

Windchill 11.0 - Client Customization



Copyright © 2014 PTC Inc. and/or Its Subsidiary Companies. All Rights Reserved.

User and training guides and related documentation from PTC Inc. and its subsidiary companies (collectively "PTC") are subject to the copyright laws of the United States and other countries and are provided under a license agreement that restricts copying, disclosure, and use of such documentation. PTC hereby grants to the licensed software user the right to make copies in printed form of this documentation if provided on software media, but only for internal/personal use and in accordance with the license agreement under which the applicable software is licensed. Any copy made shall include the PTC copyright notice and any other proprietary notice provided by PTC. Training materials may not be copied without the express written consent of PTC. This documentation may not be disclosed, transferred, modified, or reduced to any form, including electronic media, or transmitted or made publicly available by any means without the prior written consent of PTC and no authorization is granted to make copies for such purposes.

Information described herein is furnished for general information only, is subject to change without notice, and should not be construed as a warranty or commitment by PTC. PTC assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

The software described in this document is provided under written license agreement, contains valuable trade secrets and proprietary information, and is protected by the copyright laws of the United States and other countries. It may not be copied or distributed in any form or medium, disclosed to third parties, or used in any manner not provided for in the software licenses agreement except with written prior approval from PTC.

UNAUTHORIZED USE OF SOFTWARE OR ITS DOCUMENTATION CAN RESULT IN CIVIL DAMAGES AND CRIMINAL PROSECUTION. PTC regards software piracy as the crime it is, and we view offenders accordingly. We do not tolerate the piracy of PTC software products, and we pursue (both civilly and criminally) those who do so using all legal means available, including public and private surveillance resources. As part of these efforts, PTC uses data monitoring and scouring technologies to obtain and transmit data on users of illegal copies of our software. This data collection is not performed on users of legally licensed software from PTC and its authorized distributors. If you are using an illegal copy of our software and do not consent to the collection and transmission of such data (including to the United States), cease using the illegal version, and contact PTC to obtain a legally licensed copy.

Important Copyright, Trademark, Patent, and Licensing Information: See the About Box, or copyright notice, of your PTC software.

UNITED STATES GOVERNMENT RESTRICTED RIGHTS LEGEND

This document and the software described herein are Commercial Computer Documentation and Software, pursuant to FAR 12.212(a)-(b) (OCT'95) or DFARS 227.7202-1(a) and 227.7202-3(a) (JUN'95), and are provided to the US Government under a limited commercial license only. For procurements predating the above clauses, use, duplication, or disclosure by the Government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 (OCT'88) or Commercial Computer Software-Restricted Rights at FAR 52.227-19(c)(1)-(2) (JUN'87), as applicable. 01012014

PTC Inc., 140 Kendrick Street, Needham, MA 02494 USA

PRINTING HISTORY

Document No.

GS-01190_01

Date

June 15, 2010

Description

Initial Printing of:
Windchill 11.0 - Client Customization

Printed in the U.S.A

Table of Contents

Windchill 11.0 - Client Customization

Windchill 11.0 Overview	1-1
Navigation	1-2
Home Page	1-3
Search Page	1-4
Information Page	1-5
Attribute Layouts	1-6
Tables and Trees	1-7
Data Sources and Incomplete Data Sets	1-8
Data Sources	1-9
More Tables and Trees	1-10
And More	1-11
Common Client Architecture	1-12
Upgrades to JCA	1-13
GWT Architecture	1-14
Windchill 9.1 JCA Model	1-15
Windchill 10.2 JCA Model	1-16
JCA Component	1-17
Industry Standard Frameworks	1-18
JCA Evolution	1-19
Expanded Resources For Customizers	1-20
Navigation	2-1
Customization Tab	2-2
Windchill 11.0 Debugging	2-3
Finding the Code	2-4
Windchill 11.0 References	2-5
Resource Bundle Overview	2-6
Reference Resource Bundle	2-7
Action Models and Actions	2-8
Inserting a Tab	2-9
Inserting an Action	2-10
Referencing a Custom JSP	2-11
Menus for Objects	2-12
Exercise 1: Modify List of Actions	2-13
Exercise 2: Configure Eclipse for Windchill Development	2-15
Exercise 3: Add First- and Second-Level Navigation Tabs	2-28
Information Pages	3-1
Windchill 9.1 Information Page	3-2
Windchill 10.2 Information Page	3-3
Exercise 1: Modify the Customize Menu for Part	3-4
Carambola InfoPage Example Overview	3-7
Finding the Builder	3-8
Carambola Builder	3-9
Carambola Component Builder	3-10

Carambola Component Config Builder.....	3-11
MVC	4-1
Windchill Client Architecture.....	4-2
MVC	4-3
MVC Pattern	4-4
MVC — View.....	4-5
MVC — Controller	4-6
MVC — Model	4-7
Windchill Turns to Spring	4-8
Spring MVC Overview.....	4-9
Spring MVC DispatcherServlet	4-10
MVC Request Handling.....	4-11
Windchill MVC Components.....	4-12
Building Components in a Factory	4-13
Building JCA Components	4-14
Building Type-Based Components	4-15
Registering Builders.....	4-16
Supporting Legacy JCA	4-17
JCA Components	4-18
JCA Tables and Trees	4-19
Other Configs	4-20
Customization Tools.....	4-21
Builder Scan Report.....	4-22
Exercise 1: Add a Component in Navigation.....	4-23
Exercise 2: Create Custom-Type Information Page.....	4-27
Exercise 3: Create a Table Component	4-30
Exercise 4: Create a Page with Multiple MVC JCA Components	4-33
Exercise 5: Create a Tree Component.....	4-37
Client-Side Technologies.....	5-1
Windchill Client Framework	5-2
JavaScript Key Points	5-3
JavaScript with Windchill	5-4
Prototype Shortcuts	5-5
Prototype and Ajax Abstraction	5-6
Prototype and Ajax Update Operation	5-7
ExtJS Overview	5-8
ExtJS 3 and Windchill	5-9
Exercise 1: Create a Search Table	5-10

Course Description

Course Code GS-01190_01

Course Length

In this course, you will learn the preferred customization tools and methodology for customizing Windchill 11.0.

Course Objectives

- Identify, describe, and use Windchill client architectural components.
- Understand file structure and key files used in the Windchill client architecture.
- Configure and customize the Windchill client UI.

For PTC Internal Use Only

For PTC Internal Use Only

Module 1

Windchill 11.0 Overview

Module Overview

In this module, we focus on what has changed in the product since the previous version, and what was used to build the product. This includes UI changes and the architecture changes.

Objectives

After completing this module, you will be able to:

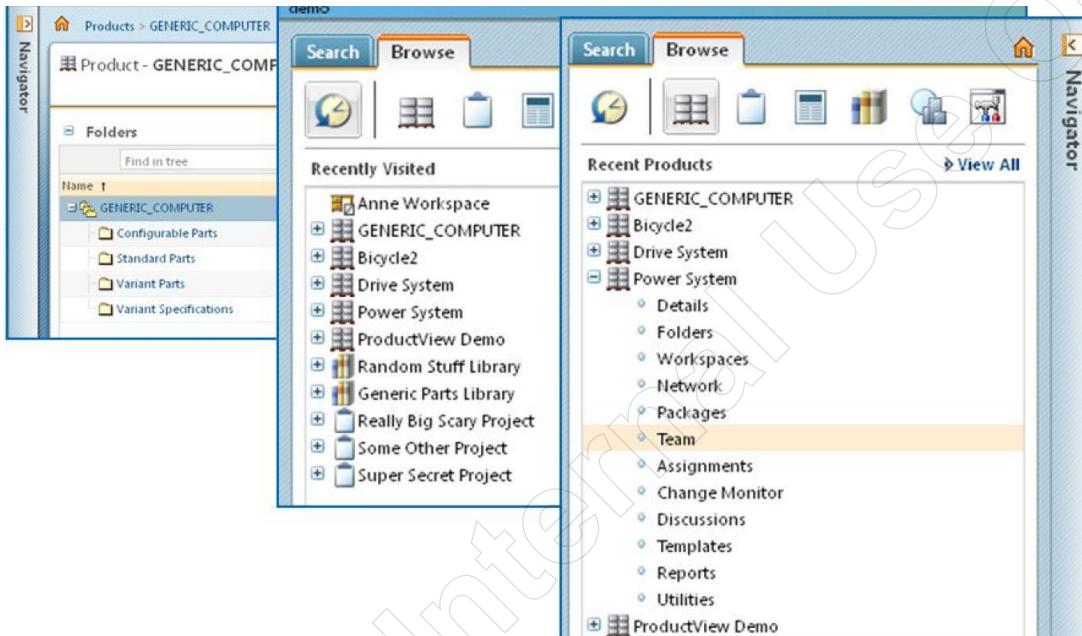
- Describe the Windchill 11.0 – Client Architecture.

Navigation

The Navigator

Highlights

- There when you need it; gone when you don't
- Recently visited contexts and workspaces tab
- Reduces clicks for navigating to recently accessed contexts
- Reduces clicks and load time for getting to second-level pages
- Integrates search into navigation

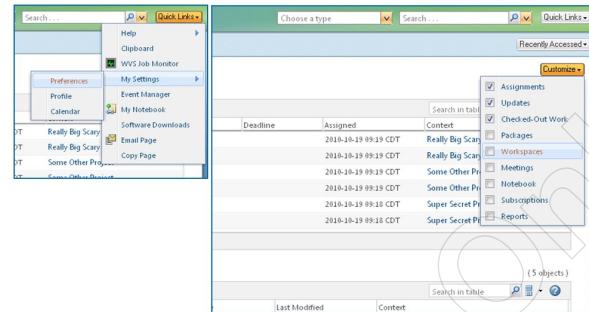


Using the Navigator

Home Page

Configurable Home Page

- Getting started page
- User chooses tables to be on home page
- Reorder tables
- Header area cleanup
- Quick links



Modifying the Home Page

Easily find the information you care about

Home
Home displays information specific to you - your assignments, checked-out work, subscriptions, and more. You can customize your Home page to show the information that you are interested in.

Advanced Search
Most of the time, Simple Search will be enough. Use Advanced Search when you need more power. Your Search history and Saved searches are also available on the Search tab.

Breadcrumb Trail
Breadcrumbs allow you to see where your object is located in the system. You can click on breadcrumbs to quickly navigate to these locations.

Simple Search
This search option is always available and allows you to quickly search upon only the object types you specify.

Help
You can revisit this page and get more detailed information in the Help menu under Quick Links.

Customize
Some pages can be customized to display only the content you want to see.

Navigator (Closed)
Here is what the Navigator

Navigation

Easier to Use

The UI is easier to use because of direct access to information :

- Immediate access to help
- Ability to customize as needed

Search Page

Search Types

Search

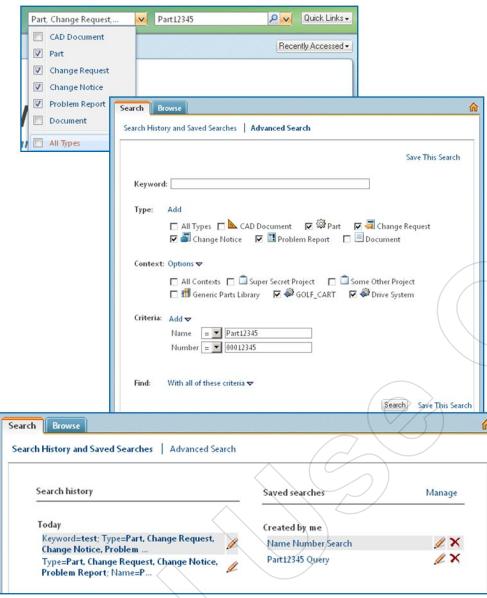
- Search of the entire system
- Search for “all types” or specific types
- Visible on all pages
- Menu with recent searches

Advanced Search

- Redesigned for enhanced usability
- Integrated into navigator
- Search results maintained for later reference

Search History and Saved Searches

- Easy access to recent and saved searches
- Edit recent searches



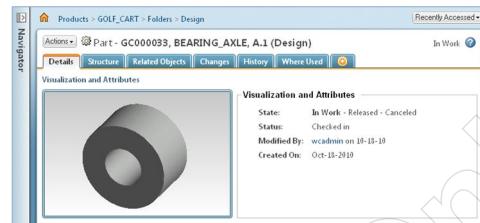
Searches

Information Page

Information Pages are Easily Configurable

Tabbed Display

- OOTB default tabs per object type
- Administrator configurable tabs
- User configurable tabs
- Command line tool for transferring tab configs from test to production



Tabbed Display

User Configuration

- Reorder tabs
- Add tabs, rename tabs, delete tabs
- Add widgets to tabs
- Reorder widgets on tabs

Tabbed Display

Easy to Configure Tabs

Administrator can:

- Import Tabs.
- Configure tabs for Everyone in a context.

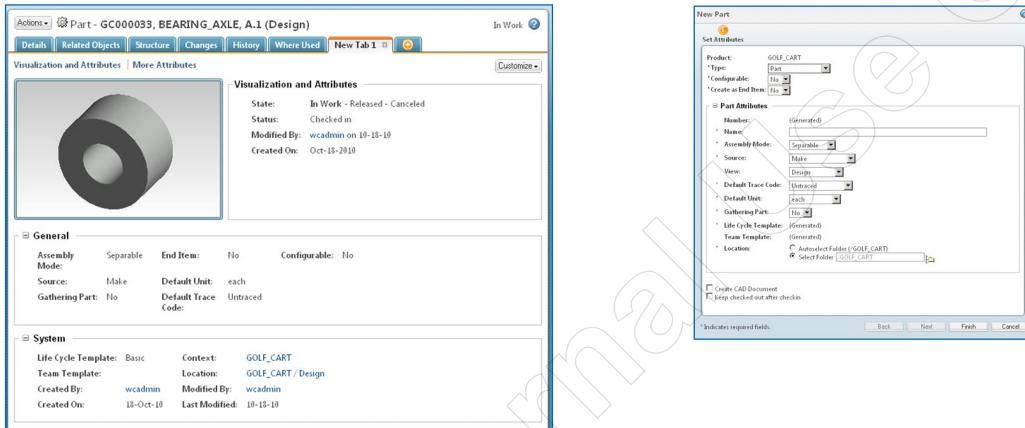
Users will:

- Inherit the tabs configured by the administrator.
- Configure additional tabs viewable only by them.

Attribute Layouts

Attribute Layouts

- Created by administrator in **Type and Attribute Management**
- Hide/Show attributes per screen type (info page, edit wizard, create wizard)
- Specify order (mix modeled and soft attributes), columns, and groupings of attributes
- Define attributes as “view only” or “editable”



Customization Is No Longer Needed

Order and placement of attributes is defined in the **Type and Attribute Management**.

Tables and Trees

Tables Perform Better

Asynchronous data loading

- Shell of table rendered first
- Data streams to client in chunks
- User has nearly immediate control
- Trees – load visible nodes only

Buffered view

- Scrolling instead of paging
- Holds all data up to the size limit
- Next/Previous when > size limit

Client-side data manipulation

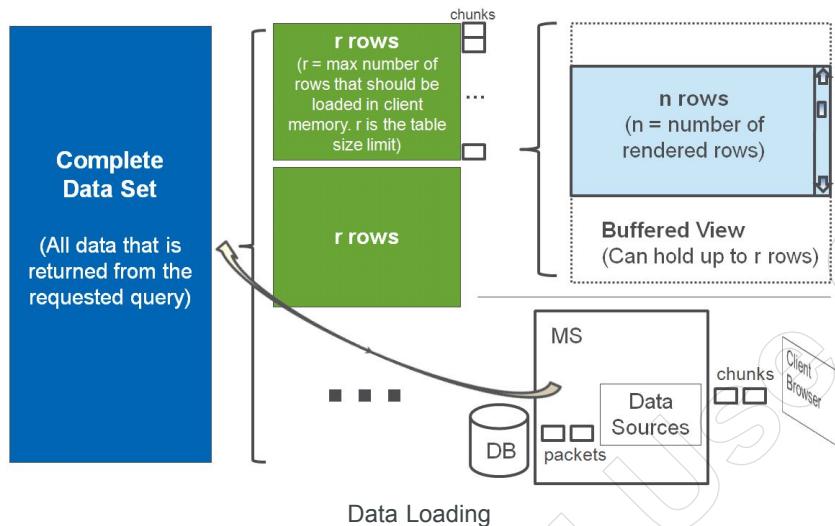
- Sorting
- Column locking
- Search in table
- Future – grouping and filtering

Number	Name	Version	State
0-512140_0.PRT	01-512140.ann	A.1	In Work
0-512140_1.PRT	01-512140.1.par	A.1	In Work
0-512140_2.PRT	01-512140.2.par	A.1	In Work
0-512140_3.PRT	01-512140.3.par	A.1	In Work
0-512140_4.PRT	01-512140.4.par	A.1	In Work
0-512140_5.PRT	01-512140.5.par	A.1	In Work
0-512140_6.PRT	01-512140.6.par	A.1	In Work
0-512140_7.PRT	01-512140.7.par	A.1	In Work
0-512140_8.PRT	01-512140.8.par	A.1	In Work
0-512140_9.PRT	01-512140.9.par	A.1	In Work
0-512140_10.PRT	01-512140.10.par	A.1	In Work
0-512140_11.PRT	01-512140.11.par	A.1	In Work
0-512140_12.PRT	01-512140.12.par	A.1	In Work
0-512140_13.PRT	01-512140.13.par	A.1	In Work
0-512140_14.PRT	01-512140.14.par	A.1	In Work
0-512140_15.PRT	01-512140.15.par	A.1	In Work
0-512140_16.PRT	01-512140.16.par	A.1	In Work
0-512140_17.PRT	01-512140.17.par	A.1	In Work
0-512140_18.PRT	01-512140.18.par	A.1	In Work
0-512140_19.PRT	01-512140.19.par	A.1	In Work
0-512140_20.PRT	01-512140.20.par	A.1	In Work
0-512140_21.PRT	01-512140.21.par	A.1	In Work
0-512140_22.PRT	01-512140.22.par	A.1	In Work
0-512140_23.PRT	01-512140.23.par	A.1	In Work
0-512140_24.PRT	01-512140.24.par	A.1	In Work
0-512140_25.PRT	01-512140.25.par	A.1	In Work
0-512140_26.PRT	01-512140.26.par	A.1	In Work
0-512140_27.PRT	01-512140.27.par	A.1	In Work
0-512140_28.PRT	01-512140.28.par	A.1	In Work
0-512140_29.PRT	01-512140.29.par	A.1	In Work
0-512140_30.PRT	01-512140.30.par	A.1	In Work
0-512140_31.PRT	01-512140.31.par	A.1	In Work
0-512140_32.PRT	01-512140.32.par	A.1	In Work
0-512140_33.PRT	01-512140.33.par	A.1	In Work
0-512140_34.PRT	01-512140.34.par	A.1	In Work
0-512140_35.PRT	01-512140.35.par	A.1	In Work
0-512140_36.PRT	01-512140.36.par	A.1	In Work
0-512140_37.PRT	01-512140.37.par	A.1	In Work
0-512140_38.PRT	01-512140.38.par	A.1	In Work
0-512140_39.PRT	01-512140.39.par	A.1	In Work
0-512140_40.PRT	01-512140.40.par	A.1	In Work
0-512140_41.PRT	01-512140.41.par	A.1	In Work
0-512140_42.PRT	01-512140.42.par	A.1	In Work
0-512140_43.PRT	01-512140.43.par	A.1	In Work
0-512140_44.PRT	01-512140.44.par	A.1	In Work
0-512140_45.PRT	01-512140.45.par	A.1	In Work
0-512140_46.PRT	01-512140.46.par	A.1	In Work
0-512140_47.PRT	01-512140.47.par	A.1	In Work
0-512140_48.PRT	01-512140.48.par	A.1	In Work
0-512140_49.PRT	01-512140.49.par	A.1	In Work
0-512140_50.PRT	01-512140.50.par	A.1	In Work
0-512140_51.PRT	01-512140.51.par	A.1	In Work
0-512140_52.PRT	01-512140.52.par	A.1	In Work
0-512140_53.PRT	01-512140.53.par	A.1	In Work
0-512140_54.PRT	01-512140.54.par	A.1	In Work
0-512140_55.PRT	01-512140.55.par	A.1	In Work
0-512140_56.PRT	01-512140.56.par	A.1	In Work
0-512140_57.PRT	01-512140.57.par	A.1	In Work
0-512140_58.PRT	01-512140.58.par	A.1	In Work
0-512140_59.PRT	01-512140.59.par	A.1	In Work
0-512140_60.PRT	01-512140.60.par	A.1	In Work
0-512140_61.PRT	01-512140.61.par	A.1	In Work
0-512140_62.PRT	01-512140.62.par	A.1	In Work
0-512140_63.PRT	01-512140.63.par	A.1	In Work
0-512140_64.PRT	01-512140.64.par	A.1	In Work
0-512140_65.PRT	01-512140.65.par	A.1	In Work
0-512140_66.PRT	01-512140.66.par	A.1	In Work
0-512140_67.PRT	01-512140.67.par	A.1	In Work
0-512140_68.PRT	01-512140.68.par	A.1	In Work
0-512140_69.PRT	01-512140.69.par	A.1	In Work
0-512140_70.PRT	01-512140.70.par	A.1	In Work
0-512140_71.PRT	01-512140.71.par	A.1	In Work
0-512140_72.PRT	01-512140.72.par	A.1	In Work
0-512140_73.PRT	01-512140.73.par	A.1	In Work
0-512140_74.PRT	01-512140.74.par	A.1	In Work
0-512140_75.PRT	01-512140.75.par	A.1	In Work
0-512140_76.PRT	01-512140.76.par	A.1	In Work
0-512140_77.PRT	01-512140.77.par	A.1	In Work
0-512140_78.PRT	01-512140.78.par	A.1	In Work
0-512140_79.PRT	01-512140.79.par	A.1	In Work
0-512140_80.PRT	01-512140.80.par	A.1	In Work
0-512140_81.PRT	01-512140.81.par	A.1	In Work
0-512140_82.PRT	01-512140.82.par	A.1	In Work
0-512140_83.PRT	01-512140.83.par	A.1	In Work
0-512140_84.PRT	01-512140.84.par	A.1	In Work
0-512140_85.PRT	01-512140.85.par	A.1	In Work
0-512140_86.PRT	01-512140.86.par	A.1	In Work
0-512140_87.PRT	01-512140.87.par	A.1	In Work
0-512140_88.PRT	01-512140.88.par	A.1	In Work
0-512140_89.PRT	01-512140.89.par	A.1	In Work
0-512140_90.PRT	01-512140.90.par	A.1	In Work
0-512140_91.PRT	01-512140.91.par	A.1	In Work
0-512140_92.PRT	01-512140.92.par	A.1	In Work
0-512140_93.PRT	01-512140.93.par	A.1	In Work
0-512140_94.PRT	01-512140.94.par	A.1	In Work
0-512140_95.PRT	01-512140.95.par	A.1	In Work
0-512140_96.PRT	01-512140.96.par	A.1	In Work
0-512140_97.PRT	01-512140.97.par	A.1	In Work
0-512140_98.PRT	01-512140.98.par	A.1	In Work
0-512140_99.PRT	01-512140.99.par	A.1	In Work
0-512140_100.PRT	01-512140.100.par	A.1	In Work

Tables

Data Sources and Incomplete Data Sets

Results Loaded When Needed



How Many Rows Should The Page Load?

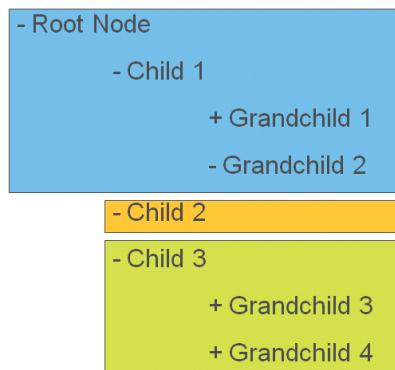
Pages load only the visible number of rows.

- UI keeps track of scrolling and reloads the view as needed based on cached information.
- A cached set of information is available on the page, but the view is not rendered until the objects come into focus.
- When the cached set of data is invalidated, it needs to be refreshed from the Method Server (over the WAN).

Data Sources

Load Visible Tree Nodes Only

Depth-First Loading (WNC 9.x)



Breadth-First Loading (WNC 10)



Comparison between Windchill 9.1 and 10

Visibility Governs the Loading of Data

The root nodes (Level 1) are more important than the sub-nodes.

- These are loaded first.
- Remaining levels are called in as needed.



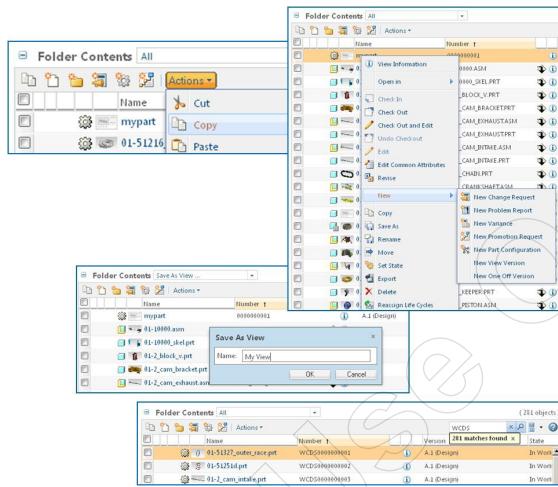
This remains same in Windchill 11.0 also.

More Tables and Trees

More Tables and Trees

Table and Tree functionality

- User Configuration
- Ad hoc Table Views
- Actions Menus
- Search in Table
- Find in Tree
- Export List to File



Folder Contents

Tables Are More Functional

- User Configuration
 - Hide/Unhide columns
 - Reorder columns
 - Resize columns
 - Table height expansion
- Ad hoc Table Views
 - Save view without going to View Manager
- Action Menus
 - Right-click actions menus
 - Single and multi-select menus
 - Shortcut toolbar actions
- Search in Table/Find in Tree
 - Act on entire data set
- Export List to File
 - Export table data to CSV, HTML, TXT, XLS, XLSX, XLS Report, and XML
 - Common action but not yet available on all tables/trees

And More

And More

- New Look and Feel
 - Combined CSS
 - New Icons
- Minimize Full Page Refreshes
- Scrollbar Annihilation
- Header Area Cleanup and “Quick Links”
- Breadcrumbs
- Combining Wizard Steps
- Inline Success Messaging



Quicker Better UI Experience

Page is delivered before the entire page is loaded

- AJAX is leveraged to focus on displaying the shell before the content.
- When content is required, it is loaded/reloaded without reloading the shell.

