

Chapter-1

Aim:- Create Custom Backoffice Extension

1. Open a command prompt.
2. Navigate to the `${HYBRIS_BIN_DIR}/platform` directory and type `./setantenv.sh` and press Enter
3. In the same directory call `ant extgen`.
4. Follow instructions that show up on the screen:

1. Type `ybackoffice` to use proper template and press ENTER.
 2. Type `myextension` for extension name and press ENTER.
 3. Type `org.myextension` for package name and press ENTER.
 4. Decide if you want a sample widget to be generated. If not, then do not forget to manually create a `widgets` folder in `myextension\backoffice\resources` directory. Here you will be adding a new directory for each component that you create. Type yes.
 5. Decide if you want a sample style sheets to be generated. For more details read [How to Replace Styles of Backoffice Application – Tutorial](#).
- New extension is generated in the following directory: `{HYBRIS_BIN_DIR}/custom`

5. Add the new extension to your `localextensions.xml` file (config folder).

To load all custom extensions automatically, add the following line to `localextensions.xml`, at the beginning of the file after other path statements:

localextensions.xml
<code><path autoload="true" dir="\${HYBRIS_BIN_DIR}/custom"/></code>

II. Build Your Project

Open a command prompt.

1. Navigate to the `${HYBRIS_BIN_DIR}/platform` directory.
2. Make sure that a compliant version is used:
 - On the Windows operating system, call the `${HYBRIS_BIN_DIR}/platform/setantenv.bat` file. Do not close the command prompt after this call as the settings are transient and would get lost if the command prompt is closed.
 - On the Unix operating system, call the `${HYBRIS_BIN_DIR}/platform/setantenv.sh` file, such as: `./setantenv.sh`.
3. Call `ant clean all` to build the entire hybris Commerce Suite.

III. Start the Application Server

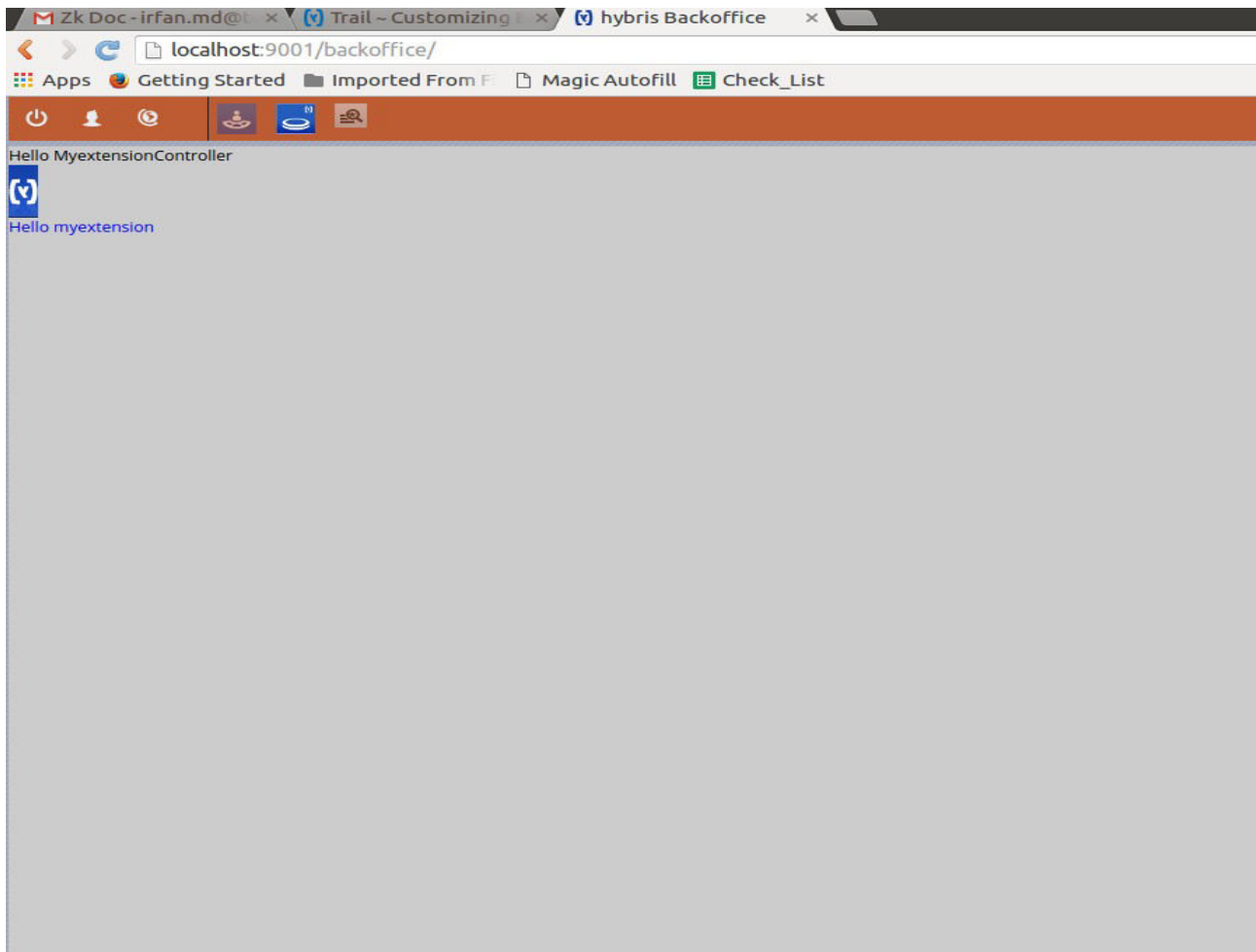
In a command prompt navigate to the `{HYBRIS_BIN_DIR}/platform` directory.

1. Type `./hybrisserver.sh` and press ENTER.
2. Open a backoffice application using for example this link <http://localhost:9001/backoffice>. The newly created `myextensionextension` is just a place where you should create your custom components.

IV. Check your Extension

open Browser type <http://localhost:9001/backoffice/>

You will see below screen



In above Screen You Can see Default Widget Which created by you when you type as [Yes] at the time of ant extgen.

NOW You Can See Your Default Code For above Widget..

There Are Some File Which automatically created in your custom extension.

1. [definition.xml](#)

2. [myextensionwidget.zul](#)

3. [MyextensionController](#)

Now check Files one by one

first Open [definition.xml](#) open your custom extension Go

backoffice --> resources --> widgets --> MyextensionWidget --> definition.xml

definition.xml

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

<widget-definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.hybris.com/schema/cockpitng/widget-
definition.xsd"
    id="com.myextension.widgets.myextensionwidget">
    <name>myextension Sample Widget</name>
    <description>myextension Sample Widget</description>
    <defaultTitle>myextension Sample Widget</defaultTitle>
    <author>Extgen</author>
    <version>1.0</version>
    <view src="myextensionwidget.zul" />
    <keywords>
        <keyword>myextension</keyword>
    </keywords>
    <controller class="com.myextension.widgets.MyextensionController" />
</widget-definition>
```

You will see above file inside that location .Just Observe that File in that File You will able to see two Config file one For View perpose called as zul file.

```
<view src="myextensionwidget.zul" />
```

myextensionwidget.zul

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

<widget xmlns="http://www.zkoss.org/2005/zul"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:h="http://www.w3.org/1999/xhtml"
    xmlns:w="http://www.zkoss.org/2005/zk/client"
    xmlns:zk="http://www.zkoss.org/2005/zk"
    xsi:schemaLocation="http://www.zkoss.org/2005/zul
http://www.hybris.com/schema/cockpitng/zul/zul.xsd"
    height="100%">

    <div height="100%" style="background: #ccc;">
        <div>
            <label id="label" value="${labels.hello} myextension" />
        </div>
        <div>
            <image src="${wr}/images/headline_icon.png"/>
        </div>
        <div>
            <label value="${labels.hello} myextension" sclass="yw-
labelstyle"/>
        </div>
    </div>
</widget>
```

second one is controller

```
<controller class="com.myextension.widgets.MyextensionController" />
```

MyextensionController

```

package com.myextension.widgets;

import org.zkoss.zk.ui.Component;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zul.Label;

import com.myextension.services.MyextensionService;

import com.hybris.cockpitng.util.DefaultWidgetController;

public class MyextensionController extends DefaultWidgetController
{
    private static final long serialVersionUID = 1L;
    private Label label;

    @WireVariable
    private MyextensionService myextensionService;

    @Override
    public void initialize(final Component comp)
    {
        super.initialize(comp);
        label.setValue(myextensionService.getHello() + "
MyextensionController");
    }
}

```

Note :- Now Apart From this , Two More Important File Is There.

Click on Your extension Go

resources ---> myextension-backoffice-widgets.xml

resources ---> myextension-backoffice-config.xml

myextension-backoffice-widgets.xml

```

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

<widgets xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.hybris.com/schema/cockpitng/widgets.x
sd">

    <widget-extension widgetId="backofficeMainSlot">
        <widget id="myextension-perspective"
widgetDefinitionId="com.myextension.widgets.myextensionwidget"
        template="false" slotId="perspectives" title="Hello myextension">
            <setting key="perspectiveImageUrl"
value="/cng/images/perspective.png"
            type="String" />
        </widget>
    </widget-extension>

    <widget-connection sourceWidgetId="myextension-explorer-tree"
outputId="nodeSelected" targetWidgetId="myextension-tree-node-adapter"
inputId="input"/>
    <widget-connection sourceWidgetId="myextension-tree-node-adapter"
outputId="true" targetWidgetId="myextension-typecode-extractor"
inputId="genericInput"/>

```

```

<widget-connection sourceWidgetId="myextension-tree-node-adapter"
outputId="true" targetWidgetId="myextension-search-enabler"
inputId="genericInput"/>
<widget-connection sourceWidgetId="myextension-tree-node-adapter"
outputId="false" targetWidgetId="myextension-search-disabler"
inputId="genericInput"/>
<widget-connection sourceWidgetId="myextension-typecode-extractor"
outputId="genericOutput" targetWidgetId="myextension-search-logic"
inputId="type"/>
<widget-connection sourceWidgetId="myextension-search-enabler"
outputId="genericOutput" targetWidgetId="myextension-text-search"
inputId="enabled"/>
<widget-connection sourceWidgetId="myextension-search-disabler"
outputId="genericOutput" targetWidgetId="myextension-text-search"
inputId="enabled"/>MyextensionController
<widget-connection sourceWidgetId="myextension-text-search"
outputId="query" targetWidgetId="myextension-search-logic"
inputId="searchtext"/>
<widget-connection sourceWidgetId="myextension-search-logic"
outputId="pageable" targetWidgetId="myextension-simple-list"
inputId="pageable"/>
</widgets>

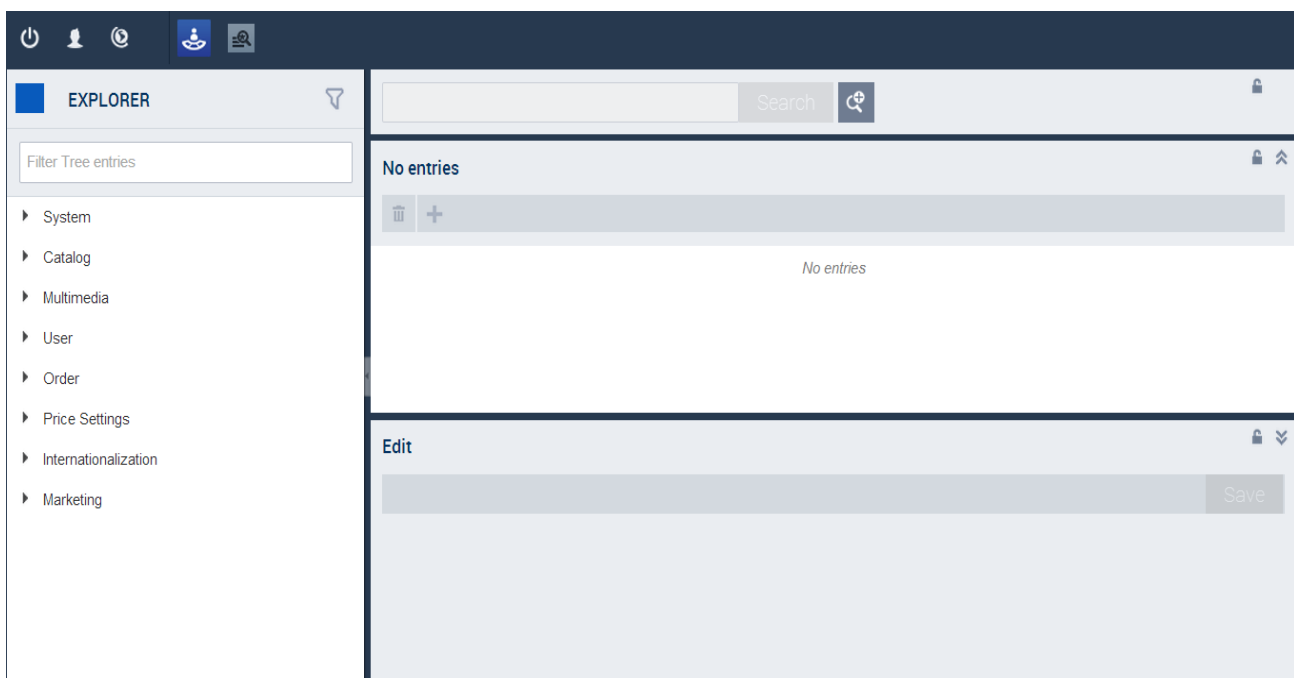
```

In Above Screen Short Shaded Area is for Adding new Widget .

here `widgetDefinitionId="com.myextension.widgets.myextensionwidget"` is definition of our Widget.

NOW LET'S SEE ONCE DEFAULT WIDGET GIVEN BY HYBRIS AS ADMIN USER

if you will open <http://localhost:9001/backoffice> you got two perspective one as your extension perspective second as Admin perspective . Inside admin perspective you can find all widget config.After logging in as admin (password:nimda), you should see the default backoffice admin area:



Chapter-2

Aim:- Understand Next Generation Cockpit Framework and Create Your Own Widget.

Overview

One of the key concepts behind the Next Generation Cockpit Framework is the idea of an application made up of a set of widgets. Each widget is a stand-alone, deployable component with a clearly-defined interface and a specific purpose. By combining and connecting widgets, an application can be designed and put together quickly using the application orchestrator. The framework allows for different tasks to be defined for different user roles, giving business users a user interface that is designed to meet their needs, one that helps them fulfill their domain-specific tasks.

