SDN Contribution

KM Search and Replace: Web Dynpro Java KM-API Example

Applies to:

This tutorial applies to the custom development IT Scenario. It utilizes Web Dynpro Java along with the KM APIs. This demo was designed and built on an SAP NetWeaver 04s system, although its core concepts should be backwards compatible with SAP NetWeaver 04.

Summary

This tutorial was inspired by the February 2006 SDN Contributors Challenge:

KM Content Search & Replace Tool, your task is to write a Java or .NET (through the .NET PDK, if possible) application that goes through a folder in KM recursively (sub-folders, folders within sub-folders, and so forth) and does a search and replace of text you specify in any documents that are text-based, i.e. XML, HTML, ASCII TEXT, etc.

One catch, this should run as an iView inside the portal!

Created on: April 18, 2006

Author Bio

Thomas Jung is an SAP NetWeaver Product Manager focusing on the development aspects across enterprise information management. Before joining SAP Labs in 2006, Thomas was an applications developer for an SAP customer company. He was involved in SAP implementations at this customer as an ABAP developer for nearly 10 years. He is also the co-author of the SAP Press Book, Advanced BSP Programming.

Table of Contents

Application Overview

We are going to build a Web Dynpro Java application that takes advantage of the KM APIs in order to fulfill the requirements of the Contributor Challenge. To support these requirements, we are going to need an application that has 3 major sections.

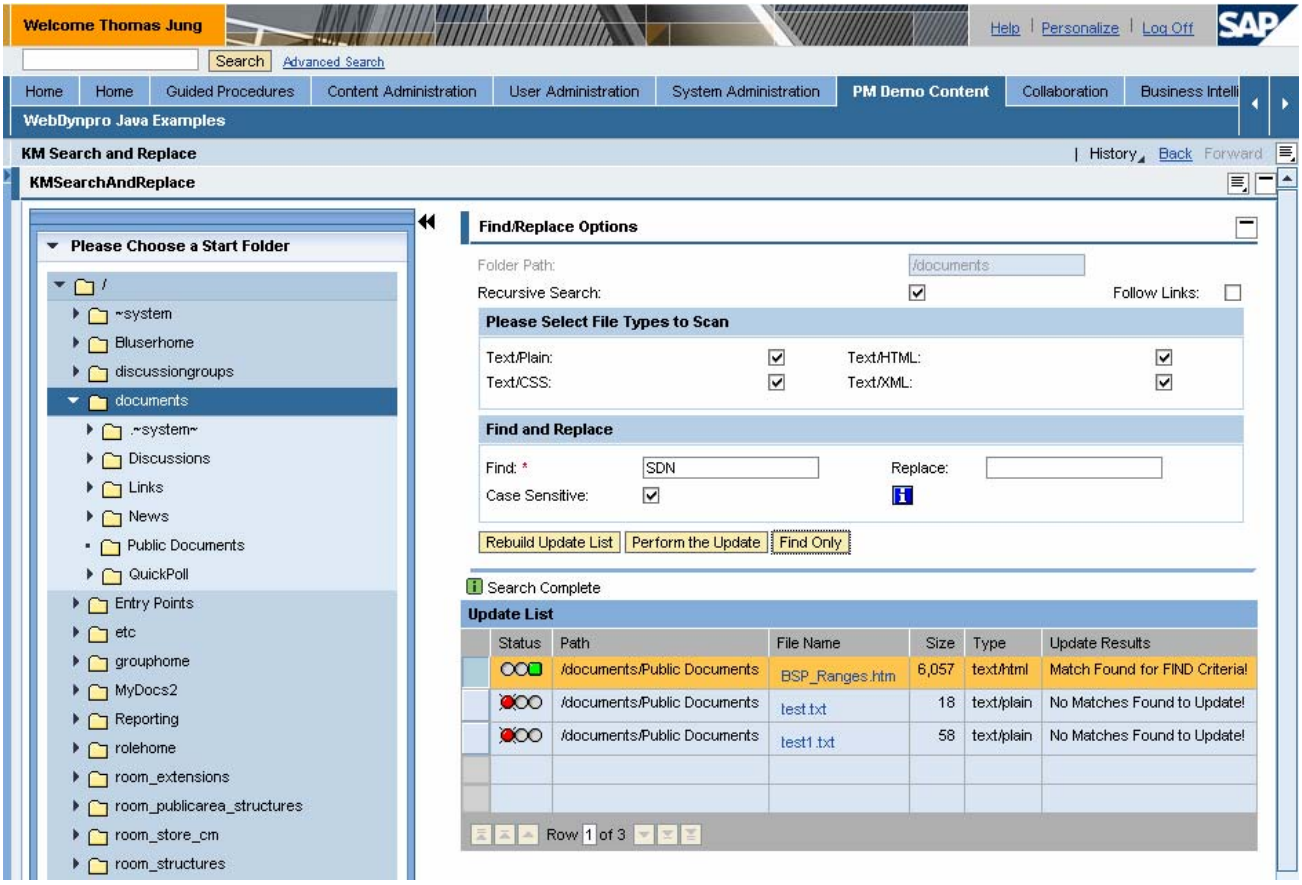


Figure 1 - Application Overview

On the left side of the application, we have the Navigation Area. Here we have displayed all the repositories and folders within the repositories as a Navigation Tree. Users can navigate down through this tree until they find the point where they want to start their search. They can then click on the folder name to select that folder.

Once the user has selected the Repository Folder to start their search from, they can hide the Navigation Tree. There is a small Icon that sits just to the right of the top of the Navigation Tree. This icon can be used to collapse or expand the Navigation Area.

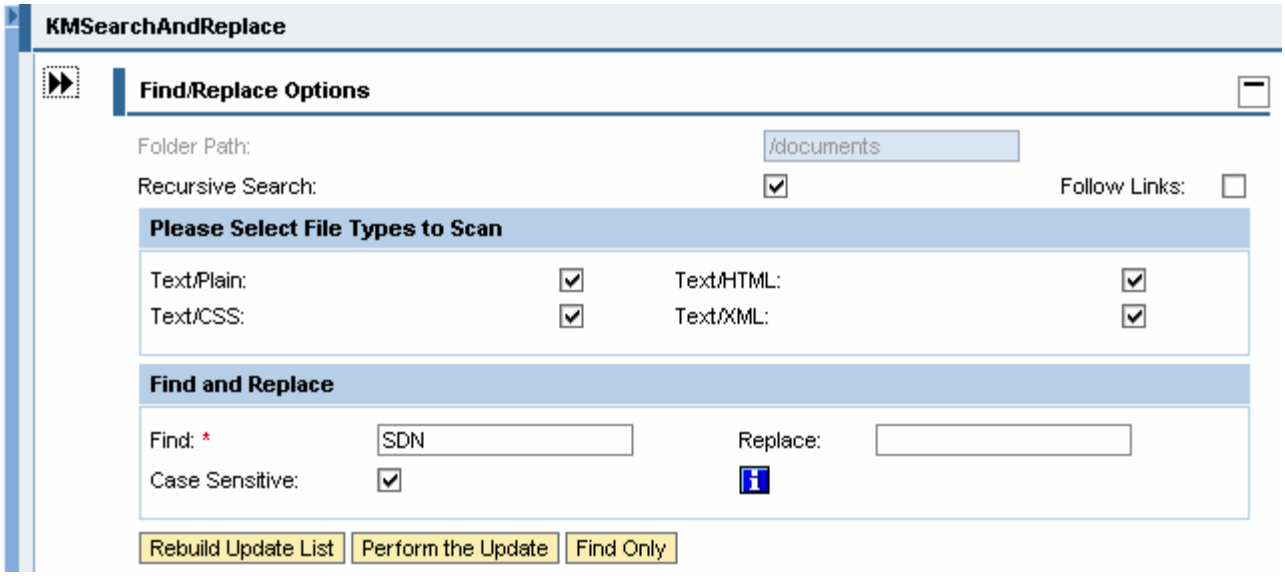


Figure 2 - Collapsed Navigation Area

On the right side at the top, we have the Search Options. We display the folder path that is currently selected. We then have two checkboxes that control the depth of the search. The user can activate/deactivate the Recursive Search or choose to follow internal links that would navigate outside the folder hierarchy that was selected.

In the selection entitled “Please Select File Types to Scan”, users can control which of the major text file types will be picked up by the search.

In the “Find and Replace” section itself, we have the input fields for users to supply their find and replace criteria. Users can keep this as a simple implicit search and replace. However these fields feed directly into the Regular Expression engine that is used to perform the search. Therefore adventurous uses can include Regular Expression syntax in these fields as well. The blue image with the white I, opens an Internet link to the Wikipedia page on Regular Expression, in case the user needs some help formatting their search. Finally users have a checkbox to choose between case sensitivity/insensitivity.

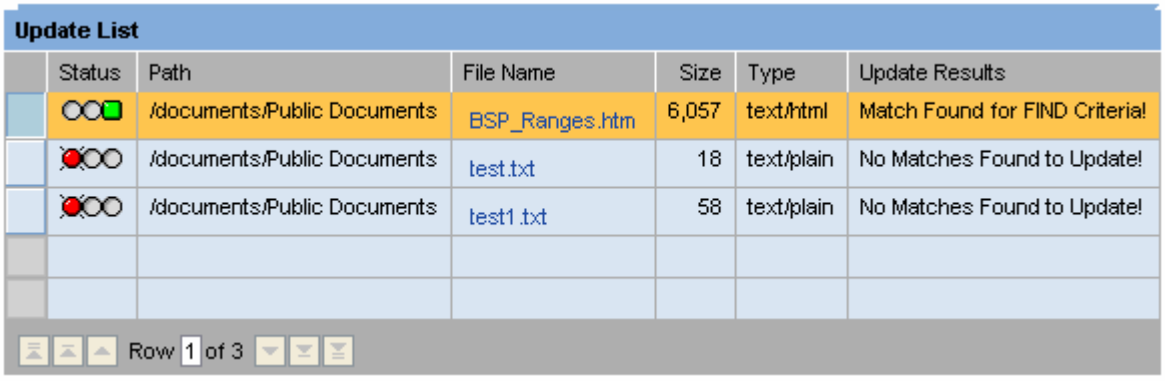


Figure 3 - Update List

Finally we have the Update List. After pressing the Rebuild Update List, this table will display all repository items that match file type criteria within the search scope (start path + follow links + recursive search).

Each item displays their current update status, the repository path, their display name (which is also a hyper link to open the actual item), the content size, the content type and any Update or Search results.

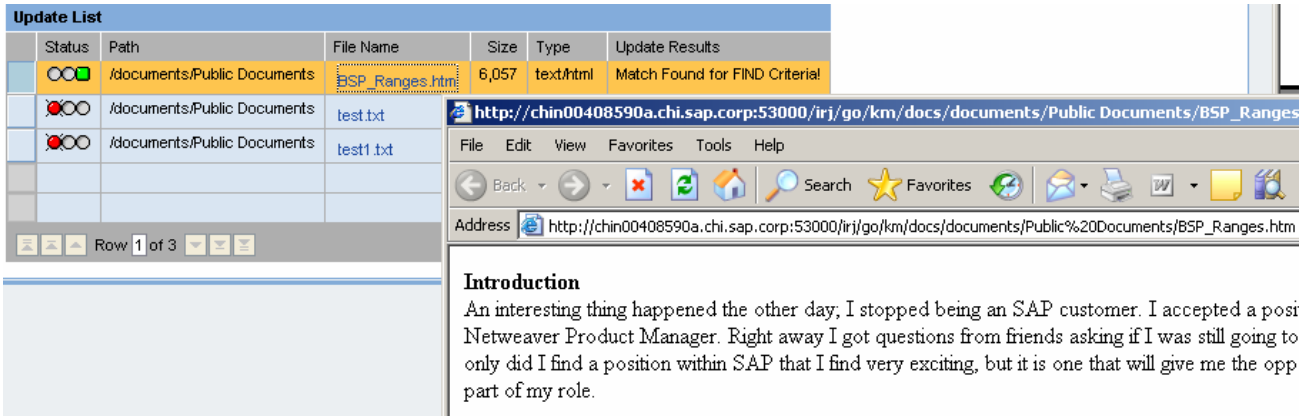


Figure 4 - All Items can be opened via Hyperlink

User can then just do a Find or they can perform a Find and Replace on this update list. If the user chooses to perform a search, a required field check is first performed.

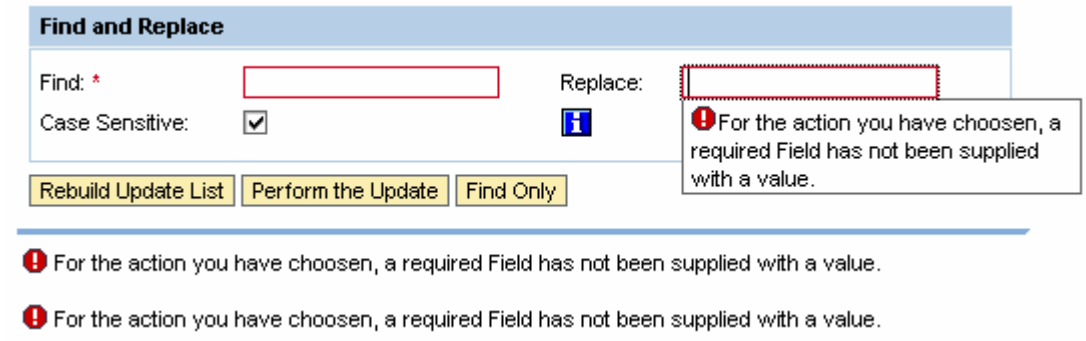


Figure 5 - Required Field Check

If the user has supplied all the required fields, they will receive a confirmation dialog before the updates actually begin.

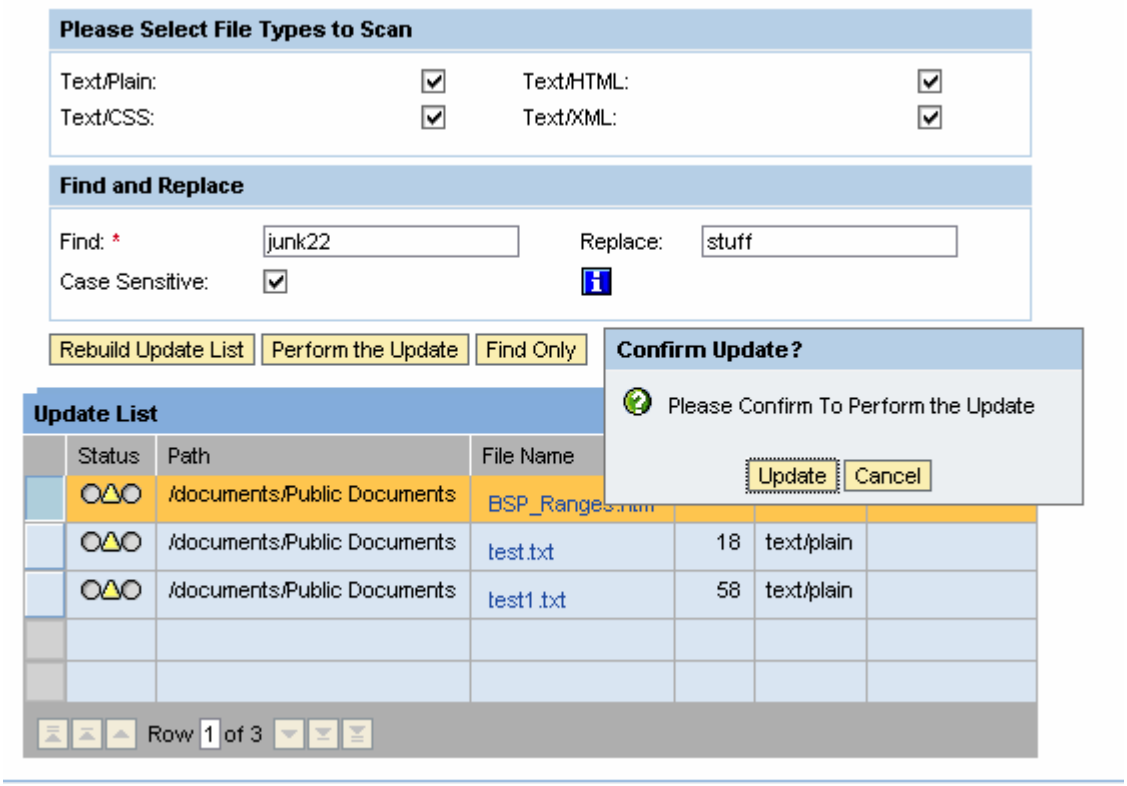


Figure 6 - Update Confirmation

Project Setup

Please note that this tutorial is a little different than most of the other Web Dynpro Java ones. The goal here is to fulfill an entire functional requirement. Therefore this demo utilizes many different UI techniques as well as a considerable amount of functionality that interacts with the KM APIs. It helps to have already studied the Web Dynpro Java tutorials on KM Integration, Trees, Recursive Context Nodes, Dialogs and Tables.

Also for the simplicity of presenting this example, all the logic and UI elements have been placed into a single View. For a real world application it would be advisable to break this single application into multiple Views or perhaps even multiple Web Dynpro Components. The KM Navigation Tree that is part of this application would make an excellent reusable Web Dynpro Component on its own.

