SAP Customer Experience

SAP Commerce Cloud Backoffice Framework Developer Training

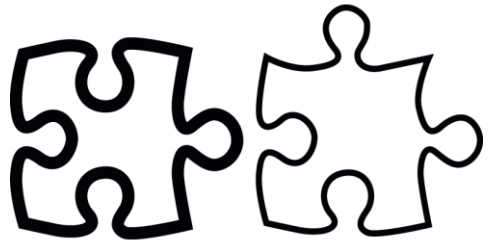
Widget Fundamentals

THE BEST RUN SAP

Creating a Widget

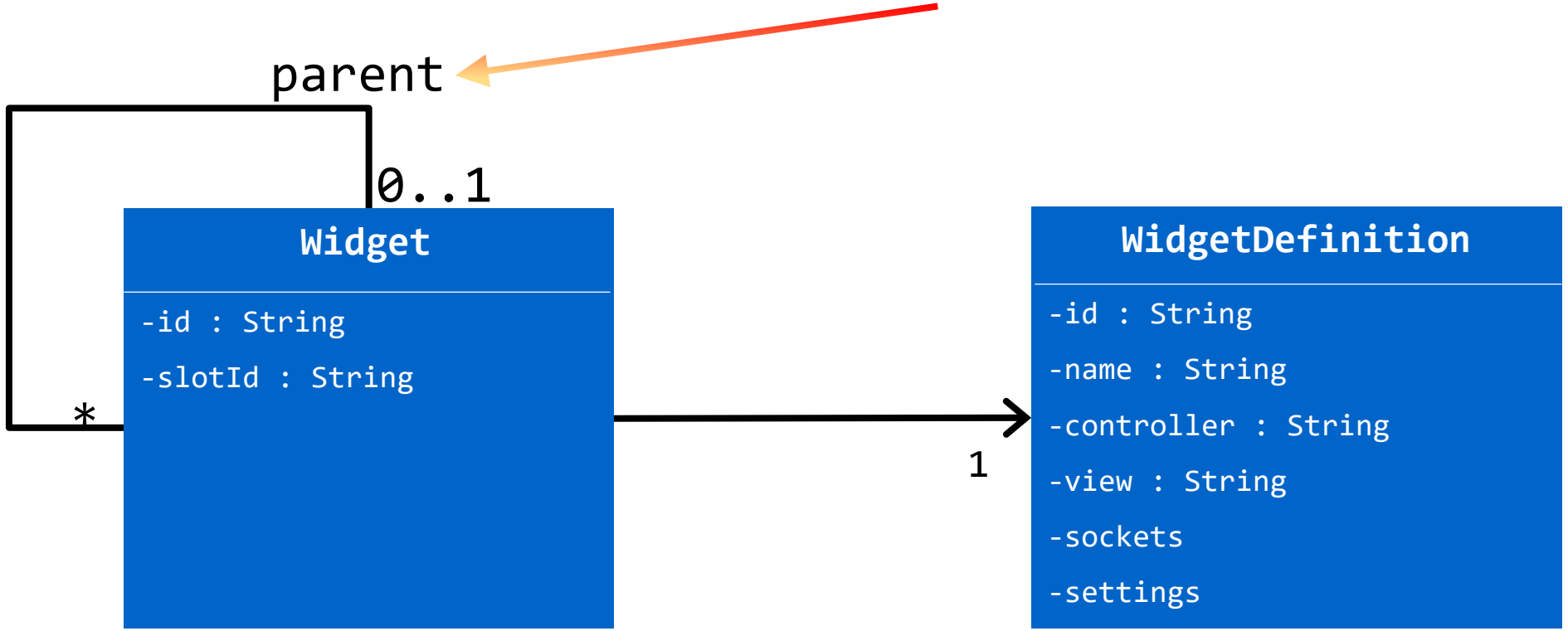
What is a widget?