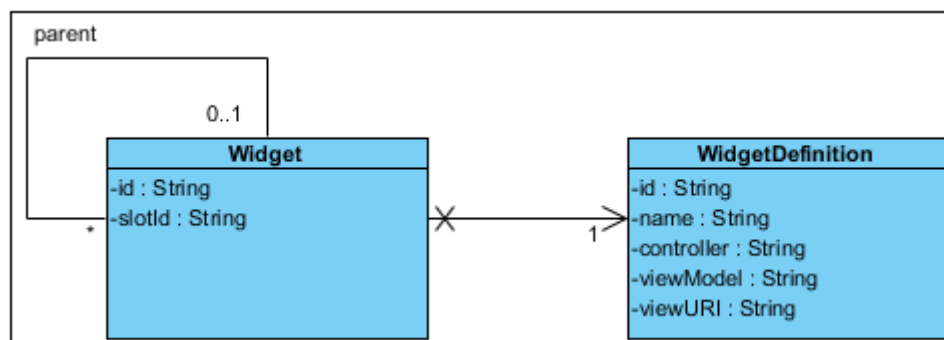
Two important extensions enable you to use the framework:

- The backoffice extension provides the framework with all the standard widgets and other components. It is a central point for the framework, which consists of the Main Backoffice Application and the ApplicationMyextensionController Orchestrator.
- The Main Backoffice Application is to be used by all users, from administrators to business users. It displays the proper application mash-up depending on his business role.
- The Application Orchestrator is where you design your custom backoffice application. From within the Application Orchestrator, you can add widgets, connect widgets together, modify widget settings, and configure access restrictions.
- The ybackoffice extension template is used to generate a custom extension within which you implement your own components. These components are then reusable in the Application Orchestrator and in the Web application of the backoffice extension.

Widgets

Widgets are organized in a tree structure. A widget tree represents an application. You can take aMyextensionController widget sub-tree (a widget group) and reuse it in other application, just like a single widget.

The following diagram illustrates the differences between a widget definition and widget instance. The widget definition describes the widget's type, id, name, view, and controller classes. A widget used in an application is an instance of the widget definition.

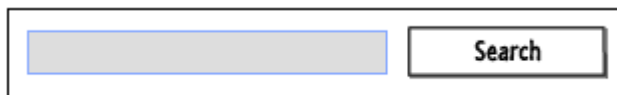


MyextensionController

How to Create a Widget

A Web application created using the Next Generation Cockpit Framework consists of components called widgets. Widgets communicate with one another and may be also grouped together. Widgets are used by application designers who create the frontend application.

This tutorial describes all steps that should be performed in order to create a widget and add it to a frontend application. As an example, the tutorial shows how to create a simple search widget made up of a text box and Search button.



Creating a Widget Definition

Widgets are defined in the definition.xml file.

1. In myextension/backoffice/resources/widgets directory create a new folder called mysearch.

2. In the mysearch folder, create a definition.xml file.

3. Add information about the search widget, like its name, description, default title, author, and version. Each widget must have a unique ID made up of the extension and widget names. For this tutorial, the widget ID is org.myextension.widgets.mysearch.

definition.xml

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

<widget-definition
id="org.myextension.widgets.mysearch" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="http://www.hybris.com/schema/cockpitng/widgetdefinition.xs
d">

<name>My Search</name>
<description>My own search widget.</description>
<defaultTitle>Search</defaultTitle>
<author>Me</author>
<version>0.1</version>
<view src="mysearch.zul" />
</widget-definition>
```

Creating a Widget View

The view of this widget is defined in a ZK ZUL file, called .zul named after the last part of the widget ID as specified in the definition.xml (in this case mysearch.zul).

In this file, you define all frontend components. For the search widget, you need two components: a text box and a button.

1. In the mysearch folder, create the mysearch.zul file.
2. Add the text box and button components, providing an ID for each, along with a button label. Your mysearch.zul file should look more or less like the following example.

mysearch.zul

```
<widget xmlns:xsi="http://www.w3.org/2001/XMLSchema-  
instance" xsi:noNamespaceSchemaLocation="http://www.zkoss.org/2005/zul">  
    <style src="{wr}/default.css"/>  
    <div>  
        <hlayout>  
            <textbox id="searchInput"/>  
            <button id="searchBtn" label="Search"/>  
        </hlayout>  
    </div>  
</widget>
```

Now add your widget Definition to *-Widget.xml

*-Widget.xml

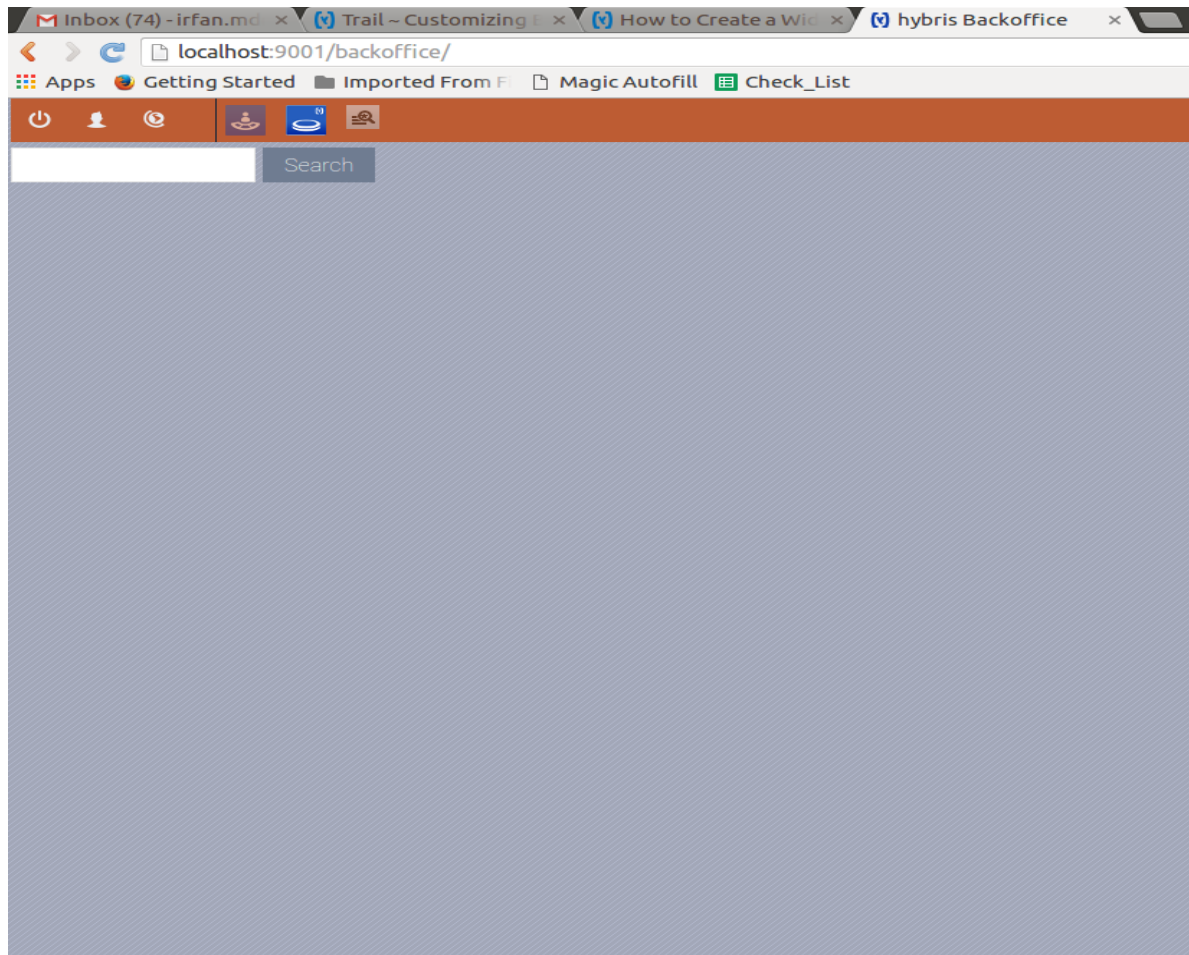
```
<widget-extension widgetId="backofficeMainSlot" >  
  
    <widget id="myserch" widgetDefinitionId="org.myextension.widgets.mysearch "  
        slotId="perspectives" template="true">  
  
    </widget>  
    - - - - -
```

Shaded area is your widget configuration.

in above xml file you are adding your widget Definition as
“org.myextension.widgets.mysearch”

Now ant build and start server...

Open the Backoffice application. For example: <http://localhost:9001/backoffice>.



You Will See the Above Screen .

As you have not yet defined any logic, nothing happens when you type in text and press the Search button.

Implementing a Search Service

before creating controller, you must add a **SearchService**, which is used by your search widget.

1. Add a new class named SearchService to the src folder within the org.myextension package. The following code is an example implementation.

SearchService.java

```
Package org.myextension;
import java.util.ArrayList;
import java.util.List;
public class SearchService
{
    public List<String> search(final String text)
    {
        List<String> result = new ArrayList<String>();
        result.add(text);
        result.add("Hello");
        result.add("Cockpit NG");
        result.add("Developer");
        return result;
    }
}
```

2. Add the new implementation to the Spring context file located in myextension/resources directory:

myextension-backoffice-spring.xml

```
<bean id="searchService" class="org.myextension.SearchService"></bean>
```

Creating a Controller

As you have not defined any action for the Search button, nothing happens if it is clicked. For this you need to create a controller.

1. In myextension/backoffice/src/org/myextension, create a controller with the following package name: org.myextension.widgets.mysearch.

It should extend DefaultWidgetController. Name it MySearchController.

2. Add the implementation for the following actions:

- When a search query is typed in the text box, it should trigger the search.
- When the Search button is pressed, the search for the search query should be executed.

MySearchController.java

```
Package org.myextension.widgets.mysearch;

import java.util.List;

import org.myextension.SearchService;
import org.zkoss.zk.ui.event.Events;
import org.zkoss.zk.ui.select.annotation.WireVariable;
import org.zkoss.zul.Messagebox;
import org.zkoss.zul.Textbox;

import com.hybris.cockpitng.annotations.ViewEvent;
import com.hybris.cockpitng.util.DefaultWidgetController;

public class MySearchController extends DefaultWidgetController
{
    private Textbox searchInput;

    @WireVariable
    private SearchService searchService;

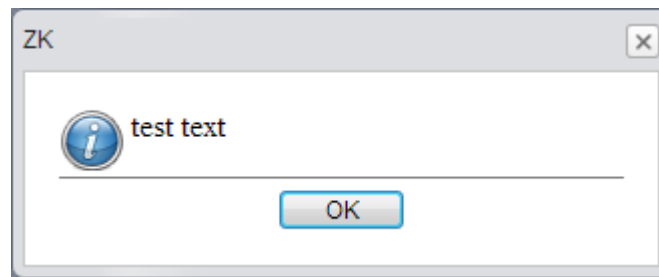
    @ViewEvent(componentID = "searchBtn", eventName = Events.ON_CLICK)
    public void doSearch() throws InterruptedException
    {
        List<String> result = searchService.search(searchInput.getText());
        Messagebox.show(result.get(0));
    }
}
```

3. Add a controller class in definition.xml (the one located in myextension/backoffice/resources/widgets/mysearch directory).

definition.xml

```
<!-- ... -->
<controller class="org.myextension.widgets.mysearch.MySearchController"/>
<!-- ... -->
</widget-definition>
```

4. Rebuild the system like you did in Deploying the Widget section.
5. Switch to the frontend application by pressing F4.
6. Click the Search button. A pop-up message appears with the same text that you typed into the text box.



Chapter-3

Aim:- understand Available Widgets and adding that widget in our extension.

Now we Are going to add different default widget in our extension .
First off all you have to understand some default widget .

1.Border Layout Widget:-

Widget Definition

Name	Border Layout
ID	com.hybris.cockpitng.borderlayout
Category	Layout
Description	It is a layout widget with north, south, west, east and center layout regions/slots.
Applied In	Not applicable.
Inputs	<div>Details of input sockets:</div> <ul style="list-style-type: none">• closeNorth • Description: Trigger to close the north region• openNorth • Description: Trigger to open the north region• closeSouth • Description: Trigger to close the south region• openSouth • Description: Trigger to open the south region• closeWest • Description: Trigger to close the west region• openWest • Description: Trigger to open the west region• closeEast • Description: Trigger to close the east region• openEast • Description: Trigger to open the east region
Outputs	Not applicable.

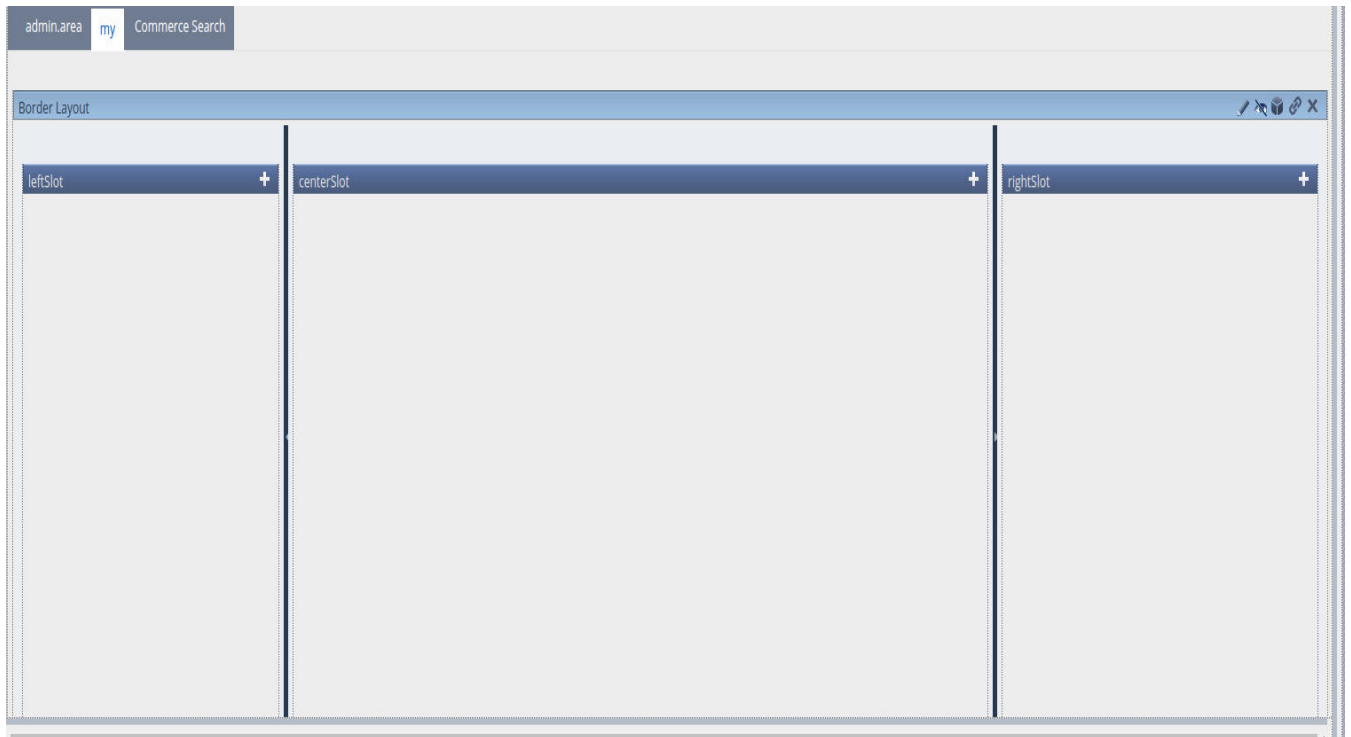
1.In above table ID is widgetdefinitionId for that widget , if you want to access this widget then you have to add that ID as widgetdefinitionId .

Example:-

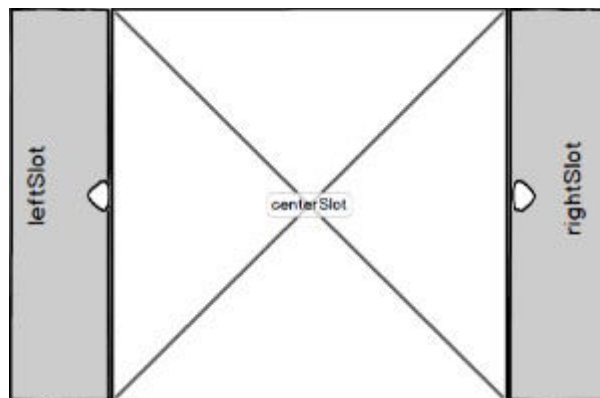
```

