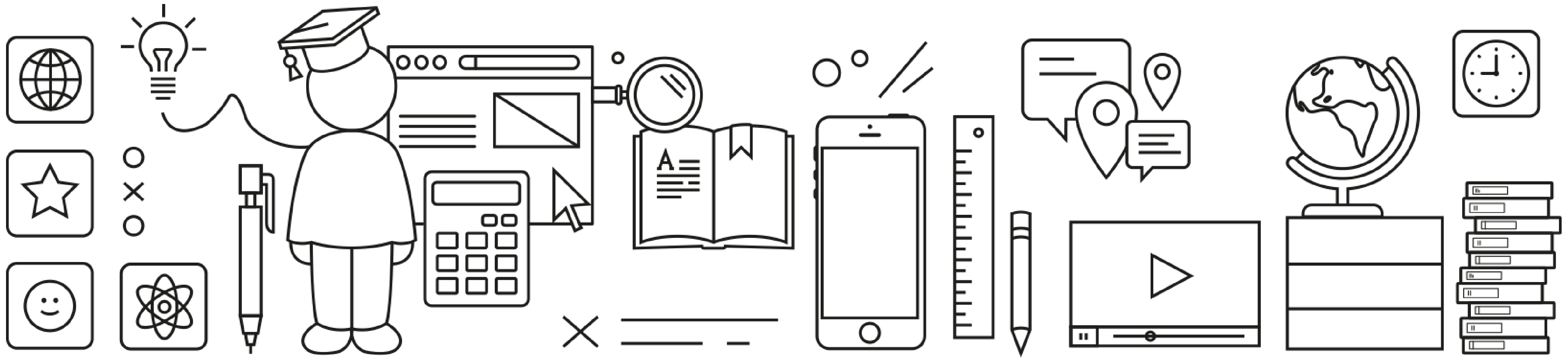
SAP Customer Experience

SAP Commerce Cloud Backoffice Framework Developer Training

Widget Communication

THE BEST RUN



Socket IO

Socket IO

- Widgets talk to each other via message objects with the help of the Backoffice Framework

- Input: listens for a ‘data receive’ event on a given input socket descriptor

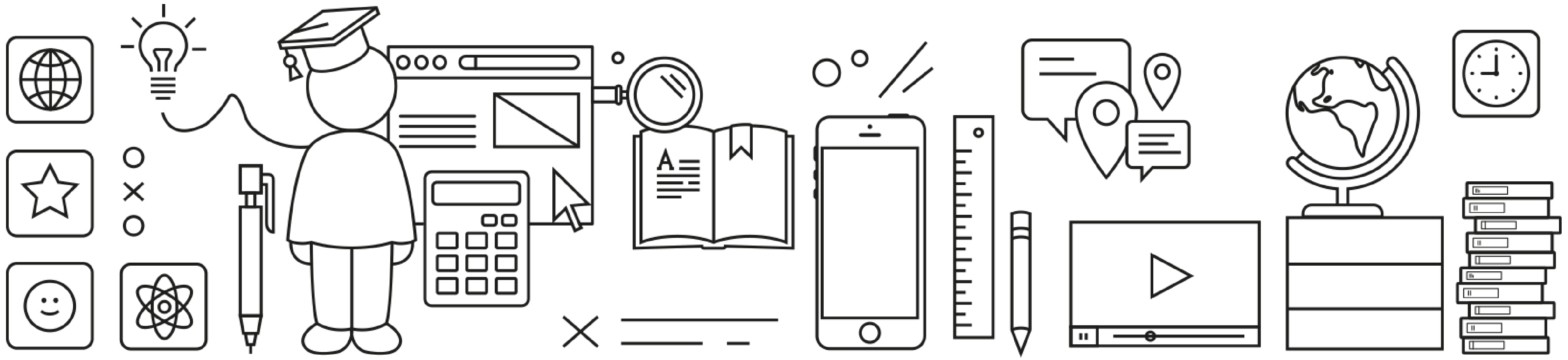
  - Fires method inside widget’s controller annotated with `@SocketEvent`:

```
@SocketEvent(socketId = "incomingMsg")
```

- Output: writes data to a given output socket descriptor

  Calls utility method inside widget’s controller:

```
sendOutput("outgoingMsg", entry);
```



Adding Input Sockets

Input Sockets

- In the `definition.xml`, define a `sockets` element with one or more child `input` elements
- Specify the socket `id` and `type` for each `input` element