Project Creation

We will start by creating a new Project in Netweaver Developer's Studio (File->New->Project). Then choose Development Component/Development Component Project.

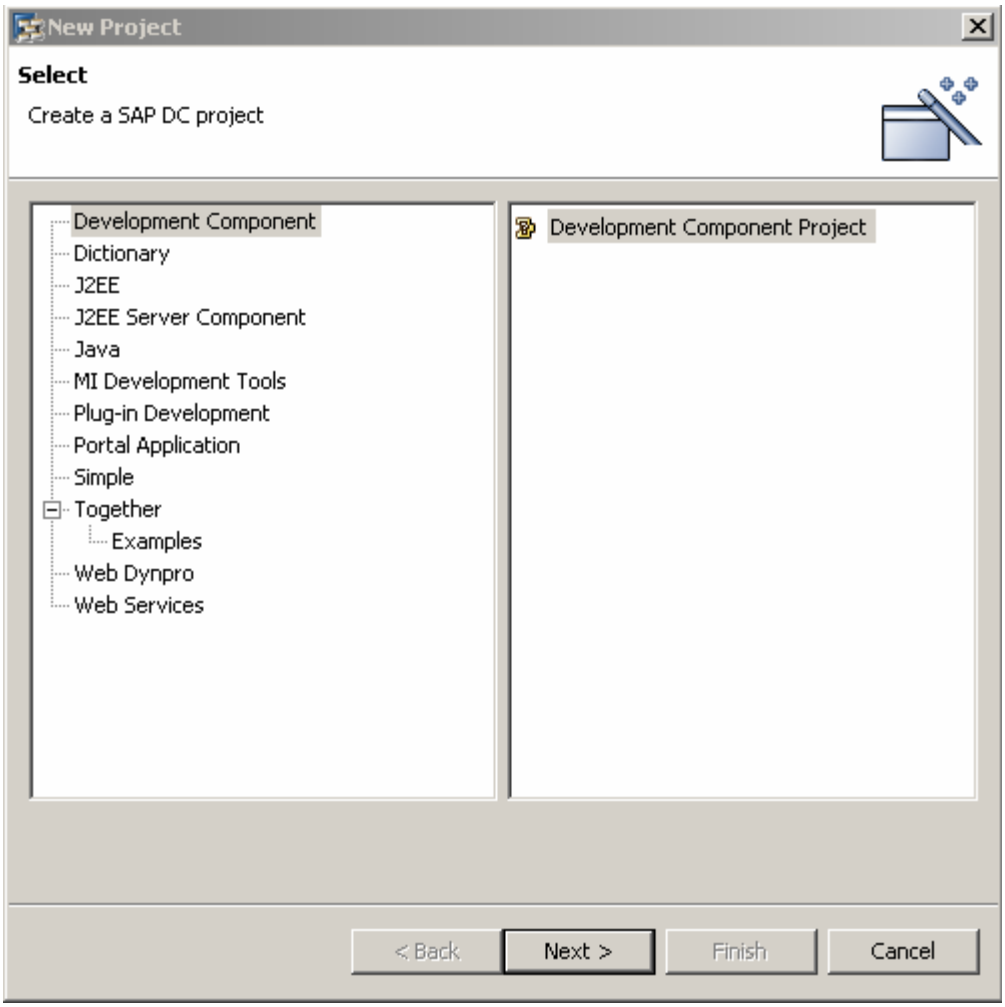


Figure 7 - Create the Development Component Project

Note: although you could choose to create a Web Dynpro directly, it is strongly advised to use Development Components for everything that you do. Please see the excellent SDN Weblog by Chris Whealy for more details.

In the next screen, you will choose the Software Component for your project. You should follow the rules of your company's NWDI if there are any. For my example, I am working with Local Development objects and I have no connection to a NWDI landscape. Therefore I simply choose Local Development->MyComponents.

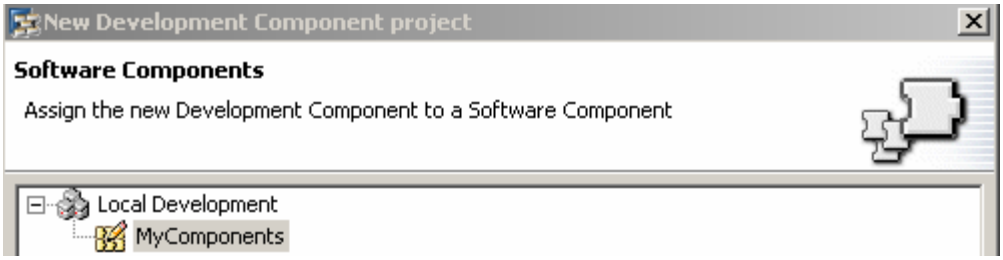


Figure 8 - Software Components

In the next screen, you will create your DC Project. You must supply the Vendor Name and the Project Name. Optionally you may supply a Caption (Description) of the Project. You can leave the Domain at the default value of Basis. Finally you should choose the Web Dynpro as the Project Type.

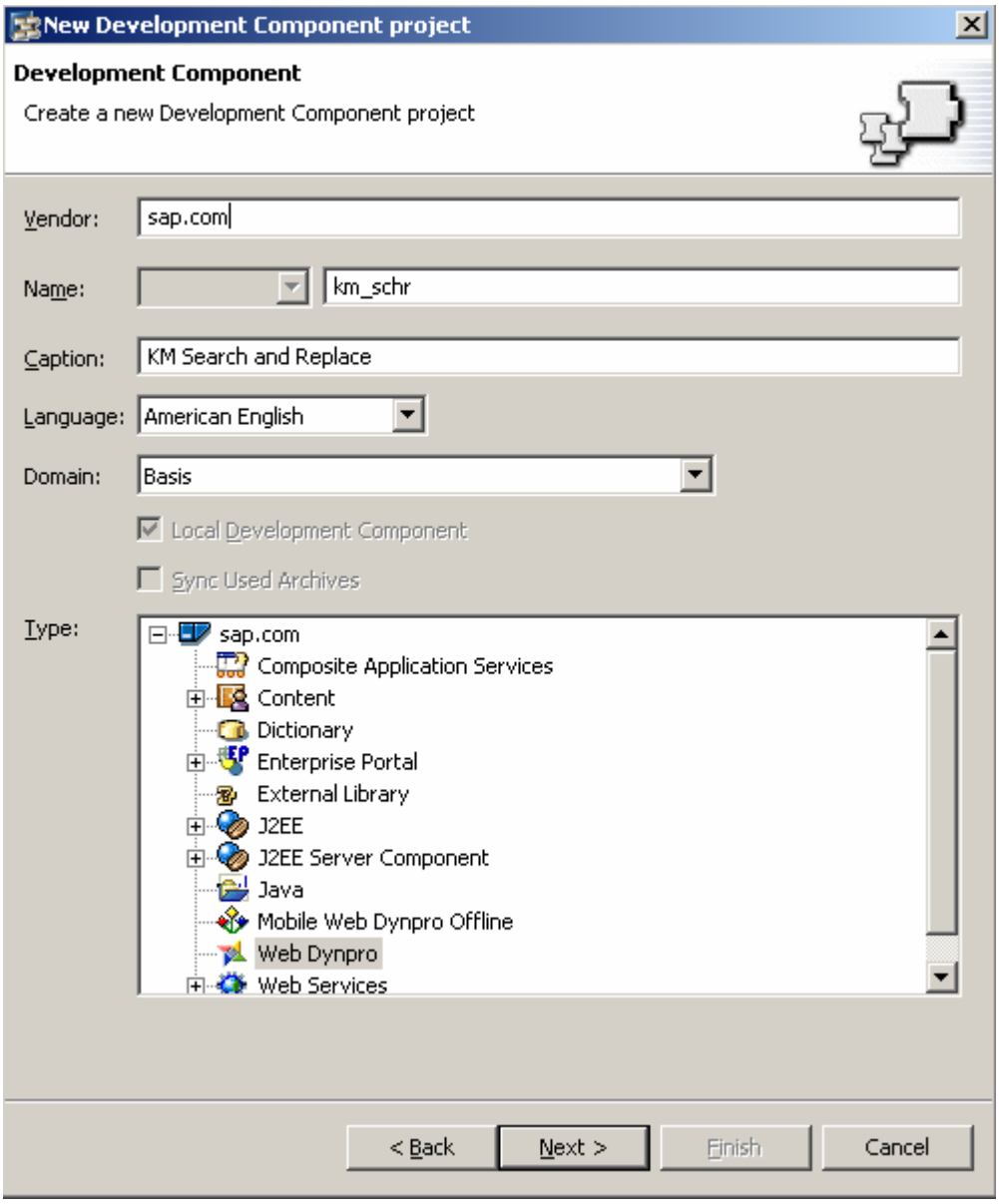


Figure 9 - DC Project Creation

In the next screen, you will choose the location to store the Project.

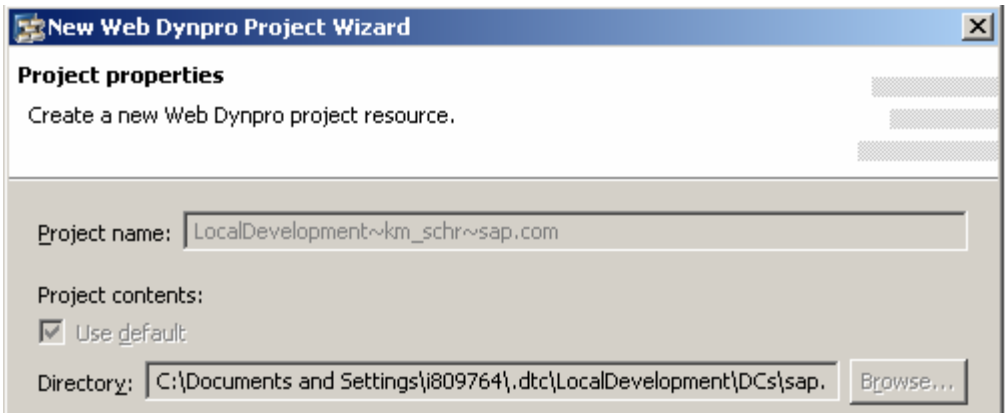


Figure 10 - Project Save Location

Project Properties

We are going to need to access several external JAR files in our project in order to work with the KM APIs. One of the easiest ways to do this is to follow the instructions from the Using Knowledge Management Functionality in Web Dynpro Applications Tutorial (Tutorial #26 on SDN under Web Dynpro Java). We will start by adding two Classpath Variables to our Netweaver Developer's Studio.

Choose Window ->Preferences.

Open *Java* in the tree and choose *Classpath Variables*. Then choose *New*…

In the dialog box displayed enter **WEBAS_HOME** as the *Name*. Enter the following *Path* for the directory:
<harddisk>\usr\sap\<instance name>\JC<instance number>\j2ee\cluster\server<server number>
(for example: c:\usr\sap\F48\JC30\j2ee\cluster\server0)

Create another variable with the *Name* **PORTAL_HOME**. Enter the following *Path* for the directory:
WEBAS_HOME\apps\sap.com\irj\servlet_jsp\irj\root\WEB-INF\portal\
(for example: c:\usr\sap\F48\JC30\j2ee\cluster\server0\apps\sap.com\irj\servlet_jsp\irj\root\WEB-INF\portal\)

You will now need to set some project properties that will allow your Web Dynpro Component to access the KM APIs.

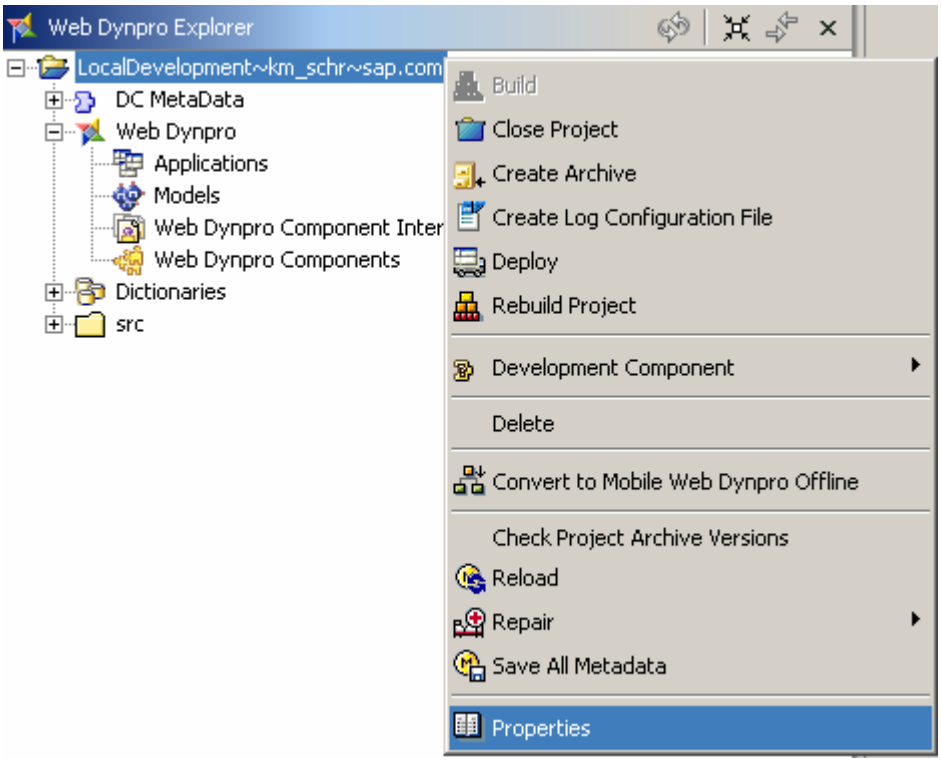


Figure 11 - Maintaining Project Properties

In the dialog box displayed, choose *Java Build Path*, *Libraries* and then *Add Variable*…

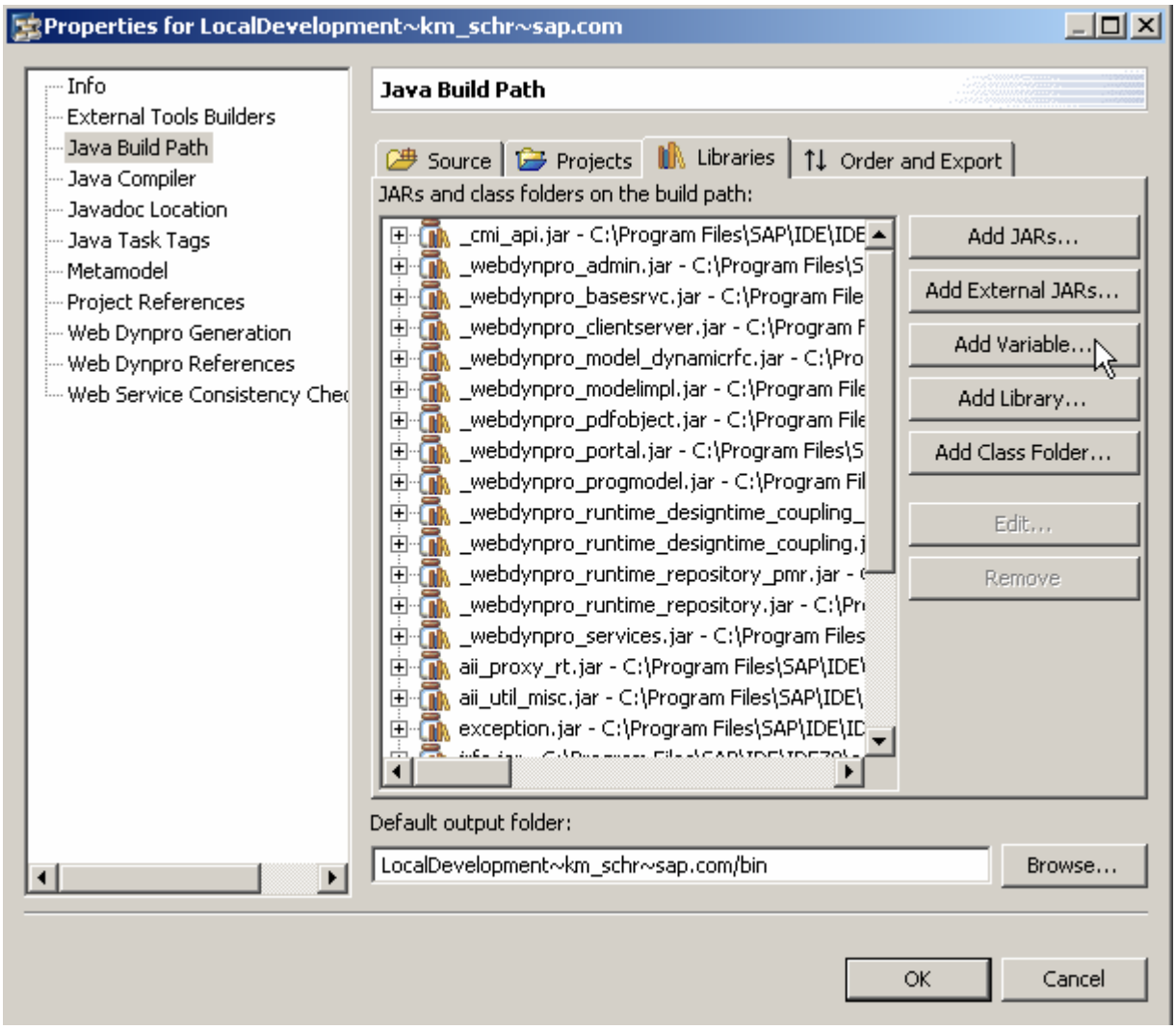


Figure 12 - Java Build Path Maintenance

You will then want to add all the Class Path references to your project that you see in Figure 13:

Figure 13 - Class Path References

While still in the Project Properties window, navigate to the *Web Dynpro References* and then choose the *Sharing References* tab. Insert a reference for the following:
PORTAL:sap.com/com.sap.km.application

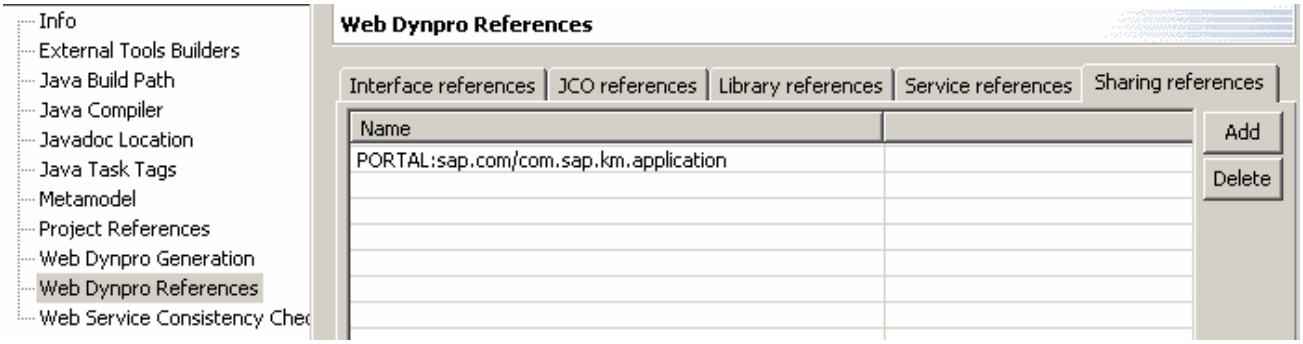


Figure 14 - Web Dynpro References

Web Dynpro Structure

For the next step, we will create the major parts of our Web Dynpro – the Component, the Window, the View, and the Application.