.....
<widget id="myborder" widgetDefinitionId="com.hybris.cockpitng.borderlayout"
slotId="perspectives" title="admin.area" template="false">
.....

```



After adding widget will be appear like that.



Note:- left ,right slot we can collapse

More Details:-

<https://wiki.hybris.com/display/release5/Border+Layout+Widget>

2.Explorer Tree Widget:-

Widget Definition

Name	Explorer Tree
ID	com.hybris.cockpitng.widgets.common.explorertree
Category	Navigation
Description	Widget for displaying a tree structure.
Applied In	Not applicable.
Based On	Not applicable.
Author	hybris
Inputs	<p>Details of input sockets:</p> <ul style="list-style-type: none">• nodeSelected Type:com.hybris.backoffice.navigation.NavigationNode Multiplicity:Single Description: Allows item selection in a tree• nodeIdSelected Type:com.hybris.backoffice.navigation.TreeNodeSelector Multiplicity:Single Description: Allows item selection in a tree using node id and gives ability to decide whether selection events should be triggered on output sockets
Outputs	<p>Details of output sockets:</p> <ul style="list-style-type: none">• nodeSelected Type:com.hybris.backoffice.navigation.NavigationNode Multiplicity:Single Description: Whenever a node is selected in this widget, it is sent out on the output socket.• dataSelected Type:java.lang.Object Multiplicity:Single Description: Whenever a node is selected in this widget, its associated data is sent out on the output socket.

1. In above table ID is widgetdefinitionId for that widget, if you want to access this widget then you have to add that ID as widgetdefinitionId.

Important note :-- for adding any node in explorer tree you have to add that node id in ***-config.xml** file

Example:-

```
*-config.xml
```

```
<context component="myexplorer-tree" merge-by="module">
  <n:explorer-tree xmlns:n="http://www.hybris.com/cockpitng/config/explorertree">
    <n:navigation-node id="Cuppytrail">
      <n:type-node id="Stadiums" code="Stadium"/>
    </n:navigation-node>
  </n:explorer-tree>
</context>
```

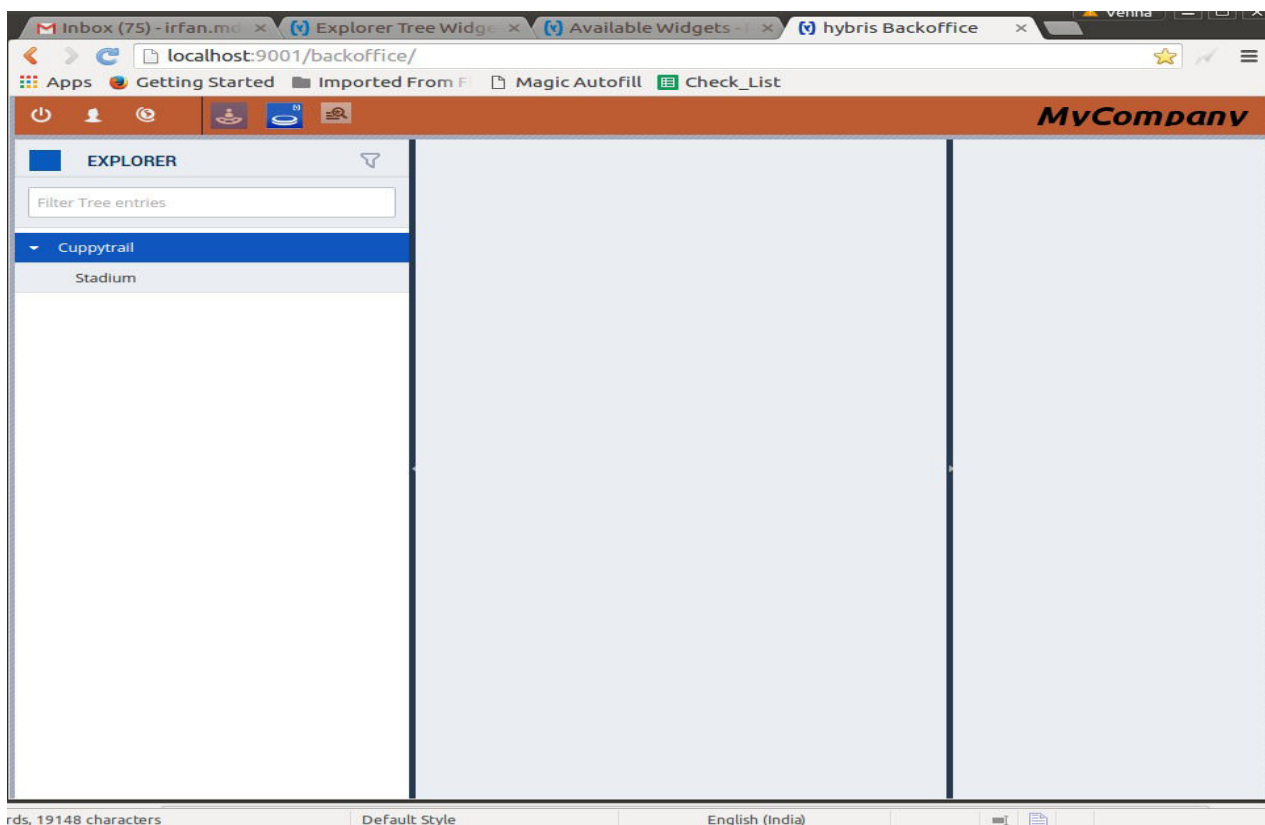
2. You have to add Default Explorer tree widget Like that:

*-widgets.xml

```
-----
<widget id="mytree"
widgetDefinitionId="com.hybris.cockpitng.widgets.common.explorertree"
slotId="rightSlot" template="false">
  <setting key="explorerTreeConfigCtx" type="String">myexplorer-tree</setting>
</widget>
-----
```

Note:-

- above in *-config.xml file “myexplorer-tree” given as component it should be config in *-widgets.xml inside <setting >
- Setting key should be available in *-widgets.xml file .
- Cuppytrail will be appear as node in explorer tree.



3. Collapsible Container Widget:-

Widget Definition

Name	Collapsible Container
ID	com.hybris.cockpitng.collapsiblecontainer
Category	Layout
Inputs	<p>collapseState</p> <ul style="list-style-type: none">•Type: com.hybris.cockpitng.widgets.controller.collapsiblecontainer.CollapsibleContainerState•Multiplicity: Single•Description: Contains information about a requested state of the panels: collapsed or expanded.
Description	This container widget hosts two or three sections that end-users can collapse or expand. The title of each section is configurable. The heights of the top and center sections can be specified as a percentage of the total height of the container.

1.In above table ID is widgetdefinitionId for that widget , if you want to access this widget then you have to add that ID as widgetdefinitionId .

*-widgets.xml

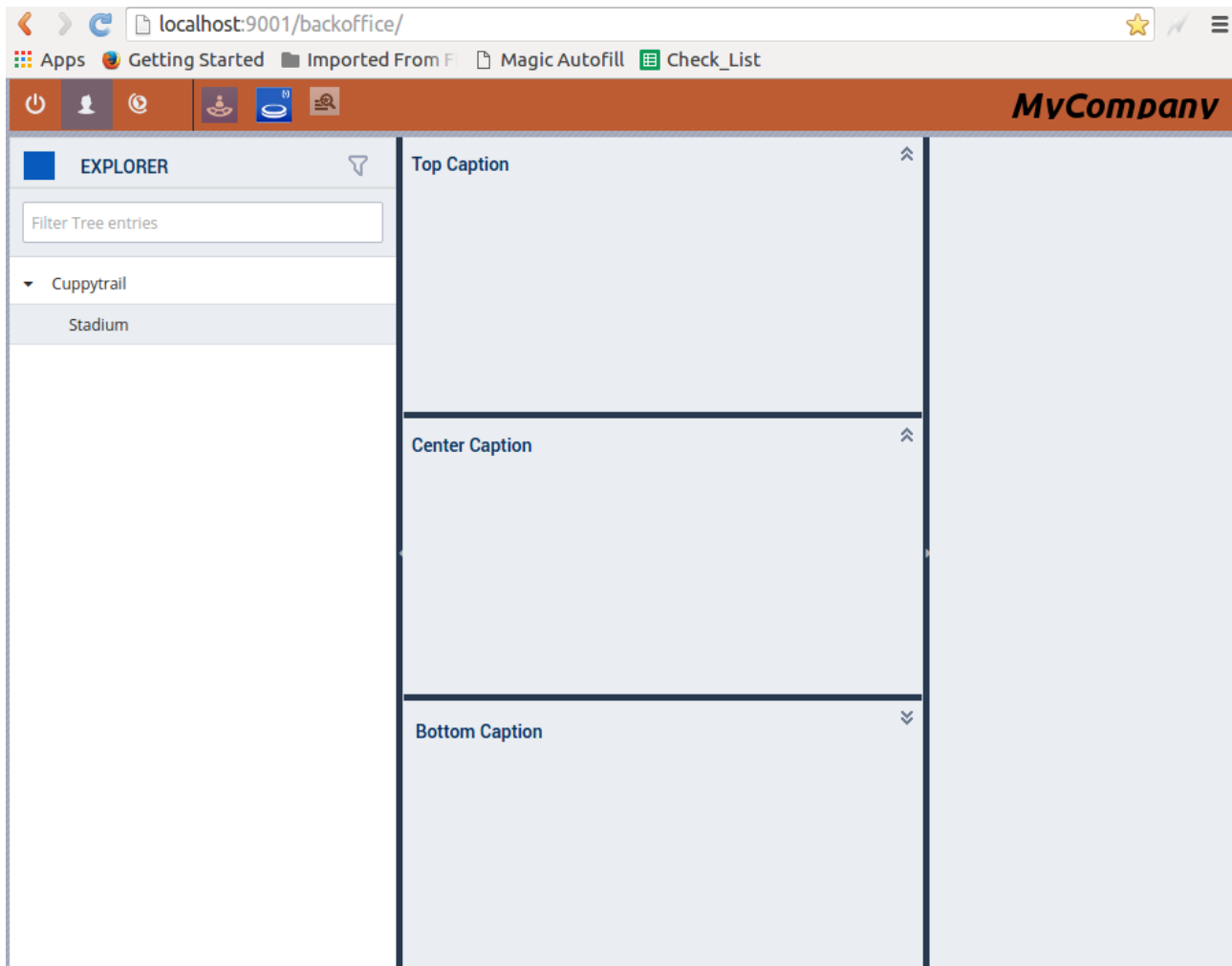
```
-----
<widget id="h" widgetDefinitionId="com.hybris.cockpitng.borderlayout"
        slotId="perspectives" title="my" template="false">
-----
<widget id="mycollapsibleContainer"
widgetDefinitionId="com.hybris.cockpitng.collapsiblecontainer"
        slotId="centerSlot" template="false">

        </widget>
</widget>
-----
```


Note:-

- collapsiblecontainer should be config inside BorderLayout widget not explorer tree.

after adding widget it will appear like that.



For More Widget :- <https://wiki.hybris.com/display/release5/Available+Widgets>

Chapter-4

Aim:- How to Pass Data Between Widgets

This tutorial shows how you can pass data between widgets using widget sockets. As an example, you are going to create a simple chat widget, which is able to send out a message on an outgoing socket and receive a message on an incoming socket. For the sake of demonstrating how to use widget sockets, the chat widget is very simple.

From a technical point of view, the chat widget should have one input and one output:

- Input: Incoming message.
- Output: A message sent out after clicking the button Send.

Create a Widget Definition

- 1.In the myextension/backoffice/resources/widgets directory, create a new folder called mychat.
- 2.In the mychat folder, create the definition.xml file.
- 3.Add information about the chat widget, like its name, description, default title, author and version. Every widget should have a unique id, composed of the extension name and the widget name, in this case: org.myextension.widgets.mychat.

Note:-

Unique Widget ID

Remember to set the unique id for your widget. If you set an id that was already used, then your new widget will not appear on the list of widgets in the Choose a widget wizard.

definition.xml

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

<widget-definition id="org.myextension.widgets.mychat"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://hybris.com/cockpitng/widget-definition">

  <name>My Chat</name>
  <description>My chat widget.</description>

  <!-- ... -->

  <keywords>
    <!-- keyword tag is used to define a category under which the widget is
visible, when adding it to your application in application orchestrator -->
    <keyword>Chat</keyword>
  </keywords>

</widget-definition>
```

4. Define the input and output for the widget using **sockets** tag. The input should be a message typed in a text box, whereas the output should be the same message that is sent when the **Send** button is pressed:

definition.xml

```
<!-- ... -->
    <sockets>
        <input type="java.lang.String" id="incomingMsg"/>
        <output type="java.lang.String" id="outgoingMsg"/>
    </sockets>
<!-- ... -->
```

5. The final definition of the chat widget should look as follows:

definition.xml

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

<widget-definition id="org.myextension.widgets.mychat"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.hybris.com/schema/cockpitng/widget-definition.xsd">
    <name>My Chat</name>
    <description>My chat widget.</description>
    <view src="mychat.zul" />
    <sockets>
        <input type="java.lang.String" id="incomingMsg"/>
        <output type="java.lang.String" id="outgoingMsg"/>
    </sockets>

