Type of Submission: Tutorial
Title: IBM Case Manager Custom Widgets
Subtitle: Creating custom widgets with the IBM Case Manager JavaScript API

Author: Mayes, Lauren
Job Title: STSM, Product architect, Case Manager and Enterprise Content Management
Email: lmayes@us.ibm.com
Company: IBM

Author: Tian, Xiao Ji
Job Title: Advisory Software Engineer, Case Manager and Enterprise Content Management
Email: tianxj@cn.ibm.com
Company: IBM

Contributor: Shenoy, Anuradha S.
Job Title: Lead developer, IBM Case Manager Client
Email: shenoya@us.ibm.com
Company: IBM

Contributor: Guo, Ming Liang
Job Title: Senior Software Engineer, Case Manager and Enterprise Content Management
Email: guoml@cn.ibm.com
Company: IBM

Job Title: Software developer, IBM Case Manager Client
Company: IBM

Contributor: Gao, Ying Ming
Email: guoyingm@cn.ibm.com

Contributor: Depuy, Stephane
Email: sdepuy@us.ibm.com

Contributor: Liu, Johnson
Email: jrliu@us.ibm.com

Contributor: Lou, Chad
Email: chadlou@us.ibm.com

Contributor: Rozhdestvensky, Eugene
Email: eugener@ca.ibm.com

Contributor: Wang, Wei
Email: wangweiu@cn.ibm.com

Contributor: Xu, Shao Hong
Email: xushaoh@cn.ibm.com

Contributor: Wang, Qi
Email: wangqqi@cn.ibm.com

Contributor: Hsieh, Wen-Chien
Job Title: Architect, IBM Case Manager
Email: shsieh@us.ibm.com
Company: IBM

Contributor: Mosher, Frankie
Job Title: Information developer, IBM Case Manager Client
Email: fmosher@us.ibm.com
Company: IBM

Abstract: This tutorial steps you through creating a custom widget package for IBM Case Manager 5.2.1. In this tutorial, you will learn how to create page widgets and actions, customize IBM Case Manager page widget, and package and deploy your custom page widgets.

Introduction

You can customize Case Manager Client by creating your own page widgets. In addition, you can customize the page widgets that are provided by IBM Case Manager to tailor them to your needs.

Audience

This tutorial is intended for developers who create Case Manager Client applications and for business analysts who design case management solutions.

Prerequisites

Before you begin this tutorial, you should be familiar with the programming model for Case Manager Client. For an overview of this model, see the developerWorks article Introduction to the IBM Case Manager JavaScript API.

You should also be familiar with the concepts and programming models that are used to build plug-ins for IBM Content Navigator. For more information, see Developing applications with IBM Content Navigator in the IBM FileNet P8 Version 5.2.1 Information Center.

To develop a widget, you must know how to use the following technologies:

- Dojo (Version 1.9.3 is packaged with IBM Content Navigator 2.0.3)
- Extensible Markup Language (XML)
- HyperText Markup Language (HTML)
- Java 2 Enterprise Edition (J2EE)
- JavaScript
- Eclipse

Downloading the sample widget package

The lessons in this tutorial are based on a custom widget package that is available in the customWidget.zip file provided with this article. The tutorial uses the sample widget package to illustrate how you build, package, and use custom page widgets and actions.

The sample package includes two actions and three widgets:

Name	Description
CustomAddCaseAction	This action adds a case from an arbitrary solution and case type. It shows how you can build a custom action based on an existing IBM Case Manager action base class and action definition file.

Name	Description
CustomAddToAttachmentAction	This action adds a case document as an attachment. It shows how a custom action interacts with the action context to retrieve the context information and publish an event.
CustomPageWidget	This simple page widget shows the basic definition file format and the source code structure for defining a page widget. This widget also shows how to create a toolbar and a pop-up menu for a widget.
CustomSearchWidget	This page widget dynamically loads the case properties to search on using search API. It only searches on single value properties, not multi-value properties. It also does not search on date time fields.
CommentWidget	This page widget shows how to embed the Comment dialog box in a page widget. The widget coordinates with Case Toolbar page widget to save a case comment when the user saves a case. The widget also supports marking or clearing the dirty state of the page as case comments are added and saved.
CustomCaseInfo	This page widget provides a custom Case Information page widget that uses the CustomAddToAttachmentAction action. The widget shows how to specify a model object as part of the action context.
CustomViewer	This page widget shows how to customize an IBM Case Manager page widget.

The sample package uses the following naming conventions:

Name	Description
ICMCustomWidgets	The name of the package
ICMCustomPlugin	The name of the IBM Content Navigator plug-in
ICMCustomWidgets	The name of the web application
com.ibm.icm.extension.custom	The common package name of the Java class
icm/custom	The common package name for custom Dojo module

IBM Case Manager 5.2.1 Custom Widget Best Practices

IBM Case Manager Page widgets are essentially Dojo based widgets also known as dijits that run inside of a Dojo based Page container framework. A normal Dojo dijit is a widget that is part of Dojo's User Interface (UI) library. In addition to the standard UI features, IBM Case Manager Page widgets offer two things:
- Events that can be used for wiring
- Widget settings

These two features allow the Business Analyst to dynamically adjust page and widget behavior with little to none programming skills.

The following guidelines should be kept in mind when creating a Page widget:

UI dijit

- First create a normal Dojo dijit which is responsible for rendering the user interface and handling user interaction to collect user input.
- Ideally, this dijit should be a designed as a reusable component that takes in a context and renders itself.
- The UI dijit should not access any IBM Case Manager Page Container related APIs, which will limit its usage to only the IBM Case Manager Case Client.
- Separating the UI dijit from the Page widget layer will allow you to reuse this dijit in a custom application or plug-in built on top of IBM Content Navigator.
- The dijit could also be reused in a non-ECM environment if it doesn't use any ECM related API.
- The dijit should also expose necessary dojo methods/events to allow interested parties (the page widget wrapper or other custom widgets) to exchange data or get notified by user interaction.

Page widget

- After the dijit corresponding to the user interface is ready, you will need to create a Page widget wrapper that extends this UI dijit and converts it into an IBM Case Manager Page widget.
- The page widget wrapper should be the place where the page container related code resides.
- The page widget wrapper accepts widget settings from the page container and passes them to the underlying dijit.
- It should also connect to Dojo events exposed on the dijit to be notified of user actions.

Event Listener (Optional)

As an optional separation, you could separate the event handling logic from the page widget when the event handling gets too complex. For e.g. if you have complex logic to

pre-process or post-process the event data, you can create a separate event listener object which holds the event handler implementations. The page widget could then delegate the event handling logic to the event listener object.

Terms of use

When developing or extending custom widgets in IBM Case Manager, avoid referencing the IBM Case Manager Case Client JavaScript code base directly or copying the Case Client JavaScript code. It is recommended to follow the IBM Case Manager JavaScript API and utilize public functions there. Private functions from the IBM Case Manager Case Client code can change between releases and will cause maintainability and functionality issues for your custom widgets.

When using public functions in your custom widgets, you can create your custom widget as a subclass to the IBM Case Manager out of the box page widgets. Only public methods can be overwritten within the subclass and as needed, call the parent class inherited function accordingly. You may overwrite the the method if full customization is needed.

Steps for creating custom page widget and actions

1. Create the web project.

    In this step, you create a web project for the custom page widget and actions. The web project defines the folder structure that is required for building, packaging and registering your custom widget package.

2. Create the registry files for the widget package.

    In this step, you create the files that are needed to register the page widgets and make them available for use in Case Manager Builder.

3. Create an IBM Content Navigator plug-in for the widget package.

    In this step, you create the classes and files that are needed to use your custom page widgets in Case Manager Client, which runs in IBM Content Navigator. These files also make any custom actions available in Case Manager Builder, Case Manager Client, and IBM Content Navigator.

4. Implement the custom page widgets and any custom actions that are used by the widgets.

    In this step, you create the JavaScript files that implement the functions performed by your custom page widgets and actions.

5. Package the page widgets and actions.

    In this step, you create a package that contains the files that are required to deploy your custom widgets.

6. Deploy the widget package.

    In this step you use the IBM Case Manager configuration tool to deploy your custom widget package

7. Add the custom page widget and actions to page

    In this step, you add the custom page widgets and actions into a page in Page Designer.