The Component

Right mouse click on *Web Dynpro Component* in the Web Dynpro Explorer View and choose *Create New Web Dynpro Component*.

In the New Web Dynpro Component screen, you should supply a component name and a component package. This screen will also allow us to create our Window and an initial View. Since we will only use one View in this example, this will complete the setup of the Window/View relationship.

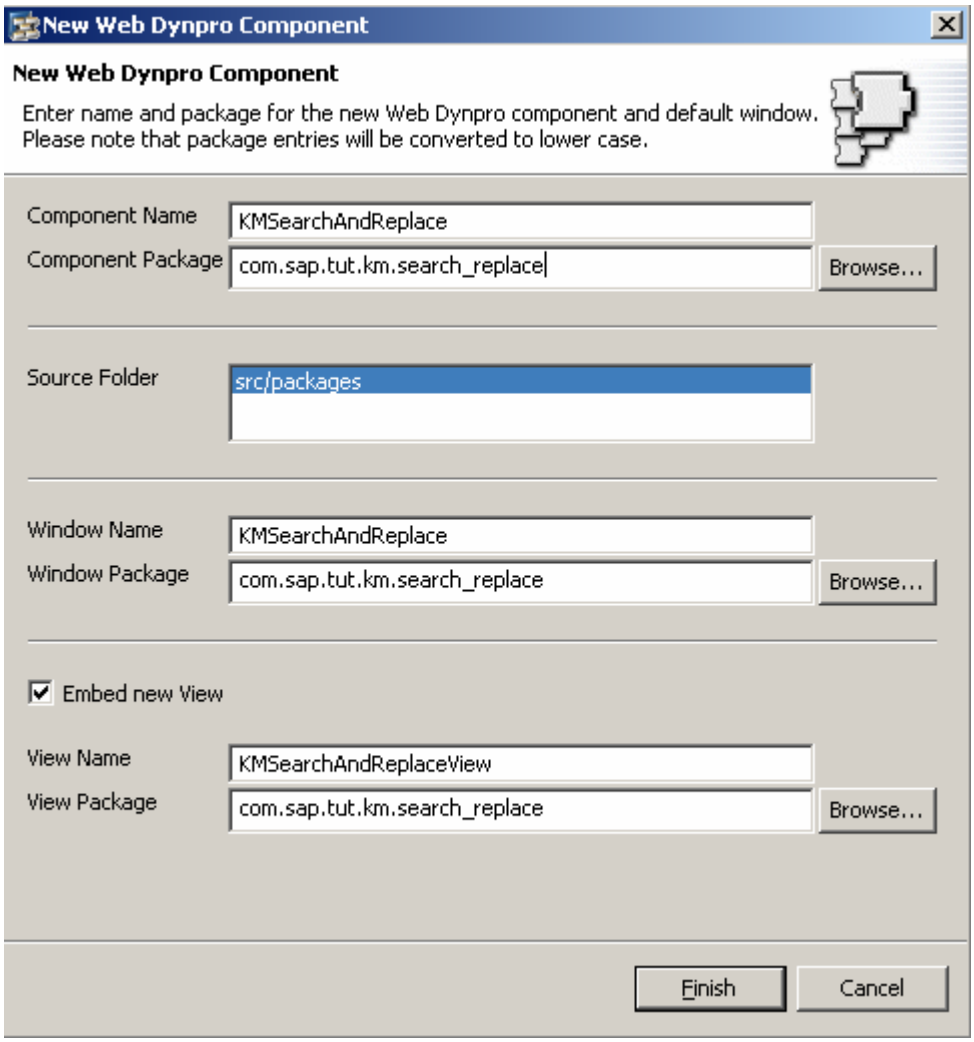


Figure 15 - New Web Dynpro Component Creation

The Application

Right mouse click on *Applications* in the Web Dynpro Explorer and choose *Create Application*.

In the New Application screen, supply the name for the application and its package.

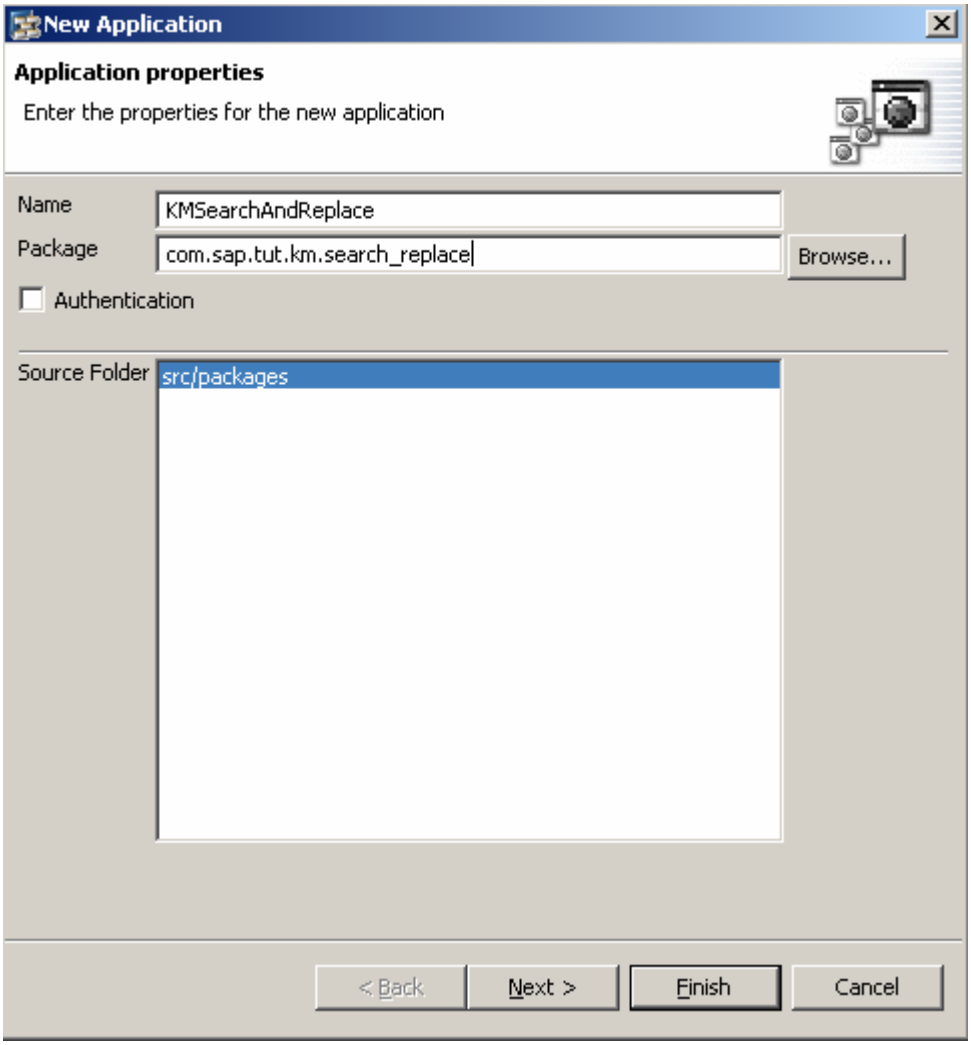


Figure 16 - New Application Creation

The Message Pool

Later in the application we will want to be able to issue an error message if all the required fields have not been filled in. At this point we can go ahead and create this message in the Message Pool.

Right mouse click on *Message Pool* in the Web Dynpro Explorer and chose *Open Message Editor*.

Press the Add Message Button on the right side of the Message Editor.

Figure 17 - Add Message Button

In the Message dialog you first assign a Key to the message. This is what we will use later in our coding to call this particular message. You can then complete the message type and message text.

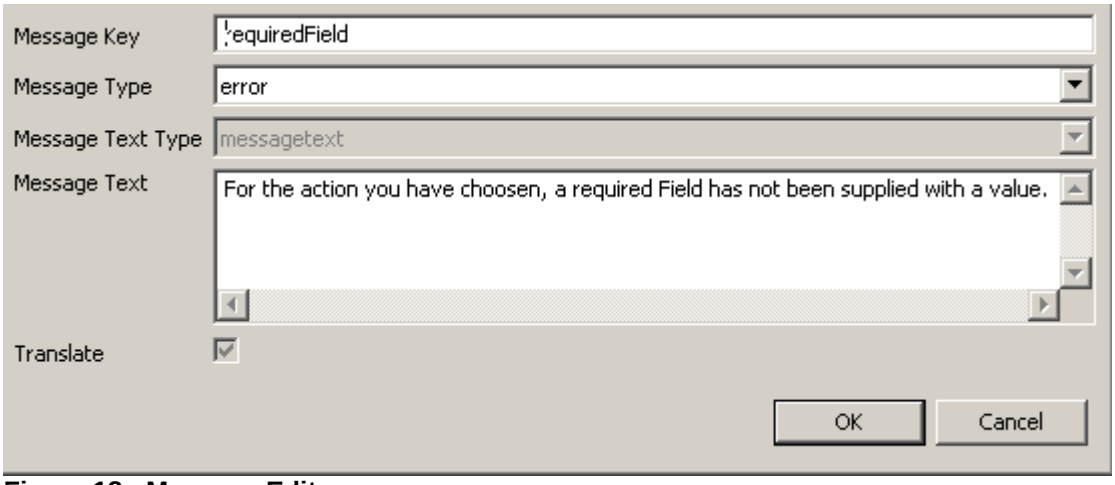


Figure 18 - Message Editor

Project Overview

If you now look at your project in the Web Dynpro Explorer it should appear as follows:

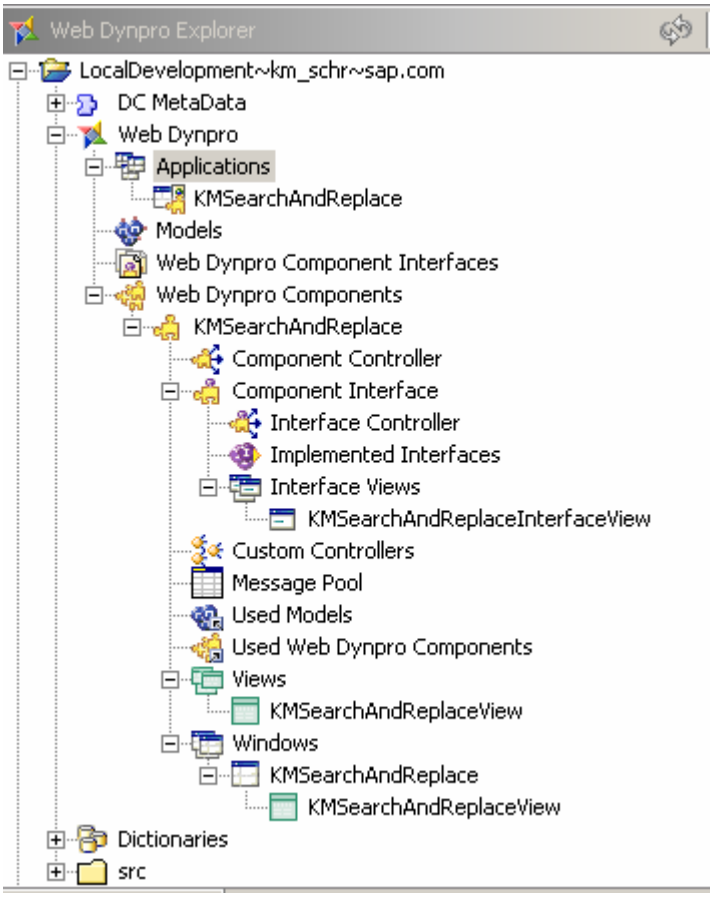


Figure 19 - Web Dynpro Project Structure

The View

The remainder of the project will all take place within the View that was generated during the component creation. We will start by defining our Context structure and our Actions. This will give us an idea of what data we want to interact with through the UI and what activities need to be triggered by the UI. Then we can design the UI in the Layout Tab. Finally we will complete the project by adding coding to the Implementation.

View Context

Our View Context will contain three nodes: FolderContent, Update, and UpdateList.

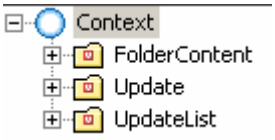


Figure 20 - View Context Nodes

FolderContent Node

The FolderContent Node will represent the data for our Navigation Tree UI element. Therefore the node itself will have a cardinality of 0..n to allow for multiple elements. We will only allow the single selection of an element in the navigation tree, therefore the selection will be set to 0…1.

Property	Value
cardinality	0..n
collectionType	list
initializeLeadSelection	true
name	FolderContent
selection	0..1
singleton	true
structure	
supplyFunction	
technicalDocumentation	
typedAccessRequired	true

Figure 21 - FolderContent Node Properties

The FolderContent Node will have several attributes to support the UI properties of a Navigation Tree Node.

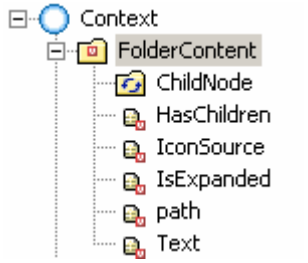


Figure 22 - FolderContent Attributes

However most important is the Recursive Node, ChildNode. You create this special node type by right mouse clicking on the parent node and choosing *New->Recursion Node*.

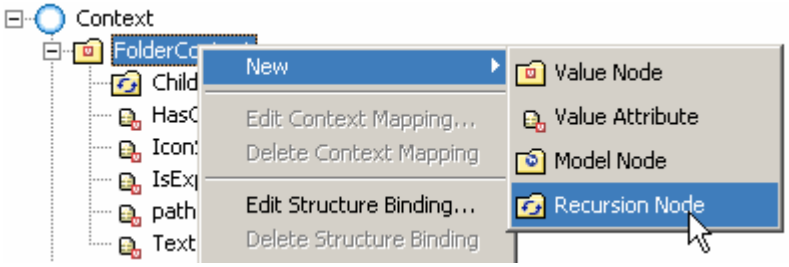


Figure 23 - Recursion Node Creation

A relationship is then created that allows the structure of the parent, FolderContent, to be repeated. This is simple way of representing the parent/child relationship that we will visualize within the Tree UI element later.

Property	Value
name	ChildNode
repeatedNode	KMSearchAndReplaceView.FolderContent

Figure 24 - Recursion Node Properties

The remaining attributes have the following types:
HasChildren type **Boolean**
IconSource type **String**
IsExpanded type **Boolean**
path type **String**
Text type **String**

Update Node

This Context Node represents the input UI elements in the upper right hand side of our application. This is simply a flat grouping of these input elements, therefore the cardinality of the node is 1..1.