    <keywords>
        <keyword>Chat</keyword>
    </keywords>

</widget-definition>
```

Create a Widget View

The view of this widget is defined in the ZK's ZUL file, named exactly as the last part of the widget ID as specified in the `definition.xml`, in this case `mychat.zul`. In this file, you can define all frontend components. For the chat widget, we need the following components: a text box, a button and a label that displays the last received message.

mychat.zul

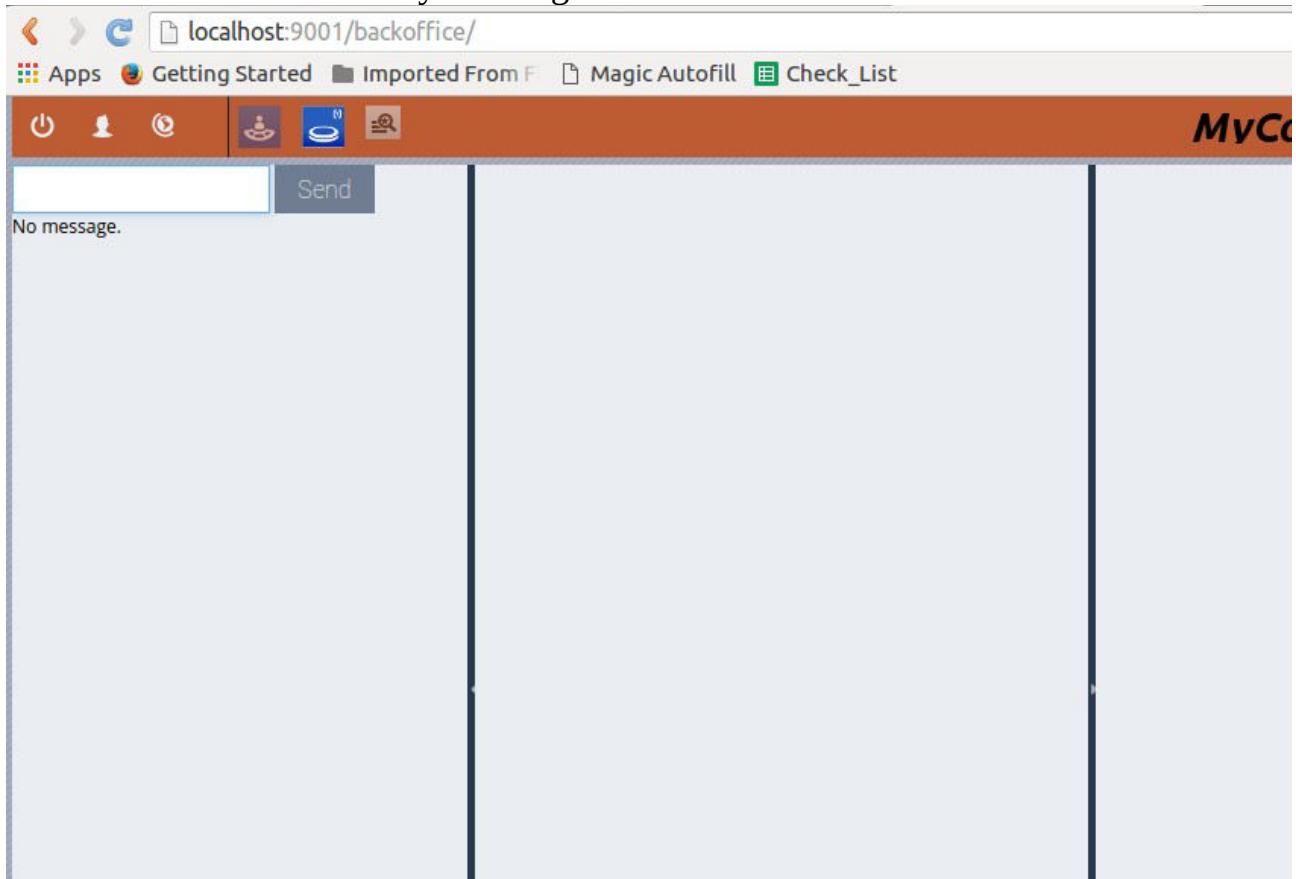
```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<widget xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

xsi:noNamespaceSchemaLocation="http://www.zkoss.org/2005/zul">
    <div>
        <textbox id="msgInput"/>
        <button id="sendBtn" label="Send"/>
    </div>
    <div>
        <label id="lastMsgLabel" value="No message."></label>
    </div>
</widget>

```

After build and start server you will get below Screen



Create a Controller

As we have not defined any action for the **Send** button, nothing happens. Now, you should:

1. In the `myextension/backoffice/src/org/myextension` directory, create a controller with the following package name: `org.myextension.widgets.mychat`. It should extend the `DefaultWidgetController`. Call it `MyChatController`.

MyChatController.java

```

Public class MyChatController extends DefaultWidgetController
{
    private Label lastMsgLabel;
    private Textbox msgInput;

    @ViewEvent(componentID = "sendBtn", eventName = Events.ON_CLICK)
    public void sendMsg()
    {
        sendOutput("outgoingMsg", msgInput.getText());
    }
}

```

```

    }

    @SocketEvent(socketId ="incomingMsg")
    public void updateTranscript(final String msg)
    {
        lastMsgLabel.setValue(msg);
    }
}

```

2.Add the controller class to the definition.xml file:

definition.xml

```

<!-- ... -->
    <controller class="org.myextension.widgets.mychat.MyChatController"/>
<!-- ... -->
</widget-definition>

```

ADD Widget into *-widget.xml

*-Widget.xml

```

-----
<widget id="border" widgetDefinitionId="com.hybris.cockpitng.borderlayout"
        slotId="perspectives" title="my" template="false">

    <widget id="mychat1" widgetDefinitionId="org.myextension.widgets.mychat"
        slotId="leftSlot" template="false">
    </widget>

    <widget id="mychat2" widgetDefinitionId="org.myextension.widgets.mychat"
        slotId="rightSlot" template="false">
    </widget>

-----

```

Note:- Above widget should be inside “Border layout widget “

after build it will look like as below.



Now Important thing is connection with the help of widget connection only we can pass data from one widget to other .

Widget Connection:-

for widget connection we already declare some input and output socket connection inside definition.xml.

Example:-

```
<sockets>
    <input type="java.lang.String" id="incomingMsg"/>
    <output type="java.lang.String" id="outgoingMsg"/>
</sockets>
```

Now we have to configure input and output socket connection inside ***-widget.xml** file.

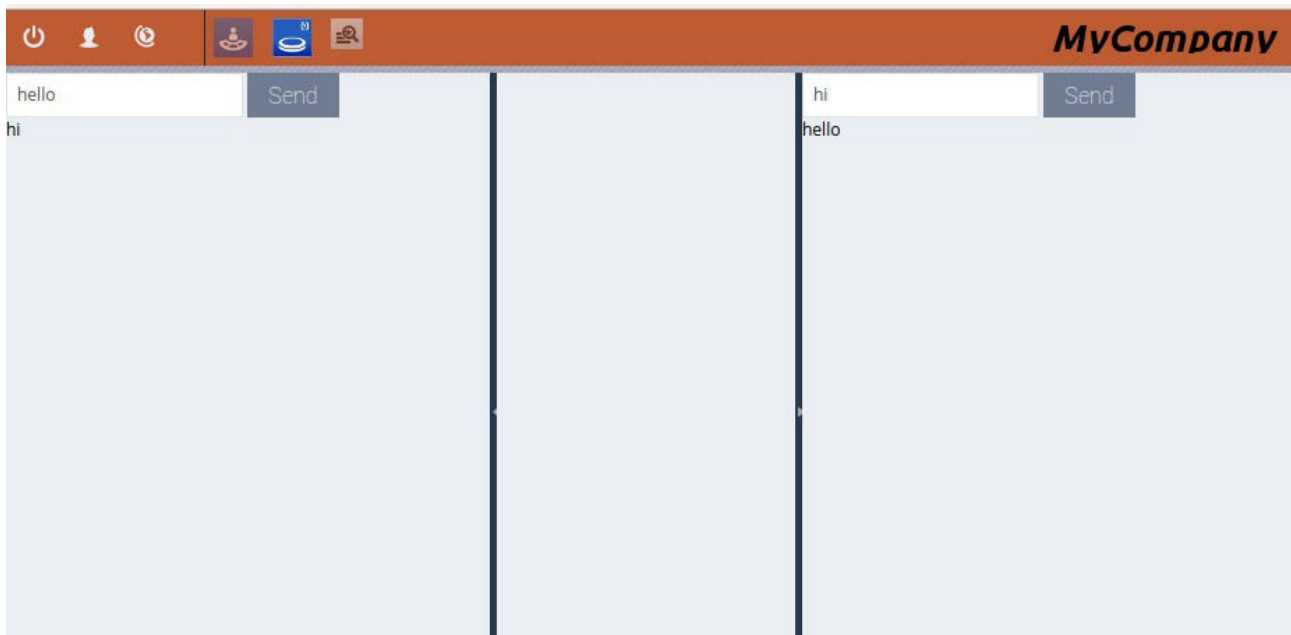
***-widget.xml**

```
-----
<widget-connection sourceWidgetId="mychat1"
    outputId="outgoingMsg" targetWidgetId="mychat2"
    inputId="incomingMsg" />
<!-- this is for send msg from left to right -->

<widget-connection sourceWidgetId="mychat2"
    outputId="outgoingMsg" targetWidgetId="mychat1"
    inputId="incomingMsg" />

<!-- this is for send msg from right to left -->
-----
```

Now you can see below screen.



Chapter- 5

Aim:- How to Create Composed Widgets

This tutorial shows how you can create a widget that can be composed with other widget or widgets. In other words, it is possible to have a widget that can be composed with one or more widgets. As an example you will extend simple chat widget, that you already created by following [How to Pass Data Between Widgets](#). You will add to this widget a slot, to which you can add additional widget (for example you may want your user to use his chat with additional list of all users or some special widget that displays history of his conversations or both of them).

From a technical point of view, the chat widget should have additional slot:

- **widgetslot:** This slot makes it possible to add and display one widget as a child component of main widget's view.
- **widgetchildren:** This slot makes it possible to add and display more than one widget as a child component of main widget's view. All widgets added to this slot will appear depending on your settings, these can be tabs, lists, portal.

Tip

Application Layout

When you think about defining a layout of an application, it is nothing else but just creating a widget that accepts many different slots located in different parts of the screen. The framework is shipped together with some layout widgets, but if you want to create your own, then you can follow this tutorial, but of course with slight modifications.

Modify a Widget View

To the existing .zul file of the chat widget you will add a widgetslot slot that will make it possible to add only one

additional widget.

1. Add the following to your widget view:

widgetid.zul

```
<!-- ... -->
    <widgetslot slotID="additionalArea" id="ChatSlot" width="100%" height="100%"/>

</widget>
```

The **slotID** is the name of the slot, displayed in the application orchestrator.

Find Out More

widgetchildren

To add widgetchildren instead of widgetslot slot, you need to add the following:

widgetid.zul

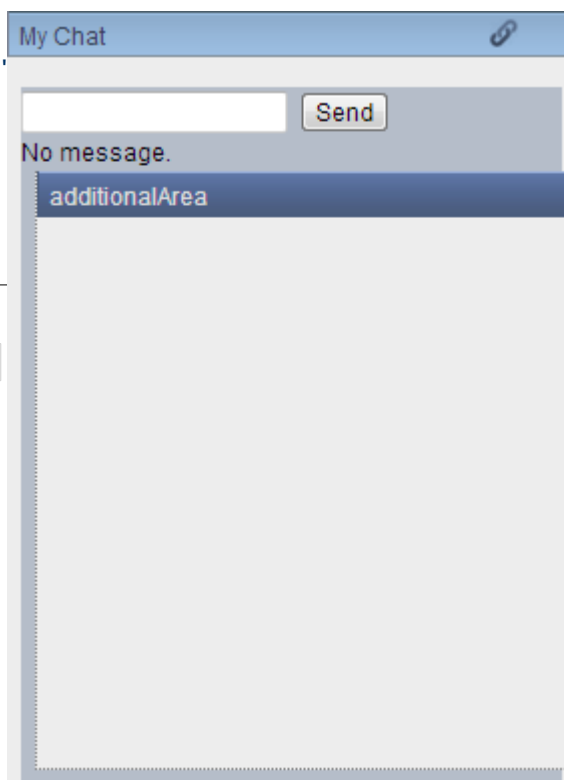
```
<!-- ... -->
<widgetchildren slotID="additionalArea"      id="ChatContainer" type="tab"
width="100%" height="100%"/>
</widget>
```

2. In the end, the view definition of the chat widget should look like that:

widgetid.zul

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

<widget xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.zkoss.org/2005/zul">
    <div>
        <textbox id="msgInput"/>
        <button id="sendBtn" label="Send"/>
    </div>
    <div>
        <label id="No message."/>
    </div>
    <widgetslot slotID="additionalArea" width="100%" height="100%"/>
</widget>
```



3. This is how chat widget with its additionalArea slot will be rendered in the application orchestrator:

now do ant buid and start the server .




Chapter- 6

Aim:- understand Application Orchestrator

The Application Orchestrator is used to create and modify Backoffice applications without having to write any code. You can access the Application Orchestrator from a Backoffice application by pressing **F4** on the keyboard.

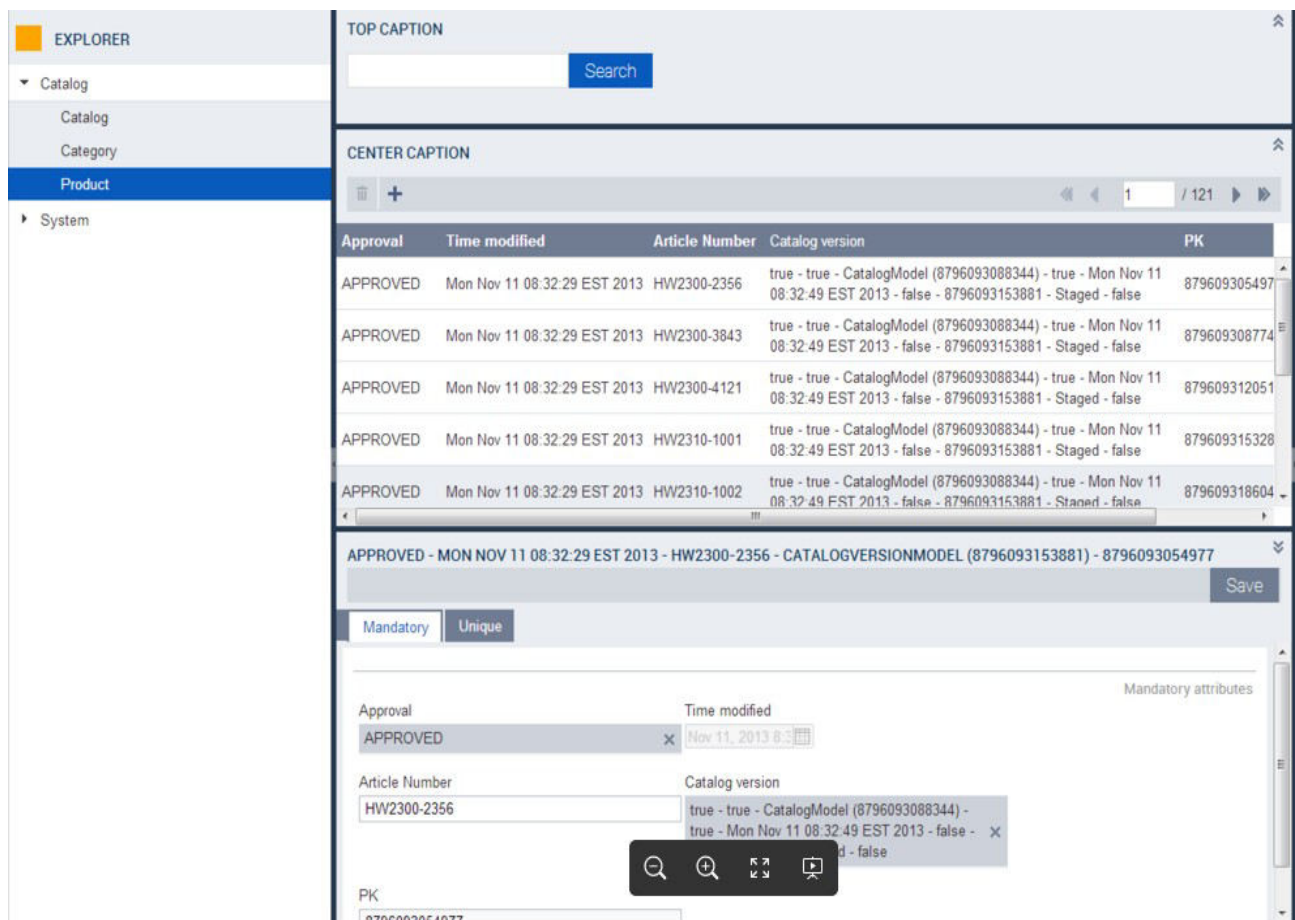
Widget Commands - Quick Reference

Button	Commands
	Edit Widget Settings
 	Create Widget Group / Ungroup
	Show/Hide Invisible Widget Children
 	Convert to Widget Template / Revert
	Connect Widgets

Button	Commands
	Add Widget
 	Remove Widget

Introduction

When you first open the Backoffice Application, the Application View is displayed. End-users interact with the Application View to perform their tasks. The view changes depending on the business role of the user.