`definition.xml`

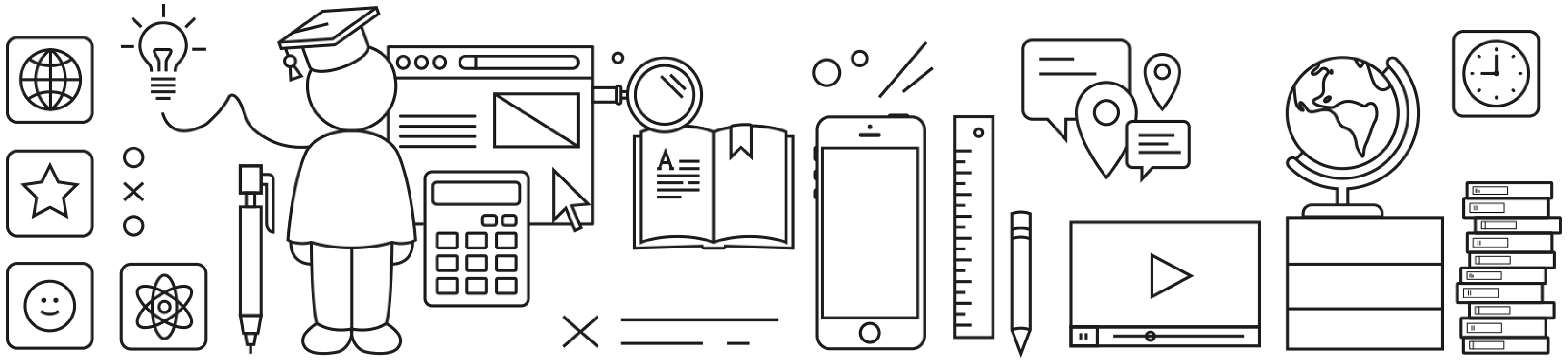
```
<widget-definition ...>
…
            <sockets>
                        <input id="incomingMsg" type="java.lang.String" />
                        …
            </sockets>
…
</widget-definition>
```

Input Socket Events

- Create a public method inside the widget's controller and annotate it using `@SocketEvent`
- The `socketId` and method input parameter type should match their definition in widget's `definition.xml`

MyChatController.java

```
@SocketEvent(socketId = "incomingMsg")
public void receiveMsg(final String msg){
		//do something with the msg
}
```



Adding Output Socket

Output Sockets

- In `definition.xml`, define a `sockets` element with one or more child `output` elements
- Specify the socket `id` and `type` for each **`output`** element

definition.xml

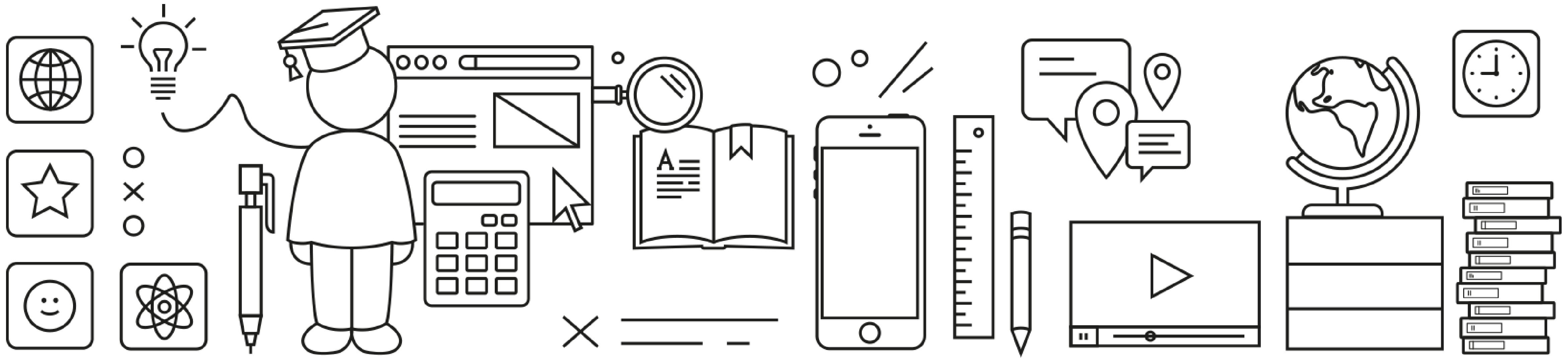
```
<widget-definition …

        <sockets>
                …
                <output id="outgoingMsg" type="java.lang.String" />
        </sockets>
…
</widget-definition>
```

Sending Data Over Output Sockets

In the widget’s controller, call `sendOutput` using the name of the `socketId` and the data content type that matches the type given to that output socket in the widget’s `definition.xml`.

MyChatController.java

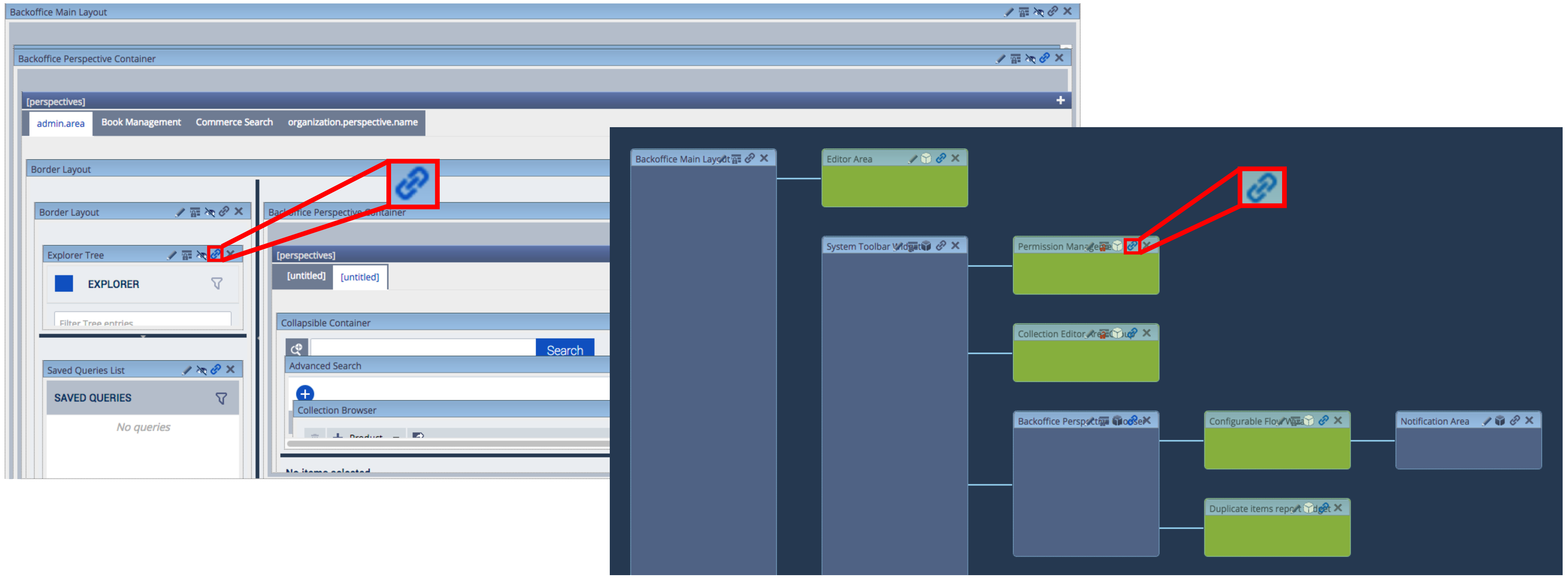
```
public void sendMsg(){
        //create some msg
        sendOutput("outgoingMsg", msg);
}
```