Property	Value
cardinality	1..1
collectionType	list
initializeLeadSelection	true
name	Update
selection	1..1
singleton	true
structure	
supplyFunction	
technicalDocumentation	
typedAccessRequired	true

Figure 25 - Update Context Node Properties

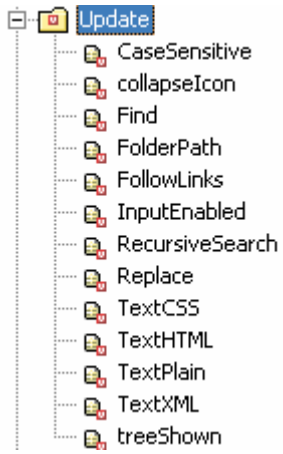


Figure 26 - Update Context Node Attributes

The node attributes have the following types:
CaseSensitive type **Boolean**
collapseIcon type **String**
Find type **String**
FolderPath type **String**
FollowLinks type **Boolean**
InputEnabled type **String**
RecursiveSearch type **Boolean**
Replace type **String**
TextCSS type **Boolean**
TextHTML type **Boolean**
TextPlain type **Boolean**

TextXML type **Boolean**
treeShow type **com.sap.ide.Web Dynpro.uielementdefinitions.Visibility**

Please note that treeShow is not a native type. Therefore to input this, you can use the input help. You can then choose Dictionary Simple Type. In the Local Dictionary, you can find Visibility under com.sap.ide.Web Dynpro.uielementdefinitions.

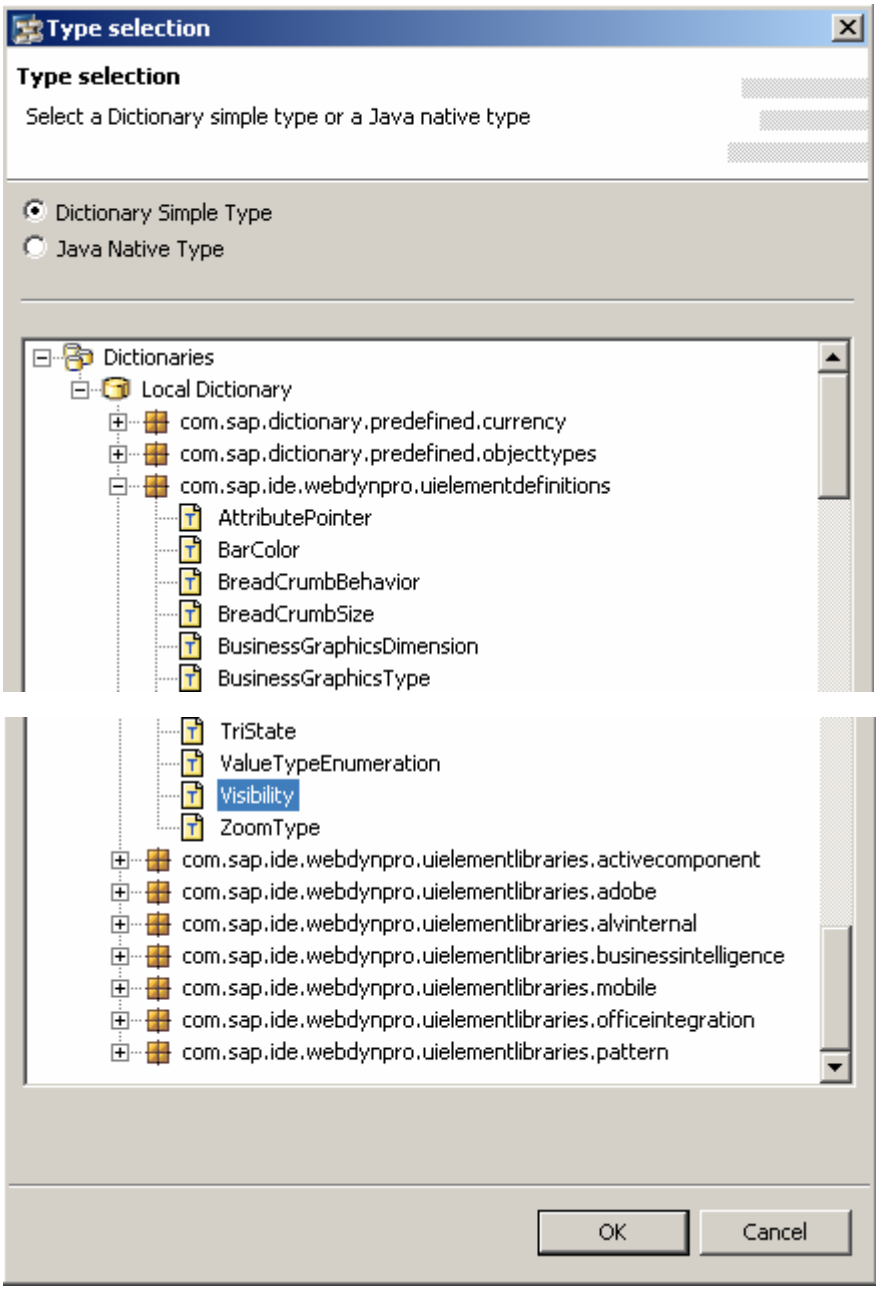


Figure 27 - Visibility Type Input

UpdateList Node

The UpdateList node context represents the data in Table UI element. Since this node will represent the rows of a table, its cardinality needs to be 0..n.

Property	Value
cardinality	0..n
collectionType	list
initializeLeadSelection	true
name	UpdateList
selection	0..n
singleton	true
structure	
supplyFunction	
technicalDocumentation	
typedAccessRequired	true

Figure 28 - UpdateList Node Properties

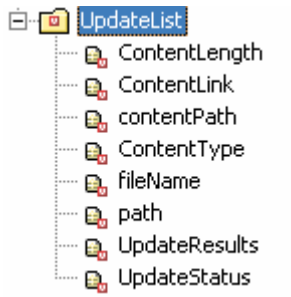


Figure 29 - UpdateList Attributes

The node attributes have the following types:
ContentLength type **Long**
ContentLink type **String**
contentPath type **String**
ContentType type **String**
fileName type **String**
path type **String**
UpdateResults type **String**
UpdateStatus type **String**

Actions

Our UI will have several activities that can fire actions. Therefore we will need to create 8 different actions:

Actions

Displays the actions of the controller

Name	W	Event handler	Text
LoadChildren	☐	onActionLoadChildren	
NodeSelection	☐	onActionNodeSelection	User has selected a Node in the…
Null	☐	onActionNull	Null Action - For Cancels that R…
RebuildUpdateList	☐	onActionRebuildUpdateList	Rebuild Update List
Update	☐	onActionUpdate	Perform the Update
findOnly	☐	onActionfindOnly	Find without the Replace
showHideTree	☐	onActionshowHideTree	Action to toggle the visibility of …
updateOK	☐	onActionupdateOK	Update OK from the Confirmatio…

Figure 30 - Actions Overview

The LoadChildren Action will be triggered when the user expands a node in the navigation tree that has no children. The NodeSelection Action will be tied to the selection of a node in the navigation tree. Both of these actions will need a single parameter to pass along the related Context element to the action. In the LoadChildren action we will name this parameter element and it will represent the parent element that we

need to lookup children for. In the NodeSelection we will call this parameter elementSelection and it will represent the element that was selected by the user. Both parameters can be added by choosing the Type Selection.

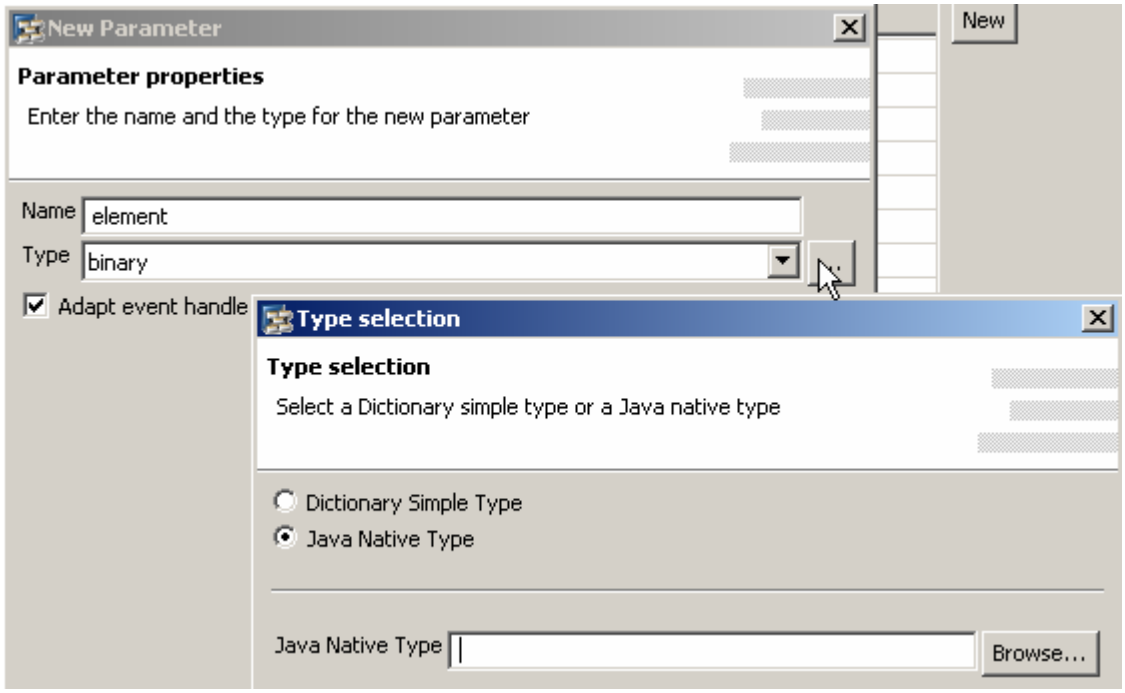


Figure 31 - New Parameter Creation

You can then choose Java Native Type. If you choose the *Browse* Button, you will have a dialog that allows you to search for the type. We will want to use the type that was created to represent our View Context element FolderContent. If you type the first few characters of the name, you should see IFolderContentElement in the listing.

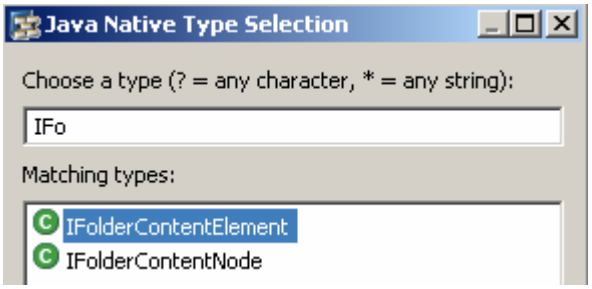


Figure 32 - Parameter Type Selection

None of the remaining actions have any parameters.

View Layout

In order to simplify the creation of the View Layout, we will break it down into several sections on only tackle one section at a time.

RootUIElementContainer

We will start at the RootUIElementContainer. We will only have a few UI elements that are actually children of the RootUIElementContainer. We will want to setup our RootUIElementContainer as a MatrixLayout.

Property	Value
Elementproperties of Transpa...	
accessibilityDescription	
defaultButtonId	
enabled	true
height	
id	RootUIElementContainer
isLayoutContainer	true
layout	MatrixLayout
scrollingMode	none
tooltip	<>
visible	visible
width	
Layout[MatrixLayout]	
stretchedHorizontally	false
stretchedVertically	true

Figure 33 - RootUIElementContainer Properties

If we look at the outline we can see that we will need to create four children elements off of the RootUIElementContainer (you can do this by right mouse clicking on RootUIElementContainer in the Outline and choose *Insert Child*). You should create a ContextualPanel, a LinkToAction, an InvisibleElement, and a TransarentContainer.

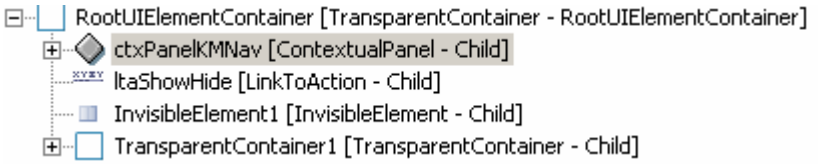


Figure 34 - RootUIElementContainer Children

Property	Value
Elementproperties of UIElement	
enabled	true
id	ctxPanelKMNav
layoutdata	MatrixData
tooltip	<>
visible	Update.treeShown
width	280
Event	
onPersonalize	
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	padless
colSpan	1
hAlign	forcedLeft
height	
vAlign	top
vGutter	none
width	

Figure 35 - ContextualPanel Properties

The ContextualPanel will eventually house all the UI elements on the Left side of our layout.

Most of the properties are quite strait forward. You might notice that the visible property is bound to the Context Attribute Update.treeShown. That way we can trigger the change in the visibility state using an action.

Property	Value
Elementproperties of LinkToAc...	
enabled	true
id	ltaShowHide
imageAlt	
imageFirst	true
imageHeight	
imageSource	Update.collapseIcon
imageWidth	
layoutdata	MatrixData
size	standard
text	
textDirection	inherit
tooltip	<>
type	function
visible	visible
wrapping	false
Event	
onAction	showHideTree
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	top
vGutter	none
width	

Figure 36 - LinkToAction Properties

Next we have the LinkToAction that sits just to the right of our Navigation area. We will use this LinkToAction to display an Image (Notice the imageSource is bound to Update.collapseIcon so that the Image can be change to match the visibility state). When pressed, this element will fire the showHideTree Action.

Property	Value
Elementproperties of Invisible...	
enabled	true
id	InvisibleElement1
layoutdata	MatrixData
tooltip	<>
visible	visible
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	10px

Figure 37 - InvisibleElement Properties

The InvisibleElement really only servers the purpose of providing a small spacer between the Navigation side of our Layout and the input and results side.

Property	Value
Elementproperties of Transpa...	
accessibilityDescription	
defaultButtonId	
enabled	true
height	
id	TransparentContainer1
isLayoutContainer	true
layout	FlowLayout
layoutdata	MatrixData
scrollingMode	none
tooltip	<>
visible	visible
width	
Layout[FlowLayout]	
defaultPaddingBottom	
defaultPaddingLeft	
defaultPaddingRight	
defaultPaddingTop	
wrapping	true
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	left
height	
vAlign	top
vGutter	none
width	

Figure 38 - TransparentContainer Properties

We will use a single TransparentContainer to house all the UI elements that fall into the right side of the screen. Inside the TransparentContainer we will change the layout type from Matrix to FlowLayout.

ContextualPanel

Inside our ContextualPanel we are going to nest several other UI elements. If you right mouse click on the ContextualPanel and choose Insert Item, you will receive a short list of the types of items that can be nested within.

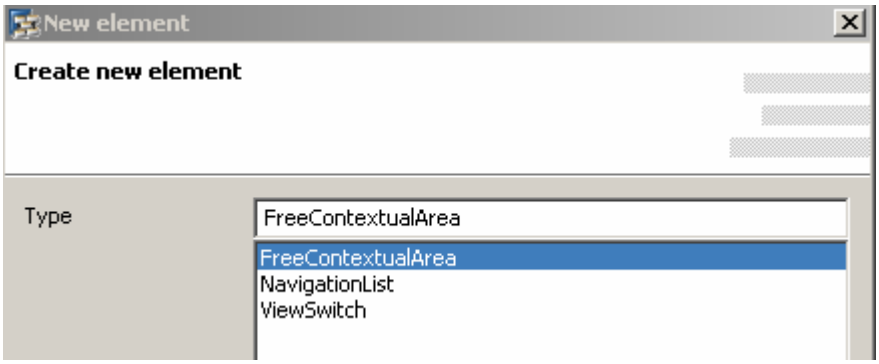


Figure 39 - ContextualPanel, Create New Element

For this example we are going to want to create a FreeContextualArea.

Property	Value
Elementproperties of ViewElement	
contentHeight	0
design	plain
id	freeCxtAreaKMNav

Figure 40 - FreeContextualArea Properties

If you right mouse click on the FreeContextualArea, the menu has two insert options: *Insert Header* and *Insert Content*. We are going to want to use each option. First use the *Insert Header* to create an ExpandableTitle.

Property	Value
Elementproperties of ViewElement	
expandable	true
expanded	true
id	freeCtxAreaHdrKMNav
title	Please Choose a Start Folder
Event	
onToggle	

Figure 41 - ExpandableTitle Properties

Next use the *Insert Content* and create a Tree UI element.

Property	Value
Elementproperties of Tree	
dataSource	FolderContent
defaultItemIconAlt	
defaultItemIconSource	
defaultNodeIconAlt	
defaultNodeIconSource	
enabled	true
id	treeKMNav
minHeight	
rootText	
rootVisible	false
selectionChangeBehaviour	auto
title	
titleVisible	none
tooltip	<>
visible	visible
width	100%

Figure 42 - Tree Properties

You can bind the dataSource for the Tree to the Context Node FolderContent using the value help.