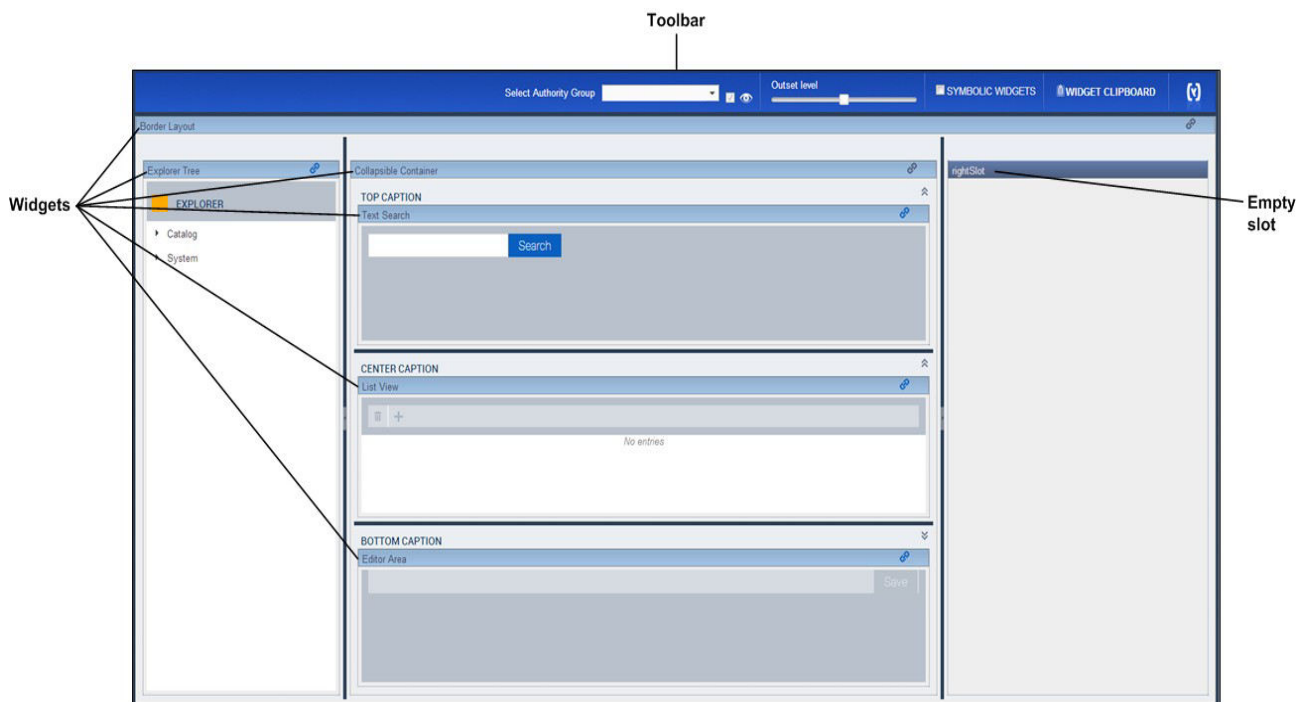
The screenshot displays the Backoffice Application interface. On the left is an 'EXPLORER' sidebar with a tree view containing 'Catalog', 'Category', 'Product', and 'System'. The main area is divided into sections: 'TOP CAPTION' with a search bar, 'CENTER CAPTION' with a table of data, and a detailed view at the bottom for a selected record. The table has columns for Approval, Time modified, Article Number, Catalog version, and PK. The detailed view shows fields for Approval, Time modified, Article Number, and Catalog version, with a 'Save' button and 'Mandatory'/'Unique' tabs.

Approval	Time modified	Article Number	Catalog version	PK
APPROVED	Mon Nov 11 08:32:29 EST 2013	HW2300-2356	true - true - CatalogModel (8796093088344) - true - Mon Nov 11 08:32:49 EST 2013 - false - 8796093153881 - Staged - false	879609305497
APPROVED	Mon Nov 11 08:32:29 EST 2013	HW2300-3843	true - true - CatalogModel (8796093088344) - true - Mon Nov 11 08:32:49 EST 2013 - false - 8796093153881 - Staged - false	879609308774
APPROVED	Mon Nov 11 08:32:29 EST 2013	HW2300-4121	true - true - CatalogModel (8796093088344) - true - Mon Nov 11 08:32:49 EST 2013 - false - 8796093153881 - Staged - false	879609312051
APPROVED	Mon Nov 11 08:32:29 EST 2013	HW2310-1001	true - true - CatalogModel (8796093088344) - true - Mon Nov 11 08:32:49 EST 2013 - false - 8796093153881 - Staged - false	879609315328
APPROVED	Mon Nov 11 08:32:29 EST 2013	HW2310-1002	true - true - CatalogModel (8796093088344) - true - Mon Nov 11 08:32:49 EST 2013 - false - 8796093153881 - Staged - false	879609318604

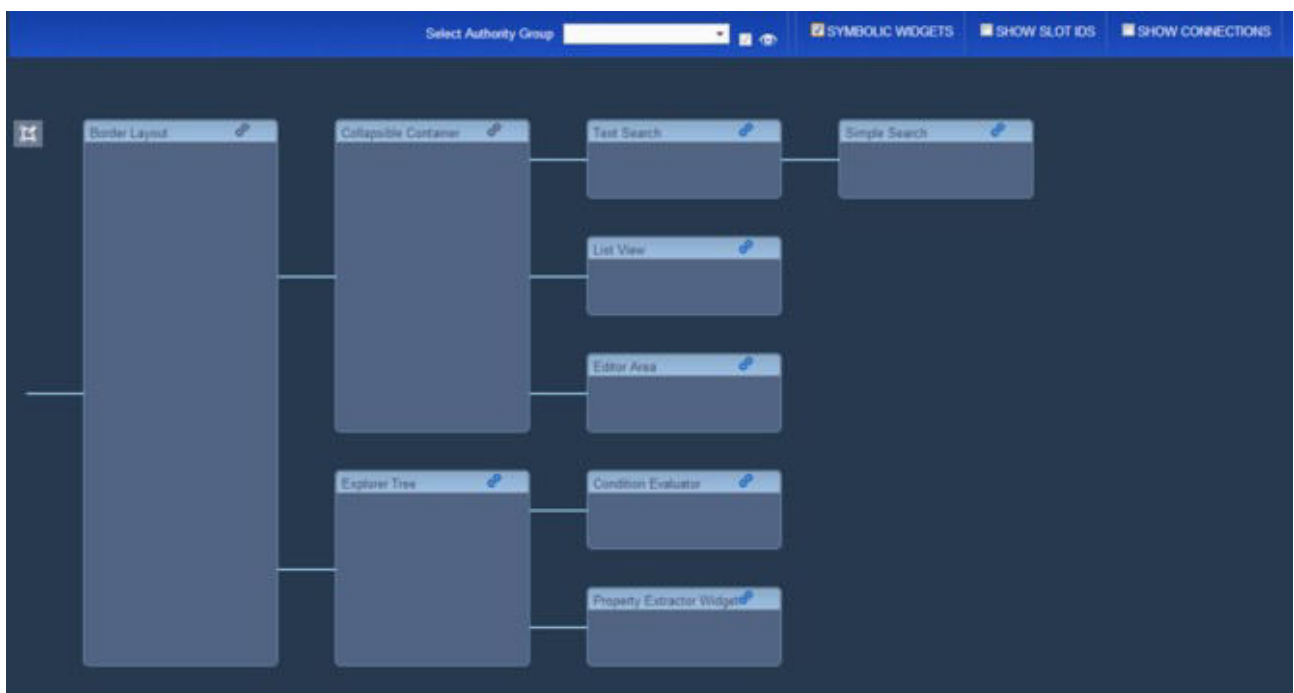
The detailed view at the bottom shows the following fields:

- Approval: APPROVED
- Time modified: Nov 11, 2013 8:32
- Article Number: HW2300-2356
- Catalog version: true - true - CatalogModel (8796093088344) - true - Mon Nov 11 08:32:49 EST 2013 - false - 8796093153881 - Staged - false
- PK: 879609305497

Pressing **F4** (or **Function+F4** on Mac) displays the Application Orchestrator and initially the Main Widgets view, which is where you design a mash-up of your application. The Main Widgets view consists of a toolbar and slots where you can place widgets. In the following example, most of the slots contain widgets, but there is one empty slot on the right.



You can also switch to the Symbolic Widgets View by clicking the **Symbolic Widgets** check box. From this view, you can see the connections and dependencies between widgets.




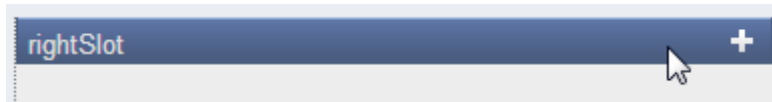
Accessing Commands and Settings for Widgets and Slots

Commands and settings for widgets and slots are accessed by pointing at the title bar of a widget or slot.

For example, when a slot contains a widget, pointing at the title bar of the widget displays various commands and settings.





Pointing at the title bar of an empty slot reveals the **Add Widget** command 



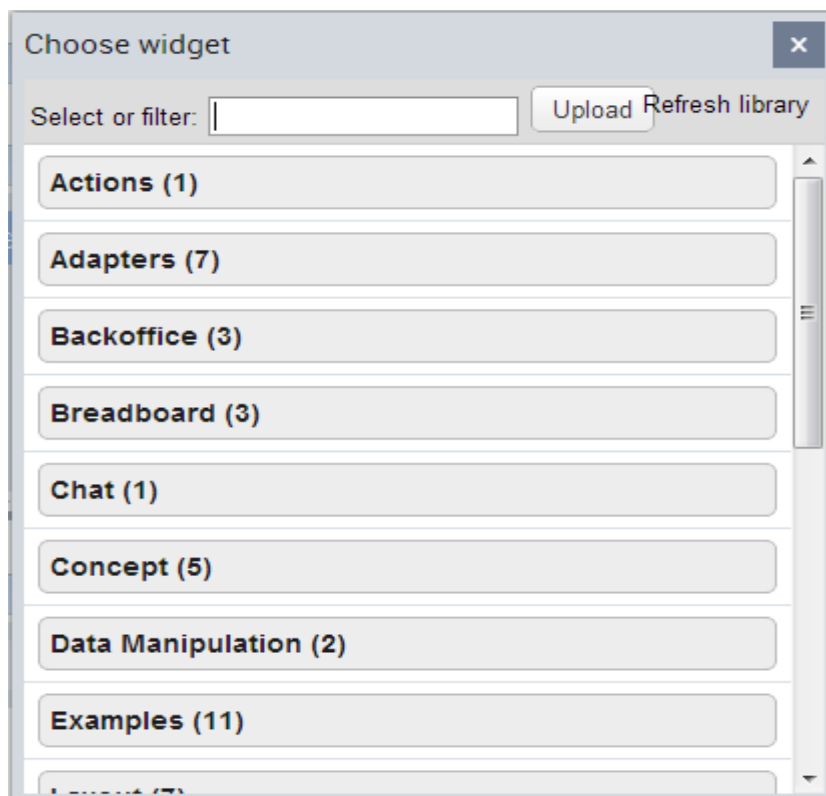
Working in the Main Widgets View

Adding Widgets in the Main Widgets View

To add a widget to a slot, point at the title bar of the empty slot, and then click the **Add Widget**  icon. In the following example, the Collapsible Container widget is added to an empty slot.

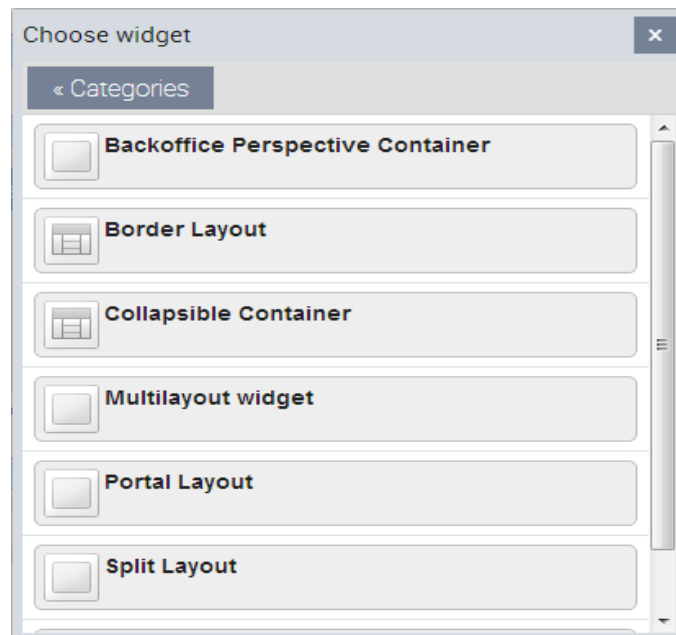
1. Point at the title bar of the empty slot, and then click the Add Widget  icon.

A list of widget categories is displayed.



2. Click the category of the widget you want to add.

The list of widgets for that category appears. In the following example, the Layout category was selected. The Layout category contains widgets that are used to organize other widgets into sections.