Connecting Sockets

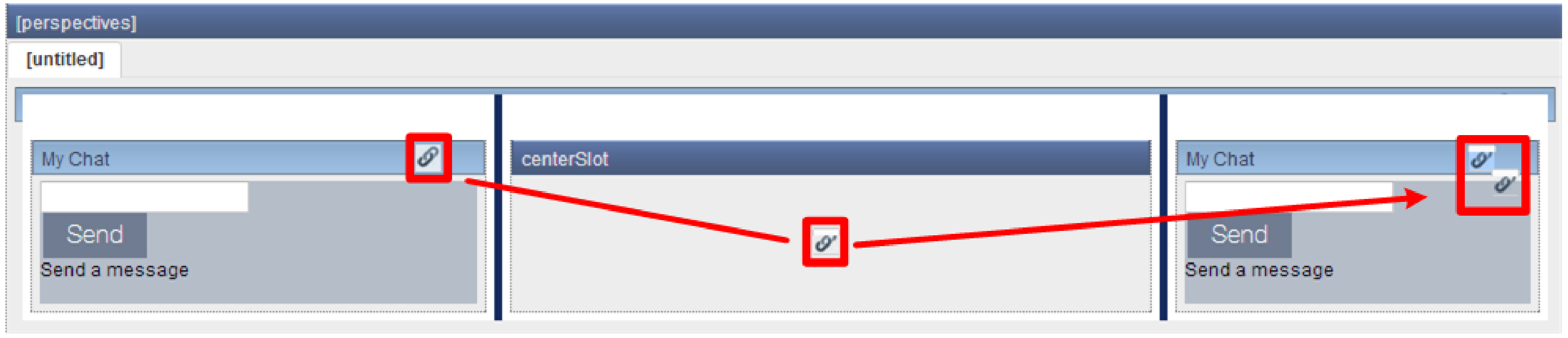
Connecting Using Application Orchestrator

- Login as *admin* and press F4 to switch into Application Orchestrator
- Can make connections in layout or symbolic widget view

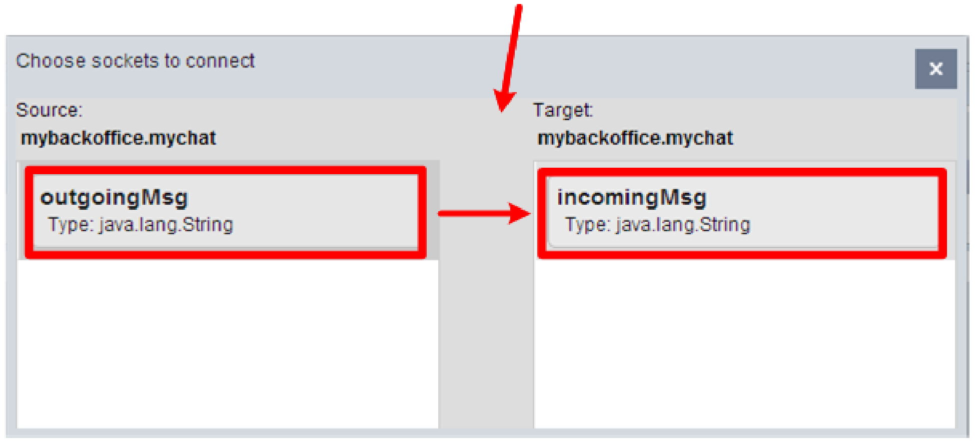


Connecting Using Application Orchestrator, cont.

- Connect source widget and target widget sockets by dragging and dropping chain link icon from source to target



- Select source socket – Application Orchestrator will suggest target sockets



Source:
mybackoffice.mychat

outgoingMsg
Type: java.lang.String

Target:
mybackoffice.mychat

incomingMsg
Type: java.lang.String

- **Socket types must match!**

Connections (`widgets.xml` , `*-backoffice-widgets.xml`)

- Define *source* and *target* widgets
- Define input and output socket IDs

`widgets.xml`

```
<widgets>
            …
            <widget-connection sourceWidgetId="myChatOne"
                                            outputId="outgoingMsg"
                                            targetWidgetId="myChatTwo"
                                            inputId="incomingMsg" />
            …
</widgets>
```

What if socket types don’t match?