8. Deploy the solution

    In this step, you save the page and deploy the solution in Case Manager Client.

9. Test your custom widgets in Case Manager Client.

    In this step, you open your solution and work with your custom widgets to determine if changes are needed.

Lesson 1: Create the web project

In this lesson, you create the primary project and folder structure for your custom widget package. This folder structure mirrors the structure that is required for packaging your custom widgets:
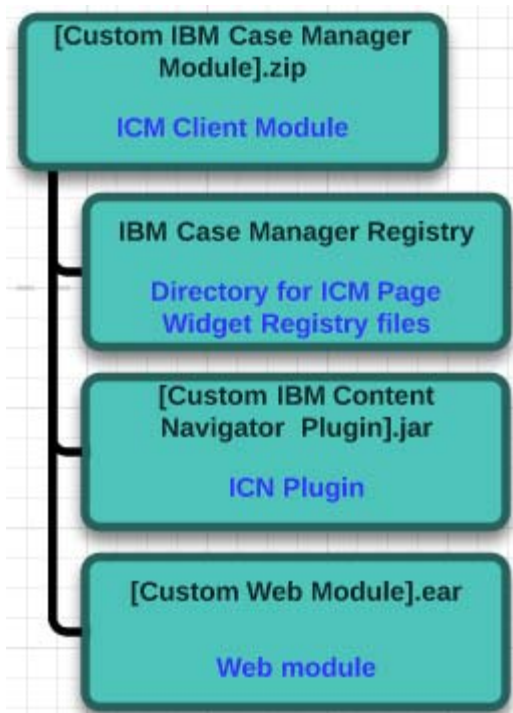
For the sample custom widget package, the project and folder structure will be as follows:

To build the web project for the ICMCustomWidgets package:

1. In your chosen development tool, create a new project that is named

2. Create the following subprojects and folder in the customWidget project:

| Subproject or folder	Description
ICMCustomPlugin	This project includes the code that is used to register the custom actions and bootstrap the runtime JavaScript code in IBM Content Navigator, You can name this project as desired.
ICMCustomWidgets	This project is used for the web module and includes the runtime code for the custom page widgets and actions. You can name this project as desired.
ICMRegistry	This folder includes the manifest file that is used to register the custom page widgets in Page Designer.

	You must name this folder .

Lesson 2: Create the registry files for the widget package

In this lesson, you create the files that are used to register your custom page widgets in Page Designer. Registering the page widgets causes them to be displayed in the Page Designer palette.
This lesson also discusses the auxiliary files that you can include. These files include the localized resource files and the image files that are used for the page widget.

The registry files are created in the `ICMRegistry` folder using the following structure:
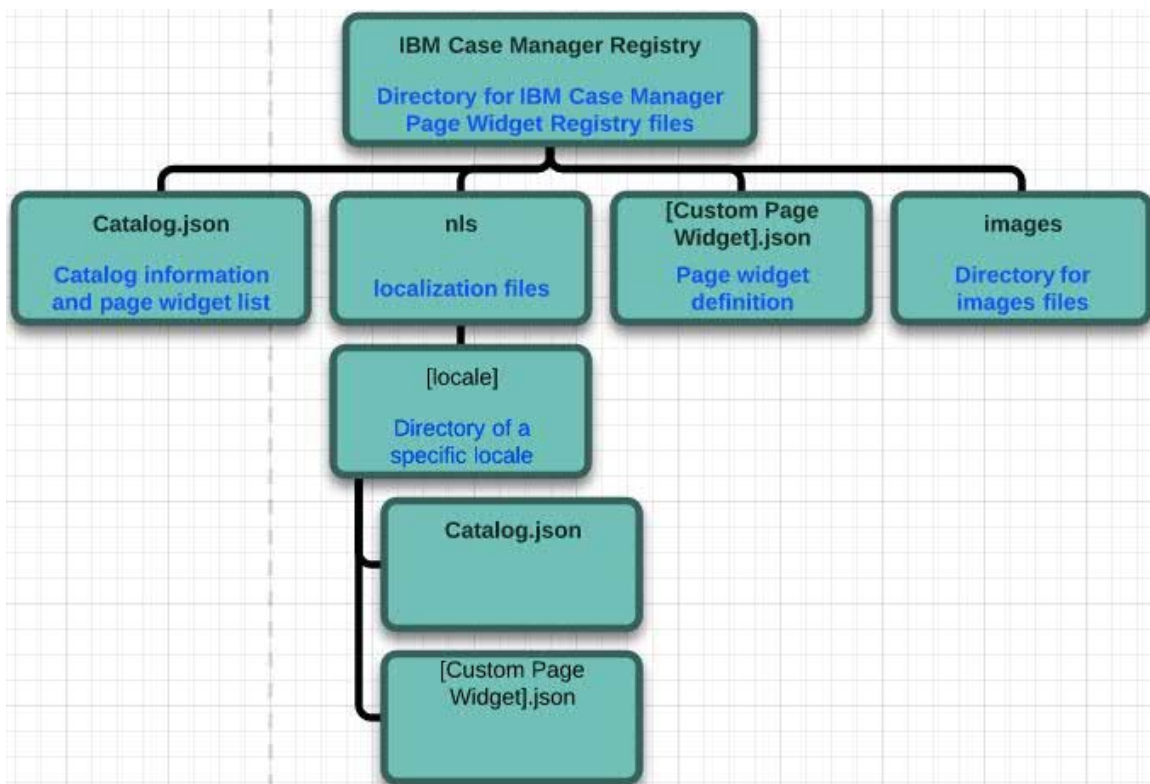
Creating the Catalog.json file

The Catalog.json file provides a description of your custom widget package and lists the widgets that the package contains.

The Catalog.json file has the following structure:

Property	Required or Optional	Type	Description
Name	Required	String	The name for your custom page widget package. You must specify a unique name for the package to avoid overriding an existing page widget package.
Description	Required	String	A description of the custom page widget package.
Locale	Required	String	The code that identifies the locale for the current catalog. This code must correspond to the subfolder name in the ICMRegistry/nls folder where the Catalog.json and widget definition file for the locale are located. If this is the default locale, set the value to "".
Version	Optional	String	The version number that is assigned to the widget package.
Categories	Optional	String	The categories in Case Manager Builder in which the custom page widgets in this package are listed. You can choose to list the page widgets in one of the following categories, which are provided by IBM Case Manager: • CaseWidgets • GenericWidgets For each category, you must provide an identifier and title.

Creating the page widget definition file

You create a JSON definition file to define the properties and events for the page widget.

Creating a JSON definition file involves the following steps.

1. For each page widget in the package, enter the following properties:

    Tip: These values are also used for the page widget in the Catalog.json file. For consistency, you can copy the values from this file into the Catalog.json file. If a value does not match, IBM Case Manager uses the value from the Catalog.json file.

Property	Required or Optional	Type	Description
id	Required	String	A unique identifier for the page widget.
category	Required	String	The identifier of the category in which the page widget is to be listed in Case Manager Builder.
title	Required	String	The name to be displayed for the page widget in Case Manager Builder.
description	Required	String	A description of the page widget.
definition	Required	String	The full path and name of the definition file for the page widget.
preview	Required	String	The relative path and name of the resource file that contains the preview image for the page widget. The image can be a .png file or a .gif file. This image is not used in IBM Case Manager V5.2.1.
icon	Required	String	The full path and name of the resource file that contains the icon for the page widget. The image can be a .png file or a .gif file. This image is used in the widget palette in Page Designer.
runtimeClassName	Required	String	The class name for the page widget as specified in the run-time plug-in for the widget package.
previewThumbnail	Required	String	The full path and name of the resource file that contains the thumbnail image for the page widget. The image can be a .png file or a .gif file. This image is not used in IBM Case Manager V5.2.1.

Property	Required or Optional	Type	Description
properties	Required	Array	An array that defines the properties that can be set for the page widget in Case Manager Builder.
events	Required	Array	An array that identifies the events that the page widget publishes and subscribes to.

2. Define the properties that can be set for the page widget when the user adds the widget to a page. These properties include the toolbars, menus, and actions that can be added to the widget.