- Standalone, deployable component
- Has a clearly-defined interface
- Has one specific purpose
- Primary component of the Backoffice framework
- Uniquely identified by an ID attribute



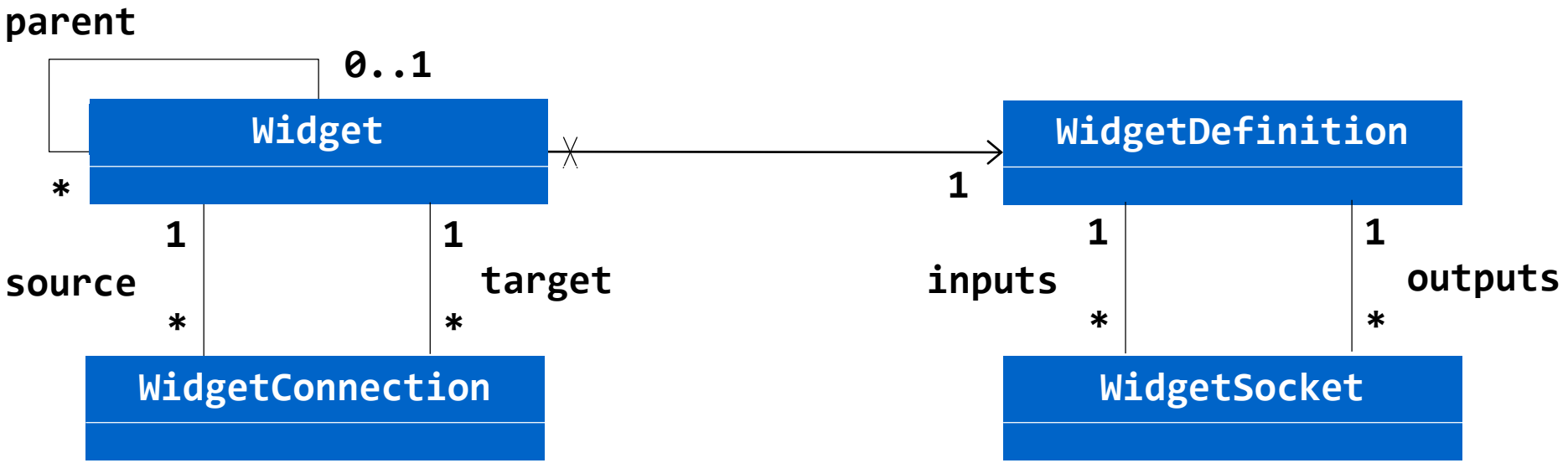
Widget *Definition*

Blueprint for all instances of a specific widget definition
(a widget instance typically lives in the slot of another widget instance)



Widget *Communication*

- To have a meaningful application, widget instances have to communicate with each other
- A widget can have inputs and outputs (“sockets”), each of which has its own ID (within the widget) and defined type
  - This helps with compatibility checking when connecting two widget instances
- Each connection references both source and target widget instances and the appropriate IDs of their input and output sockets



Components of a Widget Definition

- `<extensionName>/backoffice/resources/<folders>`
  - **`definition.xml`** (required file – must have this filename)
  - **view** file (optional) – A `.zul` file (only if widget has a user interface and wants to start with a static view template)
  - **static resources** (optional) – E.g., image, icon, .css files
  - **localization property files** (optional) – E.g., for localized UI labels
- `<extensionName>/backoffice/src/`
  - **Controller class** (optional) – A Java class, if widget is non-static