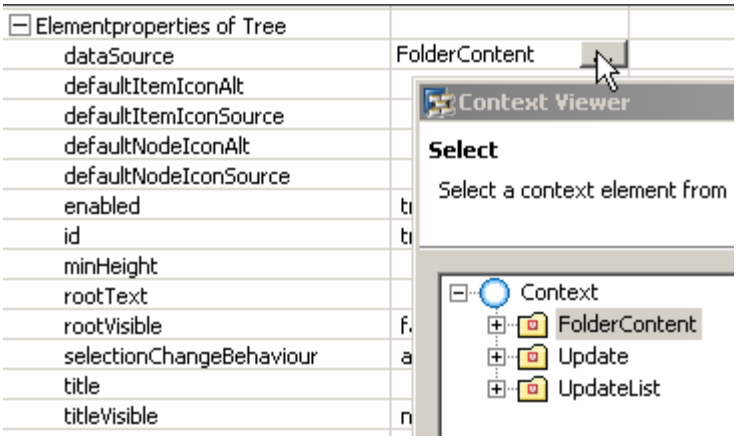


Figure 43 - Binding the dataSource

Now right mouse click on the Tree in the Outline and choose Insert *NodeType*. In the dialog that comes up, you have two choices – TreeItemType and TreeNodeType. Choose TreeNodeType.

Property	Value
Elementproperties of TreeNodeType	
dataSource	FolderContent
design	standard
expanded	FolderContent.IsExpanded
hasChildren	FolderContent.HasChildren
iconAlt	
iconSource	FolderContent.IconSource
id	treeNodeKMNav
ignoreAction	false
text	FolderContent.Text
textDirection	inherit
tooltip	
Event	
onAction	NodeSelection
onLoadChildren	LoadChildren

Figure 44 - TreeNodeType Properties

The TreeNodeType properties are quite important because this is where we are doing the majority of the definition for our navigation tree. This is also where we hooking into both of the Actions related to the navigation tree.

Once finished with all of these steps, the ContextualPanel area should look like the following in the Outline Window:

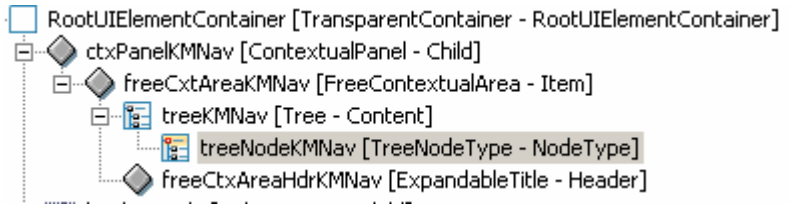


Figure 45 - ContextualPanel Outline

TransparentContainer

On the right side of our layout we have grouped all the UI elements together under a single TransparentContainer.

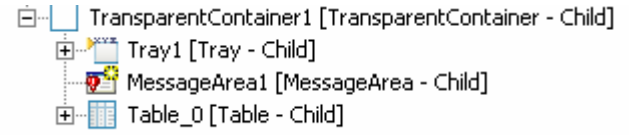


Figure 46 - TransparentContainer Inner Elements

You can Right Mouse Click on the TransparentContainer and choose *Insert Child* to create both the Tray and the MessageArea.

Property	Value
Elementproperties of Tray	
accessibilityDescription	
defaultButtonId	
design	transparent
enabled	true
expanded	true
hasContentPadding	true
height	
id	Tray1
layout	MatrixLayout
scrollingMode	none
tooltip	<>
visible	visible
width	100%
Event	
onToggle	
Layout[MatrixLayout]	
stretchedHorizontally	true
stretchedVertically	true
LayoutData[FlowData]	
cellDesign	padless
paddingBottom	none
paddingLeft	none
paddingRight	none
paddingTop	none
vGutter	none

Figure 47 - Tray Properties

Property	Value
Elementproperties of UIElement	
enabled	true
id	MessageArea1
maxVisibleMessages	0
tooltip	<>
visible	visible
LayoutData[FlowData]	
cellDesign	padless
paddingBottom	medium
paddingLeft	none
paddingRight	none
paddingTop	medium
vGutter	none

Figure 48 - MessageArea Properties

You can of course create the table in the same manner. However it is simpler to right mouse click on the TransparentContainer and choose *Apply Template*. From the Template Wizard screen, you can choose Table.

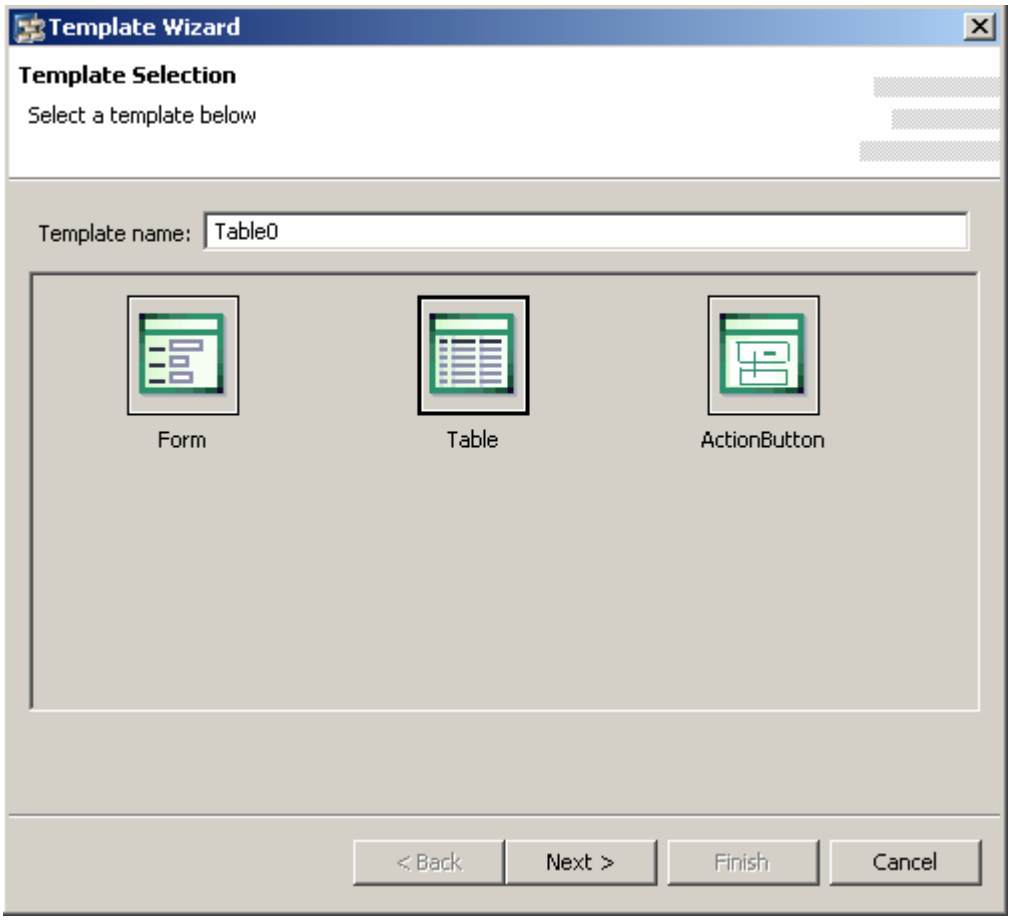


Figure 49 - Template Selection

In the next screen you can choose which context nodes you want to create binds to for the Table Wizard.

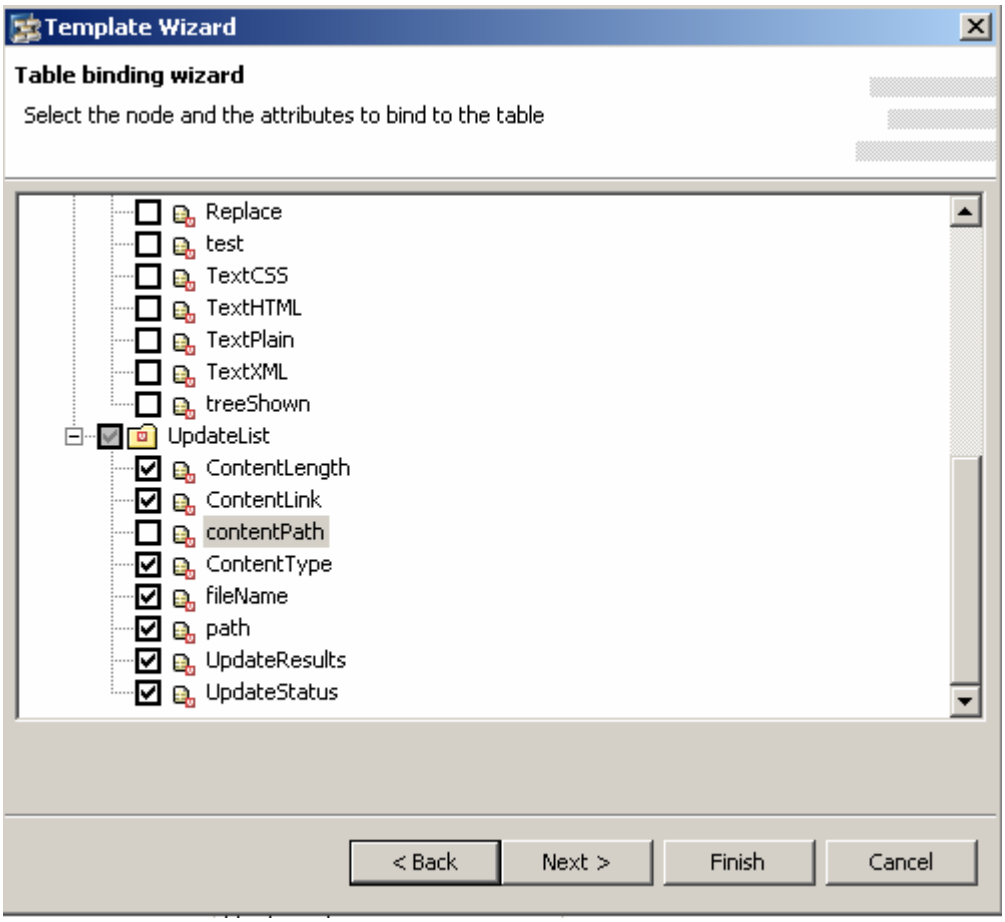


Figure 50 - Table Binding Wizard

In the final screen of the wizard you can adjust the order that the fields will appear in the table. You can also change the type of the UI elements that will be generated. For the Update status attribute we will want to change the Editor to an Image and the Binding Property to source. Also for the fileName and the ContentLink they should both point to the same UI element. This element should be changed to a ToolBarLinkToURL. The fileName context Attribute will become the text Binding Property while the ContentLink will become the reference Property.

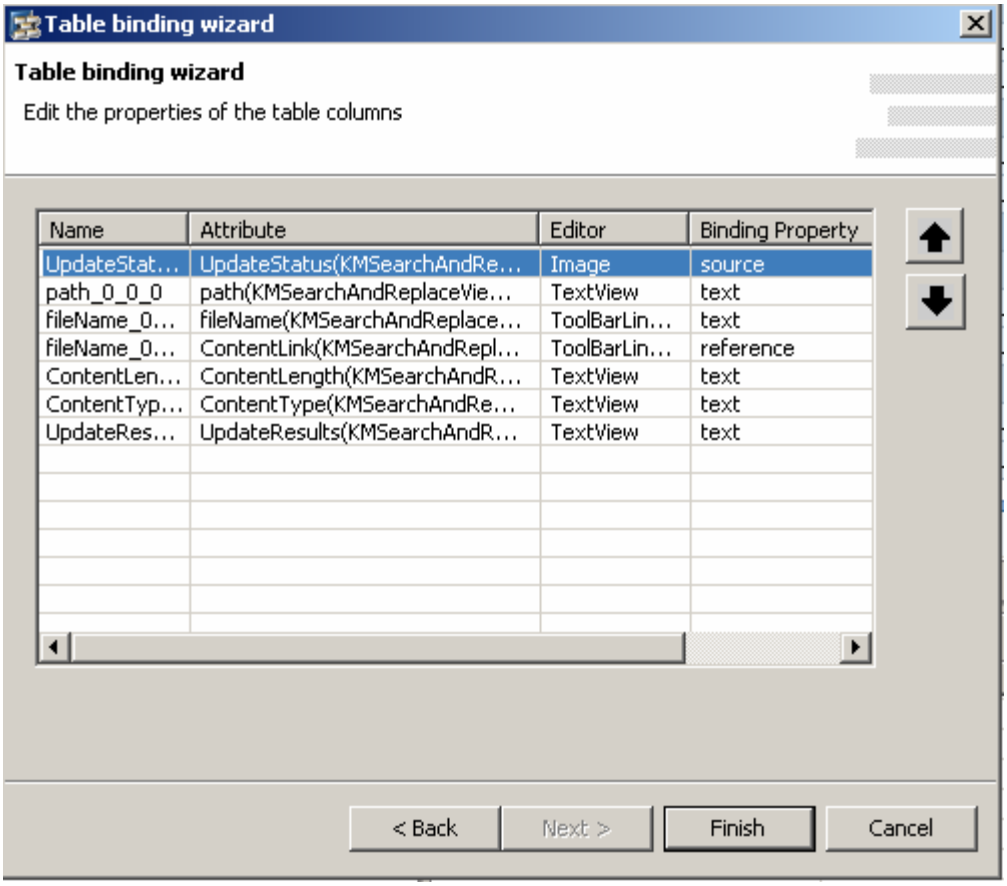


Figure 51 - Table Binding Wizard - UI Element Creation

What is generated by the Table Binding Wizard should look something like the following when viewed in the Outline.

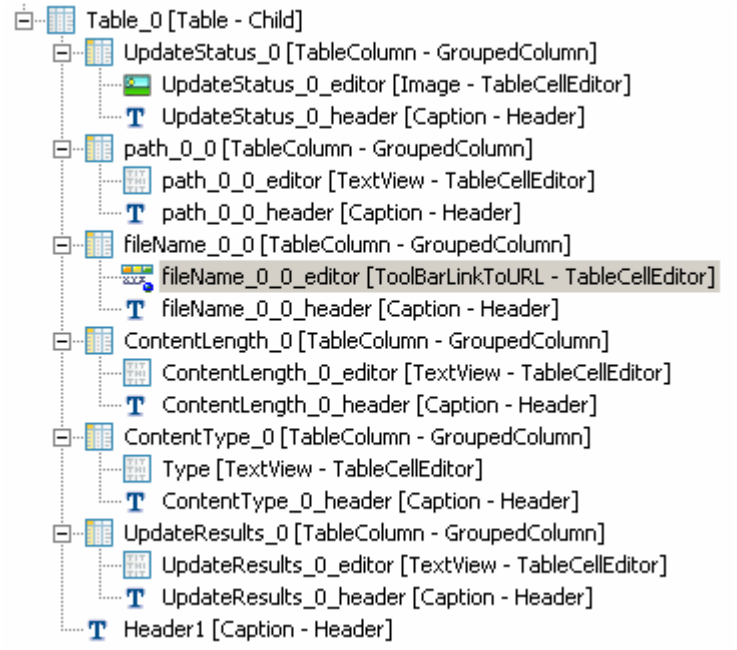


Figure 52 - Table Outline

We can now turn our attention to the UI elements that should be created inside the Tray. The Tray itself will have will have two inner Groups and a ButtonRow in addition to several InputFields and Labels.

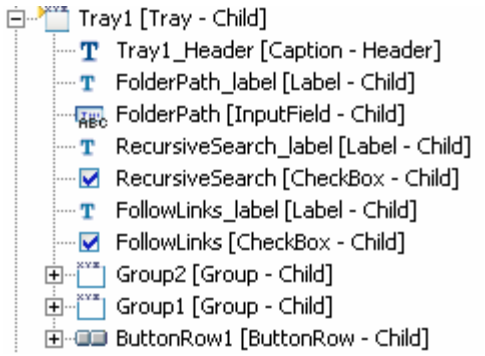


Figure 53 - Tray Outline

We can start creating these elements by right mouse clicking on the Tray. First select *Insert Header*.

Property	Value
Elementproperties of Caption	
enabled	true
id	Tray1_Header
imageAlt	
imageFirst	true
imageSource	<>
text	Find/Replace Options
textDirection	inherit
tooltip	<>
visible	visible

Figure 54 - Tray Header

Next let's use the *Apply Template* option again to run another wizard. This time we will choose the Form wizard.