This same mechanism is used to load to provide confirmation messages.

Common Client Architecture

How many client architectures are there now?

JCA stands alone ... with some exceptions.

- Resurfaced template processor (CADx)
- Ext-GWT (Product Structure Component)
- Some legacy ProjectLink UIs still use NetMarkets

No longer supported

- DCA
- NetMarkets
- Template processor

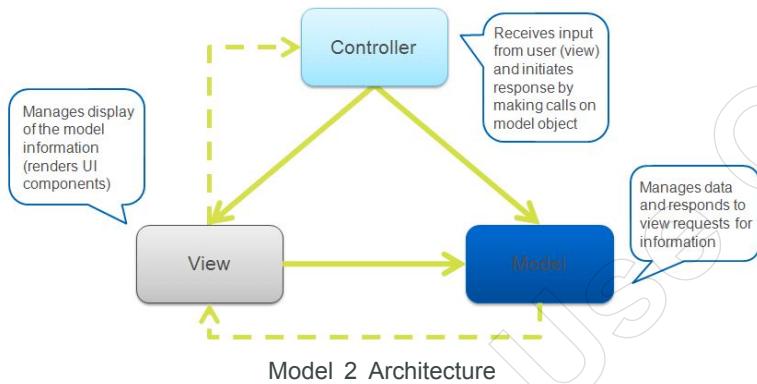


Amusing Client Technologies

Upgrades to JCA

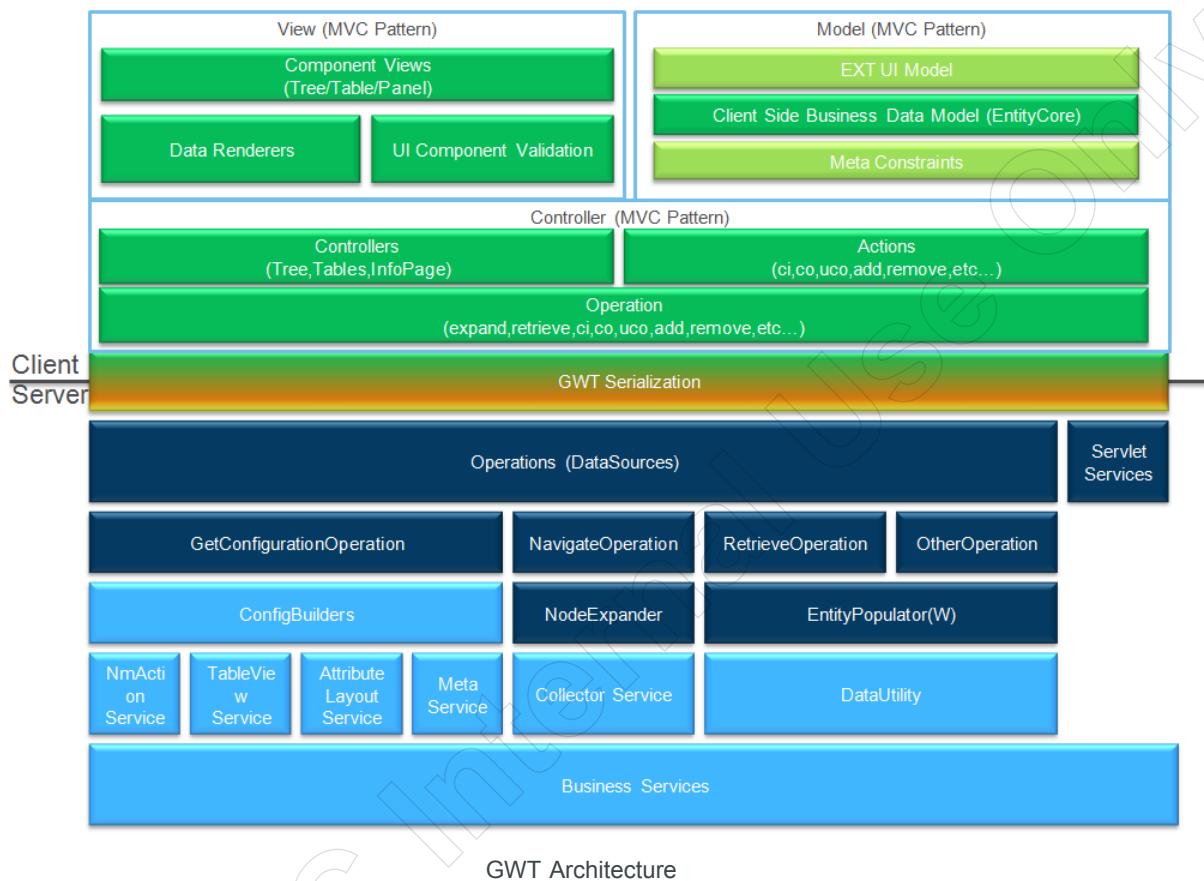
Transforming JCA from a Model 1 to a Model 2 Architecture

- Model 2 is an implementation of the Model View Controller (MVC) pattern.
- Separates business logic from display (view) logic.
- Model can be built and tested independent of visual presentation.



GWT Architecture

GWT Architecture – Client and Server

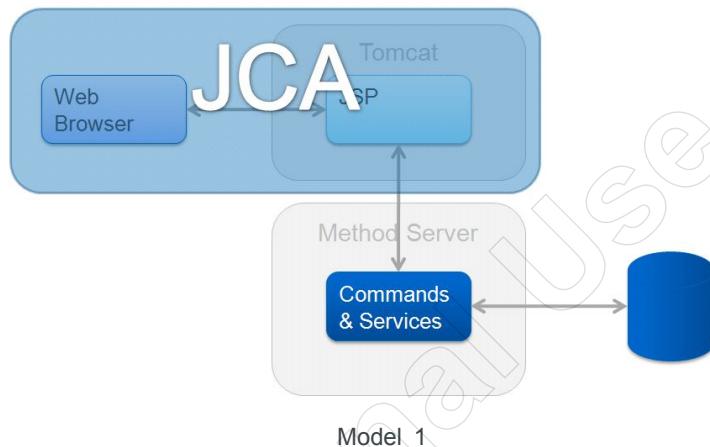


Windchill 9.1 JCA Model

Windchill Before: Model 1

JSP handles all responsibilities for the request:

- Processing the request
- Validating data
- Handling the business logic
- Generating a response



Windchill Before: Model 1

Model 1

- A simple pattern whereby the code responsible for the display of content is intermixed with logic.
- Only recommended for small applications and is mostly obsolete in modern development practices.

Model 2

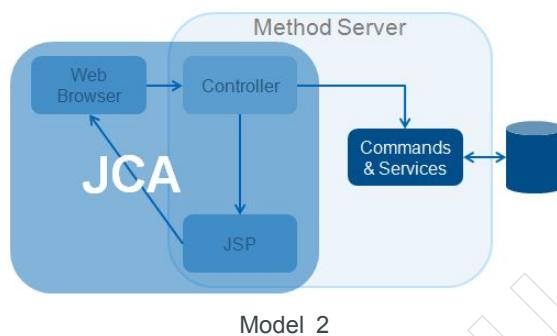
- Recommended for medium- and large-sized applications.
- More complex in that it separates the display of content from the logic used to obtain and manipulate the content.

Windchill 10.2 JCA Model

Windchill Now: Model 2

JCA incorporates all three components and keeps them distinct:

- Move business logic out of JSPs and into controller classes
- Servlet engine (Tomcat) integrated into Method Server



Why Model 2?

Model 2 is a more complex design pattern that separates the display of content from the logic used to obtain and manipulate the content:

- Easier to maintain and extend – views do not refer to each other and there is no presentation logic in the views.
- Enables you to clearly define the roles and responsibilities in large projects – enabling better coordination among team members.



This remains same in Windchill 11.0 also.

JCA Component

New JCA Components Use MVC

Basic Steps to Produce a JCA Component

1. Describe the component
2. Acquire the necessary data
3. Render the component

Authorized Objects				
	Name	Number	Authorized Revisions	Context
<input type="checkbox"/>	4922219.prt	WCAR0000002017	A	World Car Active
<input type="checkbox"/>	4907200.asm	WCAR0000000435	A	World Car Active
<input type="checkbox"/>	8150501.prt	WCAR0000002193	A	World Car Active
<input type="checkbox"/>	4922233.prt	WCAR0000000343	A	World Car Active

Table Example

- X12 JSP – *newAgreeAuthObjectsStep.jsp*
 - <jca:describeTable>
 - <jca:getModel>
 - <jca: renderTable>
- X20 JSP
 - <jsp:include page="\${mvc:getComponentURL('security.newAgreeAuthObject')}" flush="true"/>
- X20 Builder – *NewAgreeAuthObjectsStep.java*
 - Configure the component
 - Build the component data

Industry Standard Frameworks

Including Some Industry Standard Frameworks

Spring MVC (XML configuration Files)

- Provides a framework for wiring controllers, models, and views together
- Component definition moves from the JSP to a Java builder

Ext-JS (coded using JavaScript)

- A JavaScript framework with a library of components and layouts
- Built on an Ext “theme” for a new look and feel
- Ext-JS is wrapped in JCA and thus, fairly transparent

Ext-GWT (coded using Java Swing and then translated to JavaScript using a Toolkit)

- Combines the Google Web Toolkit with the Ext-JS library of components
- Uses Java as a way to generate JavaScript
- Used by the product structure browser

Licensing

ExtJS and ExtGWT are licensed

- ExtJS does not require a tool, but does require the use of a library.
 - The use of this library is implicit in any Windchill Implementation
- ExtGWT requires a tool to convert Java Swing to JavaScript.
- Implementors are not expected to use this unless they acquire a license.

JCA Evolution

JCA, JCA+, JCA++

JCA or JSP Client Architecture

- Developed for 9.0
- Adopted from the NetMarkets infrastructure used by PJL clients and originally introduced in 6.0.

JSCA or JavaScript Client Architecture

- Enhanced for 9.1
- Rendering done in JavaScript using the jsTables framework

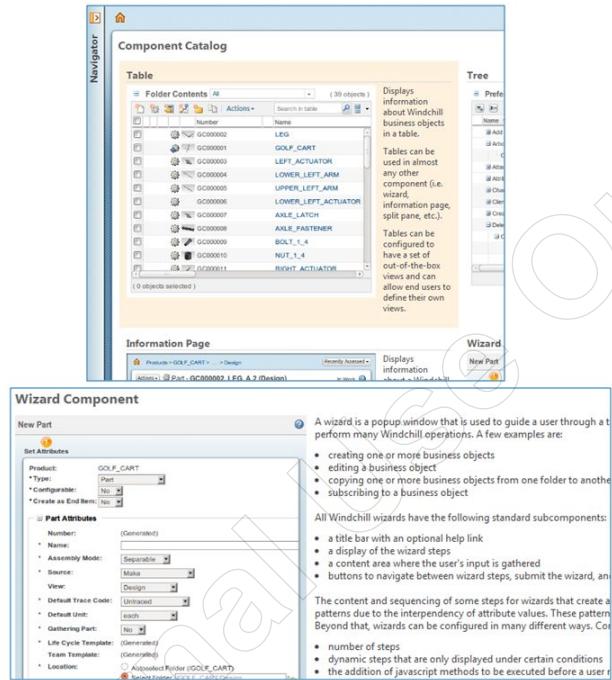
JCA or Just a Client Architecture

- Enhanced for 10.0 and continued till 11.0.
- Still uses JavaScript rendering
- Uses the Ext framework instead of the jsTables framework for rendering
- Introduces MVC

Expanded Resources For Customizers

Guide and Catalog

- Customization Guide
- Component Catalog
 - Windchill Customization Examples – /Windchill/srclib/wnc/Carambola.jar
 - Set “Client Customization” preference to yes
 - Tools and documentation for working with actions, attributes, and MVC builders



Component Catalog and Wizard Component

Module 2

Navigation

Module Overview

In this module, we revisit Navigation to see what has changed from Version 9.1, Windchill 10.0 and Windchill 10.2.

Objectives

After completing this module, you will be able to:

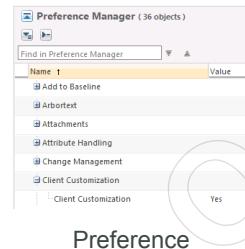
- Identify differences between 9.1 and 11.0 with respect to Navigation.
- Create a custom Tab and sub-tab.

Customization Tab

Online Tools and Examples

Windchill 11.0 offer several tools and examples for customization. These functions are organized in the **Customization** tab.

- In **Preference Manager**, set value of “Client Customization” preference to yes.
- Refresh the page and open the **Navigator**, there will be a new tab **Customization**.



Preference



Customization Tab

Customization Tab collects tools and samples

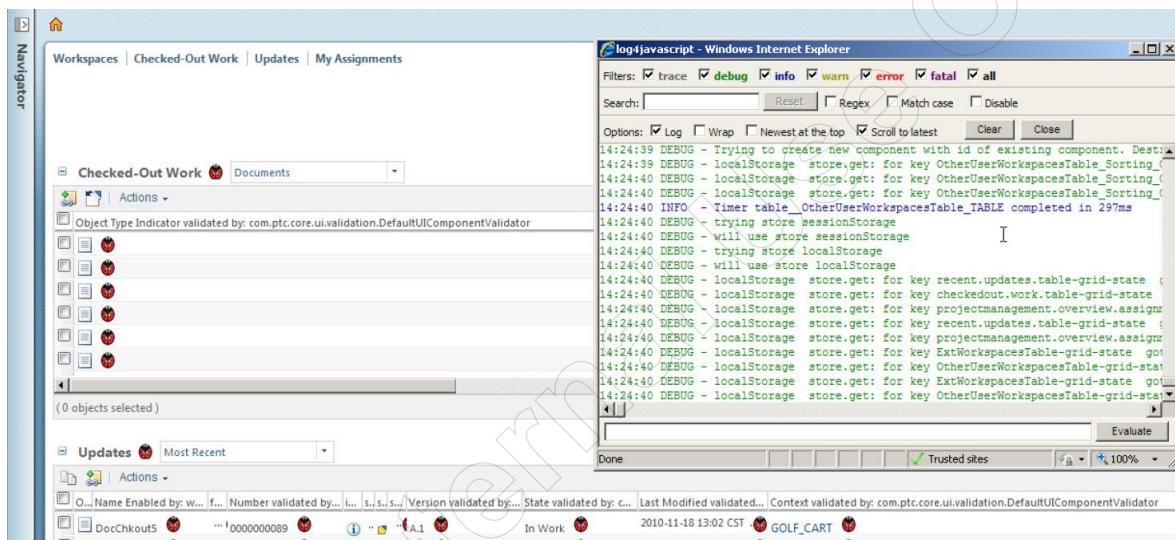
- **ComponentCatalog** for displaying examples of Windchill UI.
- **Tools** for working with actions, attributes, and MVC builders.
- **Documentation** for reference documents links.

Windchill 11.0 Debugging

Debugging Tools

JCA Debug

- Add “jcaDebug=1” to the end of the URL string to view Action and Action Model information
 - Depending on where you put this in the URL will determine which components of the UI use the debug settings.
 - Immediately after “/app” will give the most extensive output.
- Add “jsLog=1” to log JavaScript
- Add “jsDebug=1” to display debug messages in the JavaScript log.



Using Internet Explorer

Finding the Code

Alternatives for Decomposing a Customization

From a given URL, what code generates the page?

- Turn on jcaDebug
 - Tooltip shows the builder on components

The screenshot shows the PTC Windchill interface with a tooltip displayed over a component. The tooltip contains the following information:

```

PTC® Windchill®
Administrator

Products All
Name 1 Prod

ComponentId: netmarket.product.list
ComponentConfigBuilder: com.ptc.windchill.enterprise.product.mvc.builders.ProductListTableBuilder
ComponentDataBuilder: com.ptc.jca.mvc.components.DefaultJcaComponentDataBuilder
VariableRowHeights: true
DefaultColumnFreezeIndex: -1

Created On: 2016-04-19 12:00 UTC
Private Access: No
  
```

The URL shown in the browser is: <http://.../Windchill/app/?jcaDebug=true/#ptc1/...>

- Look for ptc1/comp in the URL
 - Serves a builder directly where the component ID is right in the URL.
 - ptc1/comp/netmarkets.product.list shows the netmarkets.product.list table.
 - OpenGrok for “builder netmarkets.product.list,” and there’s your builder Java class.

The screenshot shows the PTC Windchill interface displaying a table titled "Products All". The table lists various product series with their owners and last modified dates. The URL shown in the browser is: [netmarket.product.list](#)

Name	Owner	Last Modified	Description
Name 1	Jones, Mike	2011-02-28 17:15 UTC	
Chainsaw - 220 Series	Jones, Mike	2011-02-28 17:15 UTC	
Drill - 400 Series	Jones, Mike	2011-03-01 12:21 UTC	
Drill - 510 Series	Jones, Mike	2011-03-03 19:14 UTC	
Drill - 600 Series	Jones, Mike	2011-03-03 19:41 UTC	
Drill - 650 Series	Jones, Mike	2011-03-03 19:39 UTC	
Drill - 700 Series	Jones, Mike	2011-03-03 19:40 UTC	
Drill - 710 Series	Jones, Mike	2011-03-03 19:40 UTC	
Drill - 720 Series	Jones, Mike	2011-03-04 22:00 UTC	
Drill - 750 Series	Jones, Mike	2011-03-03 19:42 UTC	
Drill - 800 Series	Jones, Mike	2011-03-03 19:42 UTC	
Drill - 810 Series	Jones, Mike	2011-03-03 19:43 UTC	
Drill - 820 Series	Jones, Mike	2011-03-03 19:43 UTC	
Drill - 900 Series	Jones, Mike	2011-03-03 19:40 UTC	
Engine - 620 Series	Jones, Mike	2011-03-03 21:19 UTC	
Pump - 6600 Series	Jones, Mike	2016-03-21 19:08 UTC	

- Look for the path in the URL
 - For ptc1/org/list, look for codebase/org/list.jsp
 - Grok for “path:org/list”

The screenshot shows the PTC Windchill interface displaying a table titled "Organizations". The table lists three organizations with their subscribers and last modified dates. The URL shown in the browser is: [org/list](#)

Name	Subscriber	Last Modified
ptc	Yes	2011-02-22 10:36 UTC
PTC Outdoors	Yes	2011-03-02 13:10 UTC
PTC Power Equipment	Yes	2011-02-28 15:57 UTC

Windchill 11.0 References

Resources

JCA

- Client Infrastructure Wiki Page: http://rdwiki.ptcnet.ptc.com/wiki/index.php?title=Client_Infrastructure_Info
- Client Architecture Forum: <https://share.ptc.com/sites/rd/wpg/csi/SitePages/Home.aspx>

Windchill JavaDoc

- http://rdwiki.ptcnet.ptc.com/mediawiki/index.php?title=Windchill_Javadoc



Above links are for PTC employee use only, partners likely will not be able to access it.

Architecture and Third-Party Frameworks

- MVC Pattern: <http://java.sun.com/blueprints/patterns/MVC-detailed.html>
- Model 1 Versus Model 2 Architectures: http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/
- Spring MVC Explained: <http://jnb.ociweb.com/jnb/jnbOct2004.html>
- Ext-JS Overview: <http://extjs.com/products/extjs/>
- Googe Web Toolkit (GWT): <http://code.google.com/webtoolkit/overview.html>
- Ext-GWT: <http://extjs.com/products/gxt/>

Resource Bundle Overview

Resource Bundle Overview

- Background
 - Previously represented as rblInfo files
 - Now represented as annotated Java files (rblInfo format still supported)

- The parent class [WTListResourceBundle](#) relies on an OOTB Java resource API called [ListResourceBundle](#), but adds the annotation intelligence needed to parse entries.

Annotations

In the past, RBInfo files were used to create Java resource bundles. But now, Annotations simplify the creation of resource bundles in Java.

- Annotations are used to specify both the strings used to reference the resource class and the resource entries contained in the resource class.
- The values of the [@RBEntry](#) annotations are the values of the resources.

```
package wt.part;
import wt.util.resource.*;
@RBUUID("wt.part.partResource")
public final class partResource extends WTListResourceBundle {
    /**
     * Messages for HTML Template Processing -----
     */
    @RBEntry("Invalid CGI parameters. Parameter \'{0}\' not four
    public static final String INVALID_PARAMETERS = "12";
    @RBEntry(" {0}{1} {2} {3} of {4}")
    @RBComment("For example, \"100: 16 each of Bolt - 1234\" or
    @RBargComment0("is a line number")
    @RBargComment1("is a separator")
    @RBargComment2("is a quantity")
    @RBargComment3("is a unit")
    @RBargComment4("is and identity")
    public static final String PART_USAGE = "13";
```

Example from OpenGrok



Javadoc for [WTListResourceBundle](#)

Reference Resource Bundle

Using Resources with Actions

- Resources can be referenced in the action configuration files:
- These actions will get their localized text from the Java class resource converted in the previous slides from a legacy RBInfo file.

```
<objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld">
        <command url="netmarkets/jsp/serviceAcademy/navigationlist.jsp" windowType="page"/>
    </action>

    <action name="myProductListTable">
        <component name="com.ptc.serviceAcademy.builders.MyProductListTableBuilder" windowType="page"/>
    </action>

    <action name="tableExample">
        <component name="com.ptc.serviceAcademy.builders.TableBuilder" windowType="page"/>
    </action>

    <action name="multiComponent">
        <component name="com.ptc.serviceAcademy.builders.MultiComponentBuilder" windowType="page"/>
    </action>

    <action name="treeExample">
        <component name="com.ptc.serviceAcademy.builders.TreeBuilder" windowType="page"/>
    </action>
</objecttype>
```

Refer the Resource Bundle in the Actions

Action Models and Actions

Tabs Are Similar to Windchill 9.1 and Windchill 10.2

Action Models assemble Actions.

- Below is the Main Navigation

```
<?xml version="1.0"?>
<!DOCTYPE listofactions SYSTEM 'actions.dtd'>
<listofactions>
<objecttype name="navigation" class="" resourceBundle="com.ptc.core.ui.navigationRB">
<action name="recentContexts">
```

Main Navigation



Actions define the type of command that runs.

```
<action name="program">
<action name="product">
<action name="project">
<action name="library">
<action name="change">
<action name="site">
<action name="org">
<action name="supplier">
<action name="product.list">
<action name="program.list">
<action name="library.list">
<action name="org.list">
<command class="netmarkets" url="org/list" windowType="page"/>
<action name="search"/>
```

Actions

Similar to Windchill 9.1 and Windchill 10.2

This customization pattern is similar to what we saw in Windchill 9.1 and Windchill 10.2.

- custom-actionmodel.xml controls the assembly of actions.
- custom-actions.xml defines custom actions.
- Anything in these files overrides OOTB behavior.

Inserting a Tab

Steps for Creating a Tab

To create a tab:

1. Define/Reuse the Actions.
2. Create the sub-tab model.
3. Update Resources Bundles with icons and Text.
4. Reference the sub-tab the main tab.

```
<model name="serviceAcademy navigation"
    resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld" type="object"/>
    <action name="myProductListTable" type="object"/>
    <action name="tableExample" type="object"/>
    <action name="multiComponent" type="object"/>
    <action name="treeExample" type="object"/>
</model>
```

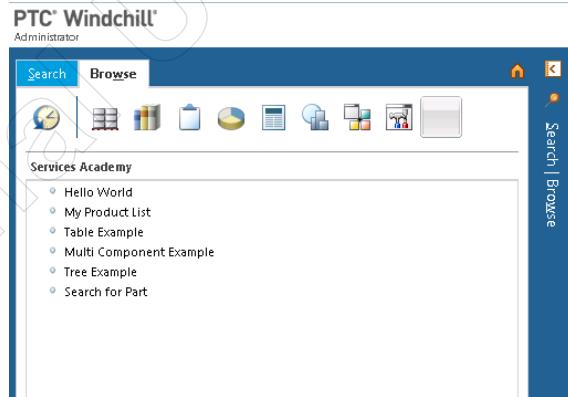
Sub-tab model

```
package com.ptc.serviceAcademy.resource;
import vt.util.resource.XMlEntry;
@ModelID("com.ptc.serviceAcademy.resource.Navigation")
public final class Navigation extends XMLResourceBundle {
    @Entry("Service Academy")
    public static final String NAVIGATION_SERVICEACADEMY_DESCRIPTION = "navigation.serviceAcademy.description";
    @Entry("Service Academy")
    public static final String NAVIGATION_SERVICEACADEMY_TITLE = "navigation.serviceAcademy.title";
    @Entry("Service Academy")
    public static final String NAVIGATION_SERVICEACADEMY_TOOLTIP = "navigation.serviceAcademy.tooltip";
    @Entry("Service Academy")
    public static final String NAVIGATION_SERVICEACADEMY_ICON = "navigation.serviceAcademy.icon";
}
```

Resource Bundle

```
<model name="main navigation" id="browseActions"
    resourceBundle="com.ptc.core.ui.navigationRB">
    <description>
        Main navigation (Tabs)
    </description>
    <action name="recentcontexts" type="navigation"/>
    <action name="product" type="navigation"/>
    <action name="library" type="navigation"/>
    <action name="project" type="navigation"/>
    <action name="change" type="navigation"/>
    <action name="supplier" type="navigation"/>
    <action name="org" type="navigation"/>
    <action name="site" type="navigation"/>
    <action name="search" type="navigation"/>
    <!-- entry for serviceAcademy tab-->
    <action name="customization" type="navigation"/>
    <action name="serviceAcademy" type="navigation"/>
</model>
```

Tab Model



End Result of Sample Customization

Inserting an Action

Completing the Action Definition

The action needs:

- Resource Bundle entries
- Action Definition

```
Resource("Table Example")
public static final String OBJECT_TABLEEXAMPLE_DESCRIPTION = "object.tableExample.description";
@Resource("Table Example")
public static final String OBJECT_TABLEEXAMPLE_TITLE = "object.tableExample.title";
@Resource("Table Example")
public static final String OBJECT_TABLEEXAMPLE_TOOLTIP = "object.tableExample.tooltip";
```

Resource Entries

```
<objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="tableExample">
        <component name="com.ptc.serviceAcademy.builders.TableBuilder" windowType="page"/>
    </action>
</objecttype>
```

Action Definition

Referencing a Custom JSP

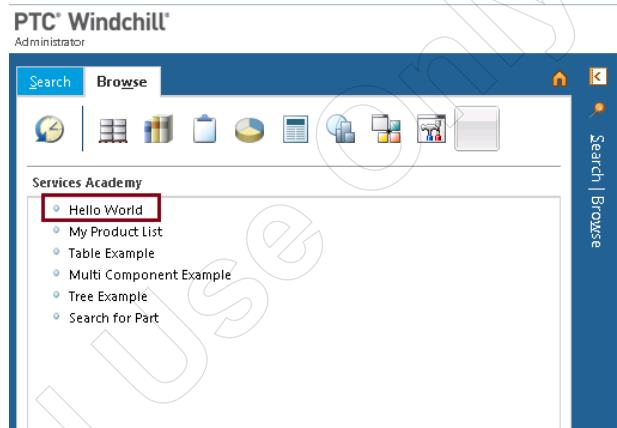
Pointing an Action to a Custom JSP

Syntax of the action is new

- clientResource has added entries (not displayed)
- URL is actually a mapping to “ptc1”

```
<objecttype name="object" class="java.lang.Object"
    resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
<action name="helloWorld">
    <rend url="/netmarkets/jsp/serviceAcademy/navigationlist.jsp?windowType=page"/>
</action>
</objecttype>
```

Action Definition for Empty Page



Navigation Entry for Empty Page

```
<%@page language="java" session="true" pageEncoding="utf-8"%>
<%@include file="/netmarkets/jsp/util/begin.jspf" %>
```

Hello World!