Several options when faced mismatched socket types:

1. Create a *new* widget definition having desired, matching socket type, plus a new controller class to go with it.

2. Change the definitions of the existing widgets to get socket types to match

3. Use an adapter widget in between

4. Adjust the `socketDataType_$T` setting, if the source widget’s socket type is of the generic type `<T>`

Adapter Widgets

- Main purpose for having adapters:

  - If you need two widgets to communicate but socket types on the two ends don’t match.

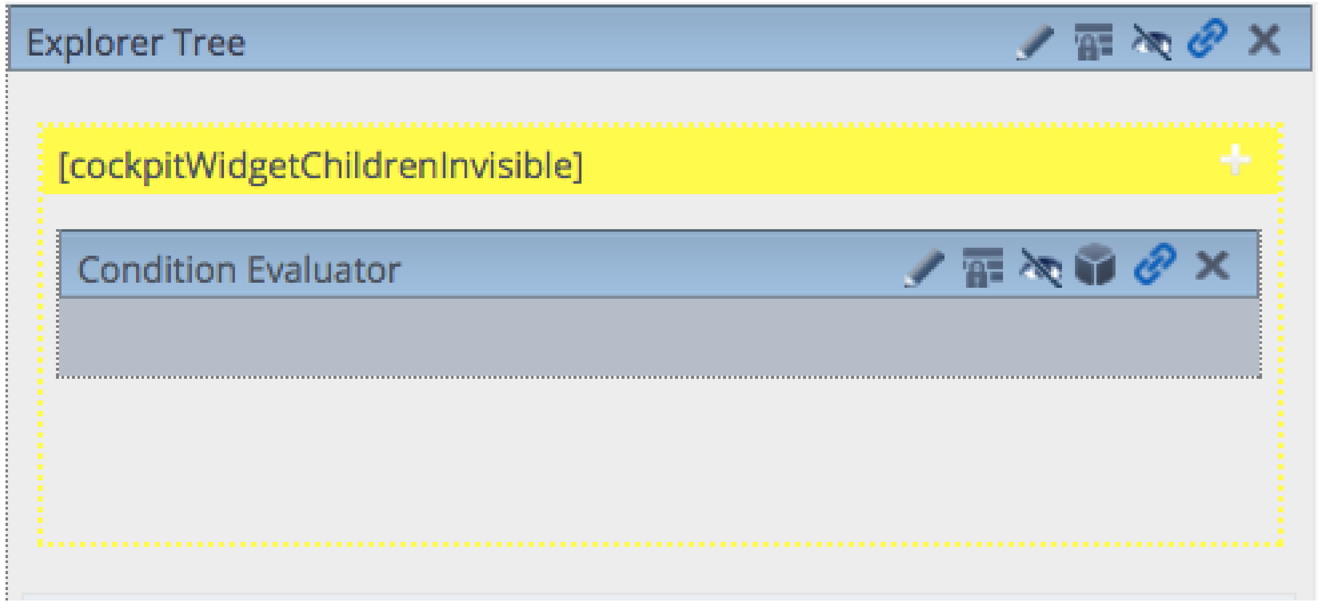
- Existing adapter widgets:

  - Logical NOT Gate Widget: to perform the logical NOT operation on a socket’s value
  - Cast Widget: to cast the type of a socket to any other type, given that it’s a *justified* cast
  - Condition Evaluator Widget: to set a condition (as a SpEL expression) to be evaluated
  - Property Extractor Widget: applies an expression on the input object and sends the result to the output
  - Event Producer Widget: emits events with the input data
  - Event Acceptor Widget: receives events of corresponding typed data

Adapter Widgets • Where should they go?