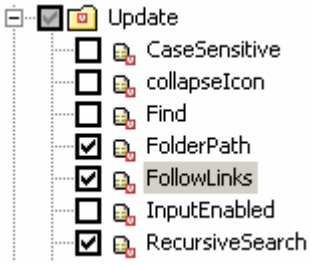


Figure 55 - Form Wizard

The results should be a single Input Field and two CheckBoxes:

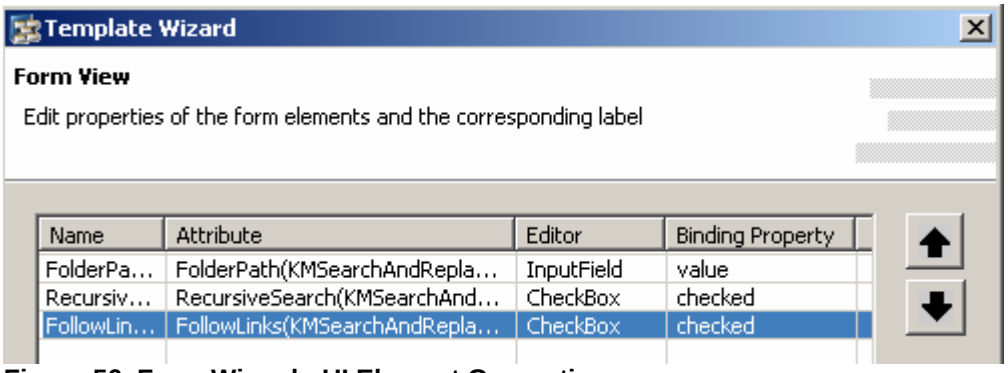


Figure 56 Form Wizard - UI Element Generation

Property	Value
Elementproperties of InputField	
alignment	auto
enabled	false
id	FolderPath
layoutdata	MatrixData
length	<>
passwordField	false
readOnly	false
size	standard
state	normal
textDirection	inherit
tooltip	<>
value	Update.FolderPath
visible	visible
width	
Event	
onEnter	
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 57 - FolderPath InputField Properties

Property	Value
Elementproperties of CheckBox	
checked	Update.RecursiveSearch
enabled	Update.InputEnabled
id	RecursiveSearch
layoutdata	MatrixData
readOnly	false
state	normal
text	
textDirection	inherit
tooltip	<>
visible	visible
Event	
onToggle	
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 58 - RecursiveSearch CheckBox Properties

Property	Value
Elementproperties of CheckBox	
checked	Update.FollowLinks
enabled	Update.InputEnabled
id	FollowLinks
layoutdata	MatrixData
readOnly	false
state	normal
text	
textDirection	inherit
tooltip	<>
visible	visible
Event	
onToggle	
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 59 - FollowLinks CheckBox Properties

We are now ready to create the two Groups. For each Group, you can repeat the *Apply Template* wizard to generate the inner elements.

Property	Value
Elementproperties of Group	
accessibilityDescription	
defaultButtonId	
design	sapcolor
enabled	true
hasContentPadding	true
height	
id	Group2
layout	MatrixLayout
layoutdata	MatrixHeadData
scrollingMode	none
tooltip	<>
visible	visible
width	100%
Layout[MatrixLayout]	
stretchedHorizontally	true
stretchedVertically	true
LayoutData[MatrixHeadData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	5
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 60 - Group2 Properties

Property	Value
Elementproperties of Caption	
enabled	true
id	Group2_Header
imageAlt	
imageFirst	true
imageSource	<>
text	Please Select File Types to Scan
textDirection	inherit
tooltip	<>
visible	visible

Figure 61 - Group2 Header

Property	Value
Elementproperties of CheckBox	
checked	Update.TextPlain
enabled	Update.InputEnabled
id	TextPlain
layoutdata	MatrixData
readOnly	false
state	normal
text	
textDirection	inherit
tooltip	<>
visible	visible
Event	
onToggle	
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 62 - TextPlain CheckBox Properties

For the remaining CheckBox, the properties should be set similar to the TextPlain example above. Notice that for all the input elements throughout the UI, the enabled property is bound to Update.InputEnabled. This allows for the input elements to be disabled until the user selects a beginning search node from the navigation tree.

Property	Value
Elementproperties of Group	
accessibilityDescription	
defaultButtonId	
design	sapcolor
enabled	true
hasContentPadding	true
height	
id	Group1
layout	MatrixLayout
layoutdata	MatrixHeadData
scrollingMode	none
tooltip	<>
visible	visible
width	100%
Layout[MatrixLayout]	
stretchedHorizontally	true
stretchedVertically	true
LayoutData[MatrixHeadData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	5
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 63 - Group1 Properties

Property	Value
Elementproperties of Caption	
enabled	true
id	Group1_Header
imageAlt	
imageFirst	true
imageSource	<>
text	Find and Replace
textDirection	inherit
tooltip	<>
visible	visible

Figure 64 - Group2 Header

Property	Value
Elementproperties of InputField	
alignment	auto
enabled	Update.InputEnabled
id	Find
layoutdata	MatrixData
length	<>
passwordField	false
readOnly	false
size	standard
state	required
textDirection	inherit
tooltip	<>
value	Update.Find
visible	visible
width	
Event	
onEnter	
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 65 - Find InputField Properties

Property	Value
Elementproperties of InputField	
alignment	auto
enabled	Update.InputEnabled
id	Replace
layoutdata	MatrixData
length	<>
passwordField	false
readOnly	false
size	standard
state	normal
textDirection	inherit
tooltip	<>
value	Update.Replace
visible	visible
width	
Event	
onEnter	
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 66 - Replace InputField Properties

Property	Value
Elementproperties of CheckBox	
checked	Update.CaseSensitive
enabled	Update.InputEnabled
id	chkCase
layoutdata	MatrixData
readOnly	false
state	normal
text	
textDirection	inherit
tooltip	<>
visible	visible
Event	
onToggle	
LayoutData[MatrixData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 67 - CaseSensitive CheckBox Properties

Property	Value
Elementproperties of UIElement	
enabled	true
id	ButtonRow1
layoutdata	MatrixHeadData
tooltip	<>
visible	visible
LayoutData[MatrixHeadData]	
cellBackgroundDesign	transparent
cellDesign	rPad
colSpan	1
hAlign	beginOfLine
height	
vAlign	baseline
vGutter	none
width	

Figure 68 - ButtonRow Properties

Property	Value
Elementproperties of Button	
design	standard
enabled	Update.InputEnabled
id	btnRebuild
imageAlt	
imageFirst	true
imageSource	<>
size	standard
text	<>
textDirection	inherit
tooltip	<>
visible	visible
width	
Event	
onAction	RebuildUpdateList

Figure 69 - btnRebuild Button Properties

Property	Value
Elementproperties of Button	
design	standard
enabled	Update.InputEnabled
id	btnUpdate
imageAlt	
imageFirst	true
imageSource	<>
size	standard
text	<>
textDirection	inherit
tooltip	<>
visible	visible
width	
Event	
onAction	Update

Figure 70 - btnUpdate Button Properties

Property	Value
Elementproperties of Button	
design	standard
enabled	Update.InputEnabled
id	btnFindOnly
imageAlt	
imageFirst	true
imageSource	<>
size	standard
text	Find Only
textDirection	inherit
tooltip	<>
visible	visible
width	
Event	
onAction	findOnly

Figure 71 - btnFindOnly Button Properties

View Implementation

The View Implementation should already contain quite a bit of generated coding. In addition to the standard Web Dynpro View methods like wdDoInit, you should see one method for each action that we created earlier. Later we will also create 12 additional private methods which will contain the majority of the KM API coding. Once complete, your code outline should look like the following:

Figure 72 - View Implementation Class Outline

Imports

As you create the following methods, you can always use CTRL+SHIFT+O to organize your imports. However the following is also a complete list of imports needed to complete the project.

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import com.sap.tut.km.search_replace.wdp.IMessageKMSearchAndReplace;
import com.sap.tut.km.search_replace.wdp.IPrivateKMSearchAndReplaceView;
import
com.sap.tut.km.search_replace.wdp.IPrivateKMSearchAndReplaceView.IFolderContentE
lement;
import
com.sap.tut.km.search_replace.wdp.IPrivateKMSearchAndReplaceView.IFolderContentN
ode;
import
com.sap.tut.km.search_replace.wdp.IPrivateKMSearchAndReplaceView.IUpdateListElem
ent;
import com.sap.tc.Web Dynpro.clientserver.uielib.standard.api.IWDTreeNodeType;
import com.sap.tc.Web Dynpro.progmodel.api.IWDMessageManager;
```

```
import com.sap.tc.Web Dynpro.progmodel.api.WDVisibility;
import com.sap.tc.Web Dynpro.progmodel.repository.IWDControllerInfo;
import com.sap.tc.Web Dynpro.services.sal.um.api.IWDClientUser;
import com.sap.tc.Web Dynpro.services.sal.um.api.WDClientUser;
import com.sap.tc.Web Dynpro.services.session.api.IWDConfirmationDialog;
import com.sapportals.portal.security.usermanagement.IUser;
import com.sapportals.wcm.repository.Content;
import com.sapportals.wcm.repository.ICollection;
import com.sapportals.wcm.repository.ILockInfo;
import com.sapportals.wcm.repository.ILockProperties;
import com.sapportals.wcm.repository.IPropertyName;
import com.sapportals.wcm.repository.IResource;
import com.sapportals.wcm.repository.IResourceContext;
import com.sapportals.wcm.repository.IResourceFactory;
import com.sapportals.wcm.repository.IResourceList;
import com.sapportals.wcm.repository.IResourceListIterator;
import com.sapportals.wcm.repository.LockProperties;
import com.sapportals.wcm.repository.PropertyName;
import com.sapportals.wcm.repository.ResourceContext;
import com.sapportals.wcm.repository.ResourceFactory;
import com.sapportals.wcm.repository.ResourcePropertyComparator;
import com.sapportals.wcm.repository.enum.LinkType;
import com.sapportals.wcm.repository.enum.LockDepth;
import com.sapportals.wcm.repository.enum.LockScope;
import com.sapportals.wcm.repository.enum.LockType;
import com.sapportals.wcm.repository.enum.SupportedOption;
import com.sapportals.wcm.service.IServiceTypesConst;
import com.sapportals.wcm.service.urlgenerator.IURLGeneratorService;
import com.sapportals.wcm.service.urlgenerator.PathKey;
import com.sapportals.wcm.util.content.IContent;
import com.sapportals.wcm.util.uri.IUriReference;
import com.sapportals.wcm.util.uri.RID;
import com.sapportals.wcm.util.usermanagement.WPUMFactory;
```

wdDoInit

In our View initialization we will want to place the call to two private methods. These methods, which we will detail later, will initialize our Context data for the navigation tree and the input fields.

```
//@@begin wdDoInit()
      initializeRepositoryTree();
      initializeContext();
//@@end
```

wdDoModifyView

On the first pass through the Modify View method, we need to make a special “hook” into our two actions that have parameters. The code here will allow us to map the path to the generating element into our actions.

```
//@@begin wdDoModifyView
      if (firstTime) {
            IWDTreeNodeType treeNode =
                  (IWDTreeNodeType) view.getElement("treeNodeKMNav");
      /* parameter mapping from parameter "path" to
       * parameter "selectedElement"
       */
```

```
            treeNode.mappingOfOnAction().addSourceMapping(
                  "path",
                  "elementSelection");
      /* parameter mapping from parameter path to
       * parameter "element"
       */
            treeNode.mappingOfOnLoadChildren().addSourceMapping(
                  "path",
                  "element");
      }
//@@end
```

onActionLoadChildren

This action is fired whenever a tree expansion is triggered in the UI, but there are no children nodes currently populated at that level.

```
//@@begin onActionLoadChildren(ServerEvent)
      addChildren(element);
//@@end
```

onActionRebuildUpdateList

There is a UI Button Element that the user can select to rebuild the Update Listing (the update listing is visualized as a Table). If selected, this action will clear the updateList Context Node and rebuild it given the current folder Path and other selection criteria.

```
//@@begin onActionRebuildUpdateList(ServerEvent)
      wdContext.nodeUpdateList().invalidate();
      addEntriesToUpdateList(
            wdContext.currentUpdateElement().getFolderPath());
//@@end
```

onActionUpdate

This action is triggered by the UI element - button. The user has triggered this action because they are ready to perform their mass update. However before we begin we actually update anything, we will first use this action to trigger a dialog confirmation so that the user has a chance to cancel their action before the mass update.

```
//@@begin onActionUpdate(ServerEvent)
      //Check for Manditory Fields
      checkManditory();
      wdComponentAPI.getMessageManager().raisePendingException();
      //Prepare the Confirmation Dialog
      IWDControllerInfo controllerInfo =
            wdControllerAPI.getViewInfo().getViewController();
      String dialogText = "Please Confirm To Perform the Update";
      IWDConfirmationDialog dialog =
            wdComponentAPI.getWindowManager().createConfirmationWindow(
                  dialogText,
                  controllerInfo.findInEventHandlers("onActionupdateOK"),
                  "Update");
      dialog.addChoice(
            controllerInfo.findInEventHandlers("onActionNull"),
                  "Cancel");
      dialog.setTitle("Confirm Update?");
```

```
      dialog.setIcon("~sapicons/s_n_ques.gif");
      dialog.show();
//@@end
```

onActionNodeSelection

This action is fired when the user selects a node in the navigation tree. This node will then be set as the current update path and we will clear and invalidate the updateList Context Node.

```
//@@begin onActionNodeSelection(ServerEvent)
      //A node (Repository) was choosen,
      //Open Input fields and set the Path as the current working one
      //Also clear the Node for the Output Table
      wdContext.currentUpdateElement().setInputEnabled(true);
      wdContext.currentUpdateElement().setFolderPath(
            elementSelection.getPath());
      wdContext.nodeUpdateList().invalidate();
      wdContext.nodeUpdateList().clearSelection();
//@@end
```

onActionupdateOK

The update dialog confirmation has been triggered by the onActionUpdate Method. If the user chooses OK from the dialog Confirmation, this action will be fired. Now we can actually call the update method (updateList).

We call updateList with the findOnly Parameter marked false. This way the updates are actually performed as opposed to the find only action and the way that it calls the same update method.

```
//@@begin onActionupdateOK(ServerEvent)
//The user choose OK from the Confirmation dialog - go ahead with the update
      updateList(false);
      wdComponentAPI.getMessageManager().reportSuccess("Updates Complete");
//@@end
```

onActionNull

This is just a dummy action that does nothing. It is used if the user selects Cancel from the Update Dialog Confirmation.

```
//@@begin onActionNull(ServerEvent)
      //Null Action for Cancel on the Confirmation dialog
//@@end
```

onActionfindOnly

This action is fired by the Find Only UI Button Element. It will trigger the updateList method, but with a flag to only execute the Find portion of the logic.

```
//@@begin onActionfindOnly(ServerEvent)
      //For Find Only (no Replace) we only need to confirm
      //that the user has supplied a value in the FIND input field
      IWDMessageManager msgMgr = wdComponentAPI.getMessageManager();
      if (wdContext.currentUpdateElement().getFind().toString().length()
                  == 0) {
            msgMgr.reportContextAttributeMessage(
```

```
            wdContext.currentUpdateElement(),
            wdContext.nodeUpdate().getNodeInfo().getAttribute(
                  wdContext.currentUpdateElement().FIND),
            IMessageKMSearchAndReplace.REQUIRED_FIELD,
                  null, true);
      }
      wdComponentAPI.getMessageManager().raisePendingException();
            updateList(true);
      wdComponentAPI.getMessageManager().reportSuccess("Search Complete");
//@@end
```

onActionshowHideTree

The Navigation tree is a useful UI element, but once the user has selected the Repository they want to search within; it doesn't serve much purpose. Therefore in order to reclain screen space and increase usibility, the Navigation tree can be hidden or displayed. There is a UI element, a button, which triggers this action.

```
//@@begin onActionshowHideTree(ServerEvent)
      //If the Tree is Visible, Hide it and change the Icon
      if (wdContext.currentUpdateElement().getTreeShown()
            == WDVisibility.VISIBLE) {
            wdContext.currentUpdateElement().setCollapseIcon(
                  "~sapicons/s_b_pagr.gif");
            wdContext.currentUpdateElement().setTreeShown(WDVisibility.NONE);
      //If the Tree is Hidden, make it visible and change the Icon
      } else {
            wdContext.currentUpdateElement().setCollapseIcon(
                  "~sapicons/s_b_pagl.gif");
            wdContext.currentUpdateElement().setTreeShown(WDVisibility.VISIBLE);
      }
//@@end
```

The remainder of the methods are all private methods that will have to be added manually to the implementation. They should all be placed within the following block at the end of the implementation section:

```
  //@@begin others
  //@@end
```

checkManditory

This method does the conditional checks for Required UI elements.