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

Simple JSP file

PTC Windchill*

Administrator

Hello World!!!

Display of Page

A Few Ways to Do This

This slide presents the legacy way to include tabs

class="netmarkets"

- This method looks for JSP file under codebase/netmarkets/jsp .

 The exercise will present a different way.

Menus for Objects

Identify and Modify Menus

Identify customization using the tools.

Action Model Report

Search By:

Action Model:	Ex: 'DefaultWizardButtons'
Description:	Ex: 'Main navigation (tabs)'
Model File:	Ex: 'actionmodels.xml'
Menu For:	Ex: 'wt.part.WTPart'
Dev Owner:	

Search Clear

Action Models: (1 objects)

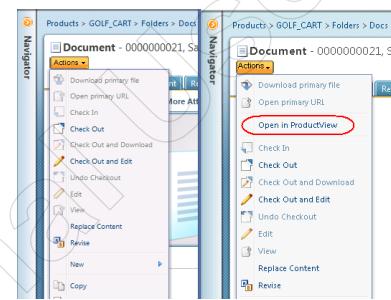
Model Name	actionDetails
docs row actions	(i)

Searching for Action Model

Modify the models identified by the tool

```
<!-- list used for document actions -->
<model name="docs row actions" menuFor="wt.doc.WTDocument">
  <action name="view" type="object"/> <!-- Info page -->
  <action name="separator" type="separator"/>
  <action name="download_primary_attachment" type="attachments"/>
  <action name="redirect_primary_attachment" type="attachments"/>
  <action name="checkin" type="wip"/> <!-- Check In -->
```

Sample Modification of a Model



Before and After Removing Open in ProductView Action

Changing the Action Menu for a Document

On this side, we show how to search for the model that controls the action menu on a document.

- In Action Model Report page under Action Model table you will see “Menu For” search field which can be used to identify the menu for a given object.
- This was not available in previous versions.

Exercise 1: Modify List of Actions

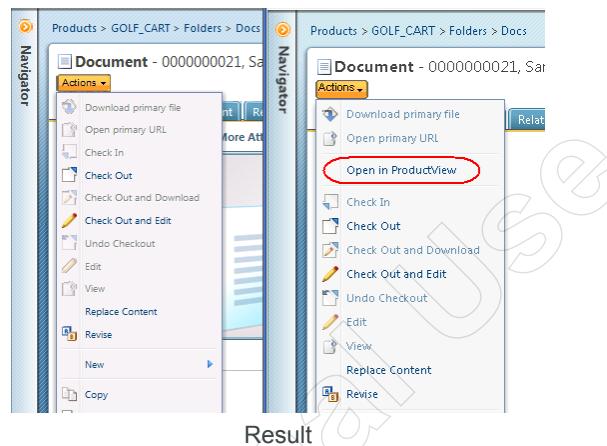
Objectives

After successfully completing this exercise, you will be able to:

- Modify List of Actions

Scenario

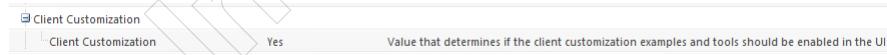
In this exercise, you will use the Windchill Tools menu to identify the menu for an object and then you will change the menu.



Task 1: Enable the Customization Tab.

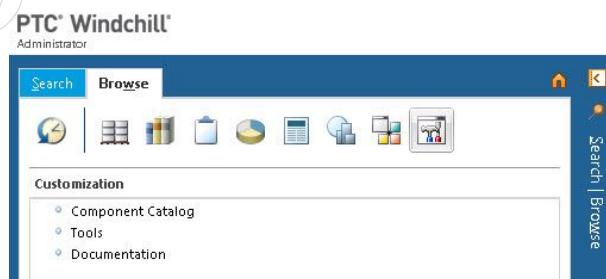
1. Navigate to Site > Utilities > Preference Management.
2. Update Client Customization>Client Customization to Yes

Example:



Result:

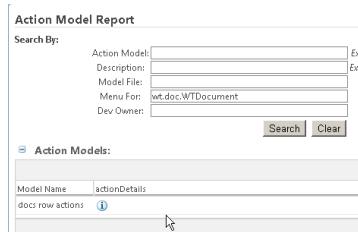
You should now see the customization Tab



Task 2: Determine the action model for Documents.

1. Navigate to the Customization >Tools Tab.
2. Run the Action Model
3. Enter "wt.doc.WTDocument" for "Menu For" for field.
4. Search for the action model
5. Click the information icon next to the action model to determine the file that holds the Action Model

Result:



Task 3: Update the Action Model.

1. Copy the “docs row actions toolbar” model from the location found in the previous task (`/config/actions/ DocumentManagement-actionmodels.xml`) into `/config/actions/Custom-actionmodels.xml`.
2. Choose an Action to Remove
3. Remove an action

Task 4: Reload the Action Model.

1. Navigate to **Customization Tab >Tools**.
2. Select **Reload Actions**.

Example:

The screenshot shows the 'Tools' section of the customization tab. It lists various tools with their descriptions. The 'Reload Action(s)' option is highlighted in red, indicating it is the selected tool. Other tools listed include Action, Action Model, Application Context Service/Resource Properties, Available Attributes, Logical Attributes Report, Property Report, MVC Builder, and Under Search.

Task 5: Test the Customization.

1. Create a document/search for an existing Document.
2. Review the **Actions** tab.
3. Verify that the action you have removed does not exist in the list of actions.

Completion Criteria

1. Students modified the list of actions.

This completes the exercise.

Exercise 2: Configure Eclipse for Windchill Development

Objectives

After successfully completing this exercise, you will be able to:

- Set up the developing environment.
- Successfully create Java project and debug.

Scenario

In this exercise, you will install the Eclipse and set up some configuration for Windchill, and create a Java project and validate the configuration.

Task 1: Check for Java version.

1. JDK should be installed on your local computer.
2. Open Command Prompt and Execute the command: **java -version** It should be 1.6 or above.

```
Administrator: Windows PowerShell (8)
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\PTCTrainingUser> java -version
java version "1.8.0_65"
Java(TM) SE Runtime Environment (build 1.8.0_65-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)
PS C:\Users\PTCTrainingUser>
```

Task 2: Set the Environmental Variables.

1. Execute the command: **javac**

The command should return “help” information for the command.

```
Administrator: Windows PowerShell (8)
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

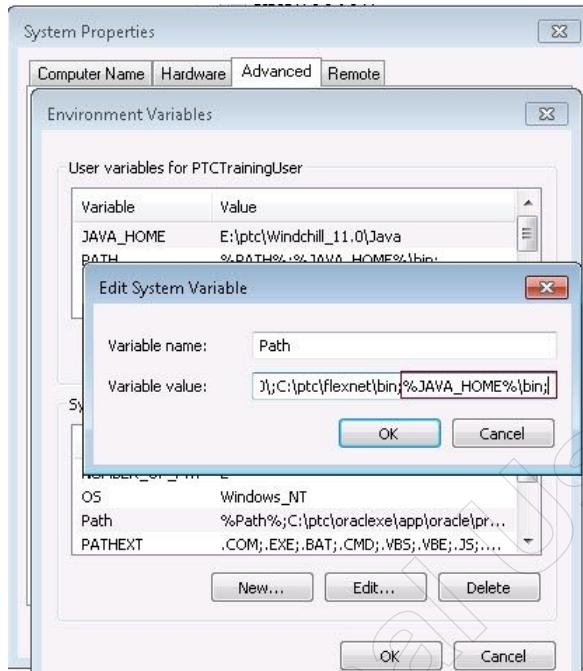
PS C:\Users\PTCTrainingUser> javac
Usage: javac <options> <source files>
where possible options include:
  -g           Generate all debugging info
  -g:none      Generate no debugging info
  -g:{lines,wars,source} Generate only some debugging info
  -nowarn      Generate no warnings
  -verbose     Output messages about what the compiler is doing
  -deprecation Output source locations where deprecated APIs are used
  -classpath <path> Specify where to find user class files and annotation processors
  -cp <path>   Specify where to find user class files and annotation processors
  -sourcepath <path> Specify where to find input source files
  -bootclasspath <path> Override location of bootstrap class files
  -extdirs <dirs> Override location of installed extensions
  -endorseddirs <dirs> Override location of endorsed standards path
  -proc:<none,only> Control whether annotation processing and/or compilation is done.
  -processor <class1>[,<class2>,...] Names of the annotation processors to run; bypasses default disco
ss
  -processorpath <path> Specify where to find annotation processors
  -parameters  Generate metadata for reflection on method parameters
  -d <directory> Specify where to place generated class files
  -s <directory> Specify where to place generated source files
  -h <directory> Specify where to place generated native header files
  -implied:<none,class> Specify whether or not to generate class files for implicitly referenced files
  -X           Provide additional options for the compiler
  -source <release> Provide source compatibility with specified release
  -target <release> Generate class files for specific VM version
  -version     Check that API used is available in the specified profile
  -help        Print a synopsis of standard options
  -Akey[=value] Options to pass to annotation processors
  -J<flag>    Pass <flag> directly to the runtime system
  -Werror     Terminate compilation if warnings occur
  R<filename> Read options and filenames from file

PS C:\Users\PTCTrainingUser>
```

If the command returns the following result, it means your **environment variable** is not set properly.

```
Administrator: Windows PowerShell (8)
PS C:\Users\PTCTrainingUser> javac
The term 'javac' is not recognized as the name of a cmdlet, function, or a path was included, verify that the path at line:1 char:6
+ javac <<<
+ CategoryInfo          : ObjectNotFound: <javac:String> [], CommandNotFoundException
PS C:\Users\PTCTrainingUser>
```

To set your ***environment variable***, navigate to Computer/Properties/Advanced System Settings/Environment Variables and set JAVA_HOME and Path as shown in Environment Variables figure.



Any spaces in the Java directory path may cause problems.

Task 3: Install Eclipse.

1. Download Eclipse from the following links. The eclipse version is: Helios 3.6 R2
 - [Eclipse Helios 3.6 R2 for Windows 64Bit](#)
 - [Eclipse Helios 3.6 R2 for Windows 32Bit](#)
2. Extract compressed file and copy to any directory on local machine.



It is widely accepted that all of the developing tools are installed in the directory: E:\DEV_TOOLS

	Name	Date modified	Type	Size
ces	configuration	4/19/2016 7:11 AM	File folder	
nts	dropins	2/18/2011 5:01 AM	File folder	
ngUser	features	4/18/2016 9:25 AM	File folder	
	p2	4/18/2016 9:25 AM	File folder	
	plugins	4/18/2016 9:25 AM	File folder	
	readme	4/18/2016 9:25 AM	File folder	
	.eclipseproduct	7/29/2010 10:37 AM	ECLIPSEPRODUCT...	1 KB
	artifacts.xml	2/18/2011 5:01 AM	XML Document	213 KB
	eclipse.exe	12/22/2010 2:08 PM	Application	43 KB
	eclipse.ini	2/18/2011 5:01 AM	Configuration sett...	1 KB
	eclipsesec.exe	12/22/2010 2:08 PM	Application	18 KB
	epl-v10.html	2/25/2005 6:53 PM	HTML Document	17 KB
	notice.html	4/27/2010 3:23 PM	HTML Document	9 KB

Task 4: Configure BIF in Windchill.

1. To Download BIF SDK, go to <https://ssp.ptc.com/wiki/display/bif/BIF> , Latest BIF SDK (jar file) can be downloaded from the “Download” tab as shown in below snapshot

Download

BIF distribution is available for all PTC employees. For more information on BIF usage for NON-PTC employees, see [BIF License Agreement](#)

BIF 2.2

- Supported release of BIF 2.2 can be downloaded from or externally link to <https://sspsvn.ptc.com/Tools/BIF/11.0/releases/2.2.0/LATEST.release>

BIF 2.1

- Create a copy of BIF to your Windchill development environment. Download the latest jar file to the location: *E:\ptc\Windchill_11.0\Windchill\BIF*



- The jar file is required to execute the BIF installer.
- The contents of the jar do not need to be extracted for execution.

- Open a Windchill shell, and execute following command;

```
java -jar <Location>/BIFSDK-2.2.x.x.jar project
```

Answer the following questions presented to create a project:

The following questions will be asked:

- The Location of the Workspace directory - Example: If the location of the workspace directory is desired to be E:\Workspace, then the location is E:\.
- The name of the project: **GSProject**.
- If third party libraries should be installed - This is only asked if Ivy is not already installed.
- Is the setup related to COTS: **n**.
- Is there an associated SVN repository: **n**.

```
Administrator: Windchill Shell
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

D:\ptc\Windchill_11.0\Windchill>java -jar D:\ptc\Windchill_11.0\Windchill\BIF\BIFSDK-2.2.0.07.jar project

=====
Executing PROJECT CREATION
=====

Please enter location of Workspace Directory [D:/ptc/Windchill_11.0/Windchill
/BIFI: D\>
Please enter your project name [SampleProject]: GSProject
Is this a setup for COTS project [y,<n>]: n
Does the project have an existing SVN URL? [y,<n>]: n
setup:

svnCheckout:
Skipped because property 'svnExecute' not set.

folderCreate:
create_project:
Execution Directory: D:/ptc/Windchill_11.0/Windchill/BIF
Copying 1 file to D:/Workspace/GSProject/srclib
Copying D:/ptc/Windchill_11.0/Windchill/BIF/BIFSDK-2.2.0.07.jar to D:/Workspace/GSProject/srclib/BIFSDK-2.2.0.07.jar
Copying 8 files to D:/Workspace/GSProject
Project GSProject development environment is ready!

COTSconfiguration:
Skipped because property 'executeCOTS' not set.

all:

=====
Executing CLEANUP
=====
```



The prompts may contain default values, values inside a set of brackets, that can be used by just pressing enter on the prompt. If the default contains a set of values, that are the only valid responses, the default value will be inside a set of parenthesis ().

- A blank project skeleton with the necessary BIF and Eclipse artifacts will be created at the location with the name specified.

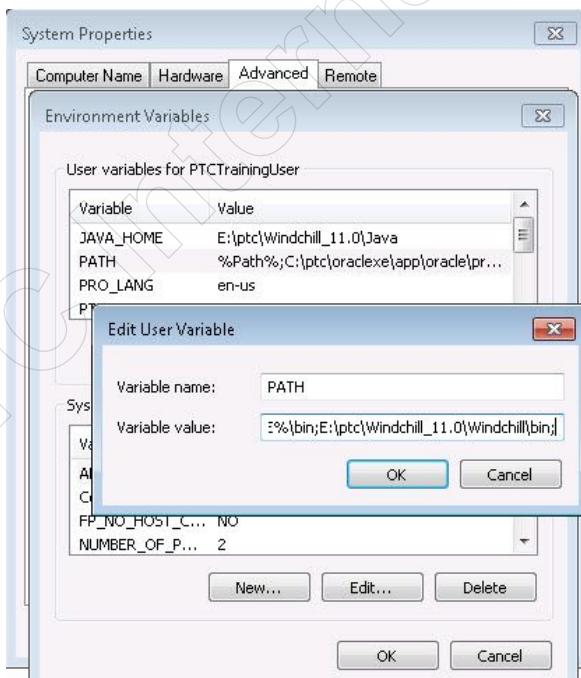
Name	Size	Type
codebase		File Folder
db		File Folder
ivy-settings		File Folder
loadFiles		File Folder
src		File Folder
srclib		File Folder
wtCustom		File Folder
wtSafeArea		File Folder
.classpath	10 KB	CLASSPATH File
.factorypath	2 KB	FACTORYPATH File
.project	2 KB	PROJECT File
bif.ant	1 KB	PROPERTIES File
component	2 KB	XML Document

5. Copy the Workspace folder to your local machine say at E:\Wokspace

Task 5: Configure eclipse and Import BIF Project.

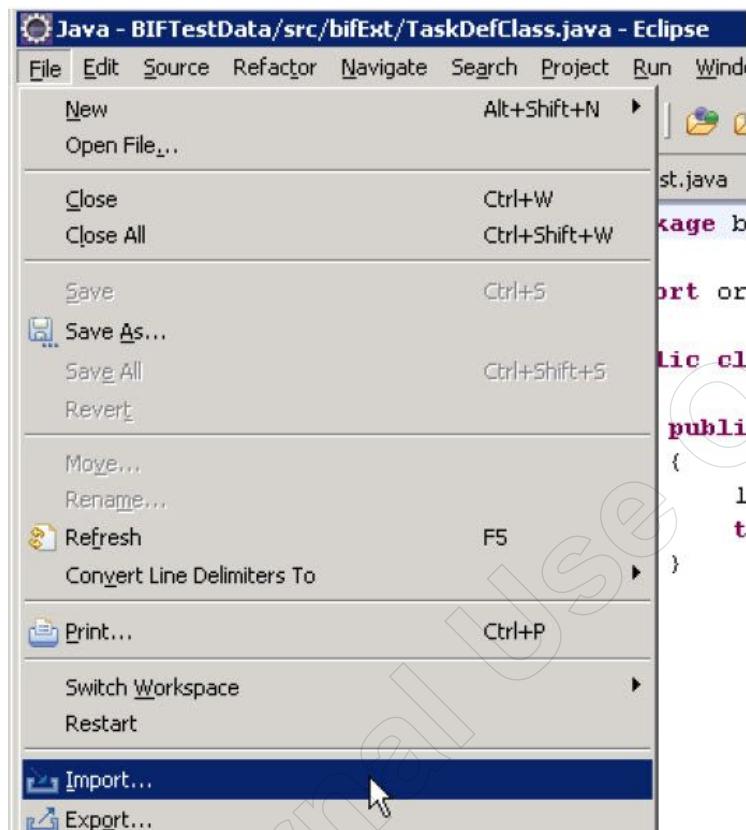
1. BEFORE OPENING ECLIPSE, Add <WT_HOME>/bin to the System Environment variable PATH.

- Right click on **My Computer** and select **Properties**
- Select the **Advanced** tab and click the **Environment Variables** button.
- Highlight the **PATH** variable under *User variables for Administrator* and click the **Edit** button
- In the *Edit User Variable* window, add <WT_HOME>/bin to the end of the *Variable value*.
- Click **OK**.

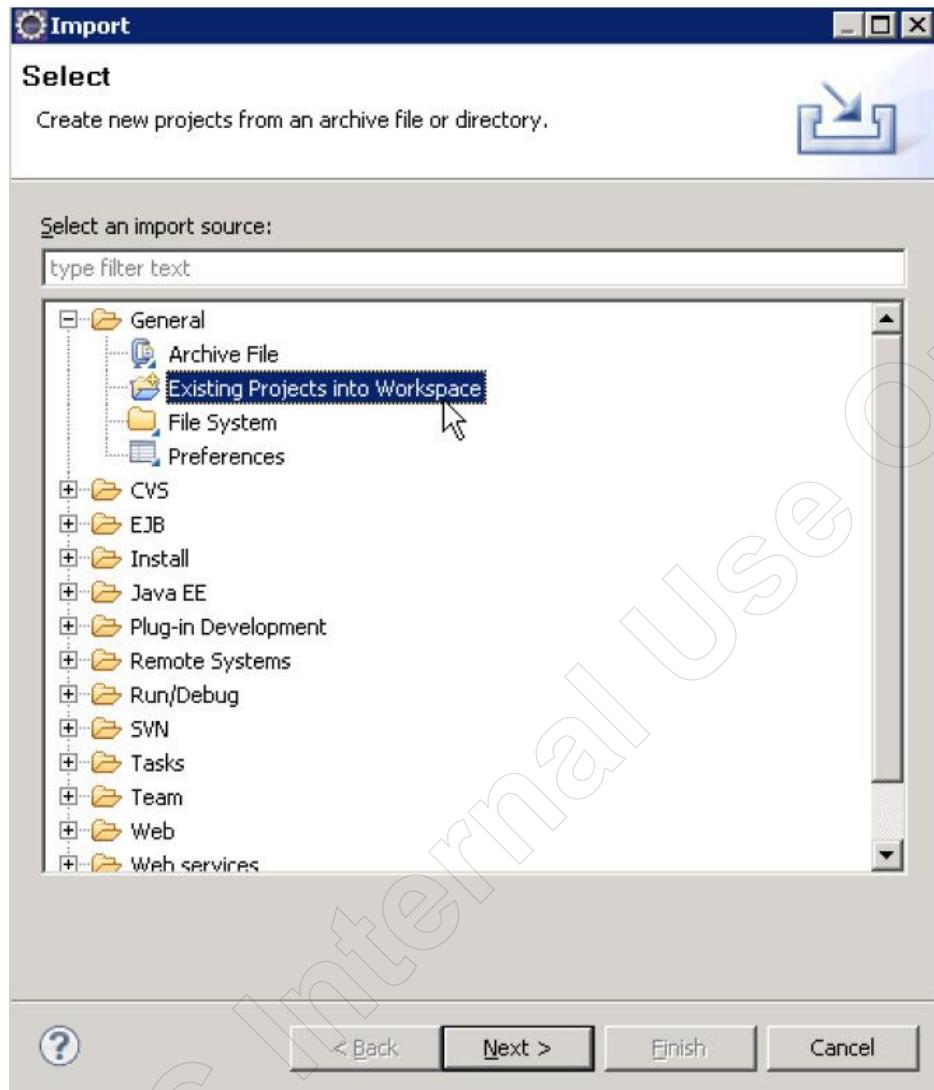


2. Import BIF Project

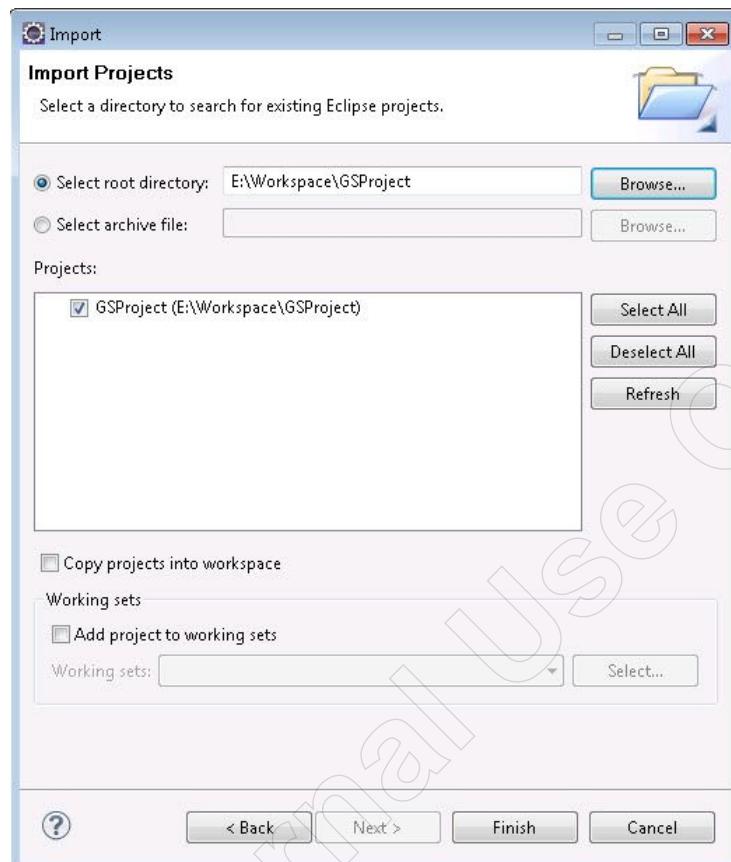
- Open Eclipse and click **File --> Import**



- Select **Existing Projects into Workspace** under the **General** folder within the Import window.



- Browse to the location of the newly created project and select the directory that contains the project.

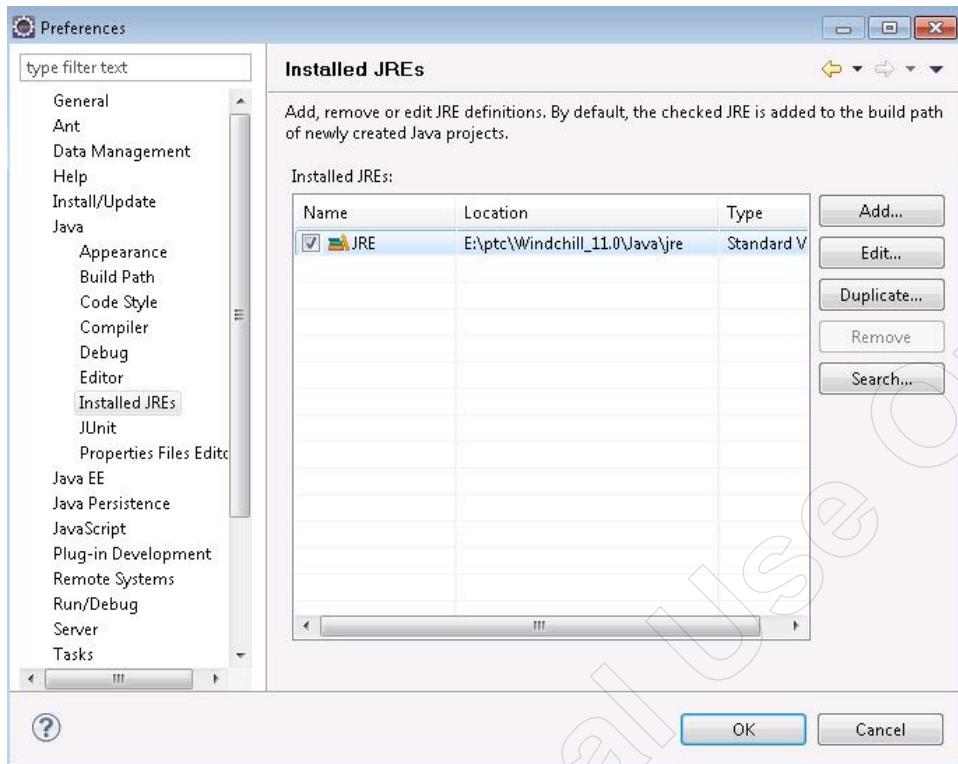


Images shown here are just for reference purpose. Project's path and name may vary depending upon student choice, it is a directory path where student have copied project folder at local machine.