`definition.xml` Example

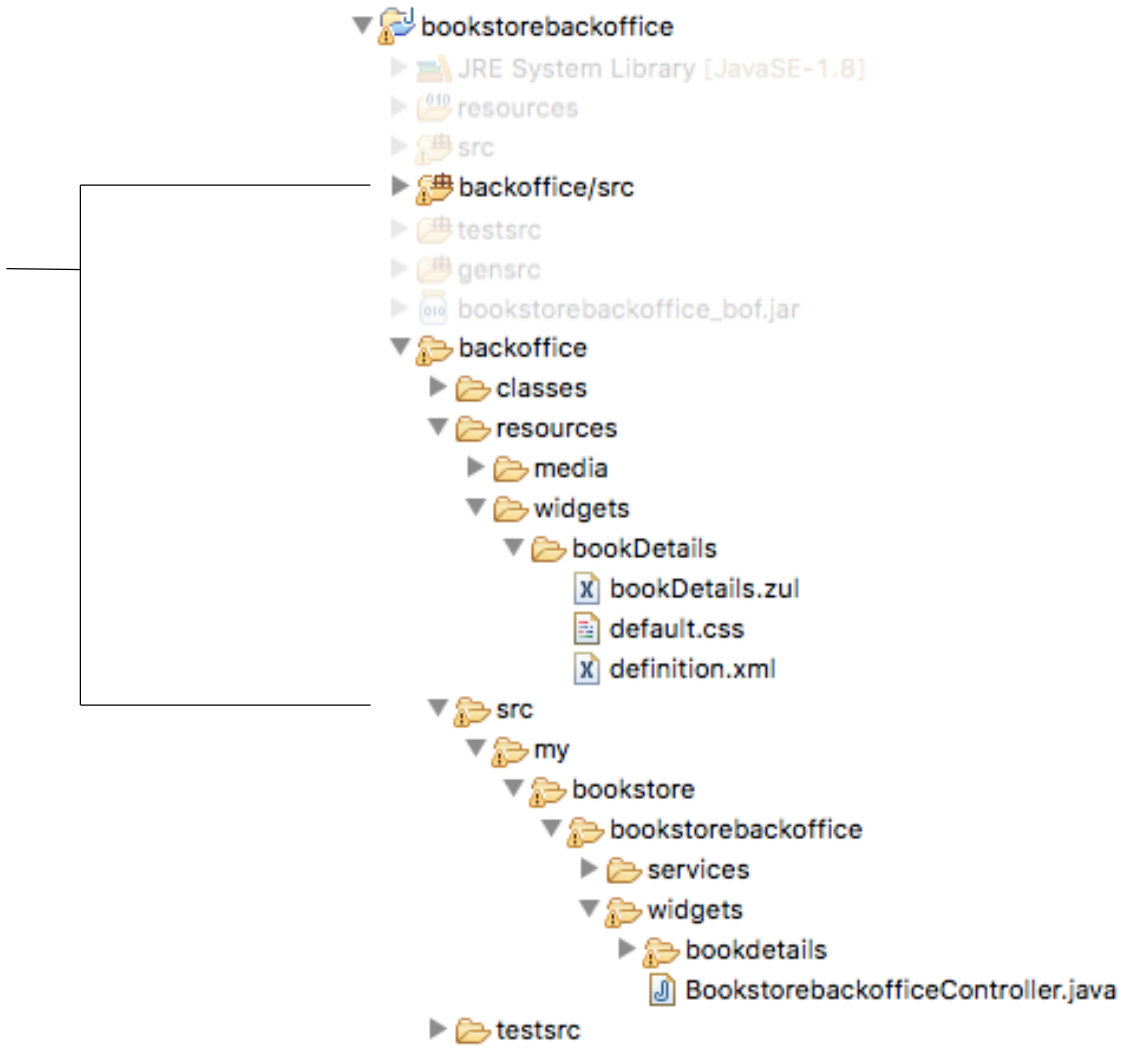
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<widget-definition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xsi:noNamespaceSchemaLocation="http://www.hybris.com/schema/cockpitng/widget-definition.xsd"
                   id="com.hybris.cockpitng.defaulttabwidget">
        <name>Tab Layout</name>
        <description>All rendered child components are displayed as tabs.</description>
        <defaultTitle>Tab Layout</defaultTitle>
        <author>hybris</author>
        <version>0.1</version>
        <controller class="com.hybris.cockpitng.widgets.controller.TabWidgetController"/>
        <view src="defaultTabWidget.zul"/>
        <sockets>
                <input id="closeAll" type="java.lang.Boolean" />     <!-- Java types used here -->
        </sockets>
        <keywords>
                <keyword>Layout</keyword>
        </keywords>
        <settings>
                <setting key="type" default-value="tab" type="String" />    <!-- Java types NOT used here -->
        </settings>
</widget-definition>
```

Widget File Structure

Creating a Widget

(Both point to the same location)

Where your Backoffice-related code should be implemented. It is where you can access and modify the Backoffice web app context.



Creating the View

UI Framework

Backoffice uses the ZK framework for creating the views

View files have `.zul` extension

In a view file, you'll find a combination of ZK XML syntax
+ SAP Hybris tags

Example view file – backofficeMainLayout.zul

myWidget.zul