3. Define the events that the page widget subscribes to and publishes. For each event, provide the following properties:

Property	Description
id	The identifier for the event.
title	The title that is displayed in the Wiring dialog box for the event.
functionName	For an event that the page widget subscribes to, the name of the function that handles the event.
type	For an event that the page widget publishes, the type of event. Set the type to Broadcast if the widget broadcasts the event to other widgets on the page. Set the type to Wiring if the event must be wired to another widget on the page.
direction	Set to either Subscribed or Published.
description	The description of the event. This description is displayed in the hover help for the event in the Wiring dialog box.

For example, the CommentWidget.json file contains the following events for the Comment Page Widget:

```
"events":[
    {
        "id": "icm.SelectCase",
        "title": "Select case",
        "functionName": "handleICM_SelectCaseEvent",
        "direction": "subscribed",
        "description": "Display the case information for the
case that is contained in the event payload."
```

```
            },
            {
                "id": "icm.SendCaseInfo",
                "title": "Send case information",
                "functionName": "handleICM_SendCaseInfoEvent",
                "direction": "subscribed",
                "description": "Display the case that is specified in
        the event payload."
            }
        ]
```

Lesson 3: Create an IBM Content Navigator plug-in for the widget package

In this lesson, you create the plug-in that makes your custom page widgets and actions available within IBM Content Navigator for Case Manager Client.

Using an IBM Content Navigator Plug-in is the standard way to extend the base function of the product. By building a custom plug-in, you can add widgets, actions, and services to IBM Content Navigator.

To create the plug-in package, you combine the classes and JavaScript files into a JAR file. The following graphic shows the JAR file that contains the directories and files that are needed for the custom page widget and actions:
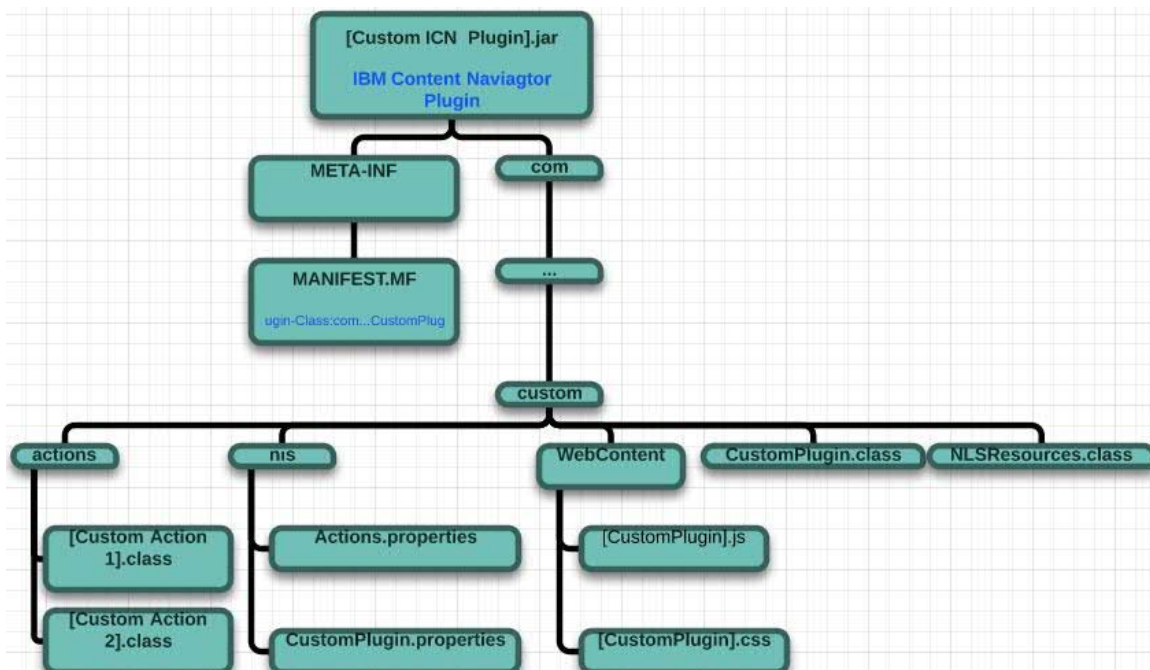
The CustomPlugin.class describes the custom plug-in. The class provides the initial JS file name for the plug-in, as well as the action list provided by the package.

To declare the plug-in class to IBM Content Navigator, add the plug-in class to the META-INF\MANIFEST.MF as follows:

```
Plugin-Class: com.ibm.icm.extension.custom.ICMCustomPlugin
```

The WebContent\[Custom Plugin].js file registers the Dojo module path for the custom runtime code. Depending on whether debug mode is on or off, the [Custom Plugin].js file can load in the source code for debugging or in the combined and compressed code for better performance.

Each custom action class in the actions folder also provides an action definition.

The Java class com.ibm.icm.extension.custom.ICMCustomPlugin is the single entry point of the sample IBM Content Navigator plug-in. This class declares the following files and data that can be loaded by IBM Content Navigator:

JavaScript file

The initial JavaScript file for the sample plug-in is ICMCustomPlugin.js. You can use the getScript() method to get this file. This JavaScript file performs the following steps to bootstrap the custom widget at runtime:

1. Load the CSS files used by the custom runtime code for the page widget and action.

2. Register the Dojo module path `icm/custom` for the custom runtime code.

ICM Content Navigator loads and evaluates the plug-in JavaScript file before the user sees the login page.

If you want to combine the JavaScript code to improve performance, consider whether you need to use debug mode. (The mode is also applicable to IBM Case Manager.) To run in debug mode, set the parameter debug=true is set in the URL. In this mode, the JavaScript file requires the minimum runtime Dojo modules that are necessary to bootstrap the custom code. If you run in non-debug mode, the JavaScript loads the entire set of compressed custom runtime code.

Custom actions

The custom IBM Case Manager actions list contains the PluginAction instances that define client-side actions provided by this plug-in. You can get this list by using the getActions() method.

The IBM Content Navigator action definition is exposed through the action Java class, providing action definitions such as the action ID, icon, and the action model class. IBM Case Manager action definitions are IBM Content Navigator actions that have been extended. The actions have additional definition information exposed through the getAdditionalConfiguration() method.

The getActions() method returns a JSON string, which enables the Page Designer to use the JSON definition and show actions in the page widget menu setting.

The first action is the CustomAddCaseAction. The following example shows the JSON definition of the action:

The JSON definition is used by Page Designer to build the configuration panel of the action.

When you design a page and add the **Add Custom Case** action to the toolbar, the following configuration pane for the action displays:

You can create a button label for the action, for example, **Launch Fraud Investigation Case**, which can later be localized for different languages. You can also enter the solution ID and Case Type for the runtime code to access.

Tip: In order for it to work at runtime, make sure to enter the Case Type field with the proper prefix of the solution. Note that 'CYJBY5' above is the name of the solution and 'CJ5' above is the corresponding solution prefix.

In addition, the **Open Add custom Case Page** event published by the action is exposed through the page widget that hosts the event. For example, after you add the action to the toolbar page widget, the event displays in the toolbar page widget's event wiring dialog. You can wire the event to the page container's Add Case event handler for opening the Add Case page.

The second custom action is CustomAddToAttachmentAction. This action is another custom action for adding a document as an attachment. You can add this action to the Case Document widget, and place it in the work detail page. This enables users to add a case document as an attachment.

The Java class com.ibm.icm.extension.custom.actions.CustomAddToAttachmentAction provides the action definition. The additional IBM Case Manager action definition is seen in the following example:

This action does not have additional configuration items. It can publish the "Add document as attachment" event which includes the attachment id and the selected document model object in payload. Then, you can wire the action to the Attachment page widget event handler to enable adding the selected document as an attachment.

To package the plug-in:

1. Create a JAR file that contains the files that are used for the plug-in.

   The JAR file must use the following structure:

   ```
   packageName
       // Include all the Java classes that extend the Plugin class
   and
       // any other Java classes that are required by the plug-in
       Java class 1
       Java class 2
       ...
       WebContent
       // Include all the image, JavaScript, and HTML files
       // that are required by the plug-in
   WEB-INF
       manifest.mf
   ```

   In the packageName folder, include the Java™ class files that are defined for the plug-in. Create the subfolders based on the package names of the Java classes. Follow the standard JAR file structure by using nested folders for nested packages.

   Important: All the Java classes must be included in a named package. You cannot use the default package.