- The imported project will be listed within the **Package Explorer**.

3. Configure JAVA used by Eclipse

- Switch the Java JRE Eclipse uses from **Window -> Preferences -> Java -> Installed JREs**
- Add the Windchill JRE and deselect the default.



4. Modify Eclipse Preference not to delete Windchill codebase

- Navigate to **Window > Preferences > Java > Compiler > Building > Output folder**
- Uncheck **Scrub output folder when cleaning projects**



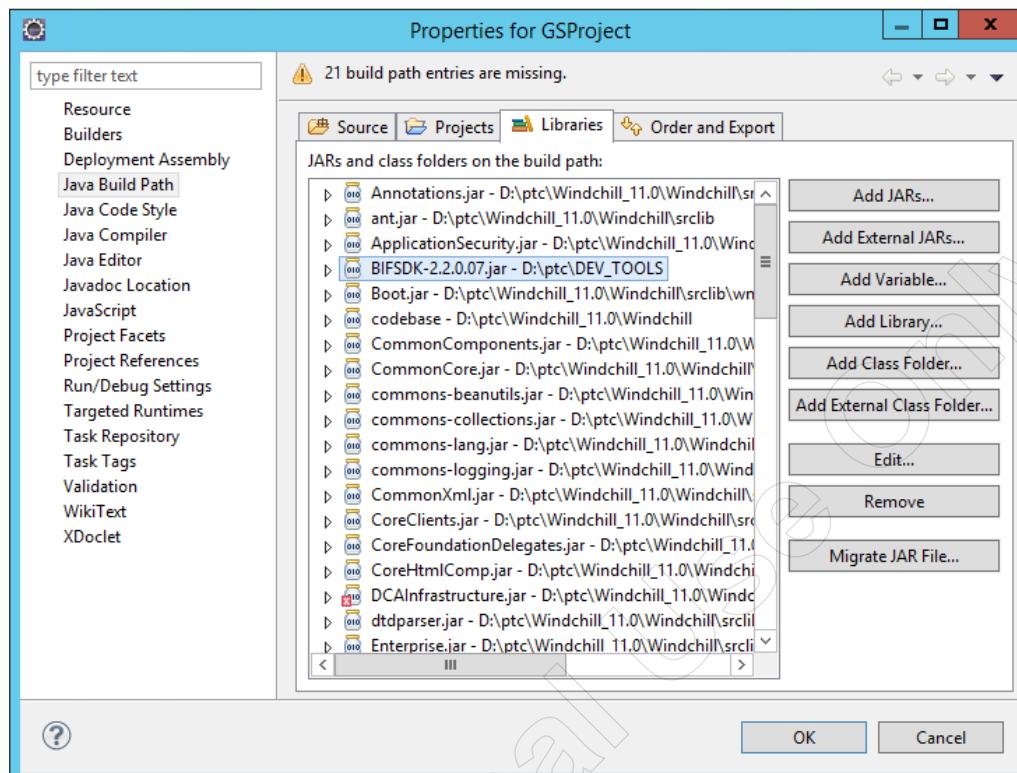
If you leave this option as “checked” then when you are building Windchill project then eclipse will DELETE your “codebase” directory, so better “uncheck” this option.

Task 6: Configure Eclipse to execute BIF SDK

1. Add BIF SDK to the project's Build Path by selecting **Project > Properties**

Within the Properties Window,

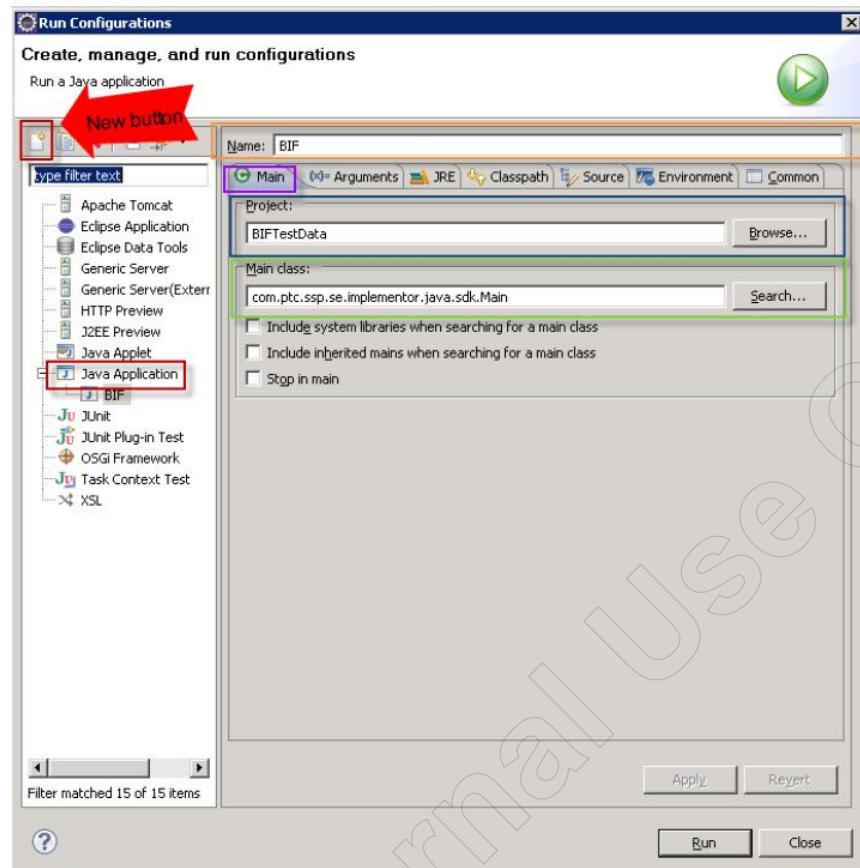
- Select **Java Build Path**
- Select the **Libraries** tab
- Select the **Add External JARs...** button
- Select the BIFSDK-2.2.0.07.jar from the global location and click **OK**
- Click **OK**



Please note that image shown above is for example purpose only, path of the file may vary as per installation.

2. Configure your project to run with the BIF by selecting **Run > Run Configurations**

- Highlight **Java Application** and click the **New** button
- Enter **BIF** for the **Name** in the **Main** tab
 1. Enter a name in the **Project** field that the BIF will execute against.
 2. In the **Main class** field, enter `com.ptc.ssp.se.implmentor.java.sdk.Main`



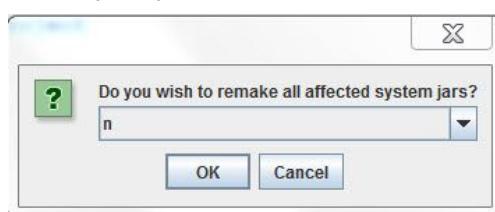
- In the **Environment** tab, create the following properties
 - ANT_HOME** with the value of <WT_HOME>/ant
 - eclipse.pdebuild.home** with the value of true.
 - JAVA_HOME** with the value of Windchill's version of JAVA For example, D:\ptc\Windchill11.0\Java
 - WT_HOME** with the value of <WT_HOME>.
- Click **Run** to execute the configuration or **Close** to save the configuration for a later date.



To execute the BIF from outside the Configuration window, click the **Run** button from the toolbar.



- During the execution, the BIF will prompt the user which action to run.



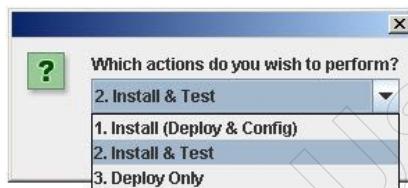


For **First Time Execution of Component**, following actions are listed. Select **Install & Test**.

Which actions do you wish to perform:

1. 1. Install (Deploy & Config)
2. Install & Test
3. Deploy Only

Enter valid response [1-3]:



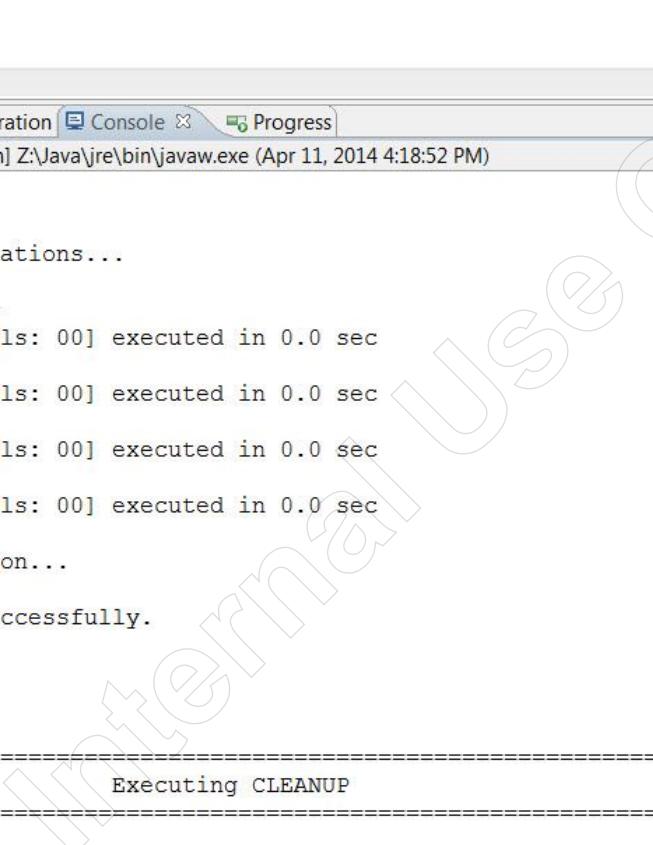
For **Subsequent Executions**, following actions are listed. You can select 'Test Only' action from it or any other action as per your requirement:

Which actions do you wish to perform:

1. Install (Deploy & Config)
2. Install & Test
3. Deploy Only
4. Config Only
5. Config & Test
6. Test Only

Enter valid response [1-6]:

4. In Console tab, you will see the BUILD SUCCESSFUL message.



MyClass.java

```

package com.ptc;
import wt.part.*;
public class MyClass {
}

```

Problems @ Javadoc Declaration Console Progress

<terminated> BIF [Java Application] Z:\Java\jre\bin\javaw.exe (Apr 11, 2014 4:18:52 PM)

-vf:

Running Component Validations...

AlterPropertiesVerifier [Passes: 00 Fails: 00] executed in 0.0 sec

WtSafeAreaVerifier [Passes: 00 Fails: 00] executed in 0.0 sec

XConfVerifier [Passes: 00 Fails: 00] executed in 0.0 sec

XMLTaskInsertVerifier [Passes: 00 Fails: 00] executed in 0.0 sec

Running Object Validation...

Validation completed successfully.

testing:

=====

Executing CLEANUP

=====

Deleting temporary BIF directory...

BUILD SUCCESSFUL

Total time: 1 minutes 42 seconds

5. A jar file with project name will be created under %WT_HOME%\codebase\WEB-INF\lib

In eclipse, If you face following error:

"The type java.lang.Object cannot be resolved. It is indirectly referenced from required .class files"

Then execute following steps:

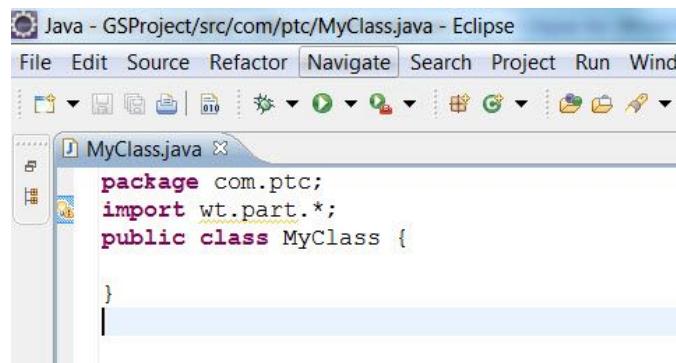
1. Right-click on project » Properties » Java Build Path
2. Select Libraries tab
3. Find the JRE System Library and remove it.
4. Click "Add Library" button at right side and Add the JRE System Library (Workspace default JRE).

Completion Criteria

1. The students must have created a new class in eclipse. Enter the code:

`import wt.part.WTPart;`

There is no error pointed out by eclipse.



```
Java - GSPProject/src/com/ptc/MyClass.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window
MyClass.java
package com.ptc;
import wt.part.*;
public class MyClass {
}
```



You can add some more code here which you want to execute.

This completes the exercise.

Exercise 3: Add First- and Second-Level Navigation Tabs

Objectives

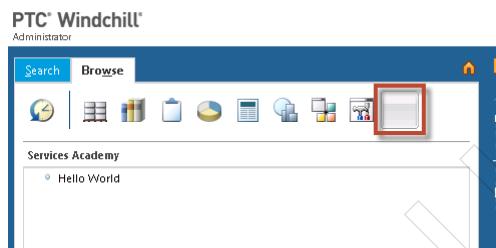
After successfully completing this exercise, you will be able to:

- Incorporate a custom JSP into the Windchill UI.
- Modify the Tab Model.
- Add Actions to the Action Framework.

Scenario

In this exercise, you will create a custom tab and sub-tab to display a JSP.

Result of Exercise screenshots shown here is the expected output for the exercise.



Result of Exercise



This is based on best practices delivered with the product:

- [Client_JCA_Overview_ActionFramework.doc](#)
- [Client_CC_Cust_Navigation_HowtoaddTabsandSubtabs.doc](#)



The above documents are only available to download for PTC internal access, partners likely will not be able to access it.

Task 1: Create the JSP.

1. In Eclipse, create a folder named **serviceAcademy** under the directory `codebase\netmarkets\jsp\`
2. In the folder **serviceAcademy**, create a new file names: **navigationlist.jsp**

Example:

```
<%@page language="java" session="true" pageEncoding="utf-8"%>
<%@include file="/netmarkets/jsp/util/begin.jspf" %>

Hello World!

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

navigationlist.jsp



Refer Lab files for navigationlist.jsp code.

Task 2: Create an icon file with the dimension: 24 X 24, named as **serviceAcademy.gif** and save it in your network drive, in the directory **codebase\netmarkets\images**.



All of the icons and the images are saved in the directory **%WT_HOME%\codebase\ netmarkets\images**.

Task 3: Create and Deploy Resource Entries.

1. In Eclipse, under the folder src, create a package named: **com.ptc.serviceAcademy.resource**.
2. In the package **com.ptc.serviceAcademy.resource**, create two files for the Resource Bundles. The two files are named as:
 - NavigationRB.java
 - NavigationRB_en_US.java



The file **NavigationRB_en_US.java** is used for Localization. ***_en_US.java** is for English, ***_zh_CN** is for Chinese, and so forth

Fill code in the two .java files, it looks like the following:

Example:

```
package com.ptc.serviceAcademy.resource;
import wt.util.resource.RBEntry;
import wt.util.resource.RBUUID;
import wt.util.resource.WTListResourceBundle;
@RBUUID("com.ptc.serviceAcademy.resource.NavigationRB")
public final class NavigationRB extends WTListResourceBundle{
  @RBEntry("Service Academy")
  public static final String NAVIGATION_SERVICEACADEMY_DESCRIPTION = "navigation.serviceAcademy.description";
  @RBEntry("Service Academy")
  public static final String NAVIGATION_SERVICEACADEMY_TITLE = "navigation.serviceAcademy.title";
  @RBEntry("Service Academy")
  public static final String NAVIGATION_SERVICEACADEMY_TOOLTIP = "navigation.serviceAcademy.tooltip";
  @RBEntry("serviceAcademy.gif")
  public static final String NAVIGATION_SERVICEACADEMY_ICON = "navigation.serviceAcademy.icon";

  @RBEntry("Hello World")
  public static final String OBJECT_HELLOWORLD_DESCRIPTION = "object.helloWorld.description";
  @RBEntry("Hello World")
  public static final String OBJECT_HELLOWORLD_TITLE = "object.helloWorld.title";
  @RBEntry("Hello World")
  public static final String OBJECT_HELLOWORLD_TOOLTIP = "object.helloWorld.tooltip";
}
```

NavigationRB.java

```

package com.ptc.serviceAcademy.resource;

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

@RBUUID("com.ptc.serviceAcademy.resource.NavigationRB_en_US")
public final class NavigationRB_en_US extends WTListResourceBundle{
    @RBEntry("Service Academy")
    public static final String NAVIGATION_SERVICEACADEMY_DESCRIPTION = "navigation.serviceAcademy.description";
    @RBEntry("Service Academy")
    public static final String NAVIGATION_SERVICEACADEMY_TITLE = "navigation.serviceAcademy.title";
    @RBEntry("Service Academy")
    public static final String NAVIGATION_SERVICEACADEMY_TOOLTIP = "navigation.serviceAcademy.tooltip";
    @RBEntry("ServiceAcademy.gif")
    public static final String NAVIGATION_SERVICEACADEMY_ICON = "navigation.serviceAcademy.icon";

    @RBEntry("Hello World")
    public static final String OBJECT_HELLOWORLD_DESCRIPTION = "object.helloWorld.description";
    @RBEntry("Hello World")
    public static final String OBJECT_HELLOWORLD_TITLE = "object.helloWorld.title";
    @RBEntry("Hello World")
    public static final String OBJECT_HELLOWORLD_TOOLTIP = "object.helloWorld.tooltip";
}

}

```

NavigationRB_en_US.java



Refer Lab Files for NavigationRB and NavigationRB_en_US java files.

If you face following issue:

"set workspace compiler compliance settings to '1.5' eclipse"

It can be resolved as follows:

- Go to Project properties -> 'Java Compiler' -> Check the box ('Enable project specific settings')
- Change the compiler compliance level to '5.0' or '1.5' & click ok.
- Do rebuild.

It will be resolved.

Task 4: Create the Action.

1. In Eclipse, under the folder `codebase\config\actions\`, update the file `custom-actions.xml`

Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE listofactions SYSTEM "actions.dtd">
<listofactions>
    <objecttype name="navigation" class="" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
        <action name="serviceAcademy">
            </action>
    </objecttype>

    <objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
        <action name="helloWorld">
            <command url="netmarkets/jsp/serviceAcademy/navigationlist.jsp" windowType="page"/>
        </action>
    </objecttype>
</listofactions>

```

Action Definitions in custom-actions.xml



Refer Lab Files for custom-actions.xml

Task 5: Update the Action Model.

1. In Eclipse, under the folder `codebase\config\actions\`, update the file `custom-actionModels.xml`

Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE actionmodels SYSTEM "actionmodels.dtd">
<actionmodels>
  <model name="main navigation" id="browseActions" resourceBundle="com.ptc.core.ui.navigationRB">
    <description>
      Main navigation (tabs)
    </description>
    <action name="recentContexts" type="navigation"/>
    <action name="product" type="navigation"/>
    <action name="library" type="navigation"/>
    <action name="project" type="navigation"/>
    <action name="program" type="navigation"/>
    <action name="change" type="navigation"/>
    <action name="supplier" type="navigation"/>
    <action name="qms" type="navigation"/>
    <action name="org" type="navigation"/>
    <action name="site" type="navigation"/>
    <action name="search" type="navigation"/>
    <!-- entry for customization tab -->
    <action name="customization" type="navigation"/>
    <!-- entry for serviceAcademy tab-->
    <action name="serviceAcademy" type="navigation"/>
  </model>
  <model name="serviceAcademy navigation" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld" type="object"/>
  </model>
</actionmodels>

```

Navigation Model in custom-actionModels.xml



Refer Lab Files for custom-actionModels.xml

Task 6: Deployment.

1. Use lightBuild to deploy the component.
2. Restart the Method Server.
3. Check the results.

Completion Criteria

1. The first level Tab and the second level Tab are displayed in the Navigator. When the second Tab is clicked, a page displayed on the right.

This completes the exercise.

For PTC Internal Use Only

Module 3

Information Pages

Module Overview

In this module, we will review the new Information Pages in the content. After a review of the basic layout, and a comparison to earlier versions of Windchill, we will review the structure of the pages and identify how to create custom pages.

Objectives

After completing this module, you will be able to:

- Relate existing Information Pages to those in previous release.
- Review the basic layout of the new Information Page.
- Describe the code behind the information page.
- Customize the Info Page.

Windchill 9.1 Information Page

Information Page in Windchill 9.1

The screenshot shows the Windchill 9.1 Information Page for a document named "DiagramForJCAOverview.pptx".

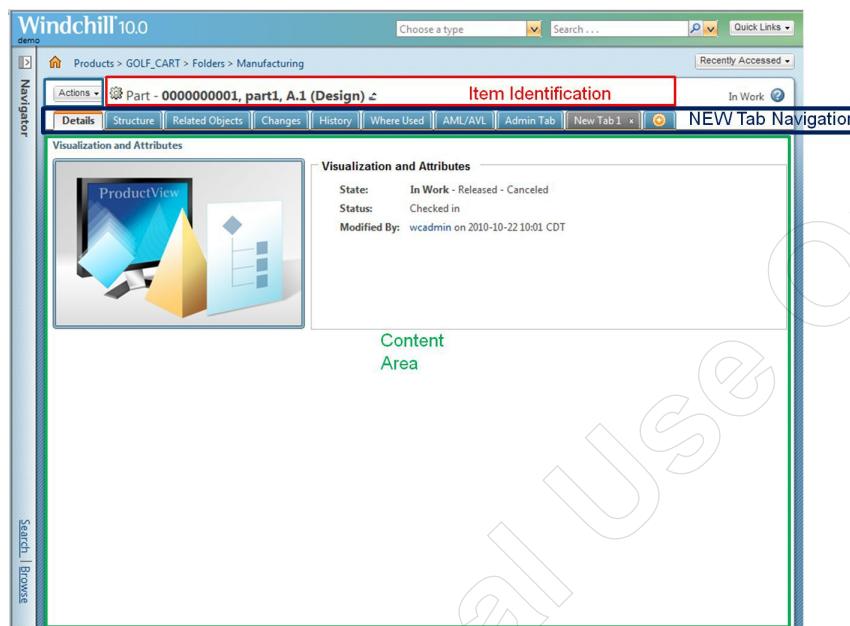
- Item Identification:** Shows basic information like Name, Primary Content, Latest Iteration, Status, Modified By, Format, Location, and More Attributes.
- Top Attributes or Primary Attributes:** Shows attributes such as State: In Work - Completed - Canceled, Status: Checked in, and Last Modified.
- Visualization:** A preview window showing a blank white page.
- Third Level Navigation:** A table showing the iteration history for the document, listing one object (A.1) with details like File Name, Size, Predecessor, State, Comment, Modified By, and Last Modified.

Key Areas of the Information Page

- Item Identification
- Actions
- Top Attributes or Primary Attributes
- Visualization
- Third level navigation area – consists of two pieces
 - Menu for navigation
 - Content area
 - ◆ Attributes, Tables, and other kinds of content.
 - ◆ Controlled by the action model

Windchill 10.2 Information Page

Information Page in Windchill 10.2



Key areas of the information page

- Item Identification
- Actions
- Tab navigation
 - Replaces third level navigation menu
- Content Area
- Attributes and Visualization are just content
 - Like any other content
 - Not always visible on the page — generally available on the details tab.



This remains same in Windchill 11.0 also.

Exercise 1: Modify the Customize Menu for Part

Objectives

After successfully completing this exercise, you will be able to:

- Modify the action menu on a part
- Augment an Information Page with additional actions.

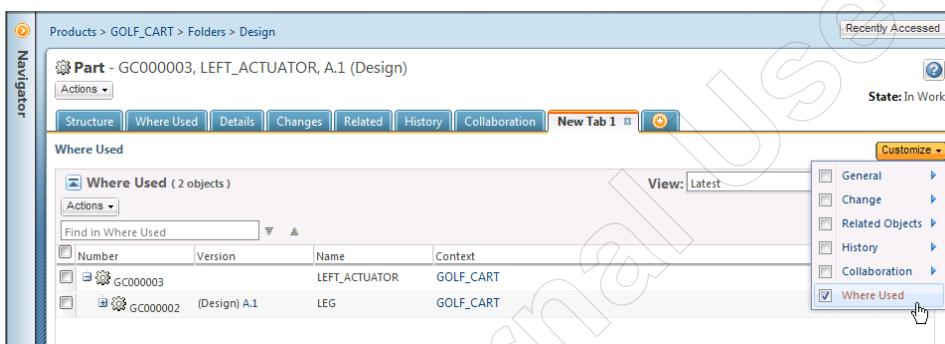
Prerequisites

Display the **Customization** tab in Windchill

Navigate to **Site > Utilities > Preference Management**. In the table, find the **Client Customization**, and change the value to Yes.

Scenario

In this exercise, we show how you can add an already registered action to the Information Page of a part.



Modified Third-Level Navigation

Task 1: Determine the action for the “Customize” menu by enabling jcaDebug=true

1. Open Windchill in the browser, browse to the product folder, click on one of the parts and then go to the Part information Pages.

There might pop up plug-in of “pvvercheck_ie.cab”; it will ask for installation. If it pops up, do install it.
2. In the end of the URL, add `&jcaDebug=1`
click **Enter** in order to refresh the page
3. Clicking the ladybug tab reveals the actionmodel file used – looking for “third_level” in that file will reveal clues.

Example:



Ladybug tab

Task 2: Register a custom Part Menu.

1. Open `%WT_HOME%\codebase\config\actions\PartClient-actionmodels.xml`

 “PartClient-actionmodels.xml” file is located in virtual machine. So open it in VM only and then update the “custom-actionModels.xml” in eclipse. Or you can directly copy it from Lab Files provided.

2. Search for

```
<model name="third_level_nav_part">
```

Result:

