects. The table will have row level “Edit” action and table level “Create” action of type “New”.

Delegate are registered for the parent step that holds the child objects.

Note: the action s are launched with “New” Type.

End

The delegate will create or update the attributes of the child object.

User submits the parent wizard after which the registered delegate for the step is called.

The edit form processor builds the JSON string and calls the java script which adds the row in the parent table.

Launch the edit wizard using edit action. Attribute populator for the edit wizard gets the persisted values for first edit or non persisted values for edit on edit case, to display the most latest updated values.

The data returned from the form processor is stored in the hidden field using a data utility.

The create form processor builds the JSON string and calls the java script which adds the row in the parent table.

Start

After entering the updated values and clicking “Finish” calls the registered edit form processor.

## Prerequisite knowledge

To apply this best practice, you need to have an understanding of the following:

* The JSP-based client architecture framework in Windchill
* The actions framework in the Windchill client architecture

## Execution of the wizards

A new table level action is defined to launch the create change action wizard. If any data is needed from the parent window on creation of the child object then wizard action window type should be set to “New”. On entering attribute values and clicking Finish a Form Processor registered for the change action item wizard is called to form the java script function and calls the java script function to add a row in the parent table.

In the form processor, the client data is retrieved from the object bean and converted to the Type Instance of the object. Then, using the type instance of the object, a JSON string is created, which is then passed to the java script function to add the row in the parent table. Please see below for the code snippet.

@Override

public FormResult postTransactionProcess(NmCommandBean clientData, List<ObjectBean> objectBeans) throws WTException {

ActionItemDataHandler handler = new ActionItemDataHandler();

//using JSON object to populate the attributes

JSONObject json = new JSONObject();

String jsScript = "";

FormResult formResult = new FormResult(FormProcessingStatus.SUCCESS);

formResult.setNextAction(FormResultAction.NONE);

// there should be only one object bean within the list since processing the the create wizard.

for(ObjectBean objBean:objectBeans) {

String tableId = ChangeManagementClientHelper.getTableId(objBean, true);

TypeInstance ti = **createAndValidateTypeInstance**(clientData, objBean, formResult);

try {

String numberValue = getValueForNumber(objBean);

if (numberValue == null || "".equals(numberValue)){

// Edit on create case, just get the generated string

NumberInputComponent numberInputComponent = new NumberInputComponent("");

numberValue = numberInputComponent.getLocalizedDisplayStringForServerGenerated();

}

addAttributeToTI(ti, "number", numberValue);

json = handler.toJSON(ti);

} catch (Exception e) {

throw new WTException(RESOURCE,changeActionItemClientResource.CHANGE\_ACTION\_ITEM\_PROCESSING\_ERROR,null );

}

jsScript = generateJavaScript(json.toString(),tableId, objBean);

formResult.setJavascript(jsScript);

formResult.setNextAction(FormResultAction.JAVASCRIPT);

}

return formResult;

}

Here is the java script code snippet that will help to add the row to the parent table.

*return {*

*handleSubmit: function (oid, jsonAttributes, tableId, isUpdate) {*

*mform = opener.Form.serialize (opener.getMainForm ());*

*mform += "&" + oid+ "=" + jsonAttributes;*

*var rows = [oid];*

*if (oid! == null && oid! == "" && oid.indexOf ("cai\_") ==-1 || isUpdate === true) {*

*opener.rowHandler.removeRows (rows, tableId, true);*

*}*

*var params = {doAjaxUpdate: true, preventDuplicates: true, iframeForm: mform};*

*opener.rowHandler.addRows(rows,tableId,null,params);*

*wfWindowClose1();*

*}*

*};*

For the conversion from JSON to Type Instance and vice versa, please see attached the ActionItemDataHandler file.

## Data Utility to hold the data within parent and child wizards

A registered data utility will handle the JSON representation to render the attributes on the child wizard and stored in the parent table.

When a child wizard form is submitted, the json string representation of the child object is added in the hidden field of the parent table. This should be done once the child form processor calls the javascript to add a row in the parent table.

Here is the sample data that will be generated and stored in the hidden field:

*{"WCTYPE|wt.change2.GenericActionItem|com.ptc.ChangeActionItem~MBA|theExecutionStatus~~NEW|-3250022980523355534~":"COMPLETED","WCTYPE|wt.change2.GenericActionItem|com.ptc.ChangeActionItem~MBA|number~~NEW|-3250022980523355534~":"(Generated)","itemTypeInstanceId":"WCTYPE|wt.change2.GenericActionItem|com.ptc.ChangeActionItem~~NEW|-3250022980523355534","WCTYPE|wt.change2.GenericActionItem|com.ptc.ChangeActionItem~MBA|name~~NEW|-3250022980523355534~":"cai","WCTYPE|wt.change2.GenericActionItem|com.ptc.ChangeActionItem~MBA|containerReference^WCTYPE|wt.pdmlink.PDMLinkProduct~~NEW|-3250022980523355534~WCP|26634|-1":{"value":"WCTYPE|wt.pdmlink.PDMLinkProduct~~WCP|26634|-1","type":"com.ptc.core.meta.common.impl.WCTypeInstanceIdentifier"}}*

The data utility also populates table column for each attribute of the child object.

Please see attached ChangeActionItemTableDataUtility file for detailed implementation of the data utility.



## Persistence of the child object

## Once the parent wizard is submitted, the form processor of the parent wizard will call the form delegate registered against the Change Action Item step, which will be responsible for saving the object. Please see section 3.5 “Code snippets converting a JSON to Type instance and vice versa” for details about converting and manipulating the attribute data.