2. Modify the manifest.mf file for your plug-in to include the following property:

   ```
   Plugin-Class: pluginClassName
   ```

   pluginClassName is the name of your Plugin.java subclass.

   You can also include any of the standard properties for a JAR file in the manifest.mf file.

3. Deploy the JAR file into your web application server.

   Important: The plug-in JAR file must be available on a URL addressable web application server or the plug-in will not work in the web client.

4. Use the IBM Content Navigator administration desktop to add the new plug-in.

Lesson 4: Implement the page widgets and actions

In this lesson, you implement your custom page widget and the actions that it uses. This lesson uses the CustomPageWidget widget to illustrate how you implement a page widget.

The following graphic shows the recommended structure for a page widget:
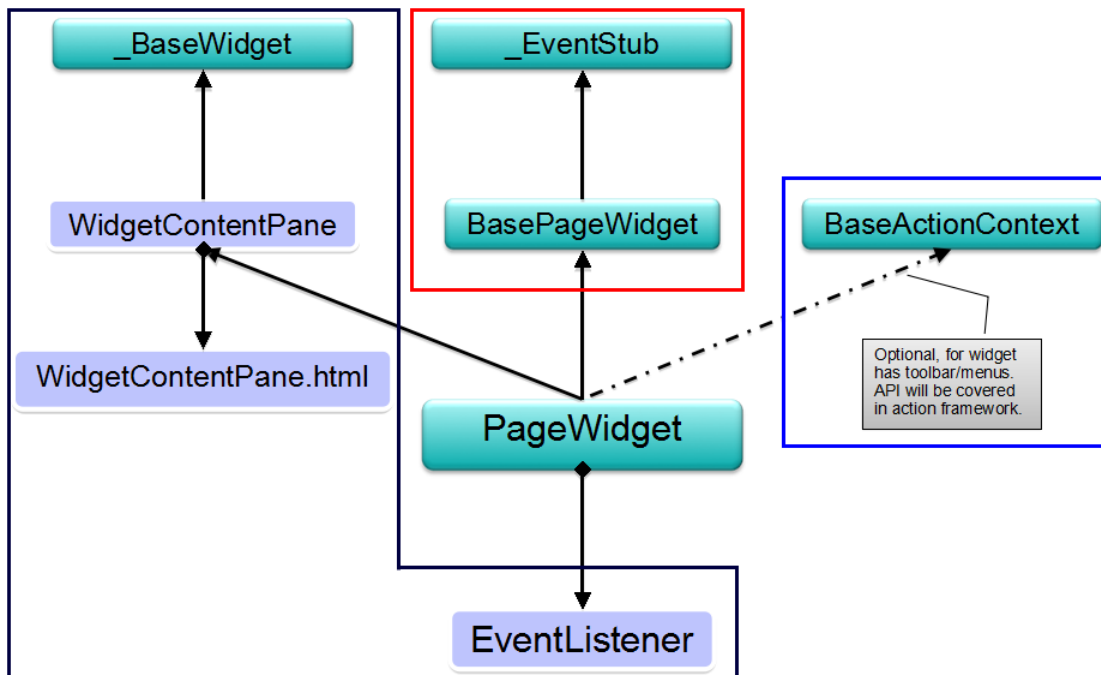
In this graphic, the classes in the black box implement the user interface for the page widget and handle the user's interaction with the interface.
The classes in the red box are base IBM Case Manager classes from which the custom page widget inherits functionality. The icm.base._BaseWidget class, which provides functions that display the widget description and show or hide the content pane.

The icm.base._EventStub class provides functions for publishing and broadcasting methods.

The class in the blue box is the base class for any page widget that hosts a toolbar or a pop-up menu.

To create the CustomPageWidget widget:

1. Create a JavaScript file that implements the WidgetContentPane class that is used to draw the user interface for the CustomPageWidget widget. This class:

- Inherits from the icm.base._BaseWidget class
- Implements a destroy function that cleans up the user interface for the page widget when the page is closed

2. Create a JavaScript file that implements the CustomPageWidget. This file:

- Inherits from the following classes: CustomWidgetContentPane, BasePageWidget, and BaseActionContext
- Creates the menu and toolbar for the custom page widget
- Adds an event handler for the custom page widget

Lesson 5: Package the page widgets and actions

In this lesson, you create three Apache Ant build script to create the package that contains the files that are required to deploy your custom widgets.

To create the build script for the custom widget package:

1. In Eclipse, open the Package Explorer

2. Right-click your project and click **New** > **File**.

3. In the New File window, type `build.xml` as the file name

4. Type the following information:

```xml
<?xml version="1.0" encoding="utf-8"?>
<project name="ICM Client Build" default="all" basedir=".">

<target name="clean">
   <delete>
      <fileset dir=".">
         <include name="*.jar" />
      </fileset>
   </delete>
</target>

<target name="buildPlugin">
   <javac destdir="./bin" srcdir="./src" debug="on">
      <classpath>
         <pathelement location="./lib/navigatorAPI.jar" />
      </classpath>
   </javac>
   <copy todir="./bin" overwrite="yes">
      <fileset dir="./src">
         <include
name="com/ibm/icm/extension/custom/WebContent/**/*.*" />
      </fileset>
   </copy>
   <jar destfile="./ICMCustomPlugin.jar" manifest="./src/META-
INF/MANIFEST.MF" update="true">
      <fileset dir="./bin">
         <include name="**/*.*" />
      </fileset>
   </jar>
</target>


<target name="all" depends="clean,buildPlugin"></target>
</project>
```

5. Save this file in the ICMCustomPlugin folder.

6. Right-click your project and click **New** > **File**.

7. In the New File window, type `build.xml` as the file name

8. Type the following information:

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

```
<project name="ICM Client Build" default="all" basedir=".">

<target name="clean">
     <delete>
          <fileset dir=".">
                 <include name="*.war" />
                 <include name="*.ear" />
          </fileset>
     </delete>
</target>

<target name="createWAR">
    <war destfile="./ICMCustomWidgets.war"
webxml="./WebContent/WEB-INF/web.xml">
        <fileset dir="./WebContent">
              <include name="**/*.*" />
        </fileset>
    </war>
</target>

<target name="createEAR">
     <ear destfile="./ICMCustomWidgets.ear"
appxml="./WebContent/META-INF/application.xml">
            <fileset dir="." includes="*.war"/>
     </ear>
</target>

<target name="all" depends="clean,createWAR,createEAR"></target>
</project>
```

9. Save this file in the ICMCustomWidgets folder.

10. Right-click your project and click **New** > **File**.

11. In the New File window, type `build.xml` as the file name

12. Type the following information:

```
<?xml version="1.0" encoding="utf-8"?>
<project name="ICM Custom Widget Build" default="all"
basedir=".">
<property name="plugin.home" value="../ICMCustomPlugin" />
<property name="webapp.home" value="../ICMCustomWidgets" />
<target name="clean">
     <delete>
          <fileset dir=".">
                 <include name="*.jar" />
                 <include name="*.zip" />
          </fileset>
     </delete>
</target>

<target name="package">
        <ant antfile="${plugin.home}/build.xml" >
```

```
            <property name="basedir" value="${plugin.home}"/>
             </ant>

          <ant antfile="${webapp.home}/build.xml">
          <property name="basedir" value="${webapp.home}"/>
          </ant>
       <zip destfile="../ICMCustomWidgets.zip">
              <fileset dir="${plugin.home}">
                    <include name="*.jar" />
              </fileset>
              <fileset dir="${webapp.home}">
                    <include name="*.ear" />
              </fileset>
              <zipfileset dir="../ICMRegistry"
prefix="ICMRegistry">
              </zipfileset>
        </zip>

</target>


<target name="all" depends="clean,package"></target>
</project>
```

13. Save this file in the build folder.

14. Run build

    a. Expand the Build folder and right-click the build.xml file.
    b. Select **Run As** and **Ant Build**.

The build generates an ICMCustomWidgets.zip file in the project folder.

Lesson 6: Deploy the widget package

In this lesson, you deploy the widget package to make your custom widget available in Case Manager Builder.