```
<!-- **** Information Page 3rd Level Navigation Menu Bar *** -->
<!-- Part information page 3rd level nav menu bar -->
<model name="third_level_nav_part">
  <submodel name="general"/>                               <!-- General -->
  <submodel name="relatedItems"/>                         <!-- Related Objects -->
  <submodel name="changes"/>                            <!-- Change -->
  <submodel name="history"/>                           <!-- History -->
  <submodel name="collaboration"/>                      <!-- Collaboration -->
  <submodel name="prodAnalytics"/>                     <!-- Product Analytics -->
</model>
```

Third-Level Navigation for Parts

3. Copy the model to the `codebase\config\actions\custom-actionModels.xml`

Task 3: Modify the custom part menu.

- Add an entry referencing the “whereUsed” action in modified `codebase\config\actions\custom-actionModels.xml`

Result:

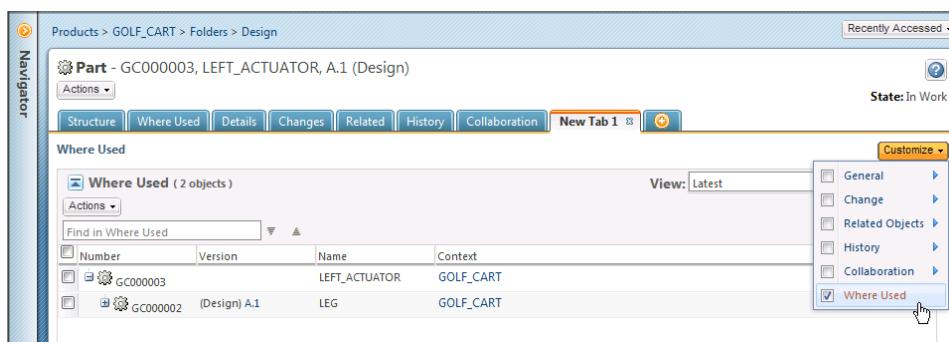
```
<!-- **** Information Page 3rd Level Navigation Menu Bar *** -->
<!-- Part information page 3rd level nav menu bar -->
<model name="third_level_nav_part">
  <submodel name="general"/>                               <!-- General -->
  <submodel name="relatedItems"/>                         <!-- Related Objects -->
  <submodel name="changes"/>                            <!-- Change -->
  <submodel name="history"/>                           <!-- History -->
  <submodel name="collaboration"/>                      <!-- Collaboration -->
  <submodel name="prodAnalytics"/>                     <!-- Product Analytics -->
  <action name="whereUsed" type="object"/>                <!--Customize the menu bar -->
</model>
```

 Refer Lab Files for `custom-actionModels.xml`

Task 4: Test it out.

- Use lightbuild to deploy the component.
- Restart the method server.
- Navigate to the **Customization >Tools**, click on **Reload Action(s)**.
- Create a part or open the existing part.
- Create a new Tab and review the Customize menu.

Result:



Modified Third-Level Navigation

- Notice the **Where Used** Action on the **Customize** menu.

Completion Criteria

1. The new Action **Where Used** is displayed in the **Customize** action model.

This completes the exercise.

For PTC Internal Use Only

Carambola InfoPage Example Overview

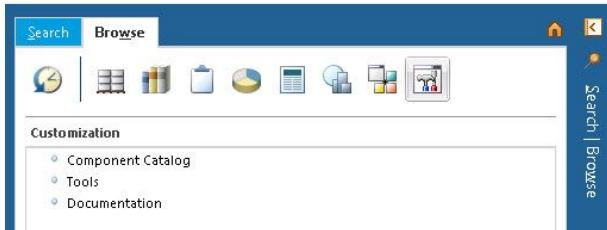
Carambola Information Page Serves as a Useful Template

The customization tab reveals an Example Information Page

- Clicking the Example reveals the Carambola InfoPage.

PTC® Windchill®

Administrator



Entry to See the Carambola Examples



InfoPage Example on Customization Tab

This screenshot shows the detailed view of the Carambola InfoPage for a star fruit. It features a large image of the fruit, its name ('Star Fruit'), a brief description ('Star fruit is one of the suj'), and a 'Health Benefits' section. Below this, a 'Part Table' is shown with three entries:

Name	Number	Version	Last Modified
Kit, Energy Delivery, S724, w/U...	D3_0000000128	A.1 (Design)	2011-04-08 13:15:00
Control Card, Thermal Manag...	D3_0000000156	A.1 (Design)	2011-04-08 13:15:00
PCB Assembly, Power Generati...	D3_0000000174	A.1 (Design)	2011-04-08 13:15:00

Carambola InfoPage

Finding the Builder

How to Find the Builder?

Search for

```
com.ptc.carambola.customization.  
examples.infoPage.Carambola.*  
to reveal:
```

- Class Name for the Builder
- Direct link to OpenGrok source code (<http://ah-opengrok.ptcnet.ptc.com/>)



OpenGrok is used to find the JAVA source code. It is currently only supported for the internal access, partners likely will not be able to access it.

The screenshot shows the 'Tools' section of the Windchill interface. The 'Tools' section lists various tools with their descriptions and links:

- Action
- Action Model
- ApplicationContext Service/Resource Properties
- Available Attributes
- Logical Attributes Report
- Property Report
- Reload Action(s)
- MVC Builder Search**
- MVC Builder Scan Report**
- MVC Builder Scan Report - Mini**
- MVC Builder Scan Report - Full**
- Modeled Objects

Starting a Builder Search

The screenshot shows the 'MVC Builder Report' search interface. It includes a search bar for 'ComponentConfigBuilder' and 'ComponentDataBuilder', a 'Find Builders' button, and a table with two results:

Bean Name	Class Name	Grok
com.ptc.carambola.customization.examples.infoPage.Carambola.infoPage	com.ptc.carambola.customization.examples.mvc.CarambolaInfoBuilder	
com.ptc.carambola.customization.examples.infoPage.Carambola.visualizationAndAttributes	com.ptc.carambola.customization.examples.mvc.CarambolaVizAndAttrBuilder	

Builder Report

Finding the Builder Is Not Always Obvious

In this case, the package naming was similar. But this is not always the case.

- jcaDebug will usually reveal the builder.

Carambola Builder

The Carambola Builder

The builder is very simple

- Few lines of code
- extends AbstractInfoConfigBuilder and implements ComponentDataBuilder
- Common commands include:
 - newInfoConfig
 - setNavBarName
 - setActionListName
 - setHelpContext
 - addComponent
 - setView
 - setType

```
CarambolaInfoConfig.java (2)
16 /* (boilerplate)
27 package com.ptc.mvc.builders.carambola;
38
49 import vt.util.VTEexception;
510
611 //TypefaceConstant = "com_ptc_carambola_customization_exemple.infoPage.Carambola"
712 ComponentIdInfoBuilder
813 public class CarambolaInfoBuilder
914     extends AbstractInfoConfigBuilder
1015     implements ComponentDataBuilder {
1116
1217     private static final String PART_RESOURCE = "com_ptc_windchill.enterprise.part.partResource";
1318     ClientMessageSource part_source = getMessageBoxSource(PART_RESOURCE);
1419
1520     private PropertyConfig newStatusesGlyph(String id) {
1621         PropertyConfig glyphs = getComponentConfigFactory().newPropertyConfig(id);
1722         glyphs.setStatus0(glyph(true));
1823         //注意:在Carambola类上使用了ClientMessageSource而不是真正的数据实用程序
1924         glyphs.setStatus1("defaultSecurity");
2025         return glyphs;
2126     }
2227
2328     @Override
2429     public InfoConfig buildInfoConfig(ComponentParams params) throws VTEexception {
2530
2631         InfoComponentConfigFactory factory = GetComponentConfigFactory();
2732         InfoConfig result = factory.newInfoConfig();
2833
2934         result.setNavBarName("Custom_infoPage_third_level_navigation");
3035         result.setActionBarName("Custom_info_page");
3136         result.setHelpTopic("Custom_info_page");
3237         result.setHelpContext("CarambolaInfoTopic");
3338
3439         result.addComponent(newStatusesGlyph("statusFamily_Share"));
3540         result.addComponent(newStatusesGlyph("statusFamily_General"));
3641         result.addComponent(newStatusesGlyph("statusFamily_Change"));
3742         result.setView("carambola/carambolaInfoPage.jsp");
3843         result.setType(Carambola.class.getName());
3944
4045         return result;
4146     }
4247
4348     @Override
4449     public Object buildComponentData(ComponentConfig config, ComponentParams params) throws Except
4550         ion {
4651             return new Carambola();
4752         }
4853     }
```

Simple Builder



Main point is to show how simple it can be, the code can fit on a single slide. Next slides go into the detail.

Carambola Component Builder

Basic Elements of a Component Builder

```

12 package com.ptc.mvc.builders.carambola; 1
13
14 import wt.util.WTEException;
15
16 import com.ptc.carambola.customization.examples.infoPage.Carambola;
17 import com.ptc.mvc.components.AbstractInfoConfigBuilder;
18 import com.ptc.mvc.components.ComponentBuilder;
19 import com.ptc.mvc.components.ComponentConfig;
20 import com.ptc.mvc.components.ComponentDataBuilder;
21 import com.ptc.mvc.components.ComponentId;
22 import com.ptc.mvc.components.ComponentParams;
23 import com.ptc.mvc.components.InfoComponentConfigFactory;
24 import com.ptc.mvc.components.InfoConfig;
25 import com.ptc.mvc.components.PropertyConfig;
26 import com.ptc.mvc.components.TypeBased;
27 import com.ptc.mvc.util.ClientMessageSource;
28
29 2 @TypeBased(value = "com.ptc.carambola.customization.examples.infoPage.Carambola")
30 @ComponentBuilder(value = ComponentId.INFOPAGE_ID) 3
31 4 public class CarambolaInfoBuilder extends AbstractInfoConfigBuilder implements ComponentDataBuilder { 5
32
33     private static final String PART_RESOURCE = "com.ptc.windchill.enterprise.part.partResource";
34
35     ClientMessageSource part_source = getMessageSource(PART_RESOURCE);
36
37     6     private PropertyConfig newStatusGlyph(String id) {
38
39         @Override
40         public InfoConfig buildInfoConfig(ComponentParams params) throws WTEException {
41
42             InfoComponentConfigFactory factory = getComponentConfigFactory();
43
44             InfoConfig result = factory.newInfoConfig();
45
46             return result;
47         }
48     }
49 }
```

1. Where is the example?
2. Java annotation to define the business object type this builder applies to
3. Java annotation that indicates that this is an information page
4. Java Annotation indicating this file is a ComponentBuilder.
5. Information page builders all extend AbstractInfo – ConfigBuilder
6. The abstract class defines one method, start with an empty config from the factory

Carambola Component Config Builder

Component Config of the Carambola Builder

```

@Override
public InfoConfig buildInfoConfig(ComponentParams params) throws WTEException {
    InfoComponentConfigFactory factory = getComponentConfigFactory();
    InfoConfig result = factory.newInfoConfig();
    result.setNavBarName("CustEx_infoPage_third_level_navigation");
    1 result.setTabSet("CustEx_infoPage_tabsSet");
    2 result.setHelpContext("CarambolaHelpTopic");
    3 result.setHelpContext("CarambolaHelpTopic");

    result.addComponent(newStatusGlyph("statusfamily_Share"));
    4 result.addComponent(newStatusGlyph("statusfamily_General"));
    result.addComponent(newStatusGlyph("statusfamily_Change"));
    result.addComponent(newStatusGlyph("statusfamily_Security"));
    5 result.setURI("/carambola/carambolaInfoPage.jsp");
    result.setType(Carambola.class.getName());

    return result;
}

@Override
public Object buildComponentData(ComponentConfig config, ComponentParams params) throws Exception {
    return new Carambola();
}

```

1. Define the menu for the Action Button
2. Define the navigation tabs
3. Help
4. Additional components can be added, like glyphs
5. Reference the JSP file that will govern the placement of objects on the page.

Each of the Subcomponents Are Also Defined by Actions

Use jcaDebug to reveal the page configuration.

The screenshot shows the PtcDebug interface with the URL <http://x20preprod.ptc/Windchill/app/jcaDebug=true?ptc1/tcomp/infoPage?typeIdForTypeLookup=com.ptc.carambola.infoPage>. The title bar says "Testing Windchill". The main content area is titled "About This Example" and shows details about the "CustEx_infoPage_tabsSet" action model. It lists the context object, navigator delegate, component ID, and component builder. Below this, it shows the "Action Model Details" section with fields for "Action Model Name" (CustEx_infoPage_tabsSet), "Dev Owner", "Description", "Overridden By", "Definition File" (/config/actions/Carambola.actionModels.xml), and "Menu For". Under "Actions In Model:", there is a table with four rows:

Order	Icon	Label	Name	Type	View Info
1		Details	carambola_details	object	View Info
2		My Tab	CustEx_default_myTab	object	View Info
3		Error Page With Generic Exception	CustEx_error_on_page	object	View Info
4		404 error	CustEx_error_404	object	View Info

(0 objects selected)

- Page Layout is revealed to demonstrate which action models are in use.
- Information can be correlated to the ComponentBuilder to show which pieces of the builder are responsible for displaying the page layout.

For PTC Internal Use Only

Module 4

MVC

Module Overview

In this module, we review the different types of components, how MVC components are defined, and how new components can be created.

Objectives

After completing this module, you will be able to:

- Demonstrate existing components.
- Explain how MVC components are defined.
- Create a new MVC component.

Windchill Client Architecture

Two Approaches to the Windchill Client Architecture

- An MVC-based architecture
 - MVC = Model, View, Controller
- Two customization paths:
 - JSP approach: Legacy Windchill 9.1, Supported in Windchill 10 and later releases till Windchill 11.0 also.
 - Java Builder approach: A new feature in Windchill 10 and continued till later releases also.

Two Customization Paths

- JSP approach
 - This approach is the same as in Windchill 9.1, which comprises of JSP pages referencing Java beans and tag libraries specifically created for the Windchill API..
 - This is a “Model 1” approach, with the JSP page serving as two components in the architecture
 - View and Controller.
- Java Builder approach
 - This approach is new to Windchill 10, comprised of SpringMVC beans (Java classes) configured to URL paths.
 - This is a “Model 2” approach, with the three architectures represented by separate components (View=JSP page, Controller=Component controller (inside Spring), Model=Component config + data).

MVC

Model–View–Controller

Architectural pattern used in software engineering.

Model

- Enterprise data and the business rules that govern access and updates to this data.

View

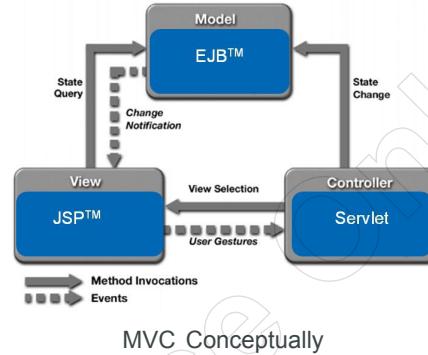
- Renders the model

Controller

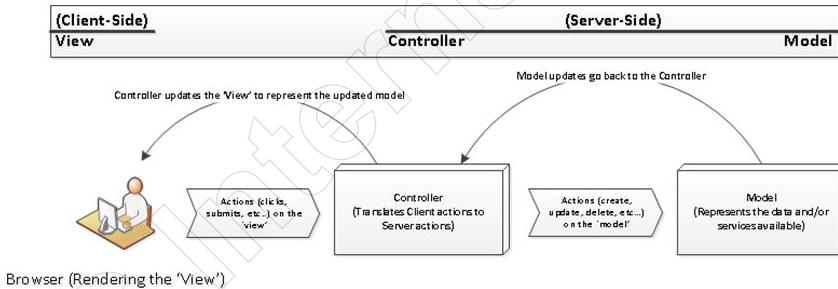
- Translates interactions with the view into actions to be performed by the model.

For Web-based clients:

- JavaServer Pages (JSP) pages render the view.
- A Servlet acts as the controller.
- Enterprise JavaBeans (EJB) represent components as the model.



High-Level Overview of Components

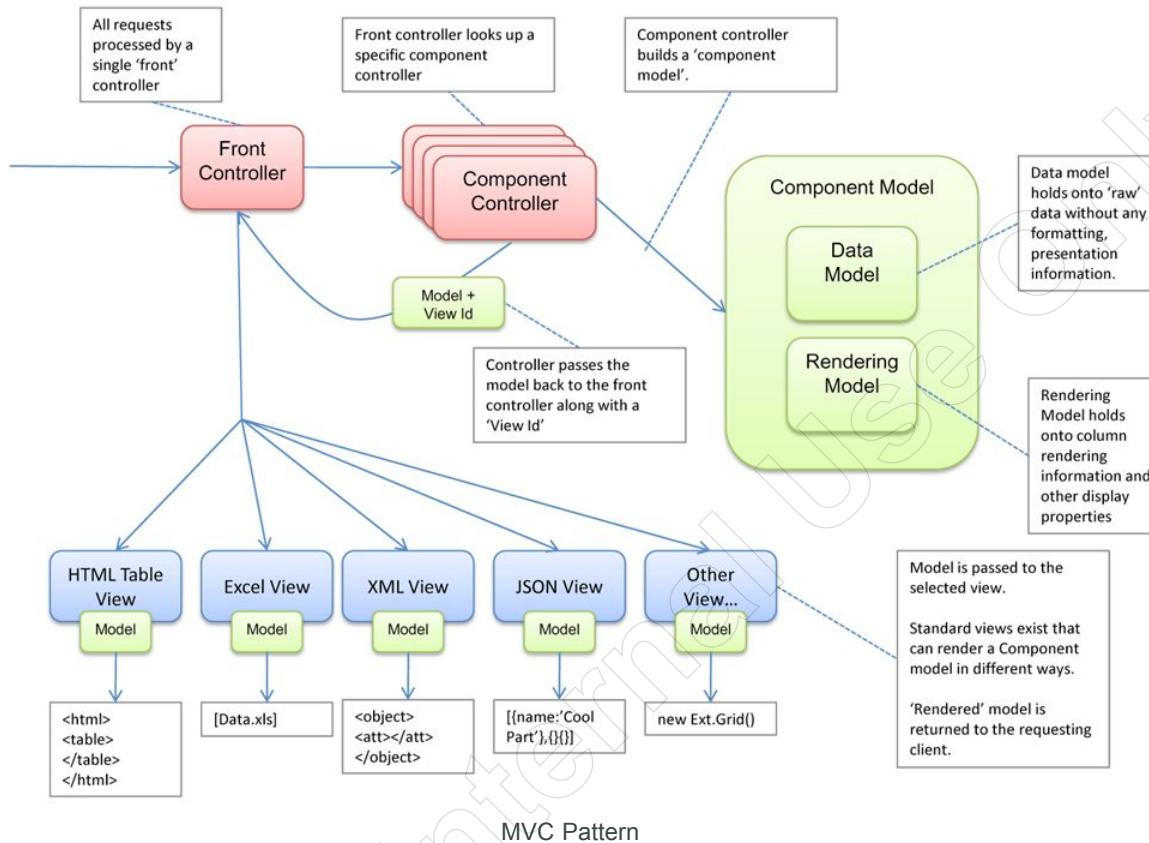


Disassembly of the UI paradigm simplifies development of complex user interfaces.

- Enables the division of development work based on Skill Area.
- Enables for the use of third-party frameworks.
- Promotes reuse of components.

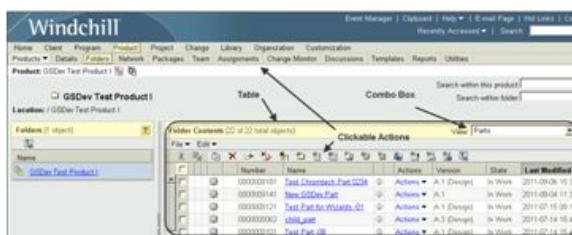
MVC Pattern

MVC Pattern – The Big Picture

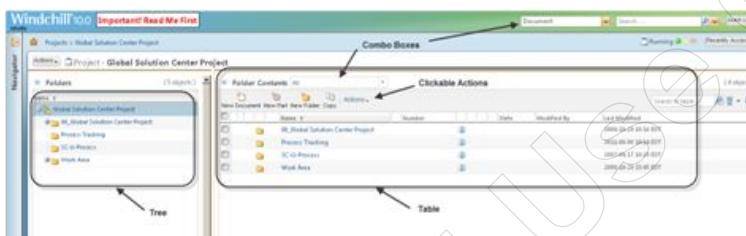


MVC — View

Examples of the Windchill Client Side



Windchill 9.1



Windchill 10



This remains same in Windchill 11.0 also

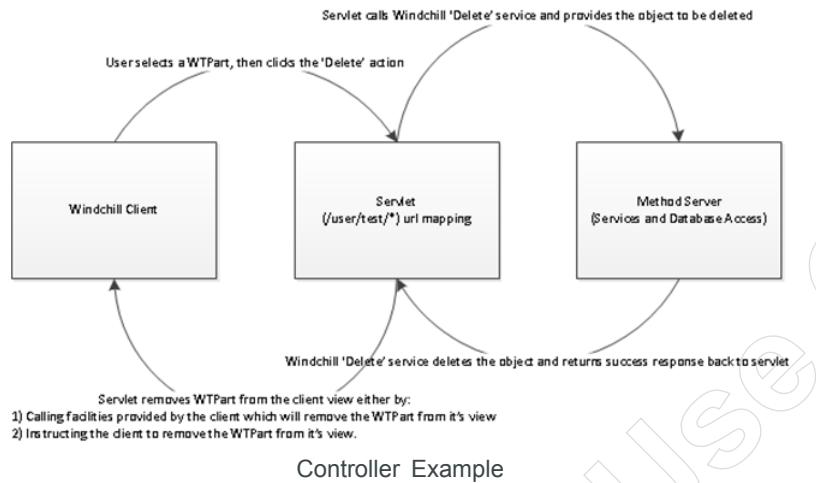
In More Detail

View

- This is the part of the application that is displayed to the user.
- Typically this includes user interface components such as input fields, property panels, layouts, tables, trees, URL links, combo boxes, and many other components we use everyday when interacting with Web sites and Web applications.
- In terms of client/server – the user interacts with the “View” components, initiating requests that are sent over the wire to the server side of the application. The view also provides facilities that enable it to be updated, either by itself or the server side of the application.

MVC — Controller

Example of a Controller



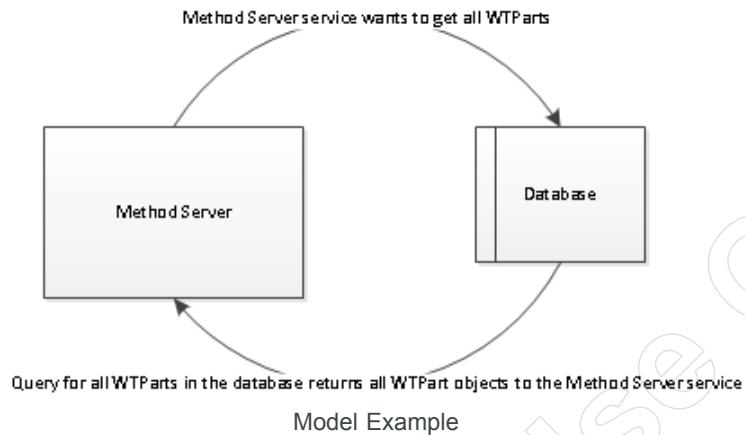
In More Detail

Controller

- The "Controller" here is implemented as a Java-based Servlet, but this has changed over time. The Controller can actually be implemented in a few different ways.
- The "Controller" is responsible for translating the actions between the two sides. How it achieves this depends on what facilities or services are available on the View and on the Model and how these correspond to one another.

MVC — Model

Example of a Model



- The WTPart and the service to get all WTParts comprise the “Model” portion of the architecture. Like the Controller portion, this portion has changed over time.

In More Detail

Model

- This is the part of the application that represents the data. This can and often does include the services available to interact with that data.
- Services called on the server side will manipulate or gather data, which in the case of Windchill will be housed in a database.
- Services called on the server side will inform the Controller of any updates made. The Controller then translates these back to updates that must be made on the view to reflect the updates.

Windchill Turns to Spring

Windchill Now Uses the Spring Framework

Benefits of Using Spring:

- Consistent way of managing business objects
- Encourages good practices such as programming to interfaces, rather than classes.
- Provides the MVC Web framework

Instead of having multiple servlets as controllers, a main Servlet is used to make control more manageable.

- Front Controller pattern
- Centralized controller

• `org.springframework.web.servlet.DispatcherServlet` acts as the Front controller.

In Windchill 9.x:

- JSP served both as the Controller and View

In Windchill 10.0,

- Separation between Controller and View



Same is continued in Windchill 11.0 also.

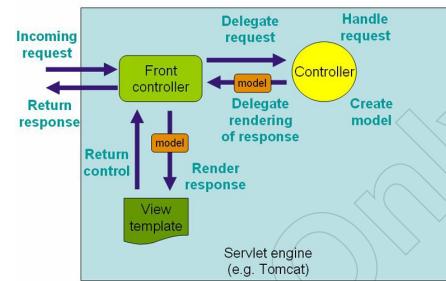
Other details can be found on the Web: <http://static.springsource.org/spring/docs/2.5.x/reference/index.html>

Spring MVC Overview

MVC Request Handling

The DispatcherServlet is an actual Servlet

- Inherits from the [HttpServletRequest](#) base class
- Declared in the *web.xml*



Request Handling in General

Requests that you want the [DispatcherServlet](#) to handle are mapped using a URL mapping.

- Standard J2EE servlet configuration

```

<servlet>
    <servlet-name>MVC Dispatcher Servlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>

```

Servlet Mapping

```

<servlet>
    <servlet-name>MVC Dispatcher Servlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>

```

URL Mapping for the Servlet

Spring MVC DispatcherServlet

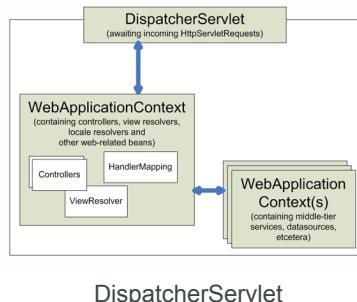
DispatcherServlet registers Handlers, Controllers, and Resolvers

The framework will look for a file named `[servlet-name]-servlet.xml` in the `WEB-INF` directory of your Web application and create the beans defined therein

- In Windchill, this file is `%WT_HOME%\codebase\WEB-INF\MVCDispatcher-servlet.xml`

The definition consists of:

- Controllers – Create, Update, Delete Objects
- Handler Mappings – Flow execution
- View Resolvers – Where to find JSP files and how to draw them.
- Theme Resolvers
- Locale Resolvers



A screenshot of a Windows Notepad++ window displaying the XML configuration file `MVCDispatcher-servlet.xml`. The code defines an application context for the MVC DispatcherServlet, including imports for beans, mvc, jca, and custom configurations, and a bean for default handler mappings.