## Code snippets converting a JSON to Type instance and vice versa

The following APIs will help with converting a type instance and attributes into a JSON object and vice versa.

/\*\*

\* Helper method to translate the JSON object to a typeinstance.

\*

\* **@param** json

\* **@param** isEdited

\* boolean flag indicating whether json is from edit. If true the

\* type instance state will be set to State.CHANGED.

\* **@return** type instance created from the json attributes

\* **@throws** Exception

\*/

**public** TypeInstance fromJSONAsTypeInstance(JSONObject json, **boolean** isEdited)

/\*\*

\* Converts the TypeInstance to JSON string representation

\*/

**public** JSONObject toJSON(TypeInstance ti)

Please see attached ActionItemDataHandler.java file for detailed implementation of the conversion APIs used.



## Usage of Attribute Populator (Edit wizard)

Edit action wizard can be launched from the parent step needs to be defined in an action with window type “New” to get the form data available. The form data is needed in order to populate the previous set attribute in the wizard that were saved in the parent form as a JSON object. The attribute populator is used to handle the persisted edit case and also the case where the object is edited and not persisted. The attribute populator will use the JSON object that is stored in the parent wizard to populate the attributes for subsequent edits. The code snippet below is an example of the prePopulateAttributeDisplayvalues implementation of an attribute populator and how the ActionItemDataHandler is used to convert the attributes from TypeInstance to JSON and updating the Type instance.

/\*\*

\* Will update the type instance with the updated attributes that are not persisted yet.

\* This can be used in scenario where the <code> GenericActionItem</code> object is edited

\* and edit action is launched again without persisting object. The updated type instance will

\* populate the last updated attributes. This will also handle the soft attributes updates.

\*/

@Override

**public** TypeInstance prePopulateAttributeDisplayValues(TypeInstance ti, NmCommandBean cb) {

NmOid oid = **null**;

TypeInstance updatedTi = **null**;

**try** {

oid = cb.getPageOid();

*logger*.debug("NmOid is: " + oid);

**if** (oid != **null**) {

String jsonString = **null**;

jsonString = ChangeManagementClientHelper.*getParameterWithObjectHandle*(oid.getHTMLId() + "\_json", cb);

**if**(jsonString == **null**){

//edit change task in the edit change notice wizard

jsonString = cb.getTextParameter(oid.getHTMLId() + "\_json");

}

*logger*.debug("json string is: " + jsonString);

**if**(jsonString!= **null**) {

ActionItemDataHandler handler = **new** ActionItemDataHandler();

JSONObject jsonObj = **new** JSONObject(jsonString);

//removing the attribute to get type instance.

jsonObj.remove("isUpdate");

updatedTi = handler.fromJSONAsTypeInstance(jsonObj, **true**);

}

}

} **catch** (Exception e) {

*logger*.warn("An exception occured while processing the GenericActionItem values.", e);

}

**if** (updatedTi != **null**) {

updatedTi.merge(ti);

} **else** {//updateTi could be null when launched from task info page

*logger*.debug("updatedTi is null. setting it to the input type instance");

updatedTi = ti;

}

**if** (*logger*.isDebugEnabled()) {

*logger*.debug("update type instance is " + updatedTi);

}

**return** updatedTi;

}

**Note**: The edit jsp might have to specify “objectHandle” to handle the child attribute.

## Solution Elements

|  |  |  |
| --- | --- | --- |
| **Element** | **Type** | **Description** |
| com\ptc\windchill\enterprise\changeActionItem\dataUtilities\ChangeActionItemTableDataUtility | Java | Data utility responsible for the rendering of the child object (ChangeActionItem) attributes. |
| com\ptc\windchill\enterprise\changeActionItem\forms\delegates\ChangeActionItemsFormDelegate.java | Java | Delegate responsible for persisting of the child object. Registered against the Change Action Item step on the change task wizard. |
| com\ptc\windchill\enterprise\changeActionItem\forms\processors\ChangeActionItemFormProcessor.java | Java | Handling the create wizard attribute to form a JSON string and call the java script api to add the row in the parent wizard table. Registered against the Change Action item create wizard. |
| com\ptc\windchill\enterprise\changeActionItem\forms\processors\EditChangeActionItemFormProcessor.java | Java | Handling the edit wizard attribute to form a JSON string and call the java script api to add the row in the parent wizard table. Registered against the Change Action item edit wizard. |
| com\ptc\windchill\enterprise\changeActionItem\forms\processors\EditChangeActionItemFormProcessor.java | Java | Handle the edit case for the edit wizard |
| com\ptc\windchill\enterprise\changeActionItem\ActionItemDataHandler.java | Java | Helper class to manipulate json object to type instance and vice versa. |
| netmarkets\jsp\changeActionItem\changeActionItemsStep.jsp | Jsp | Attributes that are rendered in the create wizard also includes the data utility usage |
| netmarkets\jsp\changeActionItem\changeActionItemsTable.jsp | Jsp | The table on the parent wizard step that holds the change action item objects |
| com\ptc\windchill\enterprise\changeActionItem\forms\populators\ChangeActionItemAttributePopulator.java | Java | Populates the attribute values for the child object (ChangeActionItem) |

</INTERNAL>

<INTERNAL>

# Other Windchill Development Considerations

JCA Wiki Page: <http://rdwiki.ptcnet.ptc.com/mediawiki/index.php?title=Main_Page_JCA_Infrastructure>

</INTERNAL>