To deploy the widget package:

1. In the Case Manager configuration tool, right-click the profile and then click **Add New Task** > **Deploy and Register Widgets Package** to create a new task.
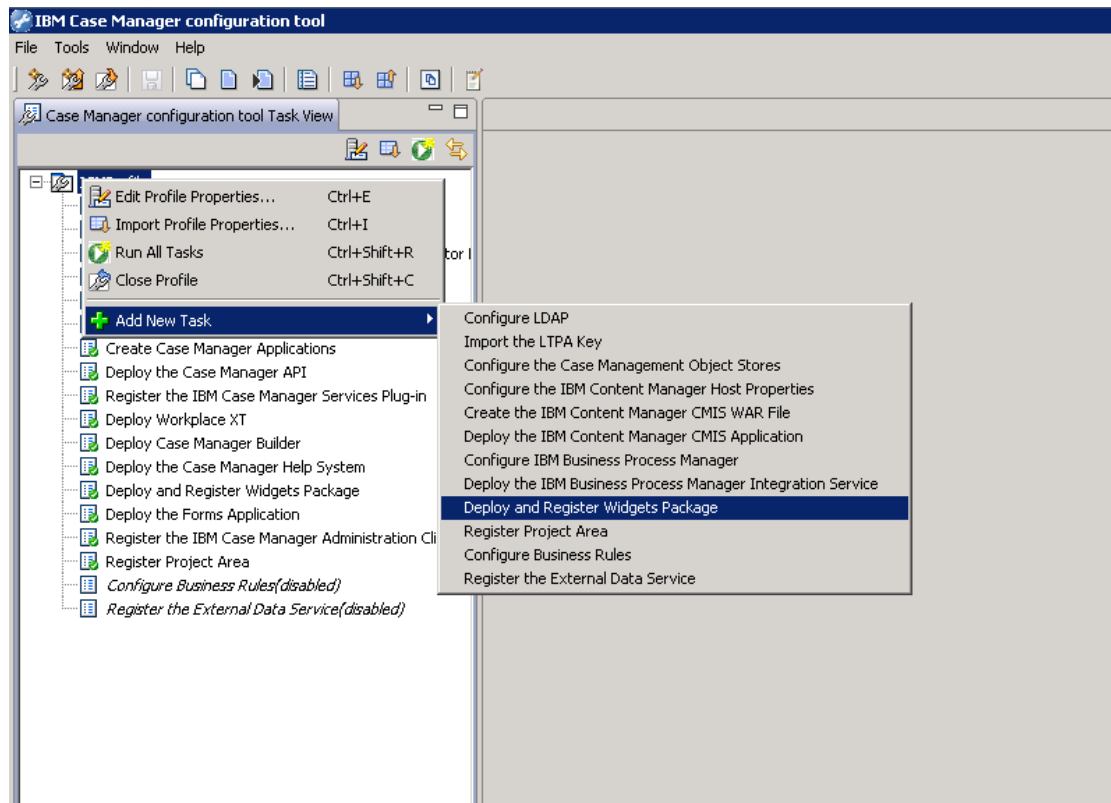
2. Add the custom widget package file to Widgets package file path, run the task.
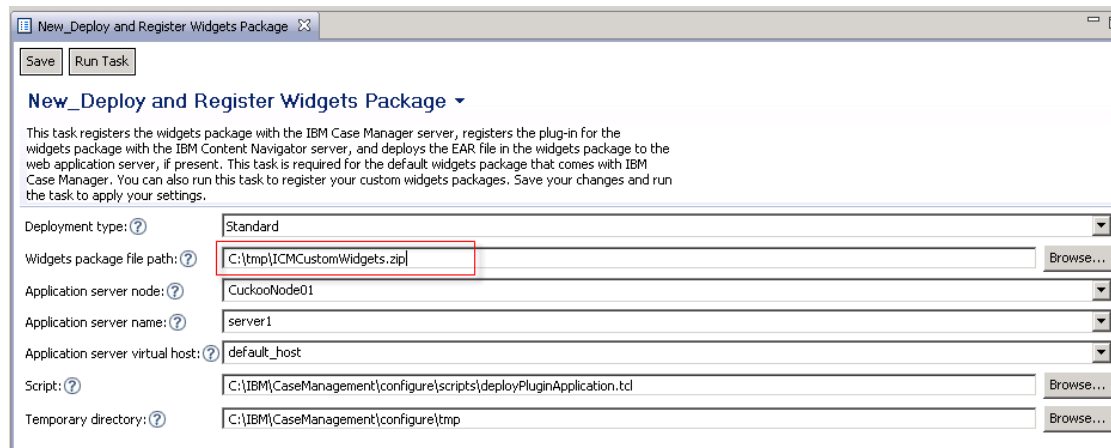
Lesson 7: Add the page widget and actions to a page

In this lesson, you add the Custom Case Information page widget to the Work Details page.

1. Add the Custom Case Information page widget to the page.
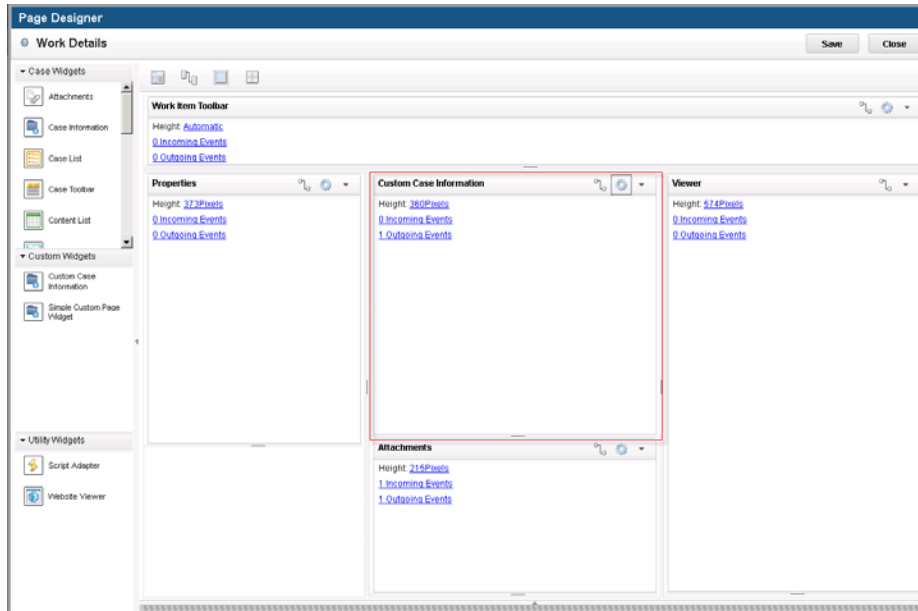
2. Disable the configuration of the 'Comments are read only'
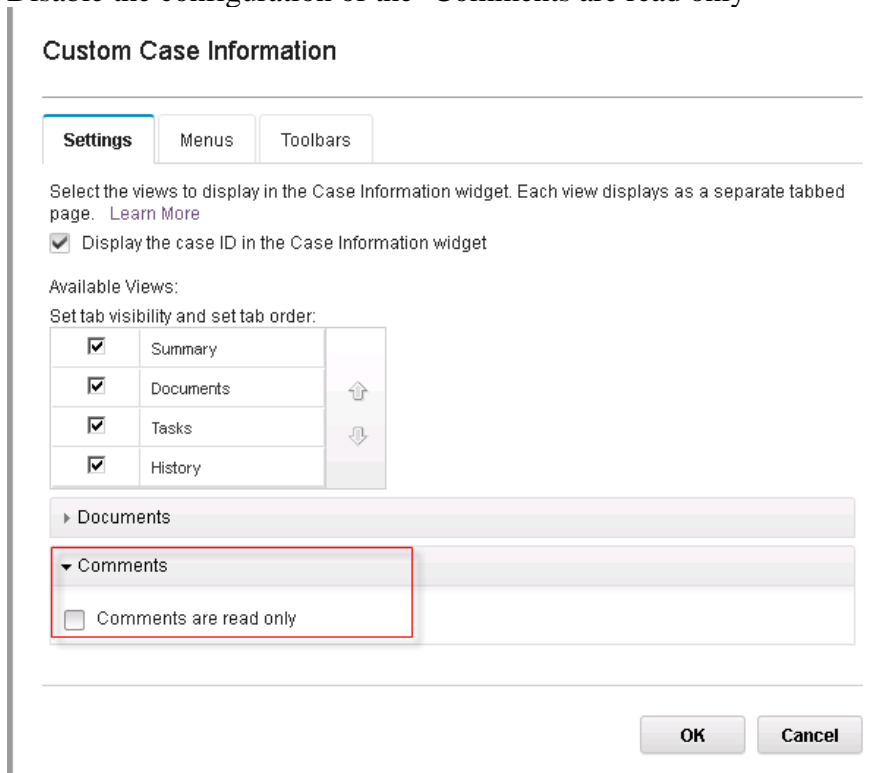
3. Add the AddDocumentAsAttachment action to document view as context menu, give the menu a name of "Add Document As Attachment"

4. The menu build by the AddDocumentAsAttachment action, where "a1" is the
   name of the attachment

5. Wire the 'Add document as attachment' event to attachment page widget's 'Add
   document attachment'

Tip: Note that the Source of the wire is 'Custom Case information', and its Outgoing event is '*Add document as attachment'. You may try skipping this step, and after deployment, you will see that there is no action triggered, until after the above wiring is established and the solution is redeployed.