```

<?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.xsd
                           http://www.springframework.org/schema/jca http://www.springframework.org/schema/jca/spring-jca.xsd
                           http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
                           http://www.springframework.org/schema/custom http://www.springframework.org/schema/custom/spring-custom.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.web.BeanFactoryConfig.PropertiesFactoryBean">
        <property name="locations">
            <list>
                <value>classpath:/config/mvc/*-urlMappings.properties</value>
                <value>classpath:/config/mvc/custom.properties</value>
            </list>
        </property>
    </bean>
</beans>
  
```

Windchill Spring Configuration File

Windchill Uses Multiple Configuration Files

`%WT_HOME%\codebase\WEB-INF\MVCDispatcher-servlet.xml` includes the following files:

- `%WT_HOME%\codebase\config\mvc\mvc.xml` – Base classes
- `%WT_HOME%\codebase\config\mvc\jca-mvc.xml` – JCA framework
- `%WT_HOME%\codebase\config\mvc*-configs.xml` – specific configurations for various product area
- `%WT_HOME%\codebase\config\mvc\custom.xml` – customization hook for implementations.

MVC Request Handling

MVC in Windchill

Spring handles a request in the following order

1. The Client requests for a **Resource** in the Web Application.
2. The Spring Front Controller intercepts the Request and then will try to find out the appropriate **Handler Mappings**
3. Handler Mappings are used to map a request from the Client to **Handler**.
4. **Handler Adapters** dispatch the Request to the Handler.

```

<?xml version="1.0" encoding="UTF-8"?>
<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.xsd
                           http://www.springframework.org/schema/context
                           http://www.springframework.org/schema/context/spring-context.xsd
                           http://www.springframework.org/schema/mvc
                           http://www.springframework.org/schema/mvc/spring-mvc.xsd
                           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.springframework.org/schema/annotation
                           http://www.springframework.org/schema/annotation/spring.annotation.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop.xsd
                           http://www.springframework.org/schema/tx
                           http://www.springframework.org/schema/tx/spring-tx.xsd
                           http://www.springframework.org/schema/web
                           http://www.springframework.org/schema/web/spring-web.xsd
                           http://www.springframework.org/schema/web-app
                           http://www.springframework.org/schema/web-app/spring-web-app.xsd
                           http://www.springframework.org/schema/context/spring-context-3.0.xsd>

<!-- Annotation Setup -->
<context:annotation-config />

<!-- Set up basic handler adapter -->
<bean id="simpleControllerHandlerAdapter" class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter" />

<!-- Allows RemoteService implementations to be resolved by Spring -->
<bean id="remoteHandlerAdapter" class="com.ptc.mer.ptc.RemoteHandlerAdapter" />

<!-- Local in this time we are not relative to /prod -->
<bean id="defaultHandlerMappings" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <!-- Property-based mappings should override the annotated mappings -->
    <property name="alwaysUseFullPath" value="true" />
    <property name="mappings" value="classpath:/META-INF/spring/handlerMappings.xml" />
</property>
<!-->
<bean id="defaultHandlerMappings" class="org.springframework.beans.factory.config.PropertiesFactoryBean" />
<!-->
<!-- Enables annotation-based handler mappings. Don't use the full path
     here, cause paths are relative to /prod -->
<-->

```

HandlerAdapter to **HandlerMapping** Example

Windchill MVC Components

ComponentBuilderResolver Identifies the Controllers That Build Components

The Handler responsible for finding JCA component builders is [ComponentBuilderResolver](#)

```

63      <!-- Default resolver -->
64      <bean id="defaultComponentBuilderResolver"
65          class="com.ptc.mvc.components.support.DefaultComponentBuilderResolver" />
66
67      <alias alias="configurableComponentBuilderResolver" name="defaultComponentBuilderResolver" />
68

```

[ComponentBuilderResolver](#) defined in *mvc.xml*

jca_mvc.xml defines Controllers which are used to create the model.

```

<bean id="componentHandleMappings"
    class="org.springframework.beans.factory.config.PropertiesFactoryBean">
    <property name="properties">
        <props>
            <prop key="/comp/**">componentController</prop>
            <prop key="/tcomp/**">typeBasedComponentController
            </prop>
        </props>
    </bean>

```

[Handler Mappings](#) in *jca_mvc.xml*

```

<bean id="typeBasedComponentController" class="com.ptc.mvc.components.ComponentController"
    parent="componentController">
    <property name="componentBuilderResolver" ref="typeBasedComponentBuilderResolver" />
</bean>

<!-- The ComponentBuilderResolver for a type-basedLook up -->
<bean id="typeBasedComponentBuilderResolver"
    class="com.ptc.jca.mvc.components.TypeBasedComponentBuilderResolver">
    <property name="beanNameFactory" ref="typeBasedBeanNameFactory" />
</bean>

```

[typeBasedComponentController](#)

```

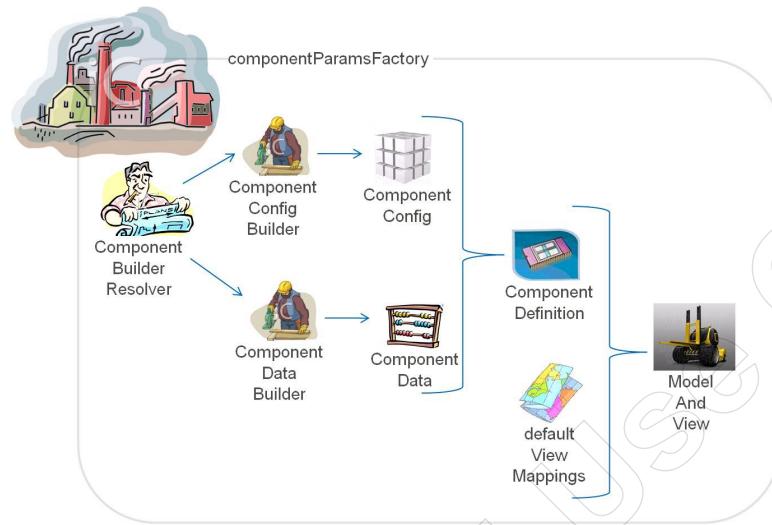
<!-- ComponentController Beans -->
<bean id="componentController" class="com.ptc.mvc.components.ComponentController">
    <property name="componentBuilderResolver" ref="jcaComponentBuilderResolver" />
    <property name="componentParamFactory" ref="componentParamFactory" />
    <bean class="com.ptc.jca.mvc.components.JcaComponentParamFactory" />
</property>
<property name="defaultViewMappings">
    <mapp...
        <entry key="com.ptc.mvc.components.TreeConfig" value="/components/tree.jsp" />
        <entry key="com.ptc.mvc.components.TableConfig" value="/components/table.jsp" />
        <entry key="com.ptc.mvc.components.AttributeTableConfig"
            value="/components/table.jsp" />
        <entry key="com.ptc.mvc.components.AttributePanelConfig"
            value="/components/attributePanel.jsp" />
    </mapp...
</property>
<property name="componentBuilderInterceptor" ref="componentBuilderInterceptor" />
</bean>

```

[componentController](#)

Building Components in a Factory

View the Spring Framework like a Factory creating ModelAndView Components



Spring Framework

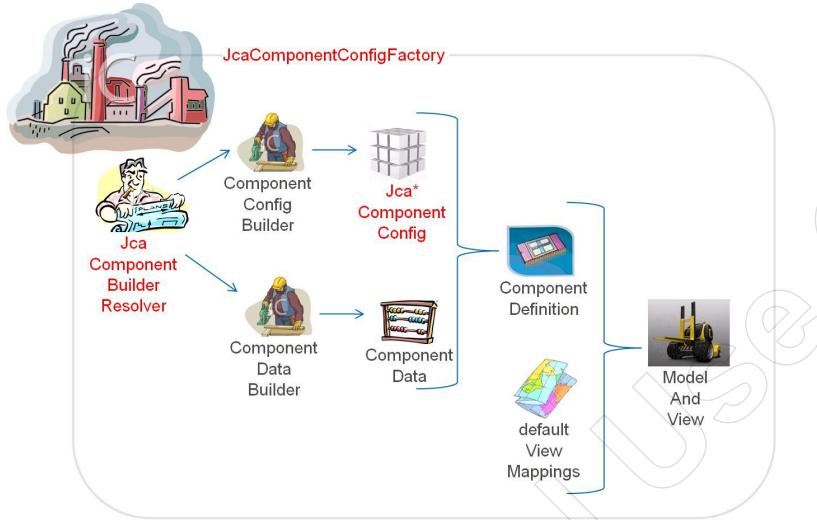
jca_mvc.xml defines Controllers which are used to create the model.

- **ComponentController** is a Handler provided OOTB to handle request to build MVC components.
 - Prepares a ModelAndView which will be handled by the DispatcherServlet
- ComponentDefinition
 - Model for **ComponentController**
 - Consist of **ComponentConfig** and **ComponentData**
- Build a component means Create a ComponentDefintion
- ComponentDefiniton is made with the help of Builders
 - **ComponentConfigBuilder**
 - **ComponentDataBuilder**
- How do we specify a component?
 - Component Id: ptc1/<controller_key>/<componentId>
- Who finds the Builders? **ComponentBuilderResolver**

Building JCA Components

JCA Components use the **JcaComponentBuilderResolver**

Identify these components with the use of comp/** in the URL



Different Java Beans Act on JCA Components

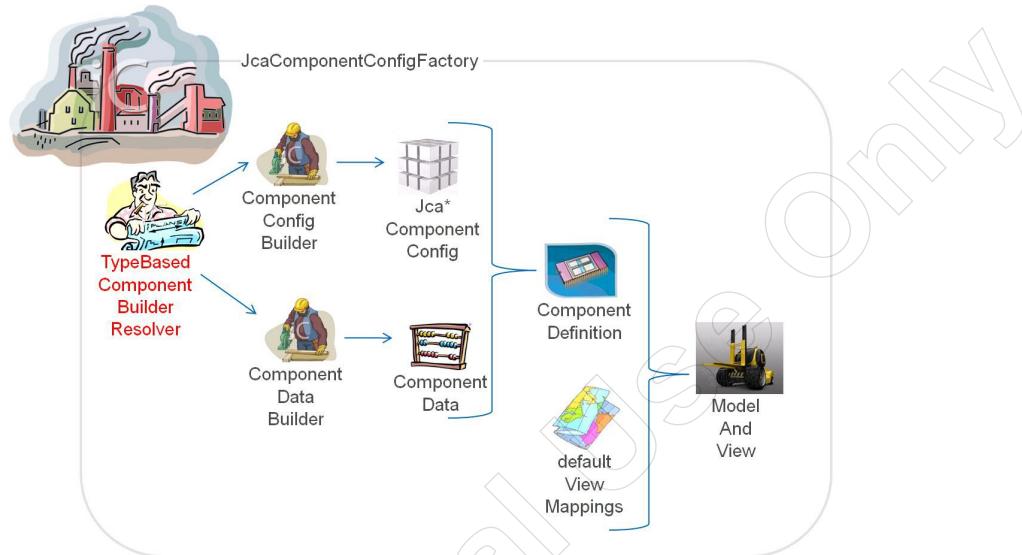
Windchill Spring knows how to use these beans based on mappings in the XML files.

- Design Pattern Remains the same
- Inheritance is from the same parent class

Building Type-Based Components

Type-Based JCA Components use the [TypeBasedComponentBuilderResolver](#)

Identify these components with the use of tcomp/** in the URL



Type-Based Components Expect Additional Parameters

Same Component Configs can be used to render these:

- There is a difference in the way the additional attributes are handled.

Registering Builders

Builders Registration Configuration

Once the builders are created, you need to let Spring infrastructure know about it. Registration can be done either via explicit configuration or by automated scanning. Automated scanning is the preferred approach for typical use cases.

- Automated scanning:
 - Configure to automatically pick up all builders within a certain package hierarchy
 - Add the configuration element to `\codebase\config\mvc\custom.xml`.

- Explicit configuration:
 - Configure to give a directly bean declaration for the builder
 - Add the configuration element to `\codebase\config\mvc\custom.xml`.

```
<?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-10.0.xsd">
    <!-- Configurations in this file override all other configurations -->
    <mvc:builder-scan base-package="com.ptc.serviceAcademy.builders"/>
</beans>
```

custom.xml

```
<?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-10.0.xsd">
    <!-- Configurations in this file override all other configurations -->
    <mvc:builder-scan base-package="com.ptc.serviceAcademy.builders"/>
    <bean class="com.ptc.servicesAcademy.builders.MyClassName"/>
</beans>
```

Builder Registration

- Automated scanning:
 - The builder-scan implementation scans the entire classpath, so you should take care to be specific with the package name you declare if using scanning. (Don't scan `com.ptc.*` for example.)
 - In addition, this means that classes that are outside your interest, but that are in the classpath and match the package hierarchy, will also get scanned.
 - From a performance standpoint (MethodServer startup) each scan adds up time.
 - You are requested to take advantage of the OOTB scan provided on “`com.ptc.mvc.builders`” base package. This means that all the builders that you author should be under `com.ptc.mvc.builders` package. If you are not using the OOTB scan, the rule of thumb is to use the scan if there are more than 10 builders available in the package.

- Explicit configuration:
 - You are encouraged not to specify the bean name while declaring the bean and use `@ComponentBuilder` annotation in the builder for proper registration of it in the Spring bean factory.

Supporting Legacy JCA

9.1 Style JCA Customizations Work in 11.0

URLS that don't have builders are handled by the default handler

```
23      |     <!-- Overridden for unit tests -->
24      |     <bean id="defaultHandlerMappings"
25      |     |       class="org.springframework.beans.factory.config.PropertiesFactoryBean">
26      |     |       <property name="location"
27      |     |         value="classpath:/config/mvc/common-urlMappings.properties" />
28      |     </bean>
```

Default Handler in *jca-mvc.xml*

```
104      |     <!-- Handles legacy netmarkets urls -->
105      |     <bean id="netmarketsController" class="com.ptc.jca.mvc.controllers.LegacyController" />
```

Registering the Legacy Controller in Spring (*jca-mvc.xml*)

Legacy Controller Enables Old Customizations to Run on Windchill 11.

This controller acts on URLs in the same manner that Windchill 10 did.

- These URLs are different from Windchill 10 and can easily be intercepted.

JCA Components

JCA Components implement **JcaComponentConfig**

JcaComponentConfig

- ... is a ComponentConfig
- ... is the super class for:
 - JcaTableConfig
 - JcaAttributesTableConfig
 - JcaTreeConfig
 - JcaAttributePanelConfig
 - JcaColumnConfig
 - JcaInfoConfig
 - JcaPropertyConfig
 - JcaPropertyPanelConfig

The screenshot shows the JavaDoc page for the **JcaComponentConfig** interface. At the top, there's a navigation bar with links for Overview, Package, Class, Use, Tree, Deprecated, Index, and Help. Below the navigation bar, there are links for PREV CLASS, NEXT CLASS, SUMMARY, NESTED, FIELD, CONSTR, METHOD, FRAMES, NO FRAMES, DETAIL, FIELD, CONSTR, and METHOD. The main content area starts with the package name **com.ptc.jca.mvc.components**. It then defines the **Interface JcaComponentConfig**. Below this, it lists **All Superinterfaces:** [ComponentConfig](#). It also lists **All Known Implementing Classes:** [AbstractJcaComponentConfig](#), [JcaAttributeConfig](#), [JcaAttributePanelConfig](#), [JcaAttributesTableConfig](#), [JcaColumnConfig](#), [JcaGroupConfig](#), [JcaInfoConfig](#), [JcaPropertyConfig](#), [JcaPropertyPanelConfig](#), [JcaTableConfig](#), and [JcaTreeConfig](#). The interface is defined with the following code snippet:

```
public interface JcaComponentConfig
extends ComponentConfig
```

A component config that can create a ComponentDescriptor

API for JCA Components



You will most likely never have to modify this class.

Javadoc is Available

Inheritance is used logically in the framework

- The code is easier to follow than previous releases – because of Spring Module

JCA Tables and Trees

JCA Table and Tree

Part Table (Sync) All

Actions		Name	Number	Version	Last Modified	Context	
<input type="checkbox"/>		MVCPT_1290196375...	0000000141		A.2	2010-11-22 12:51 CST GOLF_CART	
<input type="checkbox"/>		child1	0000000167			A.1	2010-11-22 09:31 CST GOLF_CART
<input type="checkbox"/>		Parent	0000000166		A.2	2010-11-22 08:33 CST GOLF_CART	
<input type="checkbox"/>		child4	0000000170		A.1	2010-11-22 08:32 CST GOLF_CART	
<input type="checkbox"/>		child3	0000000169		A.1	2010-11-22 08:32 CST GOLF_CART	
<input type="checkbox"/>		child2	0000000168		A.1	2010-11-22 08:32 CST GOLF_CART	
<input type="checkbox"/>		child1	0000000167			A.1	2010-11-22 08:31 CST GOLF_CART
<input type="checkbox"/>		Parent	0000000166		A.1	2010-11-22 08:30 CST GOLF_CART	
<input type="checkbox"/>		MVCPT_1290210387...	0000000148		A.1	2010-11-19 17:47 CST GOLF_CART	
<input type="checkbox"/>		MVCPT_1290208891...	0000000147		A.1	2010-11-19 17:22 CST GOLF_CART	
<input type="checkbox"/>		MVCPT_1290208460...	0000000146		A.1	2010-11-19 17:15 CST GOLF_CART	
<input type="checkbox"/>		MVCPT_1290199053...	0000000145		A.1	2010-11-19 14:38 CST GOLF_CART	
<input type="checkbox"/>		MVCPT_1290198029...	0000000144		A.1	2010-11-19 14:21 CST GOLF_CART	
<input type="checkbox"/>		MVCPT_1290197669...	0000000143		A.1	2010-11-19 14:15 CST GOLF_CART	
<input type="checkbox"/>		MVCPT_1290196581...	0000000142		A.1	2010-11-19 13:57 CST GOLF_CART	
<input type="checkbox"/>		MVCPT_1290196375...	0000000141		A.1	2010-11-19 13:53 CST GOLF_CART	
<input type="checkbox"/>		MOUNTAIN_BIKE_W...	C000004			A.1 (Design)	2010-11-18 10:59 CST Bicycle2
<input type="checkbox"/>		ROAD_RACING_WHE...	C000003			A.1 (Design)	2010-11-18 10:58 CST Bicycle2
<input type="checkbox"/>		MPart	0000000121		A.1	2010-11-18 10:08 CST NewProd	
<input type="checkbox"/>		test part	0000000109			A.1	2010-11-17 11:26 CST Drive System

JCA Table

Part Tree

Actions		Name	Number
<input type="checkbox"/>		LEG	GC000002
<input type="checkbox"/>		BOLT_1_8	GC000017
<input type="checkbox"/>		ACTUATOR_LOCK	GC000016
<input type="checkbox"/>		RIGHT_ACTUATOR	GC000011
<input type="checkbox"/>		AXLE_LATCH	GC000007
<input type="checkbox"/>		AXLE_FASTENER	GC000008
<input type="checkbox"/>		LOWER_RIGHT_ACTUATOR	GC000012
<input type="checkbox"/>		NUT_1_4	GC000010
<input type="checkbox"/>		BOLT_1_4	GC000009
<input type="checkbox"/>		UPPER_RIGHT_ARM	GC000013
<input type="checkbox"/>		LOWER_RIGHT_ARM	GC000014
<input type="checkbox"/>		NUT_1_8	GC000018
<input type="checkbox"/>		LEFT_ACTUATOR	GC000003
<input type="checkbox"/>		LOWER_LEFT_ARM	GC000004
<input type="checkbox"/>		LOWER_LEFT_ACTUATOR	GC000006
<input type="checkbox"/>		UPPER_LEFT_ARM	GC000005
<input type="checkbox"/>		NUT_1_4	GC000010
<input type="checkbox"/>		AXLE_LATCH	GC000007
<input type="checkbox"/>		AXLE_FASTENER	GC000008
<input type="checkbox"/>		BOLT_1_4	GC000009

JCA Tree

Other Configs

JcaAttributeTable, JcaPropertyPanel, and JcaAttributePanel

Attribute Table1 - CREATE

Name	Value
NAME	Test Name
NUMBER	
Attribute	
Test Attribute	
Carambola Attribute	
Quantity	21

Attribute Table

Attributes

Number:	(Generated)
Name:	
Description:	
Location:	<input type="radio"/> Autoselect Folder (/GOLF_CART) <input checked="" type="radio"/> Select Folder [/GOLF_CART]
Life Cycle Template:	(Generated)
Team Template:	(Generated)

Business

test:	
test2:	

Attribute Panel

Name: PTC Corporate Headquarters Address: 140 Kendrick Street address2: Needham MA 02494 country: USA Phone:  (781) 370-5000  Fax: (781) 370-6000	Name: PTC Corporate Headquarters Address: 140 Kendrick Street address2: Needham MA 02494 country: USA Phone:  (781) 370-5000  Fax: (781) 370-6000	Name: PTC Corporate Headquarters Fax: (781) 370-6000
--	--	---

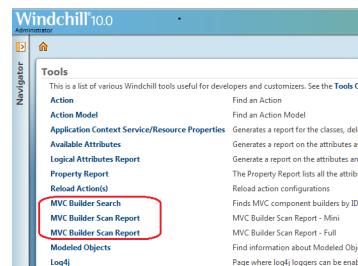
Property Panel

Customization Tools

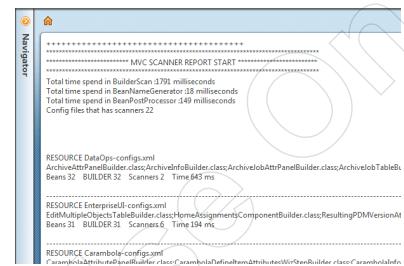
Cutomization Tools

Navigate **Customization > Tools**

- MVC Builder Search – Find the Builder given a componentId
- MVC Builder Scan Report – Summary of Files scanned on startup.
 - •log4j.logger.com.ptc.mvc.scan= INFO should be enabled



Tools



Builder Scan Report

Builder Scan Report

Builder Scan Report Summarizes the Files Found

Enable the report

- Add the following line to %WT_HOME%\codebase\WEB-INF\log4jMethodServer.properties


```
62 logger.wt.pom.properties=INFO
63 logger.wt.servlet.ServletRequestMonitor.requestMBean.start=OFF
64 logger.wt.wvs.workeragent.StandardCadAgentService=INFO
65 logger.com.ptc.mvc.scan=INFO
66
```

Enable Builder Scan Report

Tools

This is a list of various Windchill tools useful for developers and customizers. See the [Tools Overview](#)

Action	Find an Action
Action Model	Find an Action Model
Application Context Service/Resource Properties	Generates a report for the classes, delegat
Available Attributes	Generates a report on the attributes availa
Logical Attributes Report	Generate a report on the attributes and th
Property Report	The Property Report lists all the attributes
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

Run the Report from the Tools Page

```
=====
MVC SCANNER REPORT START
=====
Total time spent in BuilderScan: 3728 milliseconds
Total time spent in BeanNameGenerator: 8 milliseconds
Total time spent in BeanProcessor: 84 milliseconds
Config files that has changes: 3

=====
RESOURCE QueueAdmin-config.xml
QueueAttributes.class;QueueCreateWizardStep.class;QueueEditAnyBuilderWizardStep.class;QueueEntryArgi.cl
Beans: 9 | BUILDER: TYPEBASED 2 | Scanners: 1 | Time: 813 ms

=====
RESOURCE WfmWmc-config.xml
CustomerSupportPageBuilder.rsrc
Beans: 1 | BUILDER: 1 | Scanners: 1 | Time: 215 ms

=====
RESOURCE EnterpriseDB-config.xml
EnterpriseDBTableBuilder.class;HomeAssignmentComponentBuilder.class;ResolvingPDMVersionJobs
Beans: 30 | BUILDER: 30 | Scanners: 6 | Time: 238 ms

=====
RESOURCE Carambola-config.xml
CarambolaAttributePageBuilder.class;CarambolaDefinitionAttributeWicStepBuilder.class;CarambolaJdbc
Beans: 44 | BUILDER: 41 | TYPEBASED: 2 | Scanners: 3 | Time: 203 ms
```

Sample Report

Exercise 1: Add a Component in Navigation

Objectives

After successfully completing this exercise, you will be able to:

- Add a Web page to the Windchill UI using a Component.
- Create a component builder.
- Register a Component Builder

Scenario

In this exercise, you will create a custom Component Builder which will simply draw a table.

Screenshots below is the expected output for this exercise.

The screenshot shows the PTC Windchill interface. At the top, there's a toolbar with various icons. Below it is a navigation pane titled "Services Academy" containing links like "Hello World", "My Product List" (which is highlighted with a red box), "Table Example", "Multi Component Example", "Tree Example", and "Search for Part". The main content area displays a table titled "myTable" with the following data:

Name	Owner	Last Modified
Drive System	wcadmin	2011-04-08 12:57 EDT
GENERIC COMPUTER	wcadmin	2011-04-08 13:16 EDT
GOLF CART	wcadmin	2011-04-08 12:57 EDT
ProductView Demo	wcadmin	2011-04-08 13:16 EDT

(0 objects selected)

Task 1: Determine the table to use as a template.

1. The View Products page will be used as the example for this exercise. Click the link to reveal the URL information.

The screenshot shows the PTC Windchill interface with a toolbar at the top. Below it, under "Recent Products", there is a link labeled "Drive System".

2. Note the URL that is displayed.

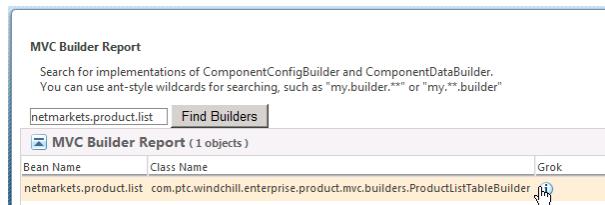
Example:

<http://server-name/Windchill/app/#ptc1/comp/netmarkets.product.list>

Task 2: Copy and modify the Builder.

1. Use the MVC Builder Search tool to determine the Builder for this page. Performing a Builder Search via **Customization Tab > Tools >MVC Builder Search**
2. Enter the keywords: **netmarkets.product.list** to find the builder.

Example:



MVC Builder Search Results

3. In Eclipse, under the **src** folder, create a package named: **com.ptc.serviceAcademy.builders**, create class named: **MyProductListTableBuilder**.
4. Update the code with below content.

Result:

```

package com.ptc.serviceAcademy.builders;

import wt.util.WTException;