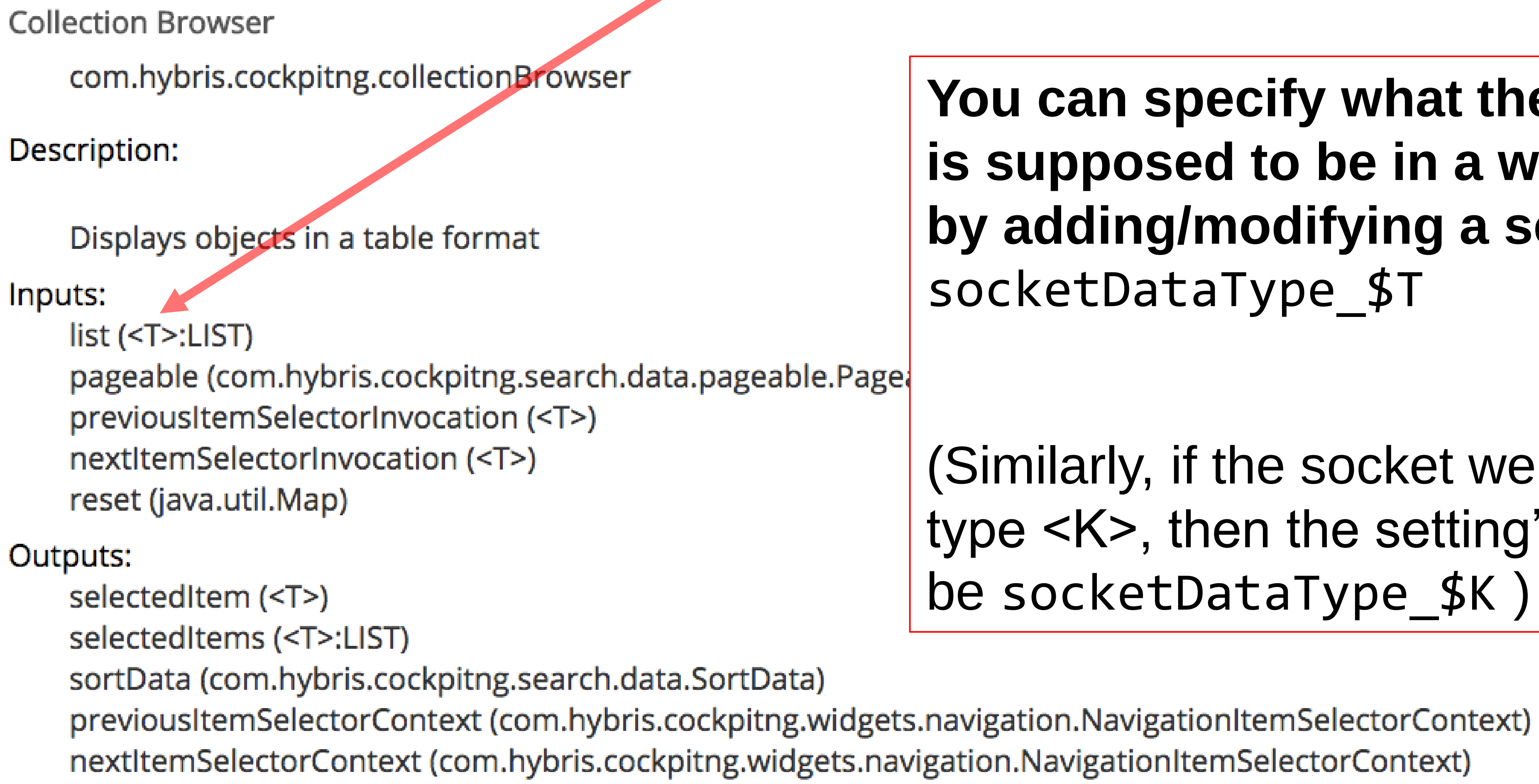
Remember cockpitWidgetChildrenInvisible slots?

- Every widget has a single invisible slot called `cockpitWidgetChildrenInvisible`
- It doesn't matter in which widget's invisible slot you put the adapter
- Only the socket connections of the adapter matter



`socketDataType_$T` **Widget Setting**

Sometimes a socket's type is in terms of the generic type `<T>`



Collection Browser

com.hybris.cockpitng.collectionBrowser

Description:

Displays objects in a table format

Inputs:

list (<T>:LIST)
pageable (com.hybris.cockpitng.search.data.pageable.Pagea
previousItemSelectorInvocation (<T>)
nextItemSelectorInvocation (<T>)
reset (java.util.Map)

Outputs:

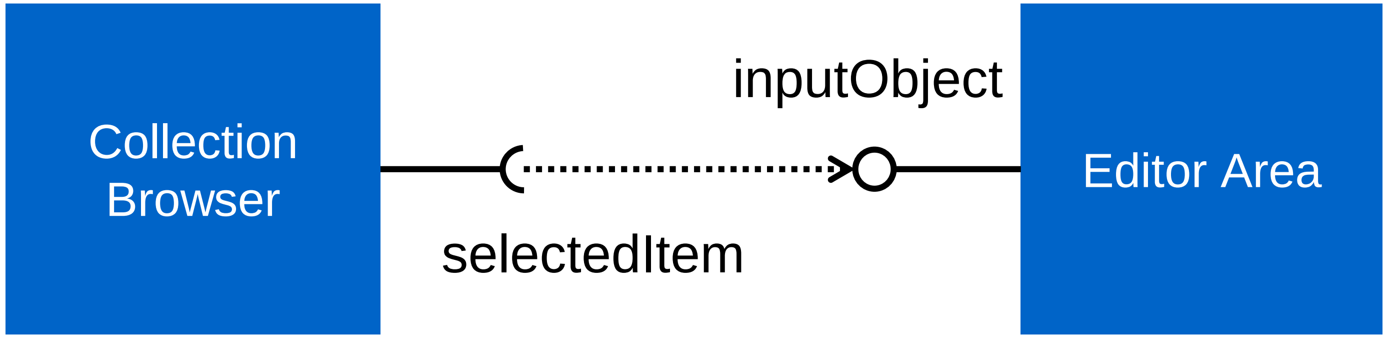
selectedItem (<T>)
selectedItems (<T>:LIST)
sortData (com.hybris.cockpitng.search.data.SortData)
previousItemSelectorContext (com.hybris.cockpitng.widgets.navigation.NavigationItemSelectorContext)
nextItemSelectorContext (com.hybris.cockpitng.widgets.navigation.NavigationItemSelectorContext)