Lesson 8: Deploy and test the solution

1. On the Manage Solutions page, find your solution and click **Deploy**. The status icon turns green as the solution is deployed to your development environment. When the solution deploys successfully, the status icon remains green and displays a check mark.

   Tip: If a solution fails to deploy, the status icon turns red and displays an X. You can view errors by clicking **More Actions** > **Errors**.

2. After the solution deploys, click **Test** and log in to Case Manager Client.

   The first time you test a solution, you are not a member of a role that has access to the solution so you must assign yourself to a role.

3. Verify the behavior of your custom page widgets and actions.

Appendix

Extending an IBM Case Manager page widget

In addition to creating custom page widgets, you can customize the page widgets that are provided by IBM Case Manager. This example extends the Case Information widget to show this type of customization.

In this example, we make the following changes to the Case Information widget:

- Add a new tab
- Add a custom action with required the context
- Participate page widget coordination

The following diagram shows the code structure of case information page widget.
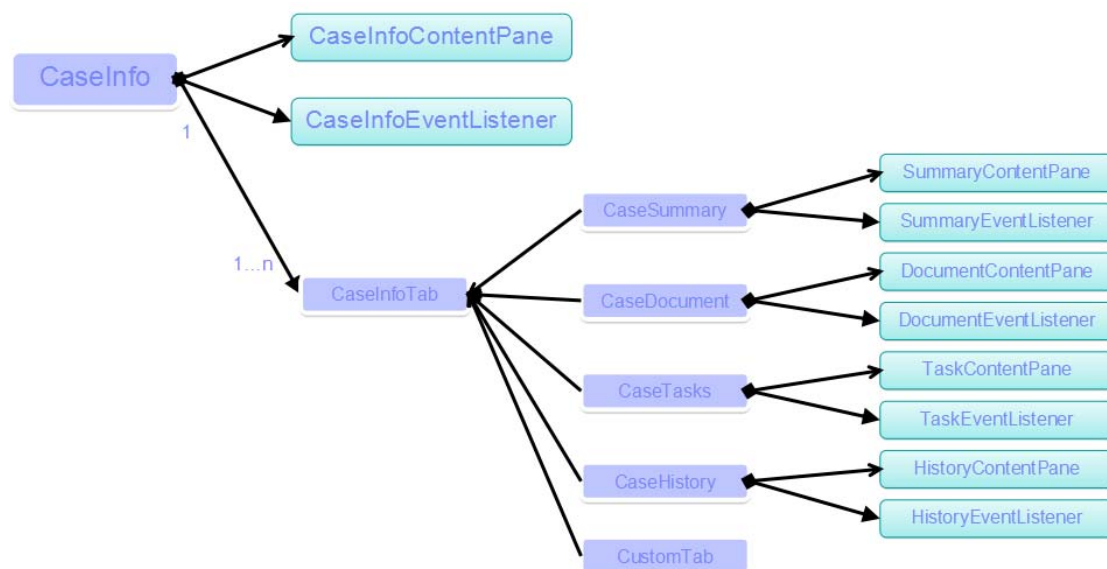
To add a tab to the Case Information widget:

1. Create a dijit to display case comment in a tab of the Case Information widget. This dijit must inherit from the icm/pgwidget/caseinfo/dijit/CaseInfoComponentContentPane, which is an abstract template dijit that defines the contracts between the Case Information widget and the dijit that is embedded in its tab.

   The CaseInfoComponentContentPane class represents a tab component used in Case Information widget. Inheriting from the CaseInfoComponentContentPane dijit enables the custom tab dijit to use a case model object inject from the Case

Information widget. It also provides a context object to enable the dijit to access the configuration for the Case Information widget and an event stub instance for publishing events. This means that the custom tab dijit need only implement the render() method to render the tab user interface.

```
CaseInfoCommentsContentPane.js  ×

 1   define(["dojo/_base/declare",
 2           "dojo/text!./templates/CaseInfoCommentsContentPane.html",
 3           "icm/pgwidget/caseinfo/dijit/CaseInfoComponentContentPane",
 4           "icm/dialog/addcommentdialog/dijit/CommentContentPane",
 5           "icm/base/Constants",
 6           "dojo/_base/array",
 7           "icm/base/Constants"],
 8           function(declare, template, CaseInfoComponentContentPane, CommentContentPane, Constants, array, Constants) {
 9
10   return declare("icm.custom.customcaseinfo.CaseInfoCommentsContentPane", CaseInfoComponentContentPane, {
11
12           templateString: template,
13
14           // Set any required data
15           setModel: function(model) {
16               this.inherited(arguments);
17               this.render(model);
18           },
19
20           //Render the tab dijit
21           render: function(model) {
22
23               if (!this.isInFocus() || !model || !model.payload || this.commentConte
24                   return;
25               }
26               this.inherited(arguments);
27
28               var caseObj = model.payload.caseEditable.getCase();
29               var coord = model.payload.coordination;
30               var readonly = this.context.widgetProperties.commentsAreReadOnly;
31               var self = this;
32
33               var context = {
34                   "artifactType": "Case",
35                   "artifactLabel": caseObj.getCaseTitle(),
36                   "commentContext": Constants.CommentContext.CASE,
37                   "caseModel": caseObj
38               };
39               this.commentContentPane = new CommentContentPane(context);
40
41               if(readonly){
42                   this.commentContentPane.commentText.disabled = true;
43               }else{ ▦|
77               }
78
79               this.contentNode.appendChild(this.commentContentPane.domNode);
80           },
81
82           // Resize function if need
83           resize: function() {
84               if(this.commentContentPane){
85                   this.commentContentPane.resize();
86               }
87           },
88
89           _eoc_: null
90
91   });
```

Callout notes: Comments content pane inherited from CaseInfoComponentPane; Get the configuration value of the comments are reado only, disable the comment if the value is true; Create the case comment dialog dijit; Add the comment dialog dijit to content node

2. Create a custom case information page widget extends from the ICM case information page widget icm/pgwidget/caseinfo/CaseInfo, creating the comment tab dijit and add it as a child of the Case Information widget.

3. Override the handler function for the icm.SendWorkItem event and then bind the received WorkItemEditable model object to its action context. This enables the user to add the custom action AddDocumentToAttachment to access the work item information from the action context.

```
CustomCaseInfo.js      ×
  1   define(["dojo/_base/declare",
  2           "dojo/_base/array",
  3           "icm/pgwidget/caseinfo/CaseInfo",
  4           "icm/custom/customcaseinfo/CaseInfoCommentsContentPane",
  5           "icm/util/Dom"],
  6       function(declare,array,CaseInfo,CaseInfoCommentsContentPane,Dom){
  7
  8       return declare("icm.custom.customcaseinfo.CustomCaseInfo", [CaseInfo], {
  9
 10           postCreate: function() {
 11               this.inherited(arguments);
 12
 13               //Load custom CSS
 14               var customCSSFile =
 15                   "/ICMCustomWidgets/icm/custom/customcaseinfo/themes/         ";
 16               Dom.loadCSS(customCSSFile);
 17
 18               //Create the tab dijit
 19               var commentPane = new CaseInfoCommentsContentPane({
 20                   title: "Comments",              //Tab title
 21                   description:"A sample widget",    //Tab description, shown when CaseInfo is initial
 22                   definitionId: "Comments",       //Tab ID.
 23                   resourceBundle: this.resourceBundle,//Resource bundle for the dijit. Could be cust
 24                   context: this                    //The custom CaseInfo instance
 25
 26               });
 27               this.addChild(commentPane);   //Add the new tab.
 28           },
 29
 30           handleICM_SendWorkItemEvent: function(payload){
 31               this.inherited(arguments);
 32               array.forEach(this.getChildWidgets(), function(widget){
 33                   widget.setActionContext("WorkItem", payload.workitem);
 34               });
 35           },
 36
 37           _eoc_: null
 38       });
 39   });
```