/**
 * MVC builder class for Product List table
 */
@ComponentBuilder("com.ptc.serviceAcademy.builders.MyProductListTableBuilder")
public class MyProductListTableBuilder extends AbstractConfigurableTableBuilder{
    private final ClientMessageSource messageSource = getMessageSource("com.ptc.windchill.enterprise.product.ProductClientResource");

    * (non-Javadoc)
    @Override
    public Object buildComponentData(ComponentConfig config, ComponentParams params) throws Exception {
        String tableId="com.ptc.serviceAcademy.builders.MyProductListTableBuilder";
        return ProductListCommand.getProducts(tableId);
    }

    @Override
    public ComponentConfig buildComponentConfig(ComponentParams params) throws WTException {
        String helpContext ="ProductsHelp";
        ComponentConfigFactory factory = getComponentConfigFactory();
        TableConfig table = factory.newTableConfig();
        table.setId("com.ptc.serviceAcademy.builders.MyProductListTableBuilder");
        table.setConfigurable(true);
        table.setType("wt_pdmlink.PDMLinkProduct");
        table.setLabel("myTable");
        table.setActionModel("product list");
        table.setSelectable(true);
        table.setHelpContext(helpContext);
    }
}

```



Refer Lab Files for **MyProductListTableBuilder.java** code.

5. Compile the Java code.



In Eclipse, the function **Build Automatically** is chosen by default. it means that every time when you save the files, the .java files will be automatically compiled. To check that, go to Project, and select **Build Automatically**.

Task 3: Update the Builder.

1. Update the custom.xml file under **%WT_HOME%\codebase\config\mvc**

Example:

```

<?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-10.0.xsd">

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

</beans>

```

2. Deploy through lightbuild if you have made the above change in eclipse and Restart Servers.
3. Verify that your builder was loaded by performing a Builder Search via **Customization Tab > Tools >MVC Builder Search**

4. Search with com.ptc.serviceAcademy.*

Example:

MVC Builder Report

Search for implementations of ComponentConfigBuilder and ComponentDataBuilder.
You can use ant-style wildcards for searching, such as "my.builder.**" or "my.**.builder"

com.ptc.serviceAcademy.

MVC Builder Report

Bean Name	Class Name	Grok
com.ptc.serviceAcademy.builders.MyProductListTableBuilder	com.ptc.serviceAcademy.builders.MyProductListTableBuilder	

Builder Search for Customization

Task 4: Register the Actions.

1. Update the custom-actions.xml under codebase\config\actions

Example:

```
<objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld">
        <command url="netmarkets/jsp/serviceAcademy/navigationlist.jsp" windowType="page"/>
    </action>

    <action name="myProductListTable">
        <component name="com.ptc.serviceAcademy.builders.MyProductListTableBuilder" windowType="page"/>
    </action>
</objecttype>
```

custom-actions.xml Entry

2. Update the custom-actionModel.xml under codebase\config\actions.

Example:

```
<model name="serviceAcademy navigation" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld" type="object"/>
    <action name="myProductListTable" type="object"/>
</model>
```

custom-actionModels.xml Entry

3. Update the resource bundles: NavigationRB.java and NavigationRB_en_US.java files.

Result:

```
@RBEEntry("My Product List")
public static final String OBJECT_MYPRODUCTLISTTABLE_DESCRIPTION = "object.myProductListTable.description";

@RBEEntry("My Product List")
public static final String OBJECT_MYPRODUCTLISTTABLE_TITLE = "object.myProductListTable.title";

@RBEEntry("My Product List")
public static final String OBJECT_MYPRODUCTLISTTABLE_TOOLTIP = "object.myProductListTable.tooltip";
```

4. Reload the Actions

Example:

Tools

This is a list of various Windchill tools useful for developers and customizers. See the [Tools](#).

Action	Find an Action
Action Model	Find an Action Model
Application Context Service/Resource Properties	Generates a report for the classes, dele
Available Attributes	Generates a report on the attributes av
Logical Attributes Report	Generate a report on the attributes anc
Property Report	The Property Report lists all the attribu
Reload Action(s)	Reload action configurations
MVC Builder Search	Finds MVC component builders by ID

Task 5: Deployment.

1. Use lightbuild to deploy the component.
2. Restart the Method Server.
3. Navigate to **Service Academy >My Product List** to check the table.

Completion Criteria

1. The second tab displayed in the **Navigator >ServiceAcademy**.
2. A product list table displayed in the page.

This completes the exercise.

Exercise 2: Create Custom-Type Information Page

Objectives

After successfully completing this exercise, you will be able to:

- Create Information Page that has tabs.
- Create custom-type Information Page.

Scenario

When “i” icon is clicked on Folder Page, you will be directed to custom-type Information Page that shows details.

Existing components are shown for contents in tab navigation.

Task 1: Define a soft type.

1. Create “MyPart” as a subtype of Part type.
2. Log on as wcadmin.
3. Click "Site > Utility > Type and Attribute Management > Manage Types."
4. Select “Part” type and create a new subtype.
5. Specify com.ptc.MyPart for Internal Name and “MyPart” for Display Name.

Task 2: Define an action model.

1. Update the custom-actionModels.xml for “myPartInfoPageTabSet” as follows:

Result:

```
<model name="myPartInfoPageTabSet" resource="com.ptc.core.ui.navigationRB">
<submodel name="partInfoDetailsTab"/>
<action name="productStructureGWT" type="psb"/>
<submodel name="changesTab"/>
</model>
```



Refer Lab Files for custom-actionModels.xml file.

2. Update the custom-actions.xml as follows:

Result:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE listofactions SYSTEM "actions.dtd">
<listofactions>
<objecttype name="navigation" class="" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
<action name="serviceAcademy">
</action>
</objecttype>

<objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
<action name="helloWorld">
<command url="netmarkets/jsp/serviceAcademy/navigationlist.jsp" windowType="page"/>
</action>

<action name="myProductListTable">
<component name="com.ptc.serviceAcademy.builders.MyProductListTableBuilder" windowType="page" />
</action>
<action name="productStructureGWT">
<component name="com.ptc.serviceAcademy.builders.MyPartInfoPageBuilder" windowType="page" />
</action>
</objecttype>
</listofactions>
```



Refer Lab Files for custom-actions.xml file.

Task 3: Modify the custom part menu.

1. Create “myPartInfoPage.jsp” for Information Page.
2. Save it in %WT_HOME%\codebase\WEB-INF\jsp\serviceAcademy

Result:



```

<%@page language="java" session="true" pageEncoding="utf-8"%>
<%@taglib uri="http://www.ptc.com/windchill/taglib/core" prefix="wc"%>
<%@taglib uri="http://www.ptc.com/windchill/taglib/icaMvc" prefix="mvc"%>
<%@include file="/netmarkets/jsp/util/begin_comp.jspf"%>
Welcome to My Part Information Page !!
<mvc:infoPage/>
<%@include file="/netmarkets/jsp/util/end_comp.jspf"%>

```



Refer Lab Files for myPartInfoPage.jsp file.

Task 4: Create a builder for Information Page.

1. Create the builder package and class as “com.ptc.serviceAcademy.builders” and “MyPartInfoPageBuilder” as follows.

Result:



```

package com.ptc.serviceAcademy.builders;
import wt.util.WTException;
import com.ptc.mvc.components.ComponentParams;
import com.ptc.mvc.components.InfoConfig;
import com.ptc.mvc.components.OverrideComponentBuilder;
import com.ptc.mvc.components.TypeBased;
import com.ptc.windchill.enterprise.part.mvc.builders.PartInfoBuilder;
@TypeBased("wt.part.WTPart|com.ptc.MyPart")
@OverrideComponentBuilder
public class MyPartInfoPageBuilder extends PartInfoBuilder {
    @Override
    public InfoConfig buildInfoConfig(ComponentParams params) throws WTException {
        InfoConfig config = super.buildInfoConfig(params);
        config.setTabSet("myPartInfoPageTabSet");
        config.setView("/serviceAcademy/myPartInfoPage.jsp");
        return config;
    }
}

```



Refer Lab Files for MyPartInfoPageBuilder.java file.

Task 5: Deploy and test.

1. Deploy the code with lightBuild.
2. Restart the Method Server.
3. Create MyPart object.
4. Show Information Page.

Welcome to My Part Information Page !!
Actions: MyPart - 0000000042, Test, A.1 (Manufacturing)

Details Structure Changes New Tab 1

Visualization and Attributes | More Attributes

Visualization and Attributes

Name:	Test
Status:	Checked in
Modified By:	Site, Administrator
Last Modified:	2016-05-18 11:09 UTC

General

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

System

State:	In Work - Released - Canceled	Location:	/GOLF_CART
Context:	GOLF_CART	Team Template:	
Life Cycle Template:	Basic	Modified By:	Site, Administrator
Created By:	Site, Administrator	Last Modified:	2016-05-18 11:09 UTC
Created On:	2016-05-18 11:09 UTC		

Completion Criteria

1. Created custom-type information page successfully.

This completes the exercise.

Exercise 3: Create a Table Component

Objectives

After successfully completing this exercise, you will be able to:

- Create a Table Customization.
- Register a new MVC builder.

Scenario

In this exercise, you will create a Table customization using MVC. In this customization, you will see additional available features for configuring table display.

Screenshots below is the expected output for this exercise.

The screenshot shows the PTC Windchill Services Academy interface. The top navigation bar has 'Search' and 'Browse' tabs. Below the navigation is a toolbar with various icons. The main content area displays a list of examples under 'Services Academy'. One item, 'Table Example', is highlighted with a red box. Below this, a 'Part Table' is shown with an 'Expand' button. The table has columns for 'Name', 'Number', and 'Last Modified'. The data rows are as follows:

Name	Number	Last Modified
handle_side_rod_900	HANDLE_SIDE_ROD_900	2011-03-03 14:46 EST
screw_skt_m6x1x15_	SCREW_SKT_M6X1X15_	2011-03-03 14:46 EST
piston_ring_900	PISTON_RING_900	2011-03-03 14:46 EST
eng_block_front_900	ENG_BLOCK_FRONT_900	2011-03-03 14:46 EST
primary_gear_shft_900	PRIMARY_GEAR_SHFT_900	2011-03-03 14:46 EST
lcrew_skt_m6x1x06_	SCREW_SKT_M6X1X06_	2011-03-03 14:46 EST
screw_skt_m5xp18x15_	SCREW_SKT_M5XP18X15_	2011-03-03 14:46 EST

Task 1: Create the Builder.

1. Create a Java file *TableBuilder.java* in the package **com.ptc.serviceAcademy.builders**



As the package **com.ptc.serviceAcademy.builders** has already been registered, so when you put other classes in this package, there is no need to register again, method server will scan the whole package to find the builders.

2. Update the code in the *TableBuilder.java* file as is displayed in the following:

Result:

```

package com.ptc.serviceAcademy.builders;

import wt.fc.PersistenceHelper;
import wt.part.WTPart;
import wt.query.QuerySpec;
import wt.util.WTException;
import com.ptc.mvc.components.AbstractComponentBuilder;
import com.ptc.mvc.components.ComponentBuilder;
import com.ptc.mvc.components.ComponentConfig;
import com.ptc.mvc.components.ComponentConfigFactory;
import com.ptc.mvc.components.ComponentParams;
import com.ptc.mvc.components.TableConfig;

@ComponentBuilder("com.ptc.serviceAcademy.builders.TableBuilder")
public class TableBuilder extends AbstractComponentBuilder{

    @Override
    public Object buildComponentData(ComponentConfig arg0, ComponentParams arg1) throws Exception {
        QuerySpec qs=new QuerySpec(WTPart.class);
        qs.setQueryLimit(10000);
        return PersistenceHelper.manager.find(qs);
    }

    @Override
    public ComponentConfig buildComponentConfig(ComponentParams arg0) throws WTException {
        ComponentConfigFactory factory = getComponentConfigFactory();
        TableConfig table = factory.newTableConfig();
        table.setLabel("Part Table");
        table.setSelectable(true);
        //set the actionModel that comes in the Table Tool bar
        table.setActionModel("mvc_tables_toolbar");

        table.addComponent(factory.newColumnConfig("name", true));
        table.addComponent(factory.newColumnConfig("number", true));
        table.addComponent(factory.newColumnConfig("type", true));
        table.addComponent(factory.newColumnConfig("thePersistInfo.modifyStamp", true));

        return table;
    }
}

```



Refer Lab Files for Tablebuilder.java code.

Task 2: Incorporate the Builder into the UI.

1. Update custom-actions.xml to create an action for the builder

Result:

```

<objecttype name="navigation" class="" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="serviceAcademy">
        </action>
</objecttype>

<objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld">
        <command url="netmarkets/jsp/serviceAcademy/navigationlist.jsp" windowType="page"/>
    </action>
    <action name="myProductListTable">
        <component name="com.ptc.serviceAcademy.builders.MyProductListTableBuilder" windowType="page"/>
    </action>
    <action name="tableExample">
        <component name="com.ptc.serviceAcademy.builders.TableBuilder" windowType="page"/>
    </action>
</objecttype>

```

Addition of Action



Refer Lab Files for custom-actions.xml.

2. Update Action Framework custom-actionModels.xml

Example:

```
<model name="serviceAcademy navigation" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
  <action name="helloWorld" type="object"/>
  <action name="myProductListTable" type="object"/>
  <action name="tableExample" type="object"/>
</model>
```



Refer Lab Files for custom-actionModels.xml.

3. Update the Resource Bundle in the two files: *NavigationRB.java* and *NavigationRB_en_US.java*.

Example:

```
@RBEEntry("Table Example")
public static final String OBJECT_TABLEEXAMPLE_DESCRIPTION = "object.tableExample.description";

@RBEEntry("Table Example")
public static final String OBJECT_TABLEEXAMPLE_TITLE = "object.tableExample.title";

@RBEEntry("Table Example")
public static final String OBJECT_TABLEEXAMPLE_TOOLTIP = "object.tableExample.tooltip";
```



Refer Lab Files for NavigationRB and NavigationRB_en_US Java files.

Task 3: Deploy the code and review the result.

1. Use lightbuild to deploy the component.
2. Restart Windchill.
3. You should see the example in the custom tab.

Completion Criteria

1. The second tab displayed in the **Navigator >ServiceAcademy**.
2. A part list table displayed in the page.

This completes the exercise.

Exercise 4: Create a Page with Multiple MVC JCA Components

Objectives

After successfully completing this exercise, you will be able to:

- Modify the View Component in MVC.
- Incorporate multiple JCA components in a single view.
- Incorporate JavaScript into a new View.

Scenario

In this exercise, you will place both a property panel and a table in a single view.

Screenshots below is the expected output for this exercise.

The screenshot shows the PTC Windchill Services Academy interface. The top navigation bar includes 'Search' and 'Browse' tabs, along with various icons for file operations like Open, Save, Print, and Email. Below the navigation is a sidebar titled 'Services Academy' containing a list of examples: Hello World, My Product List, Table Example, Multi Component Example (which is highlighted with a red box), Tree Example, and Search for Part. The main content area displays a property panel for a company with fields for Name, Phone, Fax, and Address. Below this is a 'Part Table' section with a table listing various mechanical parts, each with a checkbox, a name, a number, and a last modified date.

Name	Number	Last Modified
handle_side_rod_900	HANDLE_SIDE_ROD_900	2011-03-03 14:46 EST
screw_skt_m6x1x15_	SCREW_SKT_M6X1X15_	2011-03-03 14:46 EST
piston_ring_900	PISTON_RING_900	2011-03-03 14:46 EST
eng_block_front_900	ENG_BLOCK_FRONT_900	2011-03-03 14:46 EST
primary_gear_shaft_900	PRIMARY_GEAR_SHAFT_900	2011-03-03 14:46 EST
screw_skt_m6x1x36_	SCREW_SKT_M6X1X36_	2011-03-03 14:46 EST
screw_skt_m5xpt8x15_	SCREW_SKT_M5XPT8X15_	2011-03-03 14:46 EST
crank_900	CRANK_900	2011-03-03 14:46 EST
switch_900	SWITCH_900	2011-03-03 14:46 EST

Task 1: Create a JAVA file to make an AttributePanel builder.

1. In Eclipse, create a Java file which will be called *AttributePanelBuilder.java* and will be located in the package:
`com.ptc.serviceAcademy.builders`
2. Update the content of the files.

Example:

```

package com.ptc.serviceAcademy.builders;
import java.util.HashMap;
import java.util.Map;
import wt.util.WTEException;
import com.ptc.mvc.components.AbstractComponentBuilder;
import com.ptc.mvc.components.AttributeConfig;
import com.ptc.mvc.components.AttributePanelConfig;
import com.ptc.mvc.components.ComponentConfig;
import com.ptc.mvc.components.ComponentConfigFactory;
import com.ptc.mvc.components.ComponentParams;
import com.ptc.mvc.components.ComponentBuilder;

@ComponentBuilder(value="com.ptc.serviceAcademy.builders.AttributePanelBuilder")
public class AttributePanelBuilder extends AbstractComponentBuilder {

    @Override
    public Object buildComponentData(ComponentConfig arg0, ComponentParams arg1)
        throws Exception {
        Map<String, String> address = new HashMap<String, String>();
        address.put("name", "Parametric Technology Corp.");
        address.put("phone", "+763-957-8000");
        address.put("fax", "+763-957-8001");
        address.put("address1", "3785 Pheasant Ridge");
        address.put("address2", "Blaine MN 55449");
        return address;
    }

    @Override
    public AttributePanelConfig buildComponentConfig(ComponentParams arg0)
        throws WTEException {
        ComponentConfigFactory factory = getComponentConfigFactory();
        AttributePanelConfig panelConfig = factory.newAttributePanelConfig();
        panelConfig.addComponent(getAttributeConfig("name", "Phone"));
        panelConfig.addComponent(getAttributeConfig("phone", "Phone"));
        panelConfig.addComponent(getAttributeConfig("fax", "Fax"));
        panelConfig.addComponent(getAttributeConfig("address1", "Address"));
        panelConfig.addComponent(getAttributeConfig("address2", ""));
        return panelConfig;
    }

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

```

An override annotation indicates that the annotated method is required to override a method in a super class. If a method with this annotation does not override its super-class's method, the compiler will generate an error.

AttributePanelBuilder.java



Refer Lab Files for AttributePanelBuilder.java code.

Task 2: Create an JAVA file which will be called *MultiComponentBuilder.java* .

1. In Eclipse, create a java file which will be called *MultiComponentBuilder.java* and will be located in the package:
com.ptc.serviceAcademy.builders
2. Update the content of the file:

Example:

```

package com.ptc.serviceAcademy.builders;

import wt.util.WTEException;

import com.ptc.mvc.components.AbstractComponentConfigBuilder;
import com.ptc.mvc.components.ComponentBuilderType;
import com.ptc.mvc.components.ComponentConfig;
import com.ptc.mvc.components.ComponentParams;
import com.ptc.mvc.components.ComponentBuilder;
import com.ptc.mvc.components.MultiComponentConfig;

@ComponentBuilder(value="com.ptc.serviceAcademy.builders.MultiComponentBuilder", type=ComponentBuilderType.CONFIG_ONLY)
public class MultiComponentBuilder extends AbstractComponentConfigBuilder {

    @Override
    public ComponentConfig buildComponentConfig(ComponentParams arg0)
        throws WTEException {
        // TODO Auto-generated method stub

        MultiComponentConfig multiComp = new MultiComponentConfig();
        multiComp.addNestedComponent("com.ptc.serviceAcademy.builders.AttributePanelBuilder");
        multiComp.addNestedComponent("com.ptc.serviceAcademy.builders.TableBuilder");
        multiComp.setView("/serviceAcademy/multiComponentView.jsp");

        return multiComp;
    }
}

```

MultiComponentBuilder.java



Refer Lab Files for MultiComponentBuilder.java file.

Task 3: Create a JSP view for rendering the multiple components.

1. In Eclipse, under the folder *codebase*, create a new folder which is named: *WEB-INF*, in the folder *WEB-INF*, create a folder *jsp*, in the folder, create a folder *serviceAcademy*.

Example:

```

codebase
  config
  netmarkets
  WEB-INF
    jsp
      serviceAcademy

```

2. In the folder `codebase/WEB-INF/jsp/serviceAcademy/`, create a jsp file which will be called `multiComponentView.jsp`
3. Update the content of the file

Example:

```

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

<!-- render the attributepanel --&gt;
&lt;mvc:simpleAttributePanel/&gt;

<!-- render the table --&gt;
&lt;mvc:table compId="com.ptc.serviceAcademy.builders.TableBuilder"/&gt;
</pre>


multiComponentView.jsp


```

 Refer Lab Files for MultiComponentView.jsp file.

Task 4: Incorporate the Builder into the UI.

1. Update custom-actions.xml to create an action for the builder.

Result:

```

<objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
  <action name="helloWorld">
    <command url="netmarkets/jsp/serviceAcademy/navigationlist.jsp" windowType="page"/>
  </action>

  <action name="myProductListTable">
    <component name="com.ptc.serviceAcademy.builders.MyProductListTableBuilder" windowType="page"/>
  </action>

  <action name="tableExample">
    <component name="com.ptc.serviceAcademy.builders.TableBuilder" windowType="page"/>
  </action>

  <action name="multiComponent">
    <component name="com.ptc.serviceAcademy.builders.MultiComponentBuilder" windowType="page"/>
  </action>
</objecttype>

```

Additions on custom-actions.xml

 Refer Lab Files for custom-actions.xml.

2. Update Action Framework custom-actionModels.xml

Result:

```

<model name="serviceAcademy navigation" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
  <action name="helloWorld" type="object"/>
  <action name="myProductListTable" type="object"/>
  <action name="tableExample" type="object"/>
  <action name="multiComponent" type="object"/>
</model>

```

 Refer Lab Files for custom-actionModels.xml.

3. Update the Resource Bundle in the two files: `NavigationRB.java` and `NavigationRB_en_US.java`

Example:

```
@RBEntry("Multi Component Example")
public static final String OBJECT_MULTICOMPONENT_DESCRIPTION = "object.multiComponent.description";
@RBEntry("Multi Component Example")
public static final String OBJECT_MULTICOMPONENT_TITLE = "object.multiComponent.title";
@RBEntry("Multi Component Example")
public static final String OBJECT_MULTICOMPONENT_TOOLTIP = "object.multiComponent.tooltip";
```

Additions on the Two RB Files



Refer Lab Files for NavigationRB_en_US java files.

Task 5: Deploy and Test your code.

1. Use lightBuild to deploy the component.
2. Restart Windchill.
3. Check your results

Completion Criteria

1. The second tab displayed in the **Navigator >ServiceAcademy**.
2. The attribute panel and the part list table displayed in the same page.

This completes the exercise.

Exercise 5: Create a Tree Component

Objectives

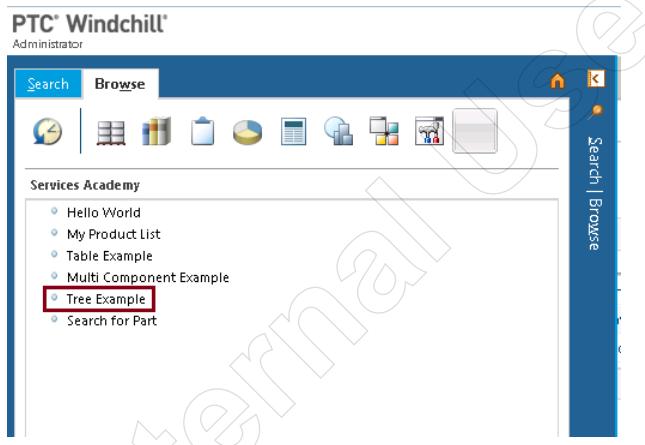
After successfully completing this exercise, you will be able to:

- Create a Tree Component.
- Use Carambola examples.
- Build data nodes for ComponentData in Tree Component.

Scenario

In this exercise, you will configure a simple tree using MVC builder.

Screenshots of "Tree Example in Menu" and "Tree Example" are the expected output for this exercise.

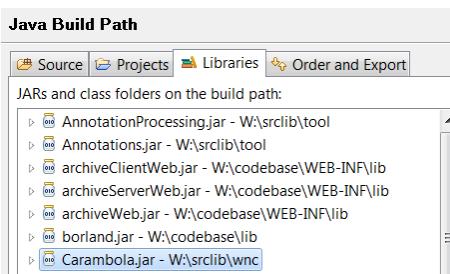


PTC® Windchill®	
Administrator	
Part Tree	
Name	Number
LOWER_BAG HOLDER	GC000022
LOWER_LEFT_ACTUATOR	GC000006
WHEELS_ASSEM	GC000031
WHEEL_HUB	GC000035
AXLE_SLEEVE	GC000034
HUB_CAP	GC000037
TIRE	GC000036
BEARING_AXLE	GC000033
WHEEL_AXLE	GC000032
UPPER_SUPPORT	GC000024
BOTTOM_SLIDER_CAP	GC000026
BOTTOM_SLIDER	GC000025
UPPER_BAG HOLDER	GC000028
UPPER_SLIDER	GC000027
CARD HOLDER	GC000030
HANDLE	GC000029
WHEEL_AXLE	GC000032
AXLE_FASTENER	GC000008
AXLE_SLEEVE	GC000034
UPPER_RIGHT_ARM	GC000013
NUT_1_4	GC000010
UPPER_ACTUATOR	GC000015
TIRE	GC000036
LEFT_ACTUATOR	GC000003
UPPER_LEFT_ARM	GC000005
LOWER_LEFT_ACTUATOR	GC000006
LOWER_LEFT_ARM	GC000004
AXLE_FASTENER	GC000008

Tree Example

Prerequisites

This example uses example code in Carambola.jar (com.ptc.carambola.customization.examples.tree package).



Modified Build Path



Please note that W:\ in above figure is WT_HOME path.

Task 1: Create the Builder.

- In Eclipse, create the Builder in the package com.ptc.serviceAcademy.builders ,name the builder : **TreeBuilder.java**.