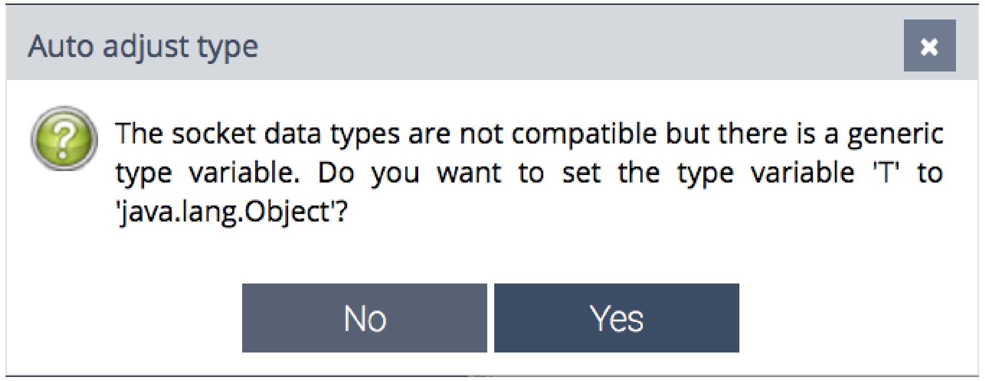
You can specify what the generic type is supposed to be in a widget instance, by adding/modifying a setting named `socketDataType_$T`

(Similarly, if the socket were of the generic type <K>, then the setting's name would be `socketDataType_$K`)

For Example

`<T> selectedItem` → `java.lang.Object inputObject`

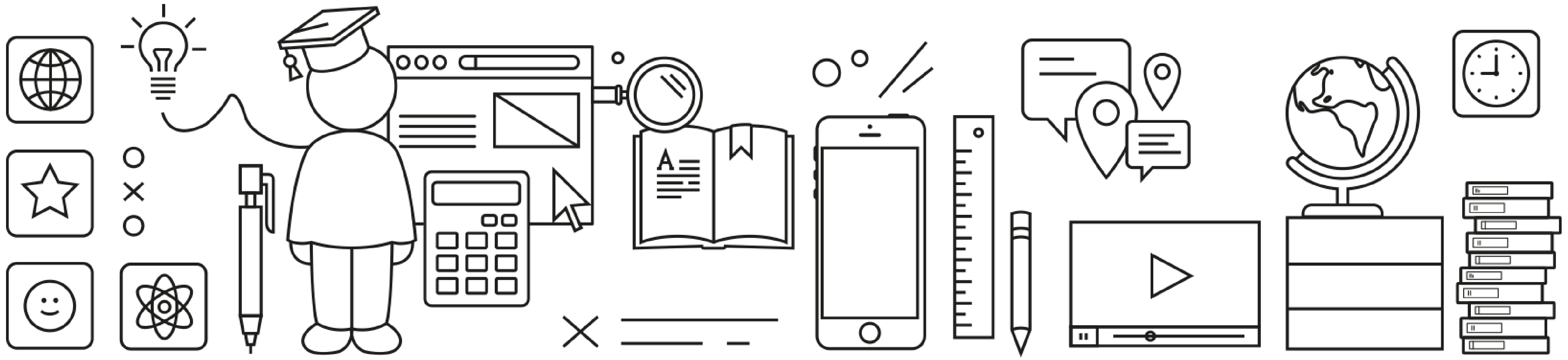



Auto adjust type

The socket data types are not compatible but there is a generic type variable. Do you want to set the type variable 'T' to 'java.lang.Object'?

Answering “Yes” will result in the `socketDataType_$T` setting being created automatically.

Demo



Mapping View Events

How To Map View Events (E.g., handling button clicks)

You can access view events in a controller, using

1. the ID of the component
2. the type of the event

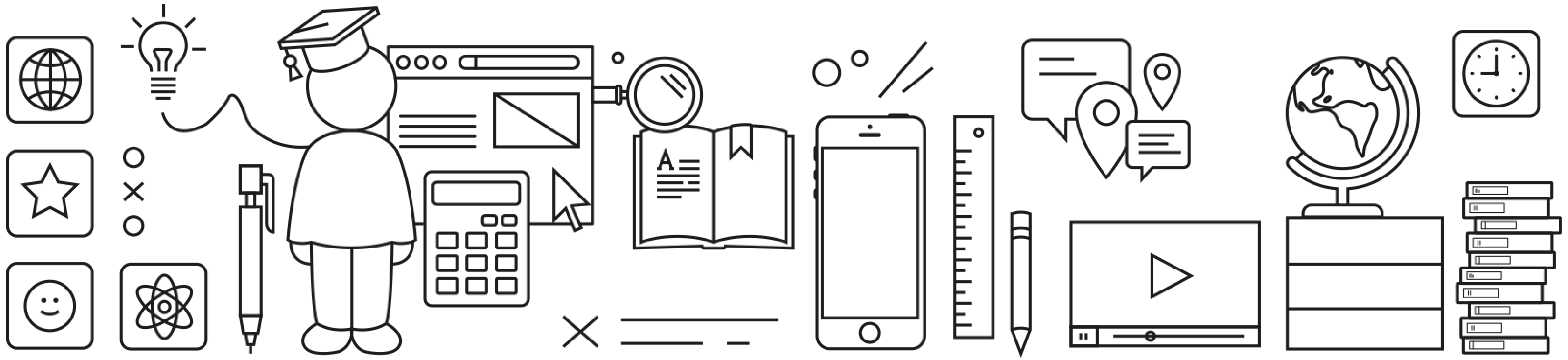
MyChat.zul

```
<?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>
```

Mapping View Events

MyChatController.java

```
public class MyChatController extends DefaultWidgetController
{
		private Label lastMsgLabel;
		private Textbox msgInput;

		...

		@ViewEvent(componentID = "sendBtn", eventName = Events.ON_CLICK)

		public void sendMsg(){

				String msg = msgInput.getText();

				sendOutput("outgoingMsg", msg);

		}

		...

}
```