Create the case comment pane and added as tab

Add workitem to action context

After you add the case comment to the Case Information widget, the widget becomes writable rather than read only. When the user adds the widget to a Case Detail page or a Work Details page, the widget will work like other page widgets in the page to:

- Mark page dirty state if there are unsaved comments, and clear the dirty state when save complete.
- Save comments when user click the save button on the toolbar.
- Notify to user if there is unsaved comments when user click close button on toolbar, such as on a Work Details page with a custom Case Information widget.

```
//Render the tab dijit
render: function(model) {

    if (!this.isInFocus() || !model || !model.payload || this.commentContentPane) {
        return;
    }
    this.inherited(arguments);

    var caseObj = model.payload.caseEditable.getCase();
    var coord = model.payload.coordination;
    var readonly = this.context.widgetProperties.commentsAreReadOnly;
    var self = this;

    var context = {
        "artifactType": "Case",
        "artifactLabel": caseObj.getCaseTitle(),
        "commentContext": Constants.CommentContext.CASE,
        "caseModel": caseObj
    };
    this.commentContentPane = new CommentContentPane(context)

    if(readonly){
        this.commentContentPane.commentText.disabled = true;
    }else{
        this.commentContentPane.addCommentButton.watch("disabled",function(){
            if(self.commentContentPane.addCommentButton.disabled === true){
                self.context.onBroadcastEvent("icm.SetDirtyState",{'dirtySate':false, 'reference':self.id});
            }else{
                self.context.onBroadcastEvent("icm.SetDirtyState",{'dirtySate':true, 'reference':self.id});
            }
        });

        coord.participate(Constants.CoordTopic.BEFORECANCEL, function(context, complete, abort){
            if(self.commentContentPane.addCommentButton.disabled === false){
                abort({message:"Comments unsaved"});
            }else{
                complete();
            }
        });

        coord.participate(Constants.CoordTopic.SAVE, function(context, complete, abort){
            if(self.commentContentPane.addCommentButton.disabled === false){
                self.commentContentPane.addCommentButton.onClick();
                complete();
            }else{
                complete();
            }
        });

        coord.participate(Constants.CoordTopic.COMPLETE, function(context, complete, abort){
            if(self.commentContentPane.addCommentButton.disabled === false){
                self.commentContentPane.addCommentButton.onClick();
                complete();
            }else{
                complete();
            }
        });
    }

    this.contentNode.appendChild(this.commentContentPane.domNode);
},
```

Callouts: Watch the add comment button, mark/clear the dirty state by broadcast 'icm.SetDirtyState' event. When page close, if there is unsaved comments, abort the page close. When user save/complete case or work item, also save the unsaved comments.

When the user adds the custom Case Information widget to a Work Details page, the add case document as attachment action supports:

- Rendering the attachment list of a work item as submenu when a case worker select a case document and bring up the context menu.

- Publishing the icm.AddDocumentAsAttachment event after a case workers selects a case document and then chooses an attachment name from the action submenu.

The getIterator() function gets the WorkItemEditable model object from action context that was injected by the parent custom Case Information page widget in the preceding

step. The function then returns the list of attachments for the WorkItemEditable model object. The action framework renders the attachment names as a list in the submenu. When a case worker specifies an attachment name, the framework calls the execute() function and the action publishes an icm.AddDocumentAsAttachment event with a case document and an attachment name in the payload.

```
define([
    "dojo/_base/declare",
    "dojo/_base/lang",
    "dojo/_base/array",
    "icm/action/Action",
    "icm/base/Coordination"
], function(declare, lang, array, Action, Coordination) {

    return declare("icm.custom.actions.CustomAddToAttachmentAction", [Action], {

        isEnabled: function()
        {
            return true;
        },

        execute: function()
        {
            var item = this.getArgument("item");
            this.publishEvent("icm.AddDocumentAsAttachment", {"attachmentName":item.id, "documents":this.getActionContext("Document")});
        },

        getIterator: function(callback){
            var self = this;
            var result = [];
            if(this.getActionContext()["WorkItem"] !== null){
                var workitem = this.getActionContext("WorkItem")[0];
                for(var k in workitem.propertiesCollection){
                    var prop = workitem.propertiesCollection[k];
                    if (prop.dataType == "xs:attachment"){
                        result.push({'title':prop.name, 'id':prop.id});
                    }
                }
            }

            callback(result);
        },

        _eoc_:null

    });
});
```
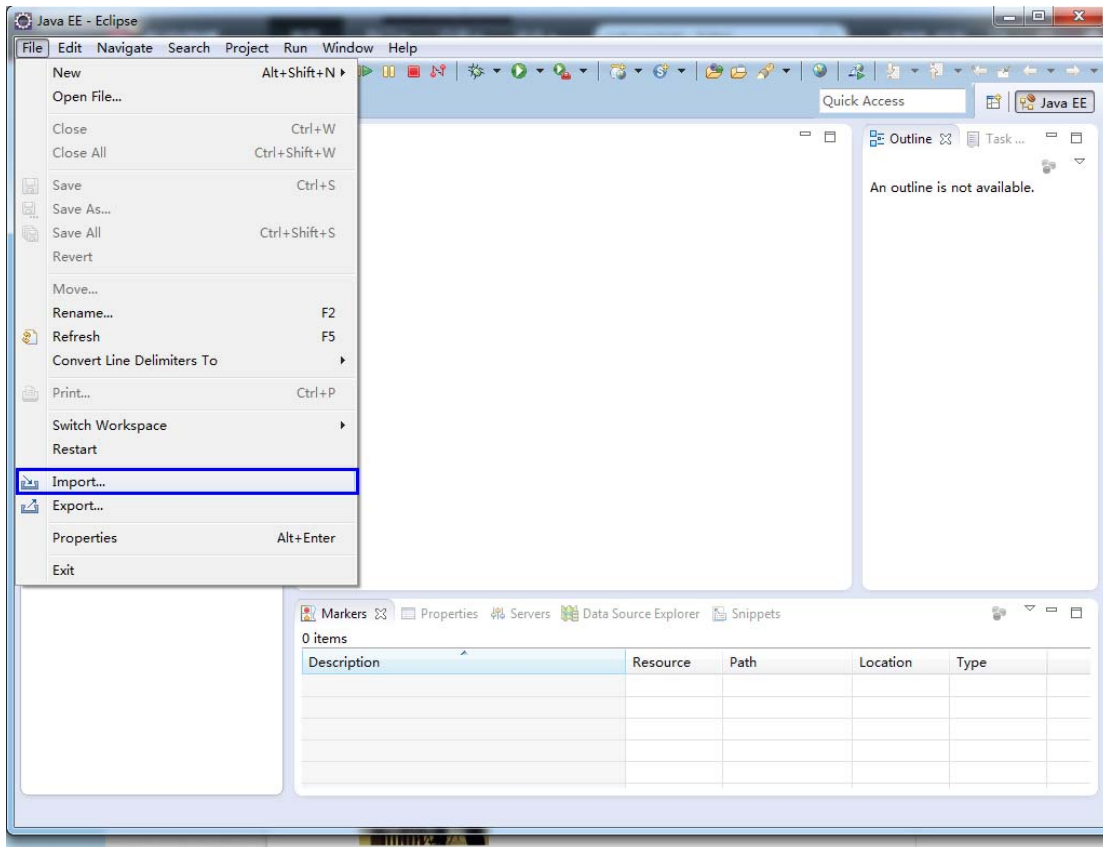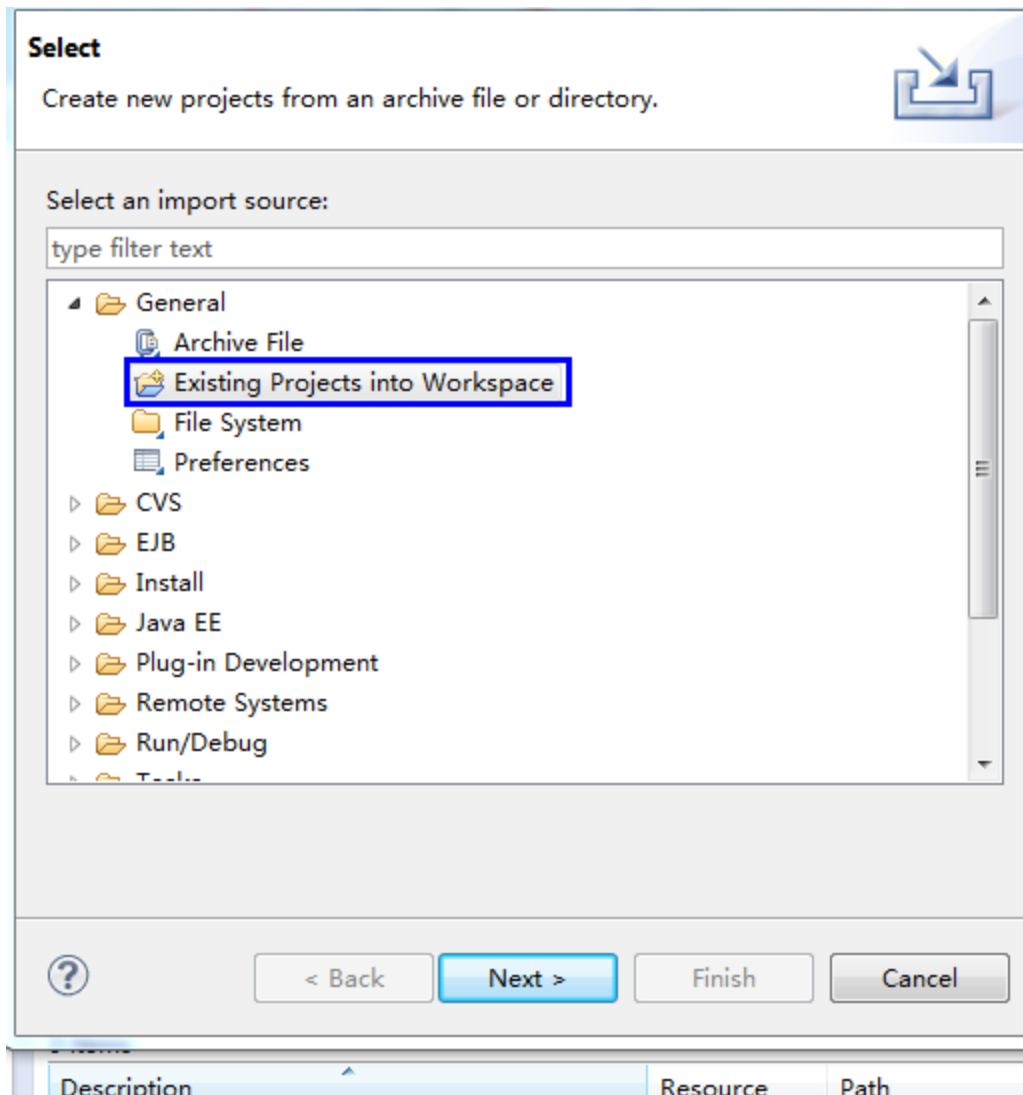
Import and build custom page widget