```
private void checkManditory() {
      //Check for a value in the Required Field REPLACE
      IWDMessageManager msgMgr = wdComponentAPI.getMessageManager();
      if (wdContext.currentUpdateElement().getReplace().toString().length()
            == 0) {
            msgMgr.reportContextAttributeMessage(
                  wdContext.currentUpdateElement(),
                  wdContext.nodeUpdate().getNodeInfo().getAttribute(
                        wdContext.currentUpdateElement().REPLACE),
                  IMessageKMSearchAndReplace.REQUIRED_FIELD,
                  null, true);
      }
      //Check for a value in the Required Field FIND
```

```
      if (wdContext.currentUpdateElement().getFind().toString().length()
            == 0) {
            msgMgr.reportContextAttributeMessage(
                  wdContext.currentUpdateElement(),
                  wdContext.nodeUpdate().getNodeInfo().getAttribute(
                        wdContext.currentUpdateElement().FIND),
                  IMessageKMSearchAndReplace.REQUIRED_FIELD,
                  null, true);
      }
}
```

addChildren

This method contains the process to add children to Repository Hierarchy Tree. It is called by the LoadChildren Action. So when the user expands a node of the tree that hasn't already been populated with children, this method is called to read those children and add them to the Context.
@param parent Reference to the Context Element that is the Parent in the Folder/SubItem Recursive Relationship

```
private void addChildren(IFolderContentElement parent) {
      IFolderContentNode folderContentNode = parent.nodeChildNode();
      IFolderContentElement folderContentElement;
      boolean hasChildren = false;
      try {
            //Get a Resouce Context from the Local Method
            IResourceContext resourceContext = buildResourceContext();
            //get a resource factory
            IResourceFactory resourceFactory = ResourceFactory.getInstance();
            //Get a RID from the current path to display the according content
            RID pathRID = RID.getRID(parent.getPath());
            //Get a Iresource object to work on
            IResource resource =
                  resourceFactory.getResource(pathRID, resourceContext);
            //cast the object to a Collection
            ICollection collection = (ICollection) resource;
            //get the Collection's Children
            IResourceList resourceList = collection.getChildren();
            //Sort Resource List by Name (Local Method)
            sortResouceListByName(resourceList);
            //Finally get an iterator to walk through the set of Children
            IResourceListIterator resourceListIterator =
                  resourceList.listIterator();
            //Now read all elements
            while (resourceListIterator.hasNext()) {
                  IResource tempResource = resourceListIterator.next();
                  if (tempResource.isCollection()) {
                        //create a new Context element for each of them
                        folderContentElement =
                              folderContentNode.createFolderContentElement();
                        //Local Method
                        addFolderToContextNode(
                              folderContentElement, tempResource, pathRID);
                        folderContentNode.addElement(folderContentElement);
                        parent.setIsExpanded(true);
                        hasChildren = true;
                  }
            }
            if (!hasChildren) {
                  parent.setHasChildren(false);
```

```
            }
      } catch (Exception e) {
            e.printStackTrace();
      }
}
```

addFolderToContextNode

This method contains the logic to add a Folder Item to the Navigation Context Node.
@param element The current Context Element
@param tempResource the current Resource (Folder in this case)
@param pathRID The Resource ID for the Parent (the current path)
@throws Exception

```
private void addFolderToContextNode(
      IFolderContentElement element, IResource tempResource,
      RID pathRID) throws Exception {
      //Entry is a Folder
      element.setHasChildren(true);
      element.setIconSource("~sapicons/s_clofol.gif");
      //Use the Name if the DisplayName is Blank
      if (tempResource.getDisplayName() == "") {
            element.setText(tempResource.getName());
      } else {
            element.setText(tempResource.getDisplayName());
      }
      //Create another RID for the directory that shall be opened
      RID directoryToOpenRID;
      //If the current directory is the root..
      if (pathRID.isRoot()) {
            //...attach the selected directory's name without a leading slash /
            directoryToOpenRID =
                  RID.getRID(pathRID.toString() + tempResource.getName());
      } else {
            //...attach the selected directory's name with a leading slash /
            directoryToOpenRID =
                  RID.getRID(pathRID.toString() + "/" + tempResource.getName());
      }
      //set the selected directory as the new path
      element.setPath(directoryToOpenRID.toString());
      element.setIsExpanded(false);

}
```

buildResourceContext

This method has the logic to take the current Web Dynpro User and cross-reference it to the EP5 User (because certain KM APIs still require the use of the EP5 User Object even in 04S). The EP5 user is used to create a resource Context. The Resource Context will be used when accessing all resources from their Repository Managers.
@return A valid Resource Context
@throws Exception

```
private IResourceContext buildResourceContext() throws Exception {
      //          create an user object from the current user
      IWDClientUser wdClientUser = WDClientUser.getCurrentUser();
      com.sap.security.api.IUser sapUser = wdClientUser.getSAPUser();
      //Create a EP user from the retrieved user
      IUser epUser = WPUMFactory.getUserFactory().getEP5User(sapUser);
      //Establish a Resource Context
```

```
      IResourceContext resourceContext = new ResourceContext(epUser);
      return resourceContext;

}
```

generateUrlForResource

This utility method will generate a URL for any valid Resource.
@param resource The Resource that we want to generate a URL for
@return The URL in the form of a simple string
@throws Exception

```
private String generateUrlForResource(IResource resource) throws Exception {
      //Generate URL for a Resource
      IURLGeneratorService ug =
            (IURLGeneratorService) ResourceFactory
                        .getInstance()
                        .getServiceFactory()
                        .getService(IServiceTypesConst.URLGENERATOR_SERVICE);
      IUriReference uriRef2;
      uriRef2 =
            ug.getRelativeUri(PathKey.CONTENT_ACCESS_PATH).appendPath(
                  resource.getRID().toExternalForm());
      return uriRef2.toExternalForm();
}
```

addEntriesToUpdateList

This method is called to process through the current list of resources. For each resource, if it matches the selection criteria, it will then be added to the updateList (visualized on the UI by a Table UI element).
@param path The Resource Path for the Current Resource

```
private void addEntriesToUpdateList(String path) {
      IUpdateListElement updateListElement;
      try {
      //Get a Resouce Context from the Local Method
      IResourceContext resourceContext = buildResourceContext();
      //get a resource factory
      IResourceFactory resourceFactory = ResourceFactory.getInstance();
      //Get a RID from the current path to display the according content
      RID pathRID = RID.getRID(path);
      //Get a Iresource object to work on
      IResource resource =
            resourceFactory.getResource(pathRID, resourceContext);
      //cast the object to a Collection
      ICollection collection = (ICollection) resource;
      //get the Collection's Children
      IResourceList resourceList = collection.getChildren();
      //Sort Resource List by Name (Local Method)
      sortResouceListByName(resourceList);
      //Finally get an iterator to walk through the set of Children
      IResourceListIterator resourceListIterator =
            resourceList.listIterator();
      //Now read through all elements
      while (resourceListIterator.hasNext()) {
            IResource tempResource = resourceListIterator.next();
            if (!tempResource.isCollection()) {
                  //Items Only
```

```
                updateListElement =
                      wdContext.nodeUpdateList().createUpdateListElement();
                updateListElement.setPath(path);
                updateListElement.setUpdateStatus("~sapicons/s_s_tl_y.gif");
                if (tempResource.getDisplayName() == "") {
                      updateListElement.setFileName(tempResource.getName());
                } else {
                      updateListElement.setFileName(
                            tempResource.getDisplayName());
                }
                //Generate URL for a Resource (Local Method)
                updateListElement.setContentLink(
                      generateUrlForResource(tempResource));
                if (LinkType.INTERNAL.equals(tempResource.getLinkType())) {
                      if (wdContext.currentUpdateElement().getFollowLinks(
                            IResource target =
                      tempResource.getTargetResource();
                            addItemsToContextNode(
                                  updateListElement, target.getContent(),
                                  target);
                      }
                      } else if (
                      LinkType.EXTERNAL.equals(tempResource.getLinkType())) {
                      //External Link - we do nothing - we can't update it.
                      } else {
                            addItemsToContextNode(
                                  updateListElement,
                                  tempResource.getContent(),
                                  tempResource);
                      }
                }
          }
          if (wdContext.currentUpdateElement().getRecursiveSearch()) {
                //Now read all elements - Read them Again for the Folders -
                //If the user selected Recursive Search
                IResourceListIterator resourceListFolderIterator =
                      resourceList.listIterator();
                while (resourceListFolderIterator.hasNext()) {
                //create a new context element for each of them
                      IResource tempResource =
                            resourceListFolderIterator.next();
                      if (tempResource.isCollection()) {
                      //Folders Only
                      //Create another RID for the directory that shall be
opened
                      RID directoryToOpenRID;
                      //If the current directory is the root..
                      if (pathRID.isRoot()) {
                      //...attach the selected directory's name without a
leading slash /
                            directoryToOpenRID =
                                  RID.getRID(pathRID.toString()
                                        + tempResource.getName());
                      } else {
                      //...attach the selected directory's name with a leading
slash /
                            directoryToOpenRID =
                                  RID.getRID(pathRID.toString() + "/"
                                        + tempResource.getName());
```

```
                        }
        addEntriesToUpdateList(directoryToOpenRID.toString());
                    }
                }
        }
        } catch (Exception e) {
                e.printStackTrace();
        }
}
```

AddItemsToContextNode

This method takes a resource and its content and then decides if it meets the criteria of being added to the updateList Node. Basically we see if the content type matches the content types that user has selected for their search.
@param updateListElement Current updateList Context Element
@param content The Resource Content Object
@param resource The currently processed Resource
@throws Exception

```
private void addItemsToContextNode(
        IUpdateListElement updateListElement, IContent content,
                IResource resource) throws Exception {
        updateListElement.setContentLength(content.getContentLength());
        updateListElement.setContentType(content.getContentType());
        updateListElement.setContentPath(resource.getRID().getPath());

        if (content.getContentType().equalsIgnoreCase("text/html")
                & wdContext.currentUpdateElement().getTextHTML()) {
                        wdContext.nodeUpdateList().addElement(updateListElement);
        }
        if (content.getContentType().equalsIgnoreCase("text/xml")
                & wdContext.currentUpdateElement().getTextXML()) {
                        wdContext.nodeUpdateList().addElement(updateListElement);
        }
        if (content.getContentType().equalsIgnoreCase("text/css")
                & wdContext.currentUpdateElement().getTextCSS()) {
                        wdContext.nodeUpdateList().addElement(updateListElement);
        }
        if (content.getContentType().equalsIgnoreCase("text/plain")
                & wdContext.currentUpdateElement().getTextPlain()) {
                        wdContext.nodeUpdateList().addElement(updateListElement);
        }
}
```

sortResouceListByName

This utility method is used to sort any Resource List by its Display Name Property
@param resourceList The resource list that we want to sort
@throws Exception

```
private void sortResouceListByName(IResourceList resourceList)
                throws Exception {
        //              create property names to index properties
        IPropertyName iPropSort =
                new PropertyName("http://sapportals.com/xmlns/cm", "displayname");
        //create a comparator to order the resource list
        ResourcePropertyComparator rRPC =
                new ResourcePropertyComparator(iPropSort, true);
```

```
      //order the resource list
      resourceList.sort(rRPC);
}
```

updateList

This method contains the logic to search through the current resources in the update list. It will perform the search using the regular expression that can be built from the FIND and REPLACE input elements.

This method is used for both Find and Find and Replace operations. The input parameter findOnly controls the flow of logic.

In the Find and Replace situation, the resources will be locked (if their repository managers support locking). Then their content will be updated and then unlocked (once again - if supported).

We set the status of either the find or the find and replace using the tradition stop lights icons.

@param findOnly A flag that specifies if we want to do a Find or a Find and Replace

```
private void updateList(boolean findOnly) {
      //Loop through our List of Items that can be updated
      for (int i = 0; i < wdContext.nodeUpdateList().size(); ++i) {
            IUpdateListElement updateListElement =
            (IUpdateListElement) wdContext.nodeUpdateList().getElementAt(i);
      try {
            //Get a Resouce Context from the Local Method
            IResourceContext resourceContext = buildResourceContext();
            //get a resource factory
            IResourceFactory resourceFactory =
                  ResourceFactory.getInstance();
            //Get a RID from the current path to display the according content
            RID pathRID = RID.getRID(updateListElement.getContentPath());
            //Create Lock Properties
            ILockProperties lockProperties =
                  new LockProperties(LockType.WRITE, LockScope.EXCLUSIVE,
                  LockDepth.SHALLOW,(60 * 1000));
            //Get a Iresource object to work on
            IResource resource =
                  resourceFactory.getResource(pathRID, resourceContext);
            //Get Content
            IContent content = resource.getContent();
            //Read Content
            InputStream IStream = content.getInputStream();
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            byte[] buffer = new byte[4096];
            int bytesread = 0;
            while ((bytesread = IStream.read(buffer)) != -1) {
                  out.write(buffer, 0, bytesread);
            }
            //Check for Resource Contraints
            if (!findOnly) {
                  if (resource.isLocked()) {
                        updateListElement.setUpdateStatus(
                              "~sapicons/s_s_tl_r.gif");
                              updateListElement.setUpdateResults(
                                    "Resource is currently Locked");
                        continue;
                  }
                  if (resource.isReadOnly()) {
```

```
                        updateListElement.setUpdateStatus(
                              "~sapicons/s_s_tl_r.gif");
                        updateListElement.setUpdateResults(
                              "Resource is marked Read-Only!");
                        continue;
                  }
            }

            //Build the Regex Pattern (Local Method)
            Pattern p = buildRegexPattern();
            String tempString;
            Matcher matcher = p.matcher(out.toString());
            //Perform the Find and Replace
            tempString = matcher.replaceAll(
                  wdContext.currentUpdateElement().getReplace());
            //If we do a find and there are no results - then there are no
matches to update
            if (!matcher.find()) {
                  updateListElement.setUpdateResults(
                        "No Matches Found to Update!");
                  if (findOnly) {
                        updateListElement.setUpdateStatus(
                              "~sapicons/s_s_tl_r.gif");
                  } else {
                        updateListElement.setUpdateStatus(
                              "~sapicons/s_s_tl_y.gif");
                  }
            } else if (findOnly) {
                  updateListElement.setUpdateResults(
                        "Match Found for FIND Criteria!");
                  updateListElement.setUpdateStatus("~sapicons/s_s_tl_g.gif");
            } else {
                  //We have a match to update - get the content and turn it into
a string
                  ByteArrayInputStream dataStream =
                        new ByteArrayInputStream(tempString.getBytes());
                  IContent newContent =
                        new Content(dataStream,
                              updateListElement.getContentType(),
                              dataStream.available());
                  //If our Repository Manager supports Locking - then we want to
lock, update, unlock
                  if (resource.getRepositoryManager()
                        .getSupportedOptions(resource)
                        .isSupported(SupportedOption.LOCKING)) {
                        ILockInfo lockInfo = resource.lock(lockProperties);
                        resource.updateContent(newContent);
                        resource.unlock(lockInfo);
                        //If our Repository Manager doesn't support Locking -
                        //then we just update the content
                  } else {
                        resource.updateContent(newContent);
                  }
                  updateListElement.setUpdateResults("Successfully Updated!");
                  updateListElement.setUpdateStatus("~sapicons/s_s_tl_g.gif");
            }
      } catch (Exception e) {
            updateListElement.setUpdateResults(
            e.getClass().getName() + " " + e.getMessage());
```

```
            updateListElement.setUpdateStatus("~sapicons/s_s_tl_r.gif");
      }
   }
}
```

buildRegexPattern

This method builds a Regex Pattern from the FIND UI element and the CaseSensitive Checkbox
@return Returns a Regex Find Pattern

```
private Pattern buildRegexPattern() {
      Pattern p;
      if (wdContext.currentUpdateElement().getCaseSensitive()) {
            p = Pattern.compile(wdContext.currentUpdateElement().getFind());
      } else {
            p = Pattern.compile(
                  wdContext.currentUpdateElement().getFind(),
                  Pattern.CASE_INSENSITIVE);
      }
      return p;
}
```

initializeRepositoryTree

This method is called from the wdDoInit - it is used for the first initialization of the Navigation Repository Tree It creates the Root Node ("/") and triggers the addChildren method for populating the first level of the hierarchy

```
private void initializeRepositoryTree() {
      //    ===Step 2: Create node elements of the type IFolderContentNode ==
      IFolderContentNode rootFolderContentNode =
            wdContext.nodeFolderContent();
      IFolderContentElement rootFolderContentElement;
      //create the new context node element of type 'IFolderConetntElement'
      rootFolderContentElement =
            rootFolderContentNode.createFolderContentElement();
      rootFolderContentElement.setPath("/");
      rootFolderContentElement.setIsExpanded(true);
      rootFolderContentElement.setText("/");
      rootFolderContentElement.setHasChildren(true);
      rootFolderContentElement.setIconSource("~sapicons/s_clofol.gif");
      //add root folder element to root folder node
      rootFolderContentNode.addElement(rootFolderContentElement);
      addChildren(rootFolderContentElement);
}
```

initializeContext

This method is called from the wdDoInit in order to do the one-off initialization of the entire Context