3 . Select the widget you want to

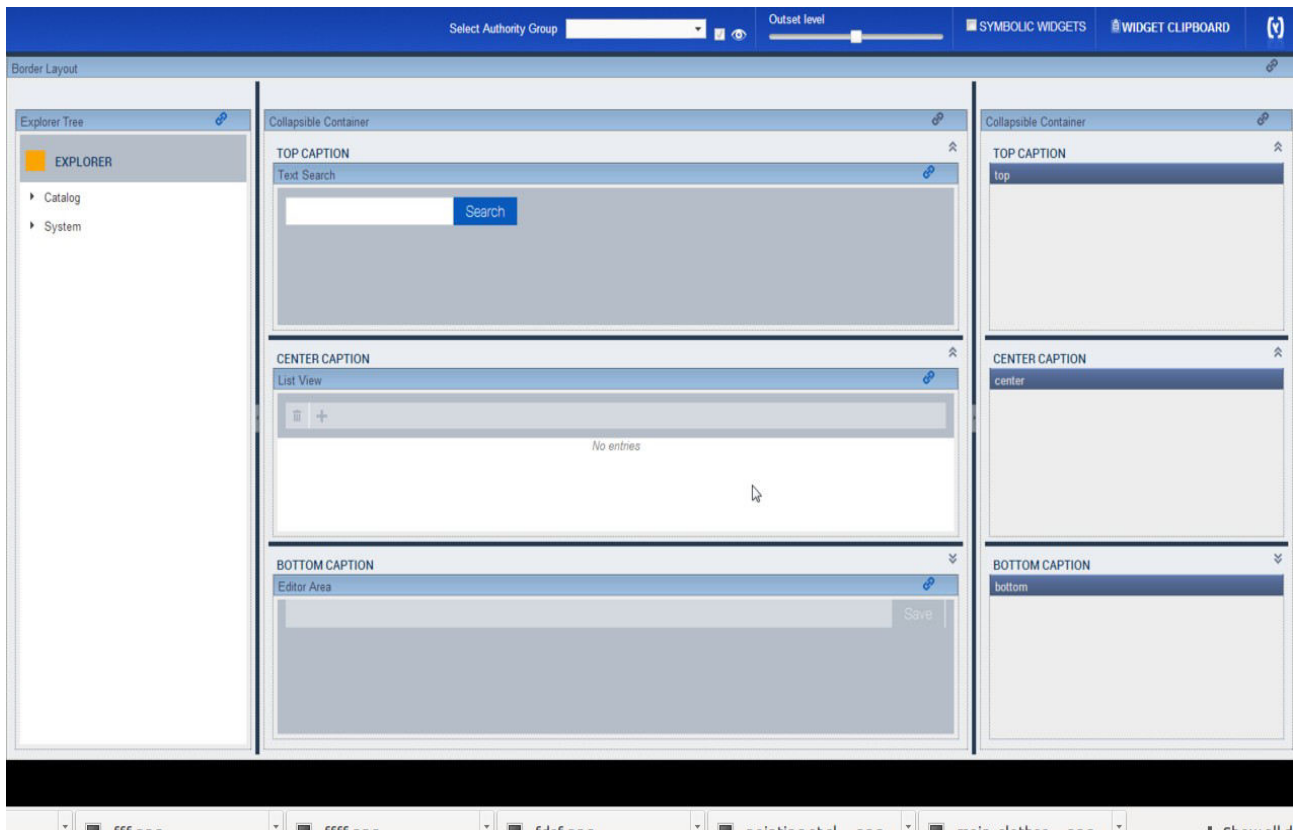
add.

The widget description appears. In the following example, the Collapsible Container widget was selected, which separates the slot into three sections that can be hidden and shown by the user.



4. Click Add & close.

The widget appears in the empty slot. In the following example, the Collapsible Container widget was added to the empty slot at the far right. Since this widget is a layout widget, the widget itself contains slots that can also contain widgets.



Moving Widgets

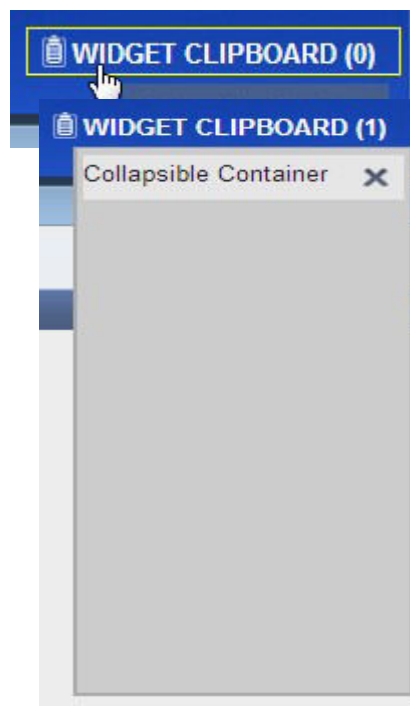
Dragging Widgets to Empty Slots

You can re-arrange widgets by dragging and dropping the widgets by their title bars.

Cutting and Pasting Widgets with the Widget Clipboard

You can "cut" a widget by dragging a widget to the Widget Clipboard in the toolbar. The clipboard is useful for when you want to move a widget to a tab, for example. The Widget Clipboard can store multiple widgets.

1. Drag a widget's title bar to the Widget Clipboard, and then let go of the mouse button.



The widget is stored in the clipboard and removed from the Main Widgets view.

2. Drag the widget from the Widget Clipboard into an empty slot.

Changing the Zoom Level in the Main Widgets view


You can change the zoom level in two ways:

- Adjust the Outset level slider that is available from the Main Widgets view.



- Adjust the zoom level for your browser. Most browsers support the keyboard commands Ctrl-Hyphen to zoom out and Ctrl-Plus to zoom in.

Widget Settings

Clicking the **Widget Settings**  icon allows you to change settings for that instance of the widget, including layout settings, localization values, and access restrictions. You can also add virtual sockets and view information about the widget.

Settings for com.hybris.cockpitng.widgets.common.simplelist

General
Localization
Virtual sockets
Access Restrictions
About

Title:

Widget ID:

6319669a-3225-410f-ba7f-

async:

true

×

maxEntryLimit:

1000

×

pageSize:

10

×

viewMode:

List

×

widgetStyleAttribute:

×

widgetStyleClass:

×


Add Setting

Close

The following table provides a brief description of each tab.

Tab Name	Description
Virtual sockets	Allows you to add additional sockets to your widget instance. For more information, see Adding a Virtual Socket .
Localization	Allows you to substitute UI localization by providing new values for localization keys, which are provided. For more information, see Substituting a Localization Value .
General	Allows you to change settings and style of the widget. You can also add additional attributes. For more information, see Adding a Setting to a Widget .
Access restriction	Allows you to define the roles that have permission to see this widget instance. For more information, see Adding Access Restrictions .
About	Displays the widget ID, a description of the widget, its input and output sockets, and other technical information. For more information, see Displaying Information about a Widget .

Changing General Widget Settings

- 1.Click the Widget Settings  icon
- 2.In the General tab, edit the settings you want to change.

Settings are saved as soon as the field loses the focus.

Each widget has its own settings. For information on the settings for a specific widget, see the documentation for that widget. For a list of all widgets, see Available Widgets.

Adding a New Setting to a Widget

You can add other settings to a widget. In the following example, the context of the UI configuration is extended by adding an attribute to the Simple List widget.

- 1.Click the Widget Settings  icon.

2.From the General tab, click Add Setting.

3.Provide the setting's key name and type, and then click Add.

For this example, the key name is config.context.perspective and the type is java.lang.String.

The screenshot shows a dialog box titled "Settings for com.hybris.cockpitng.widgets.common.simplelist". It has several tabs: "General", "Localization", "Virtual sockets", "Access Restrictions", and "About". The "General" tab is selected. Below the tabs, there are several input fields and dropdowns for configuring settings:

- Title: (empty text box)
- Widget ID: kartofel-simple-list
- async: true (dropdown)
- maxEntryLimit: 1000 (spinner)
- pageSize: 10 (spinner)
- viewMode: List (dropdown)
- widgetStyleAttribute: (empty text box)
- widgetStyleClass: (empty text box)

At the bottom left, there is an "Add Setting" button. A red box highlights this button and the "Key" and "Type" fields of the "Add Setting" form. The "Key" field contains the text "config.context.perspective". The "Type" dropdown menu is open, showing a list of Java types: java.lang.String, java.lang.Boolean, java.lang.Integer, and java.lang.Double. A red box highlights the "Type" dropdown and its list of options. To the right of the "Add Setting" form, there are several "X" buttons for each setting, and an "Add" button at the bottom right. A "Close" button is also visible at the bottom right.


4.Provide the value for the setting you added.

For this example, the value is management.

Settings for com.hybris.cockpitng.widgets.common.simplelist

General
Localization
Virtual sockets
Access Restrictions
About

Title:
Widget ID: kartofel-simple-list
async: true
config.context.perspective: management
maxEntryLimit: 1000
pageSize: 10
viewMode: List
widgetStyleAttribute:
widgetStyleClass:
Add Setting
Close

5.To complete the configuration for this example, modify the UI configuration file so that the context includes the perspective attribute, as shown.
Add additional detail that will help you later on, so that when your configuration grows larger, you can easily find the desired configuration. You can access the configuration in the cockpit-config.xml file (by clicking the icon from the  toolbar menu):

```

<context type="java.lang.String" component="simple-list" perspective="management">
  <ysl:simple-list xmlns:ysl="http://www.hybris.com/cockpitng/config/simplelist">
    <ysl:name field="#root"/>
    <ysl:description field="class.canonicalName"/>
  </ysl:simple-list>
</context>
</config>

```

Store

See also:

- [How to Parametrize a Widget - Tutorial](#)

Substituting a Localization Value

1.Click the Widget Settings  icon

2. Click **Localization**.

3. Click **Add Label**.

4. Provide the label's key name and language code (two-letter ISO code), and then click Add.
If you leave the country code blank, English is used.

Settings for com.hybris.cockpitng.widgets.common.simplelist

General Localization Virtual sockets Access Restrictions About

Add Label Key Language Add Close

loading
emptylist
showmore

5. Provide the value for a key you added.

In the following example, the key emptylist was chosen with language en (English).

Settings for com.hybris.cockpitng.widgets.common.simplelist

General Localization Virtual sockets Access Restrictions About

emptylist (en): No Results At All

Add Label Close

See also:


- [Localizing UI of the Backoffice Application](#)

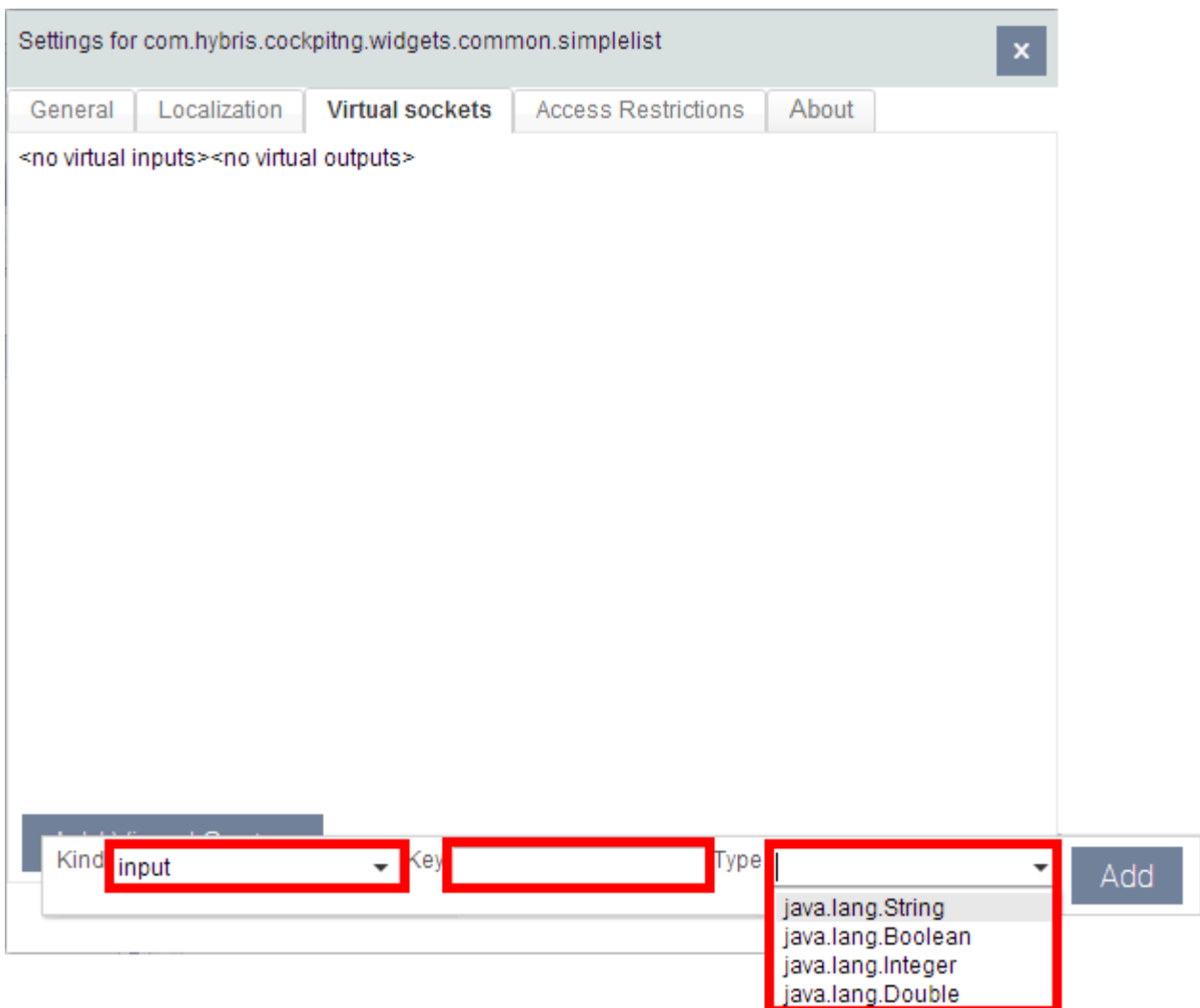
Adding a Virtual Socket

-

Virtual Sockets

Virtual sockets are similar to the regular sockets that are declared in the widget's `definition.xml` file. The main difference between the two sockets' types is that the virtual sockets can also be created at runtime via Application Orchestrator for a specific widget instance. This gives a lot of flexibility and allows to extend the capabilities of widgets. They may also be used in widget extensions to add some communication-related logic (i.e. to trigger actions on template widgets - to show the widget as a popup, etc.). Widget extension's controller may use Virtual sockets to communicate with other widgets in the same way the regular sockets are used. To learn more how to define Virtual Sockets in your extension, see [How to Customize the Widgets Structure in the Backoffice-Based Application - Tutorial](#), section (Optional) Define Widgets' Virtual Sockets.