To import and build a custom page widget:

1. Launch Eclipse V3.8 or higher and select a workspace.

2. Import the sample widget project.

3. Click **File** > **Import**.

4.   Select **Existing Projects into Workspace** and click **Next**.

Select

Create new projects from an archive file or directory.

Select an import source:

type filter text

- ⊿ 📂 General
    - 📄 Archive File
    - 📂 Existing Projects into Workspace
    - 📁 File System
    - 📊 Preferences
- ▷ 📂 CVS
- ▷ 📂 EJB
- ▷ 📂 Install
- ▷ 📂 Java EE
- ▷ 📂 Plug-in Development
- ▷ 📂 Remote Systems
- ▷ 📂 Run/Debug

[?]   [< Back]   [Next >]   [Finish]   [Cancel]

Description   Resource   Path

5. Browse to the folder that contains the custom widget project, select the customWidget project, and click **Finish**.
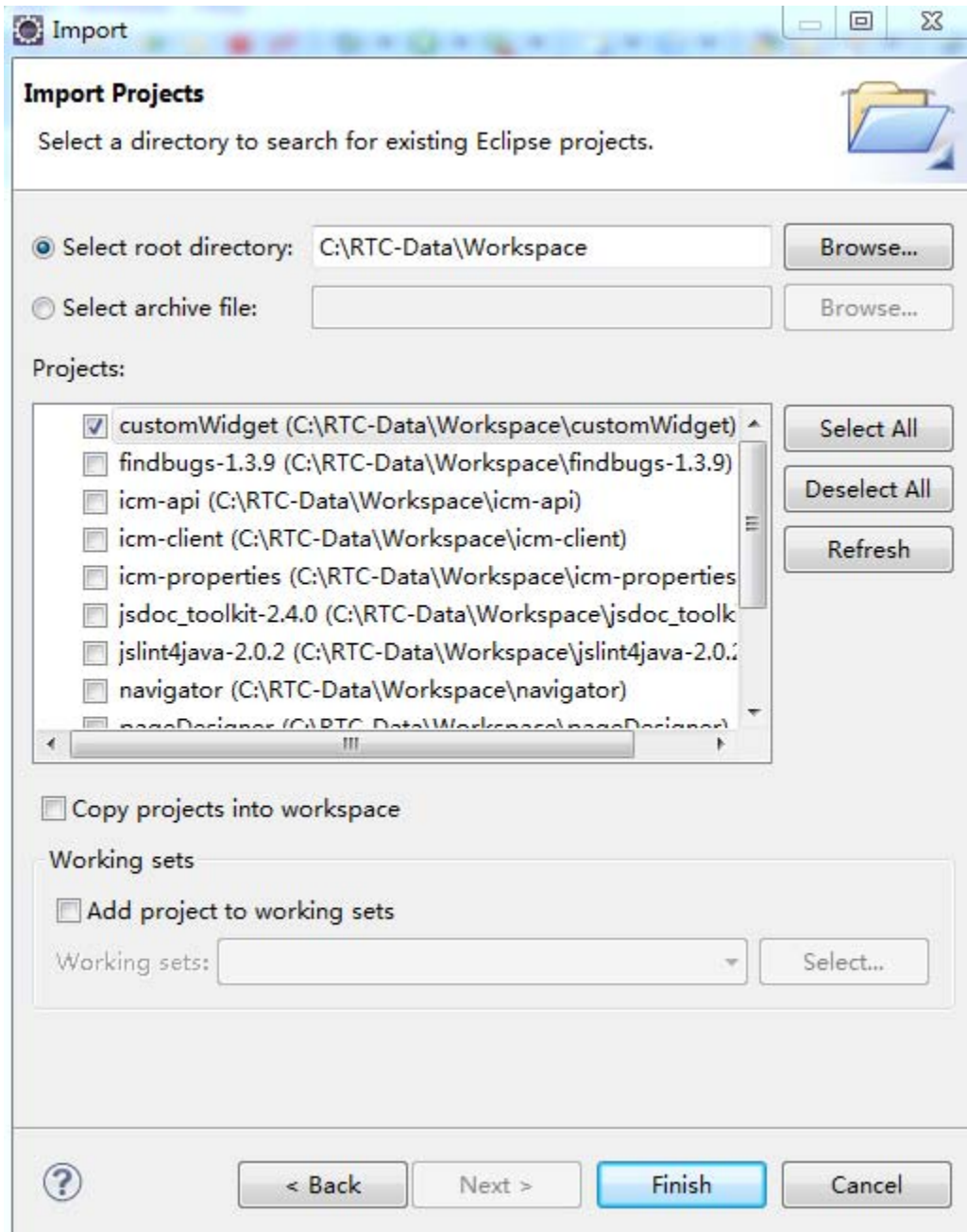
6. Run build

    c. Expand the Build folder and right-click the build.xml file.
    d. Select **Run As** and **Ant Build**.

The build generates a ICMCustomWidgets.zip file in the project folder.