```
private void initializeContext() {
      wdContext.nodeUpdate().createElement();
      wdContext.currentUpdateElement().setInputEnabled(false);
      wdContext.currentUpdateElement().setTextCSS(true);
      wdContext.currentUpdateElement().setTextHTML(true);
      wdContext.currentUpdateElement().setTextPlain(true);
      wdContext.currentUpdateElement().setTextXML(true);
      wdContext.currentUpdateElement().setReplace("");
      wdContext.currentUpdateElement().setFind("");
```

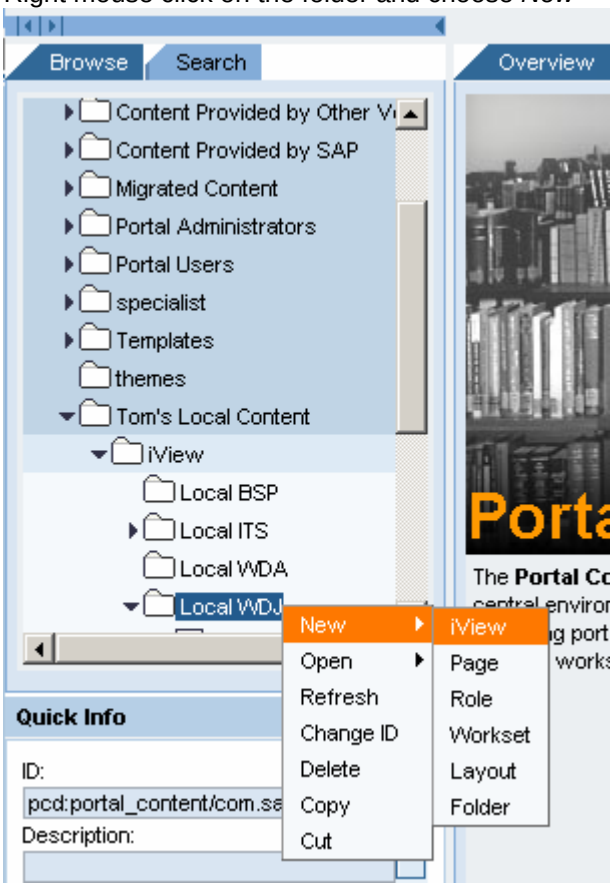
```
        wdContext.currentUpdateElement().setCaseSensitive(true);
        wdContext.currentUpdateElement().setCollapseIcon(
              "~sapicons/s_b_pagl.gif");

        wdContext.currentUpdateElement().setTreeShown(WDVisibility.VISIBLE);
}
```

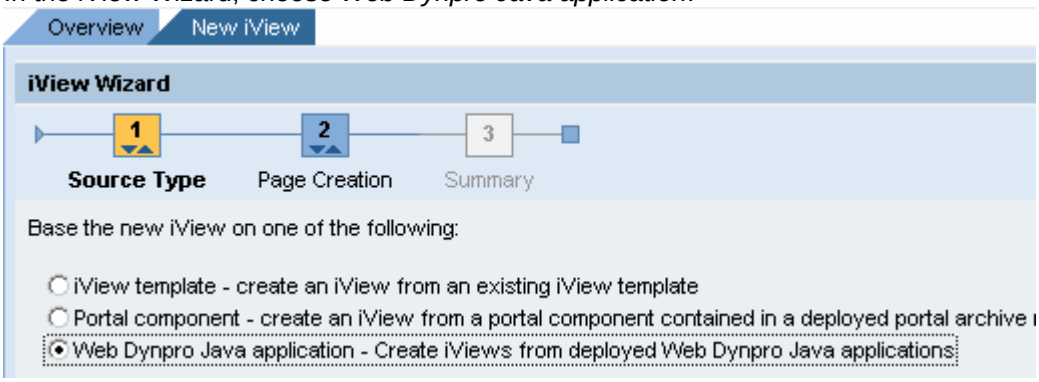
Portal Installation

After building and deploying your application we are ready to setup the Portal iView for it.

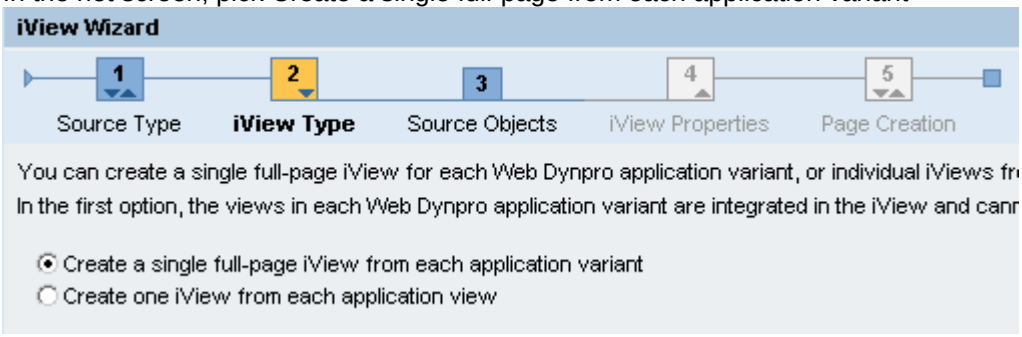
- Start by going to the Content Administration screen in your portal.
- Choose the Folder that you want to create your portal content within.
- Right mouse click on the folder and choose *New -> iView*



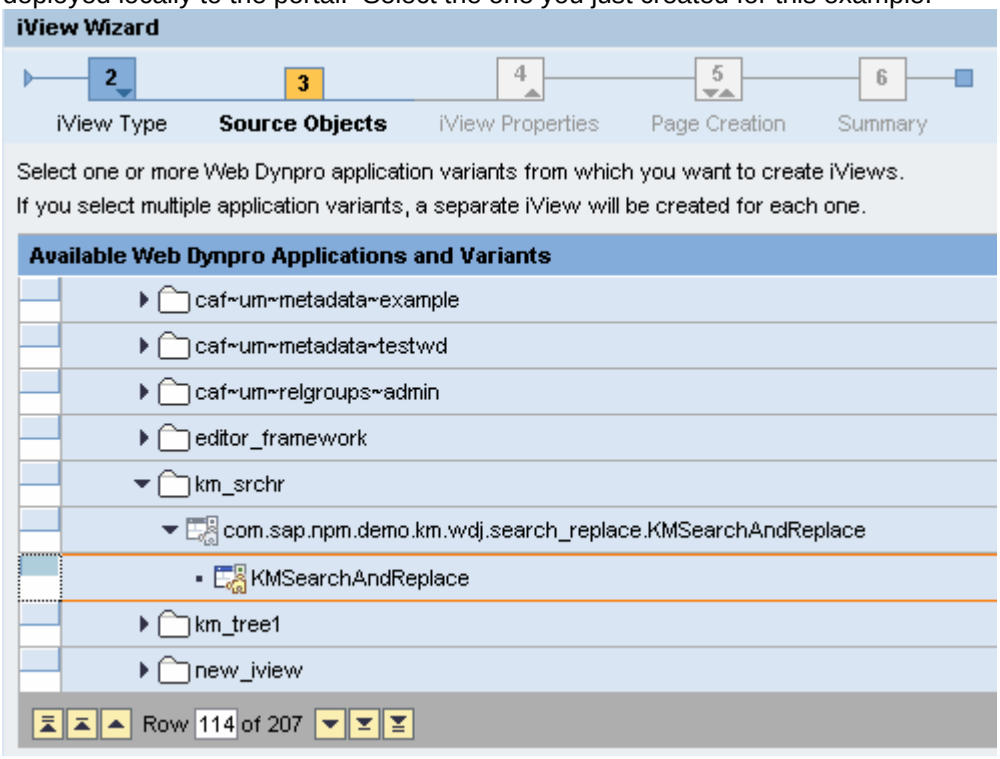
- In the iView Wizard, choose *Web Dynpro Java application*.



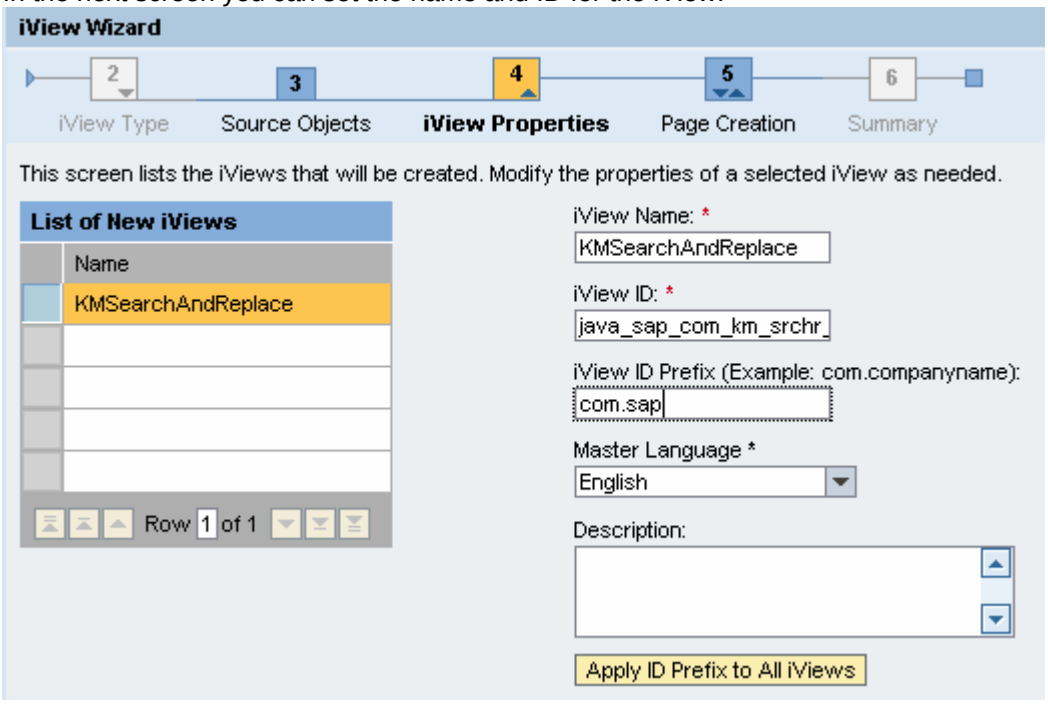
- In the net screen, pick Create a single full-page from each application variant



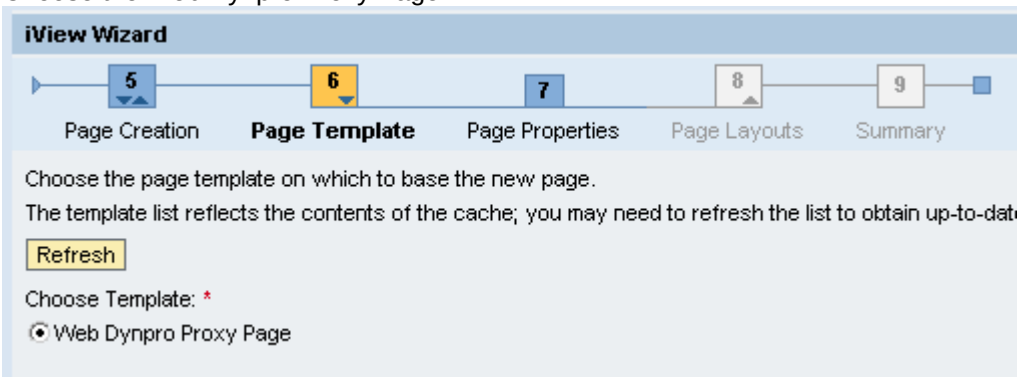
- You now come to a screen that shows you all the Web Dynpro Java applications that have been deployed locally to the portal. Select the one you just created for this example.



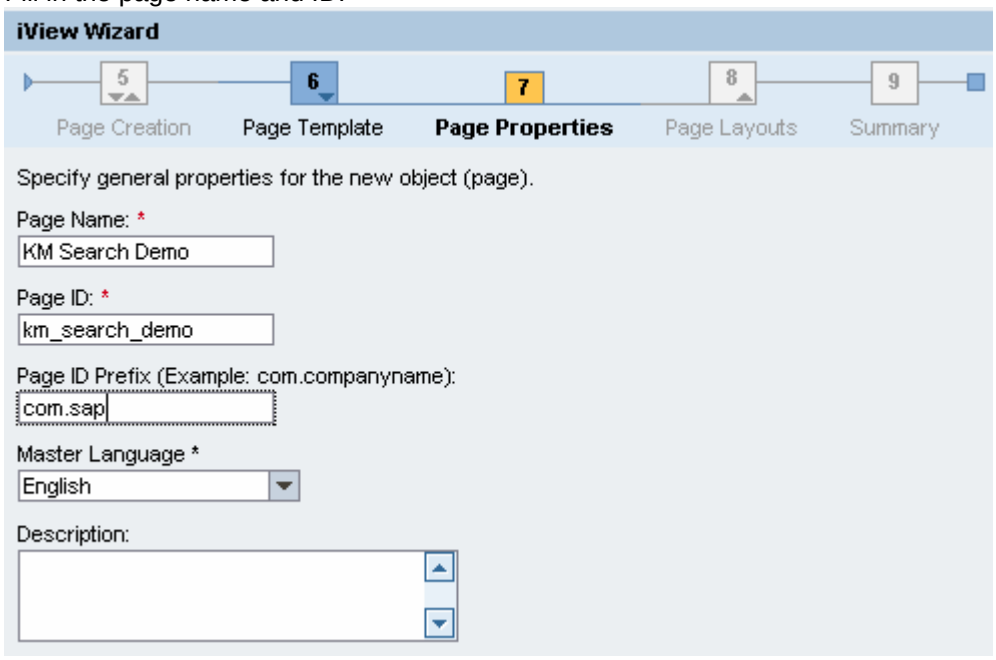
- In the next screen you can set the name and ID for the iView.



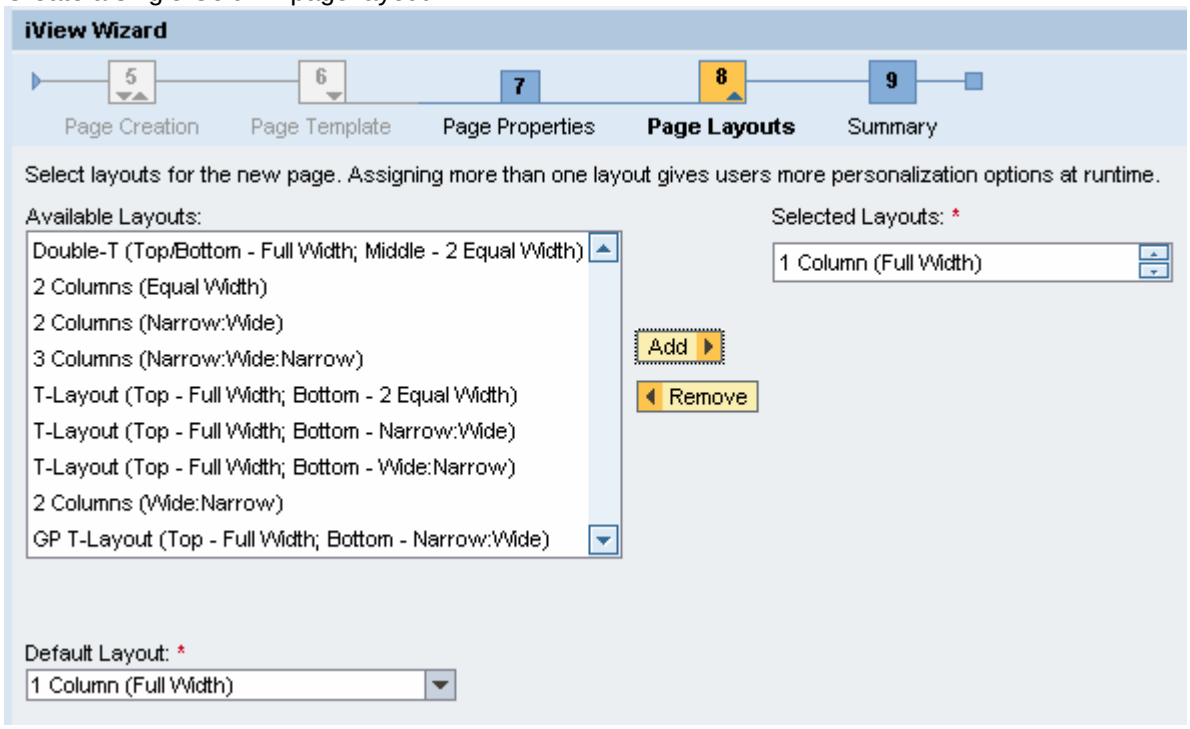
- Acknowledge the next screen and then the Page Creation wizard will be launched.
- Choose the Web Dynpro Proxy Page



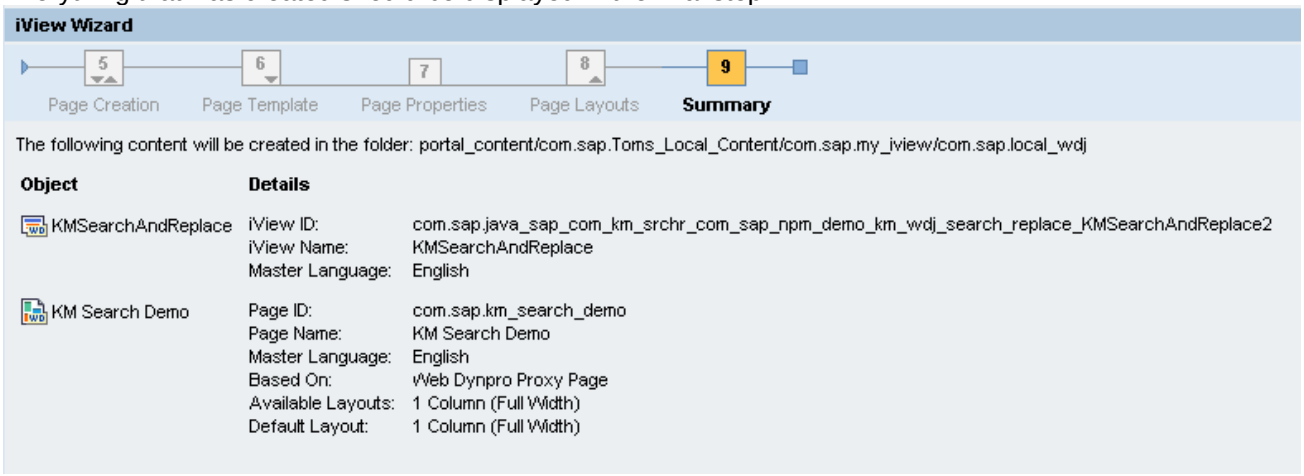
- Fill in the page name and ID.



- Create a single Column page layout.



Everything that was created should be displayed in the final step.



Copyright