Trapping Global Events

Global Events – Annotation

- Based on the Spring Event System

- Denote the listener method by using the @GlobalCockpitEvent annotation and specify which CRUD event you're trapping

The `ObjectCRUDHandler` interface has constants for all standard model-based event names that can be trapped (notice: ***OBJECT_CREATED_EVENT*** is singular, unlike the others):

- ObjectCRUDHandler.***OBJECT_CREATED_EVENT***
- ObjectCRUDHandler.***OBJECTS_UPDATED_EVENT***
- ObjectCRUDHandler.***OBJECTS_DELETED_EVENT***

`Controller class`

```
@GlobalCockpitEvent(eventName = ".." <, scope>)
public void handleEvent(final CockpitEvent event) { ... }
```

Global Events – Publishing

The controller class of your Widget, Editor, or Action can also *publish* a CRUD Global Cockpit Event to trigger refresh on all listening components:

MyXyzController.java

```
public class MyXyzController ... {
  @Resource
  private CockpitGlobalEventPublisher cockpitGlobalEventPublisher;

  @Resource
  private ModelService modelService;

  public ReturnType controllerMethod(final SomeContext<ProductModel> ctx) {
    final ProductModel currentProd = ctx.getData();

    currentProd.setSomeAttribute( newValue );
    try {
      modelService.save(currentProd);
      cockpitGlobalEventPublisher.publish(   //only on success
         ObjectCRUDHandler.OBJECTS_UPDATED_EVENT, currentBook, null);
    } catch (final ModelSavingException ex) { ... }
```

Global Events – Scope

Scope is optional!

Available scopes:

- CockpitEvent.*DESKTOP* (default, if unspecified)
- CockpitEvent.*SESSION*
- CockpitEvent.*USER*
- CockpitEvent.*APPLICATION*

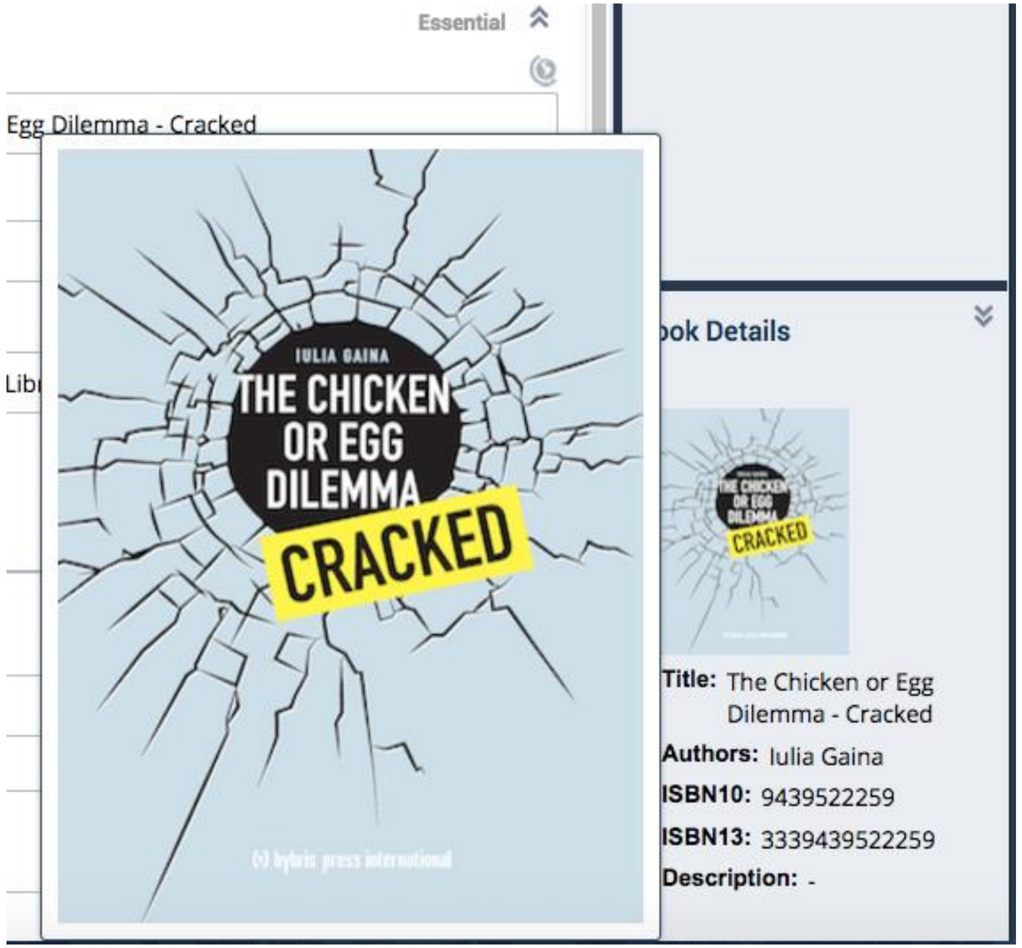
Exercise 6

Exercise 6 – Connect Your Widgets

- Create a *smarter* widget that

- Creates a popup preview ONLY if there is an existing image

- Can handle updates and deletions, and in turn update the view



Title: The Chicken or Egg Dilemma - Cracked
Authors: Iulia Gaina
ISBN10: 9439522259
ISBN13: 3339439522259
Description: -