```
<?xml version="1.0" encoding="UTF-8">
<?taglib uri="http://www.zkoss.org/dsp/web/core" prefix="c"?>

<widget width="100%" height="100%"
      xmlns="http://www.zkoss.org/2005/zul"
      xmlns:xsi="http://www.w3.org/2001/SMLSchema-instance"
      xmlns:h="http://www.w4.org/1999/xhtml">

  <vlayout id="mainContainer" width="100%" height="100%">
    <widgetslot slotID="headerArea" />
    <widgetslot slotID="mainArea" vflex="1" />
  </vlayout>

  <div id="roleSelectorContainer" sclass="yw-roleselector-container">
    <widgetslot slotID="roleSelectorSlot"/>
  </div>
</widget>
```

ZUL Schema

ZK XML syntax + SAP Hybris tags

myWidget.zul

```
<widget   xmlns="http://www.zkoss.org/2005/zul"
           xmlns:xsi="http://www.w3.org/2001/SMLSchema-instance"
           xsi:schemaLocation="http://www.zkoss.org/2005/zul
           http://www.hybris.com/schema/cockpitng/schemas/zul.xsd">
```

ZK	SAP Hybris
	<widget>
<hlayout>	<action>
<vlayout>	<widgetchildren>
<button>	<widgetslot>
<listbox>	
…	

Access to Controller and Model

Access methods and data of the widget instance's controller from a .ZUL file using SpEL syntax with the `widgetController` and `widgetModel` objects

myWidget.zul