1. Click the Widget Settings  icon
2. Click Virtual sockets.
3. Click Add Virtual Socket .
4. Provide the virtual socket's kind (input or output), key name, and type, and then click Add.



Settings for com.hybris.cockpitng.widgets.common.simplelist

General Localization **Virtual sockets** Access Restrictions About

<no virtual inputs><no virtual outputs>

Kind: Key: Type:

java.lang.String
java.lang.Boolean
java.lang.Integer
java.lang.Double

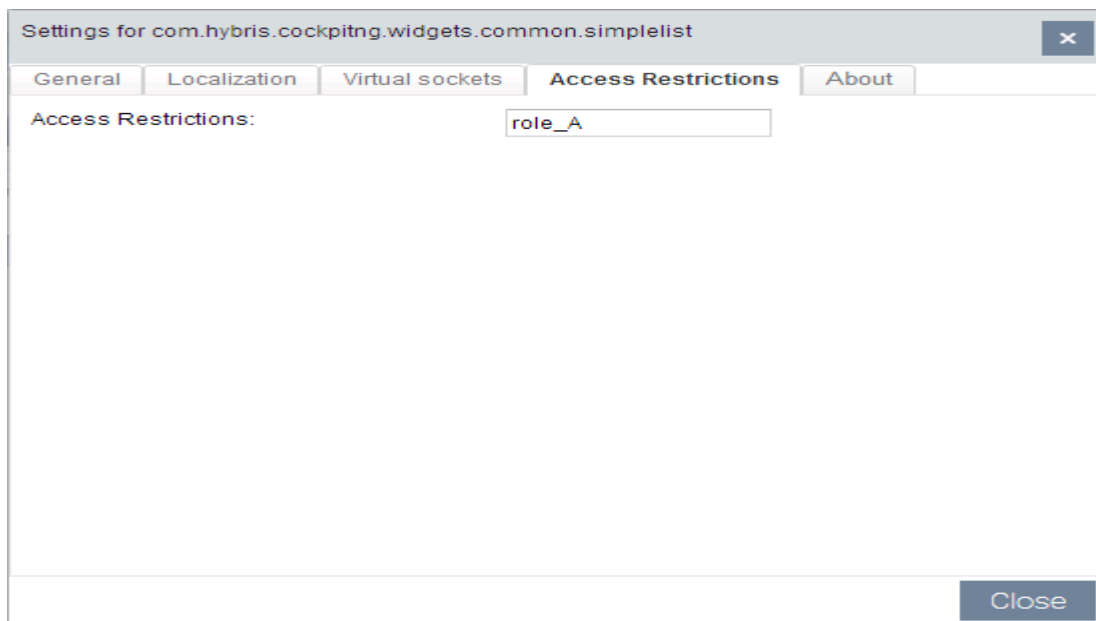
Add

Adding Access Restrictions

1. Click the Widget Settings  icon.

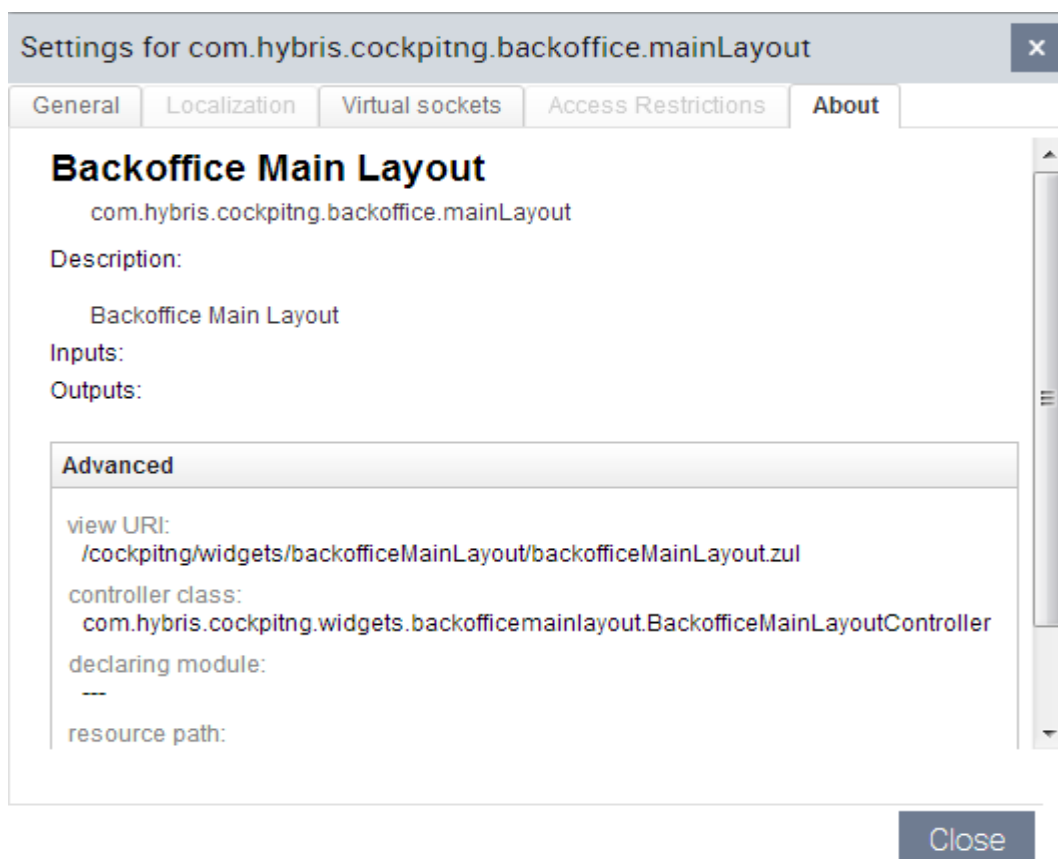
2. Click **Access Restrictions**.

3. Type the role names into the **Access Restrictions** box. Separate role names by commas.



Displaying Information about a Widget

Click the **Widget Settings** icon, and then click **About**. Click **Advanced** to see more information, such as controller class, resource module, and classpath location path.







Connecting Widgets in order to Share Data and Respond to Events

Connecting one widget to another allows widgets to share data and to react to changes that occur in

other widgets.



Widgets receive data on input sockets and send data on output sockets. Socket data types and other information is described in the documentation for each widget: [Available Widgets](#).

Widgets can be connected to other widgets by clicking the Connect  icon in a widget's title bar or by dragging the **Connect**  icon from one widget to another. If you drag click the **Connect**  icon, you can only connect to sockets for the target widget only. If you click the **Connect**  icon, all compatible widgets in the application, both input and output, are shown.

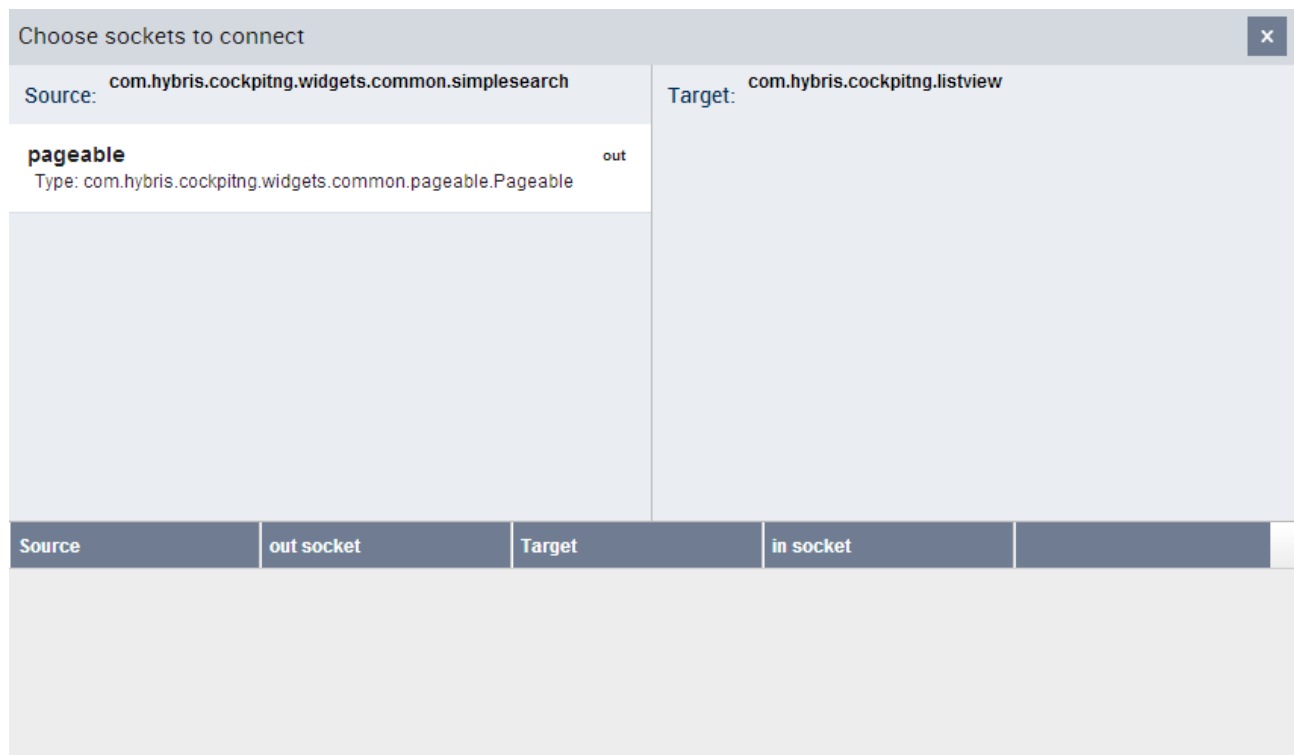
Info:-

Editors and actions can also send and receive data using sockets. For more information, see [How to Create Socket-Aware Actions and Editors – Tutorial](#).

Connecting Widgets by Dragging the Connect Icon onto another Widget

1. Drag the Connect  icon from the source widget onto the **Connect**  icon of another widget, and then let go of the mouse button.

The **Choose sockets to connect** dialog box appears.



2. From the Source column, select the socket that will be sending data.

If the target widget has compatible input sockets, a list of such sockets appears in the Target column. In the following example, the Simple Search widget's pageable output socket can be connected to the List View's pageable input socket.

Choose sockets to connect

Source: com.hybris.cockpitng.widgets.common.simplesearch

pageable

Type: com.hybris.cockpitng.search.data.pageable.Pageable

out

Target: com.hybris.cockpitng.listview

pageable

Type: com.hybris.cockpitng.search.data.pageable.Pageable

in

Source	out socket	Target	in socket	

3. Click the target output socket that you want to connect with.

The connection is created, and a description for the connection appears at the bottom of the dialog box.

Choose sockets to connect

Source: com.hybris.cockpitng.widgets.common.simplesearch

pageable


Type: com.hybris.cockpitng.search.data.pageable.Pageable

out

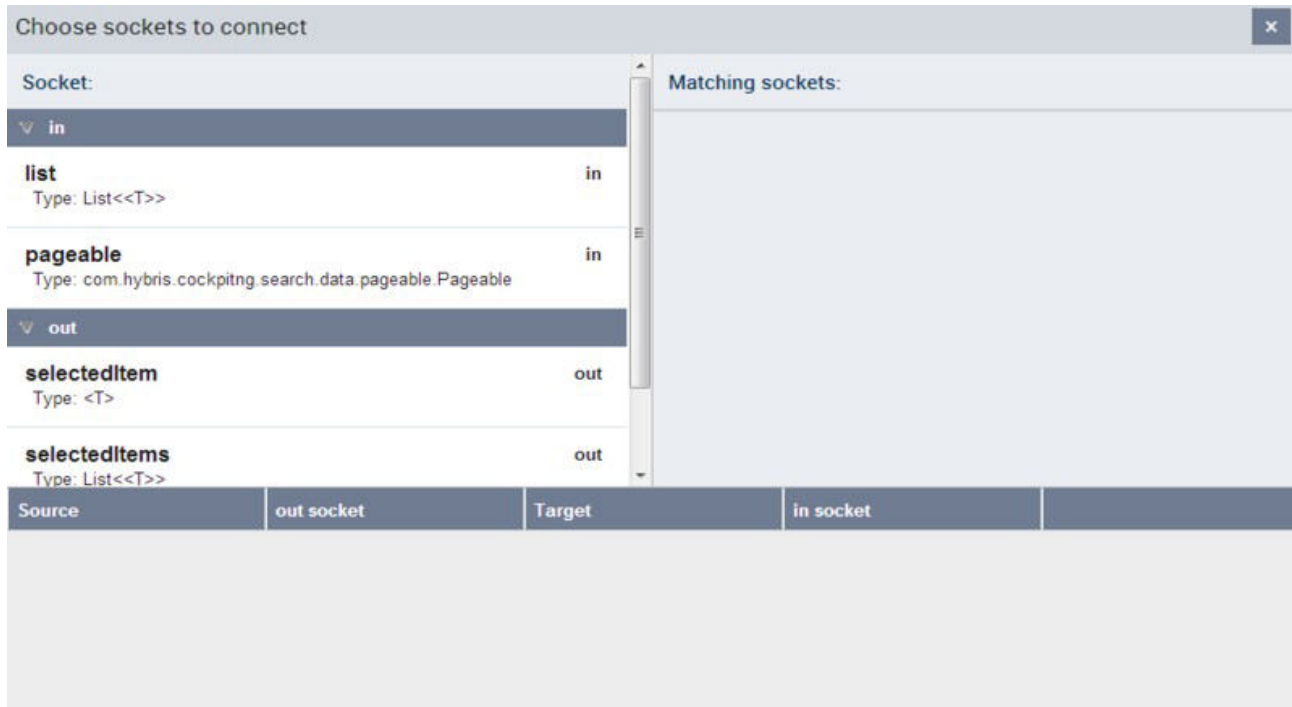
Target: com.hybris.cockpitng.listview

Source	out socket	Target	in socket	
34aaa03e-bcfc-468c-bb6d-8c940dcc44f3 : Simple Search	pageable	034bf489-ec7c-4bc9-af89-1fe936d3fdd8 : List View	pageable	X