1. Look at the Existing Socket Handler

Have a look at `handleSelectedBook(BookModel)`

BookDetailsController.java

```
@SocketEvent(socketId = SOCKET_SELECTED_BOOK)
public void handleSelectedBook(final BookModel book)
{
	LOG.info("Socket event is caught with Book: " + (book != null ? book.getName() :			"no book available"));
	setSelectedBook(book);
	render();
}
```

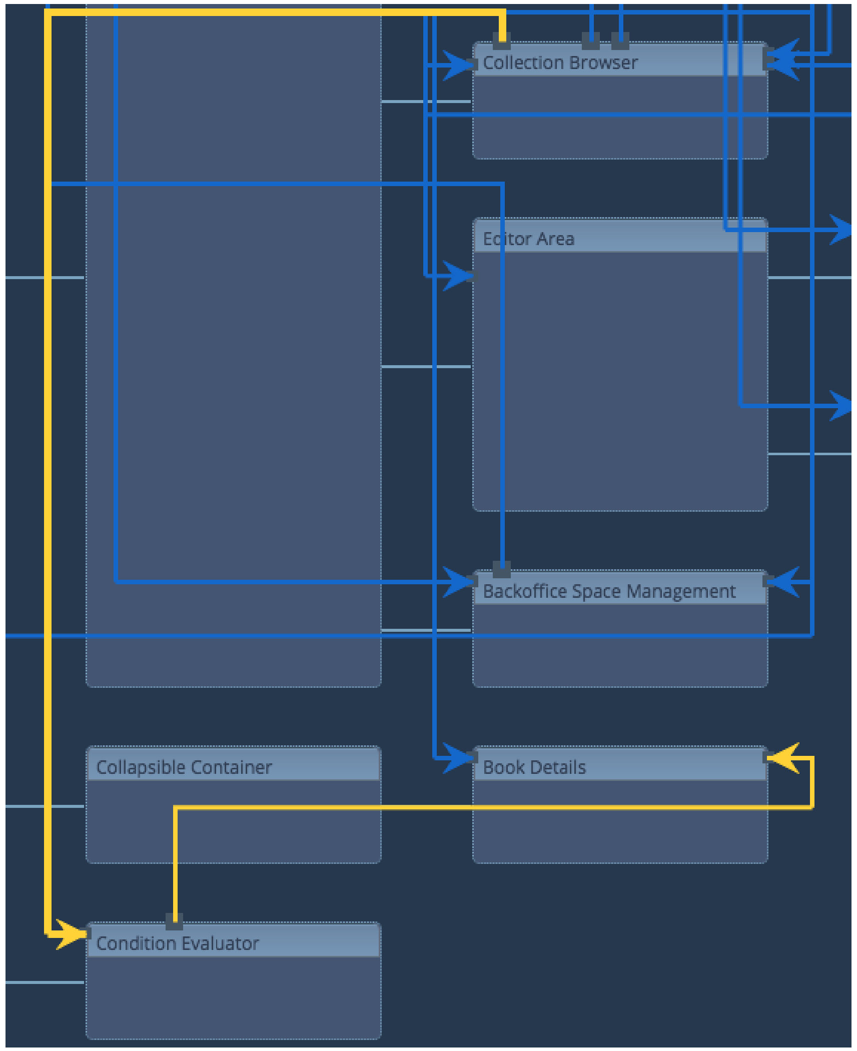
2. Add a New Input Socket

- **Create a `java.lang.Boolean` input to enable/disable previewing an image.**
  - **Widget definition**
  - **Widget controller (`handleAllowPreview(Boolean)`)**

3. Connect Through an Adapter

- Set the input using an adapter. The adapter checks if there's an image for the selected book.

- If an image exists, the adapter outputs `true` (if not `false`) to Book Details.



4. Trap Global Events

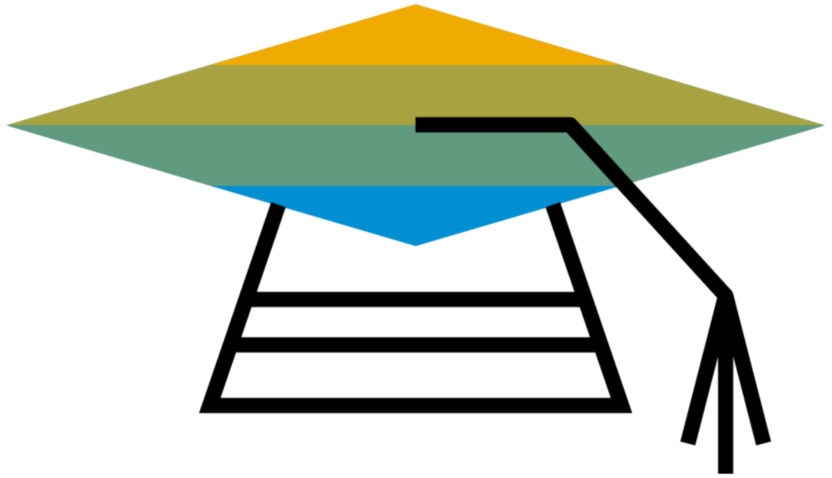
Create methods for handling updates and deletions.

– Trap update event (`handleObjectsUpdatedEvent(CockpitEvent)`)

– Trap delete event (`handleObjectsDeletedEvent(CockpitEvent)`)

SAP Customer Experience

Thank you.



THE BEST RUN