```
<label value="${widgetModel.MODEL_KEY}"/>

<label value="${widgetController.getRandomNumber()}"/>
```

Wiring between Controller and View

Each element in the `zul` file is wired to a property in the controller through convention: by matching type and ID

MyChat.zul

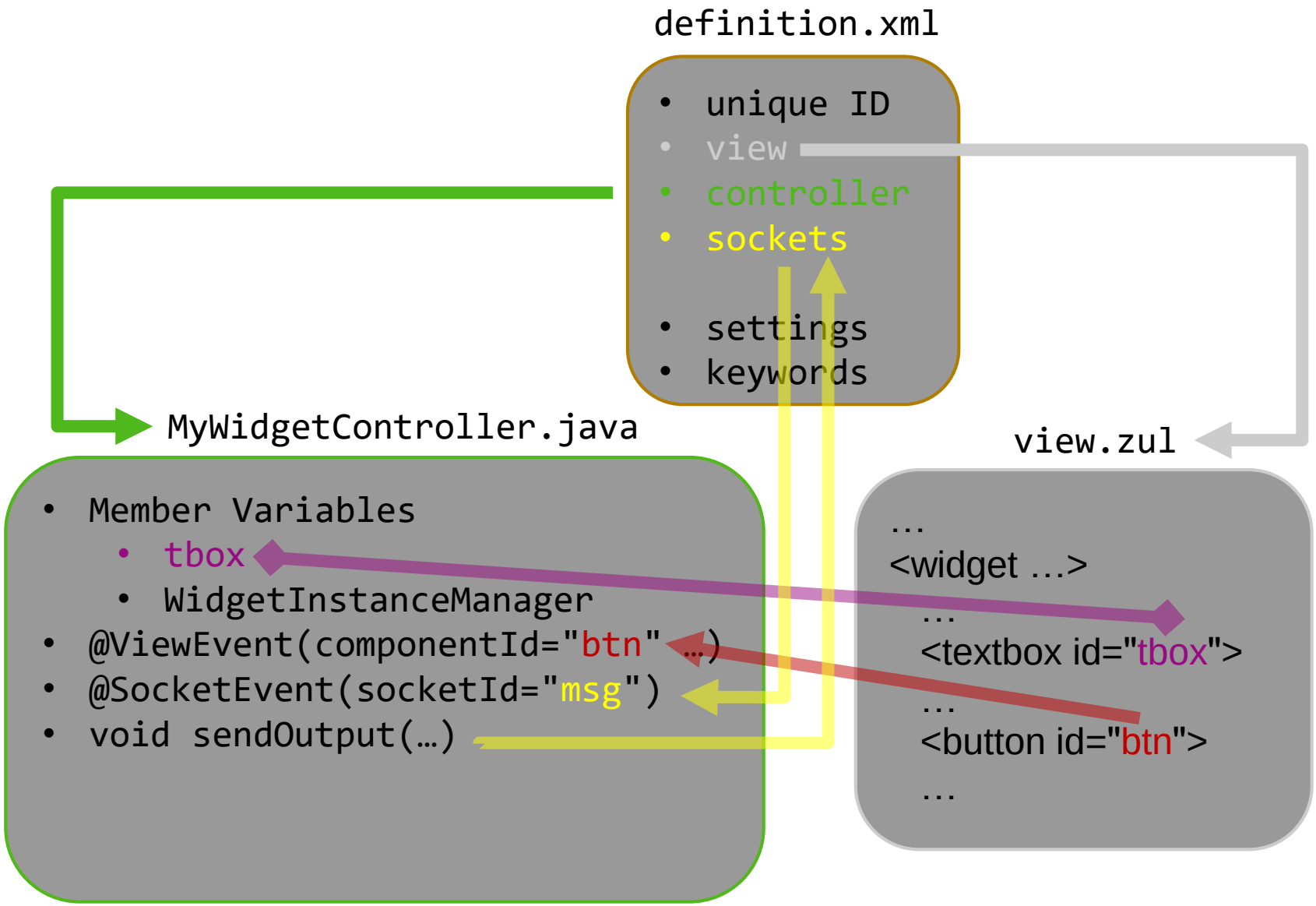
```
...
<div>
    <textbox id="msgInput"/>
...
```

MyChatController.java

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

Note that no mutator (setter) method is necessary. The value gets set automatically by the Backoffice Framework.

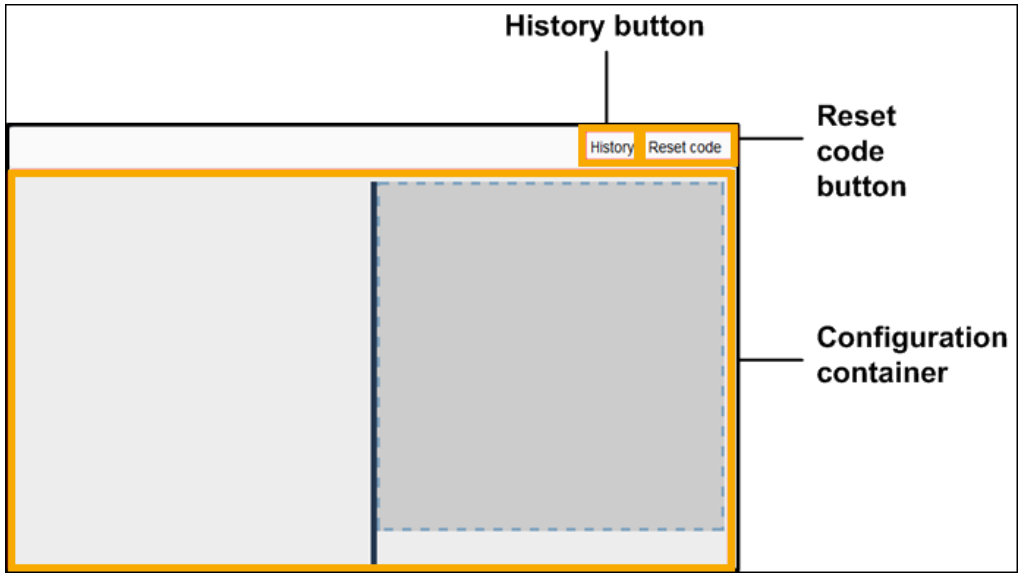
Wiring Between Widget Definition, Controller, & View



ZUL Playground Widget

- For testing ZUL file content on the fly

- Should be used in combination with
  - InputTestWidget widget
  - Cast widget



For more information on how to use this widget, refer to this page on the SAP Commerce Help:

ZUL Playground widget on Help

Creating the Controller

Custom Widget Controller

- Extend `DefaultWidgetController`
- Automatically *wire* view elements
  - By convention, the view element's id is identical to the controller's member variable name
- Use the `WidgetInstanceManager` to access context params
  - model data, instance settings, current user role, etc.

MyChatController.java