Result:

```
package com.ptc.serviceAcademy.builders;

import wt.util.WTException;

import com.ptc.mvc.components.AbstractComponentBuilder;
import com.ptc.mvc.components.ComponentBuilder;
import com.ptc.mvc.components.ComponentConfig;
import com.ptc.mvc.components.ComponentConfigFactory;
import com.ptc.mvc.components.ComponentParams;
import com.ptc.mvc.components.TreeConfig;
import static com.ptc.core.components.descriptor.DescriptorConstants.ColumnIdentifiers.NUMBER;
import static com.ptc.core.components.descriptor.DescriptorConstants.ColumnIdentifiers.NAME;

@ComponentBuilder("com.ptc.serviceAcademy.builders.TreeBuilder")
public class TreeBuilder extends AbstractComponentBuilder {

    @Override
    public Object buildComponentData(ComponentConfig config, ComponentParams params) throws WTException {
        return new TreeHandler();
    }

    @Override
    public ComponentConfig buildComponentConfig(ComponentParams params) throws WTException {
        ComponentConfigFactory factory = getComponentConfigFactory();
        TreeConfig tree = factory.newTreeConfig();
        tree.setLabel("Part Tree");
        tree.setSelectable(true);
        tree.setActionModel("CustEx_tree_toolbar");
        tree.setNodeColumn(NUMBER);
        tree.setExpansionLevel("full");
        tree.setDisableAction("false");
        tree.addComponent(factory.newColumnConfig(NAME, true));
        tree.addComponent(factory.newColumnConfig(NUMBER, true));
        return tree;
    }
}
```

TreeBuilder.java



Refer Lab Files for TreeBuilder.java code.

There will be a new Class for implementing the method created in the next steps.

Task 2: Create the Class **TreeHandler.java**.

1. In the package com.ptc.serviceAcademy.builders, create a new Class named:

TreeHandler.java

Example:

```

package com.ptc.serviceAcademy.builders;

import java.util.*;
import com.ptc.mvc.builders.carambola.tree.CustExTreeHandler;
import wt.fc.Persistent;
import wt.fc.collections.WTArrayList;
import wt.part.WTPart;
import wt.part.WTPartHelper;
import wt.util.WTEException;
import wt.vc.config.ConfigSpec;

public class TreeHandler extends CustExTreeHandler {

    private ConfigSpec configSpec;
    /**
     * Get the root node from the command bean,
     * and get a config spec based on the root node
     */
    public List<Object> getRootNodes() throws WTEException {
        return getRootAsParts();
    }
    /**
     * Get the child parents for the given list of parent parts
     */
    public Map<Object, List<Object> > getNodes(List<Object> parents) throws WTEException {
        if (configSpec == null) {
            configSpec = getDefaultConfigSpec();
        }
        Map<Object, List<Object> > result = new HashMap<Object, List<Object> >();
        //API returns a 3D array where the 1st dim is the parent parts,
        //the 2nd dim is the list of children for a given parent,
        //and the 3rd dim is 2 element array w/the link obj at 0 and the child part at 1
        Persistent[][][] all_children = WTPartHelper.service.getUsesWTParts(new WTArrayList(parents), configSpec);
        for (ListIterator<Object> i = parents.listIterator(); i.hasNext();) {
            WTPart parent = (WTPart)i.next();
            Persistent[][] branch = all_children[i.previousIndex()];
            if (branch == null) {
                continue;
            }
            List<Object> children = new ArrayList(branch.length);
            result.put(parent, children);
            for (Persistent[] child : branch) {
                children.add(child[1]);
            }
        }
        return result;
    }
}

```

TreeHandler.java



Refer Lab Files for TreeHandler.java code.

Task 3: Add the actions.

1. Create the action for navigating the tree builder

Example:

```

<objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld">
        <command url="netmarkets/jsp/serviceAcademy/navigationlist.jsp" windowType="page"/>
    </action>

    <action name="myProductListTable">
        <component name="com.ptc.serviceAcademy.builders.MyProductListTableBuilder" windowType="page"/>
    </action>

    <action name="tableExample">
        <component name="com.ptc.serviceAcademy.builders.TableBuilder" windowType="page"/>
    </action>

    <action name="multiComponent">
        <component name="com.ptc.serviceAcademy.builders.MultiComponentBuilder" windowType="page"/>
    </action>

    <action name="treeExample">
        <component name="com.ptc.serviceAcademy.builders.TreeBuilder" windowType="page"/>
    </action>

```

</objecttype>

custom-actions.xml



Refer Lab Files for custom-actions.xml file.

2. Add the action to the ActionModel

Example:

```

<model name="serviceAcademy navigation" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld" type="object"/>
    <action name="myProductListTable" type="object"/>
    <action name="tableExample" type="object"/>
    <action name="multiComponent" type="object"/>
    <action name="treeExample" type="object"/>
</model>

```

custom-actionModels.xml



Refer Lab Files for custom-actionModels.xml file.

Task 4: Update the ResourceBundles.

1. Update the files **NavigationRB.java** and **NavigationRB_en_US.java**

Example:

```

@RBEEntry("Tree Example")
public static final String OBJECT_TREEEXAMPLE_DESCRIPTION = "object.treeExample.description";

@RBEEntry("Tree Example")
public static final String OBJECT_TREEEXAMPLE_TITLE = "object.treeExample.title";

@RBEEntry("Tree Example")
public static final String OBJECT_TREEEXAMPLE_TOOLTIP = "object.treeExample.tooltip";

```



Refer Lab Files for NavigationRB and Navigation_en_US java files.

Task 5: Deployment.

1. Use lightBuild to deploy the component.
2. Restart Windchill.
3. Check the result.

Completion Criteria

1. The second-level tab displayed in the **Navigator >ServiceAcademy**.
2. A part list tree displayed in the page.

This completes the exercise.

For PTC Internal Use Only

Module 5

Client-Side Technologies

Module Overview

In this module, we review the Client-side technologies, describe JavaScript and the two JavaScript third-party libraries which are used in Windchill.

Objectives

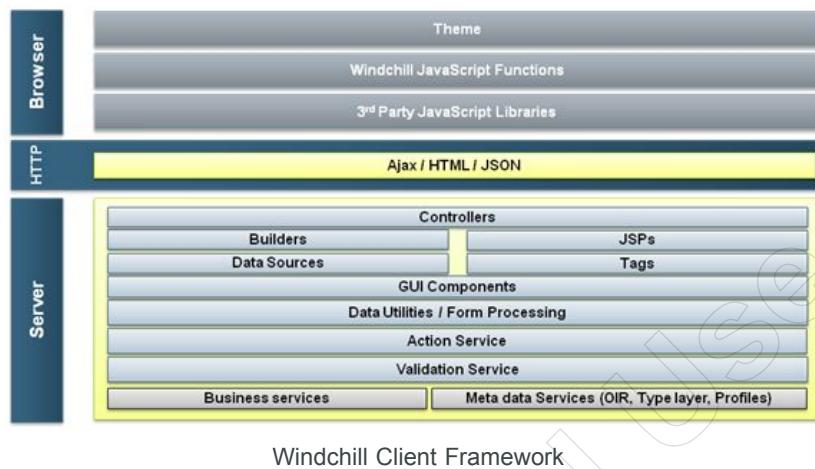
After completing this module, you will be able to:

- Describe the client-side technologies in Windchill.
- Explain the two JavaScript third-party libraries: Prototype and ExtJS.
- Explain how JavaScript is used in Windchill.
- Create a search Page and make the MVC table builder to display the result.

Windchill Client Framework

Windchill Client Architecture

The Windchill Client Architecture provides a framework for building pages easily and consistently by providing a standard set of components and services that are configurable and customizable.



Windchill Client Framework

The Framework Explanation

- The set of components includes, but is not limited to, a table, a tree, an information page, a wizard, and a dynamic JavaScript menu.
- At a lower level, these container components use other common components, called actions, action models and GUI components, to display data inside them.
- GUI Components are objects that define how an attribute is to be rendered. They are controlled through properties set in the Data Utilities. A GUI Component also contains a renderer that is responsible for writing out the HTML and/or JavaScript.

JavaScript Key Points

A Few Things to Understand Upfront about JavaScript

JavaScript is a prototype-based scripting language that is dynamic, weakly typed and has first-class functions. It is a multi-paradigm language, supporting object-oriented, imperative, and functional programming styles.

- JavaScript has prototypal inheritance.
 - This means that objects in JavaScript inherit from other objects, not classes.
- JavaScript is dynamic and weakly typed.
 - This means that most of the type checking normally done by the compiler is done at runtime in JavaScript.
 - It also means that most type conversions as well as ad hoc polymorphism (object property and function inheritance) are done at runtime.
- JavaScript has a global namespace.
 - This means that JavaScript code begins in a global container and anything created in that container can be referenced anywhere else in the container.
- JavaScript gives first-class status to functions.
 - This means that functions are objects and can have properties and methods, can be created, assigned to variables, passed as arguments, returned by other functions, and manipulated like any other object.

JavaScript Used in Windchill 11.0

- JavaScript is being used to a far greater extent in Windchill 10.0 than in previous releases such as Windchill 9.1.
- Though many customizations can be done without involving any custom JavaScript functionality, it is more vital now to understand how to best integrate custom JavaScript functionality into Windchill 11.0, as well as the best practices for writing custom JavaScript code.

JavaScript with Windchill

Using JavaScript with Windchill

- JavaScript can be integrated into either JSP pages (using the JSP or legacy approach to MVC) or into JSP views (using the Spring or builder approach to MVC).
- JavaScript can be included in one of three ways:
 - When created inside a *.js file and included as a source library in a JSP.
 - When created directly inside <script> tags in the JSP page or JSP view.
 - When created as a *.jsfrag file and compiled into main.js



This is a Windchill only option.

- There are two JavaScript third-party libraries which are used in Windchill:
 - Prototype
 - ExtJs



The detailed description about the two third-party libraries will be introduced in the following pages.

Three Ways to Include JavaScript in Windchill

- Option 1: Create a *.js library file of JavaScript functions and/or objects and include the file as a source library in a JSP.

```
<html>
<script type="Javascript" src="netmarkets/javascript/components/wizard.js"/>
<head>
<title>Insert title here</title>
</head>
<body>
</body>
</html>
```

- Option 2: Create the JavaScript directly in <script> tags in a JSP file:

```
<html>
<script type="Javascript" src="netmarkets/javascript/components/wizard.js"></script>
<head>
<title>Insert title here</title>
</head>
<body>
<script type="Javascript">
var myVar = "hello";
function myFunction()
{
    alert("some javascript");
}
</script>
</html>
```

- Option 3: Build a custom JavaScript library into Windchill's main.js file which is included in all pages.

- This is a good option if you are planning to extend the Windchill JavaScript libraries and will be using these extensions throughout many different contexts.
- There are two steps to accomplish this:

1. Create a *.jsfrag file and move it to *WT_HOME/codebase/netmarkets/javascript/js frags*
2. Run the following Windchill jsfrag compile script: From a Windchill shell on the *WT_HOME/bin* path, run the command: *ant -f jsfrag_combine.xml*



- Make sure the *.jsfrag file contains no errors, otherwise this will break main.js and the entire jsTables JavaScript libraries for rendering
- A tool such as jslint can help find errors: www.jslint.com

Prototype Shortcuts

Prototype JavaScript Library and DOM Shortcuts

The Prototype API, available in Windchill 9.1, Windchill 10 and Windchill 11.0, offers many convenient shortcuts to accessing and manipulating DOM elements.

- `document.getElementById('elementId')` is replaced with a shorter version `$('elementId')`.

```
<script type="Javascript">

    function myFunction()
    {
        document.getElementById("someId");      // this...
        $("someId");                          // is the same as this...
    }

</script>
```

- The prototype shortcut is much cleaner syntax when longer functions are performed

```
<script type="Javascript">

    function myFunction()
    {
        document.getElementById("someId").onclick=function() { alert("hello"); };
        $("someId").onclick = function() { alert("hello"); };
    }

</script>
```

- Retrieval of elements by css selectors and tag and/or attribute names are realized with the following shortcuts
 - `$$("tagName")` > `getElementsByTagName`
 - `$$("attr=value")` > `getElementsByTagAttributeValue` (not available elsewhere)

```
function prototypeDemo()
{
    // get element by id
    var htmlTable = $$("tableId");

    // get all links with a href of #
    var local_links = $$('a[href="#"]');

    // get all div tags with a 'foo' class
    var foo_divs = $$('div.foo');
}
```



Developers can see the API here: <http://api.prototypejs.org/>
 The Prototype library is available under the open source MIT license.

Prototype and Ajax Abstraction

Prototype and Ajax Abstraction

With prototype, Ajax calls are made simpler:

```
<script type="text/javascript">
//'PTC' is the namespace global object created for Windchill 10
//a namespace global object in the case of javascript is just an object that is created
//that we extend(prototypal) to define our state,methods, and inner objects..basically
//our API is contained within one object.
PTC.onReady(function(){
    $('#serviceAcademy_table').onclick=function(event){
        console.debug(event.target);

        //setup the ajax request based on a parameter
        new Ajax.Request('ptcl/custom/relatedItemsData',{
            method:'post',
            parameters:{changeRecord:'002'},
            onSuccess:function(result){
                alert('Ajax call returned successfully with '+result.responseText);
            },
            onFailure:function(result){
                alert('Ajax call returned unsuccessfully with '+result.responseText);
            }
        });
    });
</script>
```

- The Ajax request setup in the example above sends data to a Spring service (static data for the example) and handles the return result through separate “onSuccess” and “onFailure” callback functions.
- The prototype API handles all of the underlying browser differences and provides a single, simple interface from which to create Ajax calls and callbacks.

Prototype and Ajax Update Operation

Prototype and Ajax Update Operation

Prototype can also create an Ajax update operation

- This type of Ajax procedure links an Ajax call directly into a target component in the page.
- In this way, the target component can be updated automatically by the result of the Ajax call.

```
<script type="text/javascript">
//'PTC' is the namespace global object created for Windchill 10
//a namespace global object in the case of javascript is just an object that is created
//that we extend(prototype) to define our state,methods, and inner objects...basically
//our API is contained within one object.

PCT.onReady(function(){
    $('#serviceAcademy.table').onclick=function(event){
        //setup the ajax request based on a parameter
        new Ajax.Update('contentPanel', 'ptcl/custom/relatedItemsData',{
            method:'post',
            parameters:{changeRecord:'002'},
            insertion:'top'
        });
    };
});
</script>
```



In the above example, the “contentPanel” div element in the page is updated by the result of the Ajax call every time it is fired.



In this specific setup, the Ajax call will just add string content to the top of the content area every time the table on the page is clicked.

ExtJS Overview

What is ExtJS?

ExtJS is a JavaScript library for building interactive Web applications using techniques such as Ajax, DHTML, and DOM scripting.

It was originally an add-on library for YUI (Yahoo! User interface library) but has evolved into a full JavaScript library intended for the implementation of rich Internet applications (RIA).

- RIA Views with ExtJS
 - “RIA” is a term used in the industry to describe a Web-based application which shares many of the characteristics of desktop application software.
 - ExtJS uses Ajax, DHTML, and DOM scripting to mimic these characteristics.



For more information about ExtJS, please go to ExtJS official Website:<http://www.sencha.com/products/extjs/>

GUI Controls

Ext JS includes a set of GUI-based form controls (or "widgets") for use within Web applications:

- Text field and text area input controls
- Date fields with a pop-up date-picker
- Numeric fields
- List box and combo boxes
- Radio and checkbox controls
- Html editor control
- Grid control (with both read-only and edit modes, sortable data, lockable and draggable columns, and a variety of other features)
- Tree control
- Tab panels
- Toolbars
- Desktop application-style menus
- Region panels to allow a form to be divided into multiple sub-sections
- Sliders
- Vector graphics charts

Many of these controls are able to communicate with a Web server using Ajax.

ExtJS 3 and Windchill

ExtJS JavaScript Library

- Windchill 11.0 utilizes the ExtJS JavaScript library to render much of its UI.
- The ExtJS library starts under a namespace called “Ext” and all of its functionality is encapsulated within that namespace.

```
<%@page language="java" session="true" pageEncoding="utf-8"%>
<%@include file="/netmarkets/jsp/util/begin.jspf" %>

<script language="Javascript">

Ext.onReady(function(){
    alert('Hello World!'));
}</script>

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

ExtJS Code Example



ExtJS UI Example

Plug-In for Editing ExtJS

Use the Aptana Studio 3 plugin with Eclipse for code complete with ExtJS and Prototype.

- Eclipse Help > Install New Software
- Link to Add: <http://download.aptana.com/studio3/plugin/install>
- Download the ExtJS 3.2.1 Library: <http://www.sencha.com/products/extjs/download/ext-js-3.2.1>
- Download the ExtJS sdocml file for Eclipse projects: <http://wiki.appcelerator.org/display/tis/JavaScript+Library+Support>



Use the 3.3.0 version as there is no 3.2.1 version.

Exercise 1: Create a Search Table

Objectives

After successfully completing this exercise, you will be able to:

- Create a Search page for searching objects in Windchill
- Transfer the parameters using JavaScript
- Build a MVC Table builder to display the search result

Scenario

In this exercise, you will create a Search page used for searching Windchill object: Part, and make the results displayed in a MVC builder Table.

Screenshots below is the expected output for this exercise.

PTC® Windchill®
Administrator

Services Academy

- Hello World
- My Product List
- Table Example
- Multi Component Example
- Tree Example
- **Search for Part**

PTC® Windchill®
Administrator

Part Name:
Part Number:

Part Table

Name	Number	Version	Last Modified	Context	State
WHEEL...	GC000035	A.1 (Design)	2015-10-13 21:34 U...	GOLF_C15-10-13 21:34 U...	In Work
TIRE	GC000036	A.1 (Design)	2015-10-13 21:34 U...	GOLF_CART	In Work
HUB_C...	GC000037	A.1 (Design)	2015-10-13 21:34 U...	GOLF_CART	In Work
RT_ARM	GC000038	A.1 (Design)	2015-10-13 21:34 U...	GOLF_CART	In Work
LT_ARM	GC000039	A.1 (Design)	2015-10-13 21:34 U...	GOLF_CART	In Work
WHEEL_ASSY	GC000040	A.1 (Manufacturing)	2015-10-13 21:34 U...	GOLF_CART	In Work
GOLF_CART	GC000001	A.1 (Manufacturing)	2015-10-13 21:34 U...	GOLF_CART	In Work
01-51368d.prt	WCDS000023	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-51296.prt	WCDS000105	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-2_piston_pin.prt	WCDS000045	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-512...	WCDS000...	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-2_cam_exhaust.asm	WCDS000494	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-31002.prt	WCDS000317	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-513...	WCDS000...	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-52003.prt	WCDS000110	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-513...	WCDS000...	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-51373.prt	WCDS000427	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-51251d.prt	WCDS000692	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work
01-512105.prt	WCDS000665	A.1 (Design)	2015-10-13 21:35 U...	Drive System	In Work

Part Search Table

Task 1: Create the Builder.

1. In Eclipse, Create a Java file `PartSearchBuilder.java` in the package `com.ptc.serviceAcademy.builders`.



As the package `com.ptc.serviceAcademy.builders` has already been registered, so when you put other classes in this package, there is no need to register again, and the method server will scan the whole package to find the builders.

Result:

```

package com.ptc.serviceAcademy.builders;

import static com.ptc.core.components.descriptor.DescriptorConstants.ColumnIdentifiers.CONTAINER_NAME;
@SuppressWarnings("deprecation")
@ComponentIdentifier("com.ptc.serviceAcademy.builders.PartSearchBuilder")
public class PartSearchBuilder extends AbstractComponentBuilder {
    @Override
    public Object buildComponentData(ComponentConfig config, ComponentParams params) throws Exception {
        String number = (String) params.getParameter("number");
        String name = (String) params.getParameter("name");
        if(name != null && number.length() > 0) number = number.trim();
        if(name != null && name.length() > 0) name = name.trim();
        QueryResult result = null;
        QuerySpec spec = new QuerySpec(WTPart.class);

        if (number != null && number.length() > 0) {
            spec.appendWhere(new SearchCondition(WTPart.class, "master>number", SearchCondition.LIKE, "%" + number + "%"));
        }
        if(name!=null&name.length()>0) {
            if (number != null && number.length() > 0) {
                spec.appendAnd();
            }
            spec.appendWhere(new SearchCondition(WTPart.class, "master>name", SearchCondition.LIKE, "%" + name + "%"));
        }
        LatestConfigSpec latestCSpec = new LatestConfigSpec();
        spec = latestCSpec.appendSearchCriteria(spec);
        result = PersistenceHelper.manager.find(spec);
        return result;
    }
    @Override
    public ComponentConfig buildComponentConfig(ComponentParams params) throws WTEException {
        ComponentConfigFactory factory = getComponentConfigFactory();
        ClientMessageSource messageSource = getMessageSource(RESOURCE);
        TableConfig table = factory.newTableConfig();
        table.setLabel(messageSource.getMessage("PART_TABLE_LABEL"));
        //make row in the table selectable
        table.setSelectable(true);
        table.addComponent(factory.newColumnConfig(NAME, false));
        //add columns
        table.addComponent(factory.newColumnConfig("smallThumbnail", false));
        table.addComponent(factory.newColumnConfig(NUMBER, false));
        table.addComponent(factory.newColumnConfig(VERSION,true));
        table.addComponent(factory.newColumnConfig(LAST_MODIFIED,true));
        table.addComponent(factory.newColumnConfig(CONTAINER_NAME, false));
        table.addComponent(factory.newColumnConfig(STATE, true));
        return table;
    }
}

```

PartSearchBuilder.java



Refer Lab Files for PartSearchBuilder.java code.

Task 2: Create the Search UI Page.

1. Create a JSP file *partSearch.jsp* in the folder location *codebase/netmarkets/jsp/serviceAcademy/*.
2. Draw a search page using HTML language.

Example:

```

<%@page language="java" session="true" pageEncoding="UTF-8"%>
<%@ taglib prefix="jca" uri="http://www.ptc.com/windchill/taglib/components"%>
<%@ taglib prefix="mvc" uri="http://www.ptc.com/windchill/taglib/jcaMvc"%>
<%@include file="/netmarkets/jsp/util/begin.jspf" %>



|                                                                                                                                                                                                                                                                                                     |  |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| Part Name: </td> <td> <input type="text" size="25px" id="partSearchName" name="partSearchName"> </td>                                                                                                                                                                                               |  |
| Part Number: </td> <td> <input type="text" size="25px" id="partSearchNum" name="partSearchNum" value="" onkeyup="javascript:this.value=this.value.toUpperCase();"> </td>                                                                                                                            |  |
| <td class="wizFormLabel"> <input type="button" size="15px" id="partSearchSubmit" name="partSearchSubmit" value="Search" onClick="submitDocSearch();"> &ampnbsp&ampnbsp&ampnbsp <input type="button" size="15px" id="partSearchClean" name="partSearchClean" value="Clean" onClick="clean();"> </td> |  |



<mvc:tableContainer compId="com.ptc.serviceAcademy.builders.PartSearchBuilder" height="600" />


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

```

partSearch.jsp

3. Fill the JavaScript functions.

Example:

```

<script>
    function submitDocSearch() {
        var ecnImg = document.getElementById('mvcTable');
        ecnImg.style.display="";
        submitParameters();
    }

    function submitParameters() {
        var number = document.getElementById('partSearchNum').value;
        var name = document.getElementById('partSearchName').value;
        var params = {
            number : number,
            name : name
        };
        //reload the table builder
        PTC.jca.table.Utils.reload('com.ptc.serviceAcademy.builders.PartSearchBuilder', params, true);
    }

    function clean(){
        //clear the textbox number
        var filetextNum = document.getElementById("partSearchNum");
        if (filetextNum.outerHTML) {
            filetextNum.outerHTML = filetext.outerHTML;
        } else {
            filetextNum.value = "";
        }
        //clear the textbox name
        var filetextName = document.getElementById("partSearchName");
        if (filetextName.outerHTML) {
            filetextName.outerHTML = filetext.outerHTML;
        } else {
            filetextName.value = "";
        }
    }
</script>

```



Refer Lab Files for partSearch.jsp code.

Refer the below link to enable the Java Code Validation and ways to configure JavaScript with eclipse using JSHint plugins.

<https://ssp.ptc.com/wiki/display/lms/Eclipse+Optimizations>

<https://ssp.ptc.com/wiki/display/lms/JSHint+Usage>

Task 3: Incorporate the Builder into the UI.

1. Update custom-actions.xml to create an action for the builder.

Result:

```

<objecttype name="object" class="java.lang.Object" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
    <action name="helloWorld">
        <command url="netmarkets/jsp/serviceAcademy/navigationlist.jsp" windowType="page"/>
    </action>

    <action name="myProductListTable">
        <component name="com.ptc.serviceAcademy.builders.MyProductListTableBuilder" windowType="page"/>
    </action>

    <action name="tableExample">
        <component name="com.ptc.serviceAcademy.builders.TableBuilder" windowType="page"/>
    </action>

    <action name="multiComponent">
        <component name="com.ptc.serviceAcademy.builders.MultiComponentBuilder" windowType="page"/>
    </action>

    <action name="treeExample">
        <component name="com.ptc.serviceAcademy.builders.TreeBuilder" windowType="page"/>
    </action>
    <action name="partSearch">
        <command url="netmarkets/jsp/serviceAcademy/partSearch.jsp" windowType="page"/>
    </action>
</objecttype>

```

Additions on custom-actions.xml



Refer Lab Files for custom-actions.xml.

2. Update Action Framework custom-actionModels.xml.

Result:

```

<model name="serviceAcademy navigation" resourceBundle="com.ptc.serviceAcademy.resource.NavigationRB">
  <action name="helloWorld" type="object"/>
  <action name="myProductListTable" type="object"/>
  <action name="tableExample" type="object"/>
  <action name="multiComponent" type="object"/>
  <action name="treeExample" type="object"/>
  <action name="partSearch" type="object"/>
</model>

```

Additions on custom-actionModels.xml

3. Update the Resource Bundle in the two files: *NavigationRB.java* and *NavigationRB_en_US.java*.

Example:

```

@RBEEntry("Search for Part")
public static final String OBJECT_PARTSEARCH_DESCRIPTION = "object.partSearch.description";

@RBEEntry("Search for Part")
public static final String OBJECT_PARTSEARCH_TITLE = "object.partSearch.title";

@RBEEntry("Search for Part")
public static final String OBJECT_PARTSEARCH_TOOLTIP = "object.partSearch.tooltip";

```

Additions on the two RB files



Refer Lab Files for NavigatonRB and NavigationRB_en_US java files.

Task 4: Deploy and Test your code.

1. Deploy code using lightBuild and restart the Method Server.
2. Check your results.

Completion Criteria

1. The second tab displayed in the **Navigator >ServiceAcademy**.
2. The search conditions are displayed in the table, when the search button clicked, the result table displayed in the page.
3. All Students taking the GS-00663 - Client Customization Workshop after this course are expected to retain the same Virtual Machine (with exercises performed in GS-00338) for the Workshop (GS-00663).

This completes the exercise.