Connecting a Widget to any other Widget by Clicking the Connect Icon

1. Click the **Connect**  icon for the widget in question (the widget can be the source or target).

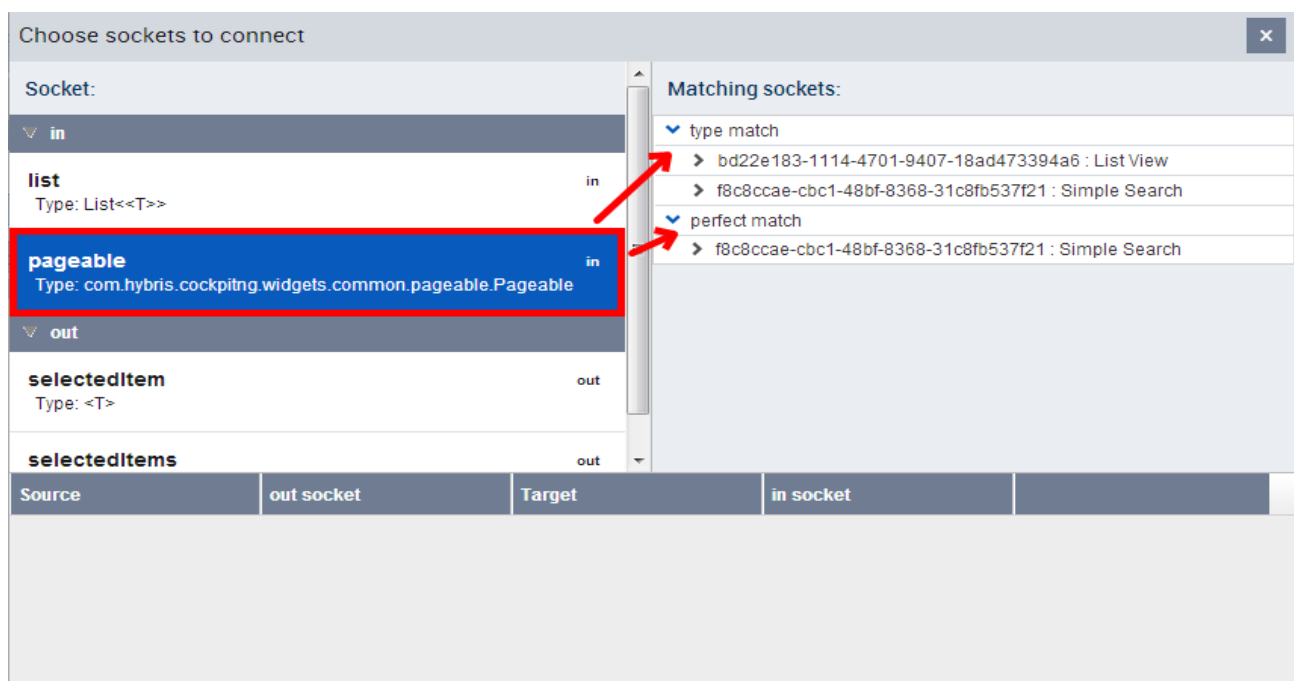
The **Choose sockets to connect** dialog box appears. The **Sockets** column lists all possible sockets (both inputs and outputs) for the widget instance.



2. From the **Source** column, select the socket you want to use for the connection.

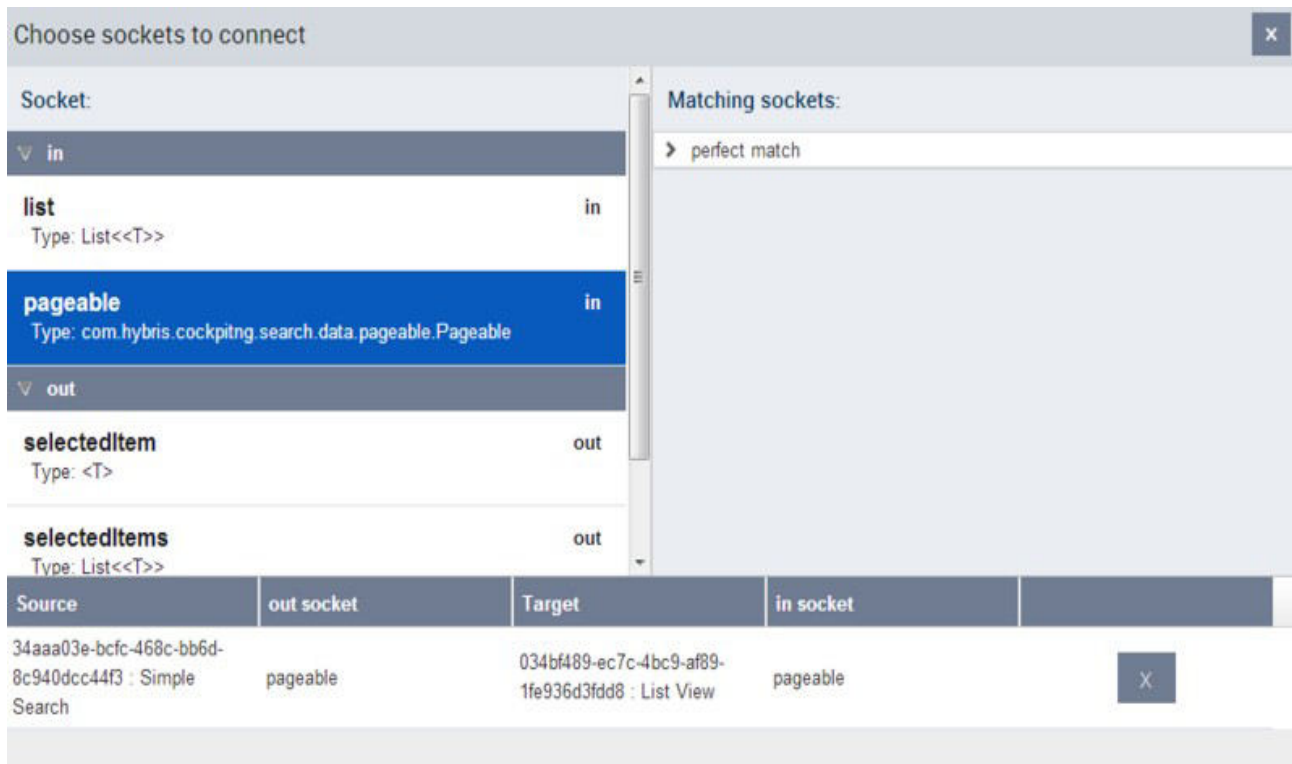
If any other widgets contain compatible sockets, a list of such sockets appears in the **Matching sockets** column. Click the items in the **Matching sockets** column to expand the list.

In the following example, the **pageable** input socket matches the output sockets of two other widgets based on type. Simple Search is listed a second time as a perfect match, meaning the widgets have the same socket ID, type, and multiplicity.





3. From the Matching Sockets column, select the socket that you want to connect with.

The connection is created, and a description for the connection appears at the bottom of the dialog box. In the following example, the Simple Search widget's **pageable** output socket was selected.



Displaying Widget Connections

To display widget connections, you can:

- Click the the Connect  icon. The widget's connections are displayed at the bottom of the dialog box.
- Right-click the Connect  icon. A small window appears with the connections listed.

Input connectors:

com.hybris.cockpitng.widgets.common.propextractor: genericOutput -> com.hybris.cockpitng.textsearch: enabled X
com.hybris.cockpitng.widgets.common.propextractor: genericOutput -> com.hybris.cockpitng.textsearch: enabled X

Output connectors:


com.hybris.cockpitng.textsearch: query -> com.hybris.cockpitng.widgets.common.simplesearch: searchtext X

In either case, to remove a connection, click the **X** icon at the right of the connection.

See also:

- [How To Orchestrate a Backoffice Application – Tutorial](#) section Connect Widgets
- [How to Pass Data Between Widgets - Tutorial - Prior 5.7.0](#)
- [How to Create Socket-Aware Actions and Editors - Tutorial](#)

Invisible Widget Children

All widgets can host invisible child widgets, which are hidden by view. Click the Toggle Invisible Widget Children  icon of a widget to show or hide a widget's invisible widget children.

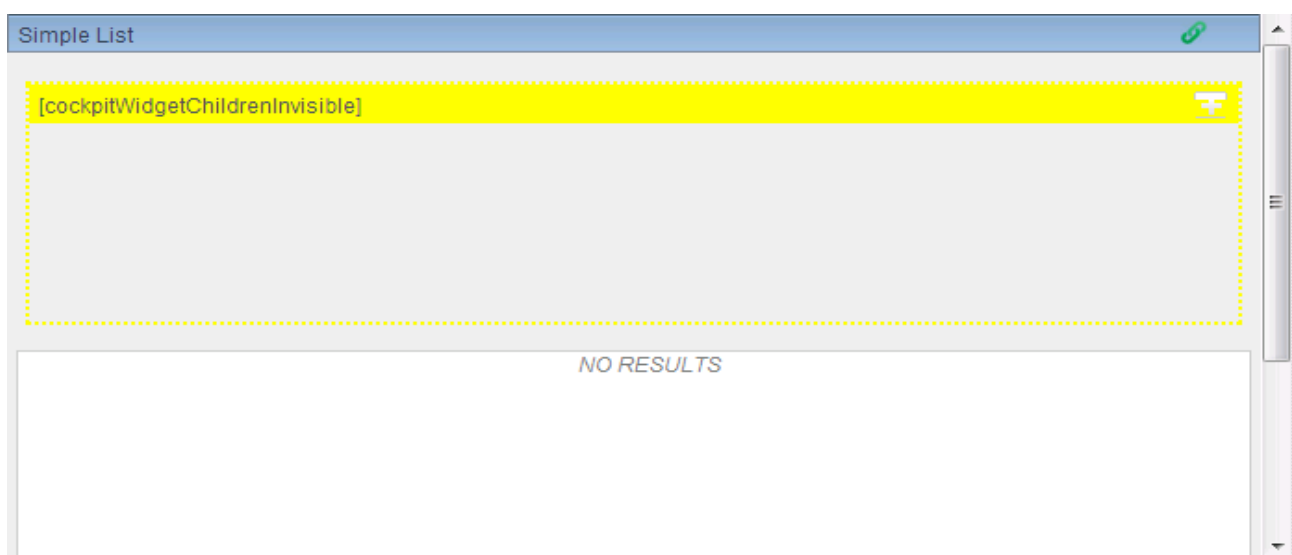
Invisible widget children are used when:


- The widget in question does not provide a user interface ([Condition Evaluator](#), for example)
- You want to use a widget's logic but hide its user interface
- The widget is to be displayed as a popup dialog (see [Widget Templates](#) for more information)

Adding Widgets to Invisible Child Slots from the Main Widgets View

1.Click the **Toggle Invisible Widget Children**  icon

The container for invisible child widgets appears.



2.Point at the title bar for the invisible children, and then click  at the far right.


The standard **Choose widget** dialog appears.

3.Select the widget group and then the widget itself.

See also:

- [How to Create Composed Widgets – Tutorial](#)

Widget Groups


Click the Create Widget Group  icon to create a widget group. Widget groups are made from widgets that can host other widgets, not including invisible widget children. For example, you can create a widget group from the [Collapsible Container](#) widget.

Widget groups can be reused in your application just like a normal widget, making it easier to design your application and to reuse the same widget mash-up in different locations.

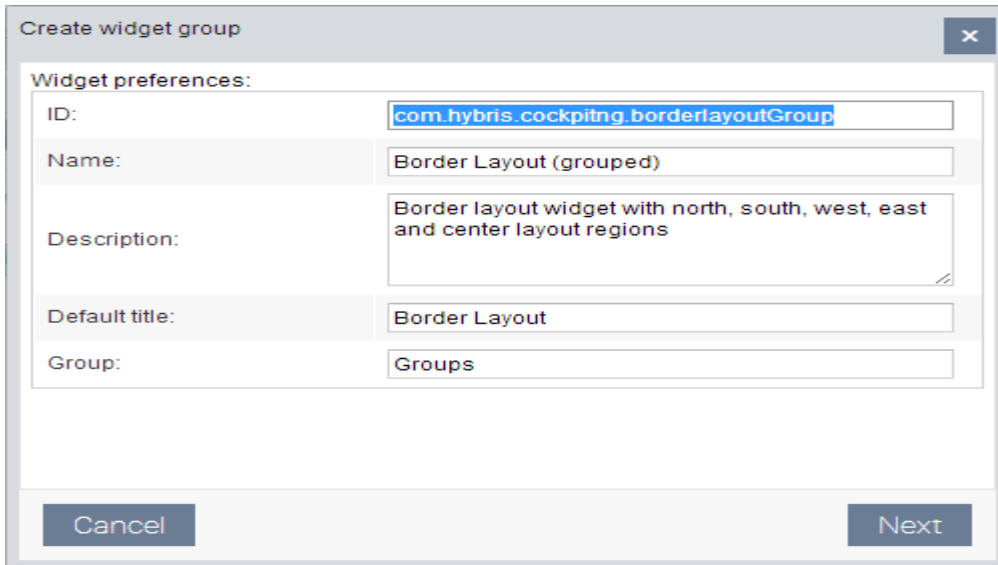
When a widget group is created, by default it is part of the Groups category. When you later add the widget group, you select it from the Groups category. This can be changed when you create the widget group.

Widgets in a group are not exposed to other widgets outside of the group, unless you configure specific sockets while creating a group.

Creating a Widget Group

1. Create the widget set that will be saved as a group.
2. From the the parent widget, click the Create Widget Group  icon.

In the following example, the parent widget is a **Border Layout** widget



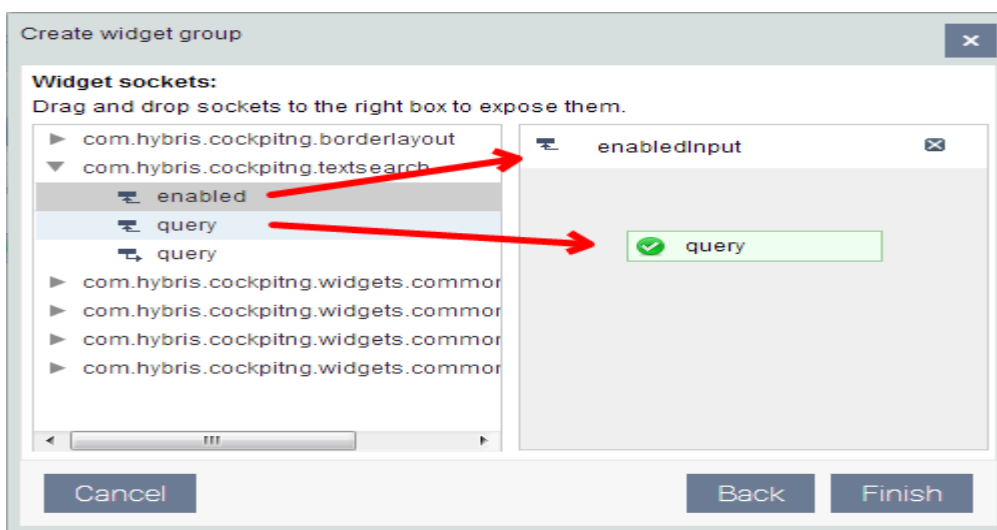
The image shows a 'Create widget group' dialog box with a close button (X) in the top right corner. The dialog contains a 'Widget preferences:' section with the following fields:

ID:	<code>com.hybris.cockpitng.borderlayoutGroup</code>
Name:	Border Layout (grouped)
Description:	Border layout widget with north, south, west, east and center layout regions
Default title:	Border Layout
Group:	Groups

At the bottom of the dialog are two buttons: 'Cancel' on the left and 'Next' on the right.

3. Edit the widget preferences, and then click **Next**.

4. Select the sockets that will be exposed to widgets outside of the group.



5. Click **Finish**.