```
widgetInstanceManager.getWidgetSettings().getBoolean("online");
```

The Widget Controller Java Class

```
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 receiveMsg(final String msg)
	{
		lastMsgLabel.setValue(msg);
	}
}
```

Accessing the Settings

Accessing Widget Settings

definition.xml

```
<widget-definition ...>
    ...
    <settings>
        <setting key="mySetting" type="java.lang.String" default-value="123"/>
    </settings>
    ...
</widget-definition>
```

- Then inside your Controller code:

MyWidgetController.java

```
getWidgetSettings().getString("mySetting");
```

- Or inside your View code:

myWidgetView.zul

```
<label id="aLabel" value="${widgetSettings.mySetting}" />
```

Changing Widget Model

Widget Model

- Model lifecycle
  - Not to be confused with SAP Commerce Cloud platform's Model
  - Model is created when widget is created
  - Stored in session
  - Destroyed when session is closed or widget is removed
- Adding data to Model

```
WidgetModel model = widgetInstanceManager.getModel();
model.put("product", product);
```

- Retrieving data from Model

```
ProductModel product = model.getValue("product", ProductModel.class);
```

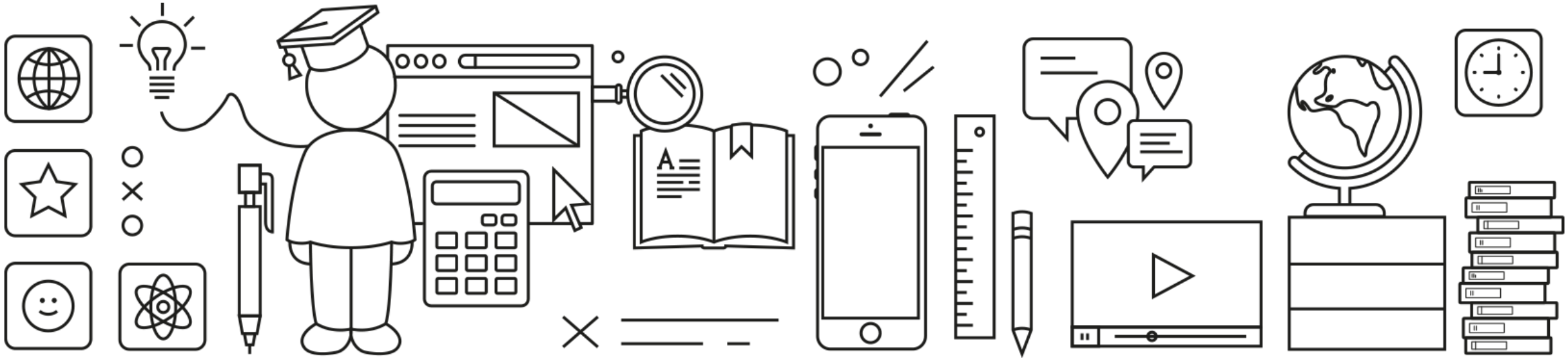
Widget Model, cont.

- Accessing values of an Object in the Model

```
ProductModel product = model.getValue("product", ProductModel.class);
String productCode = model.getValue("product.code", String.class);
String categoryCode = model.getValue("product.category.code", String.class);
```

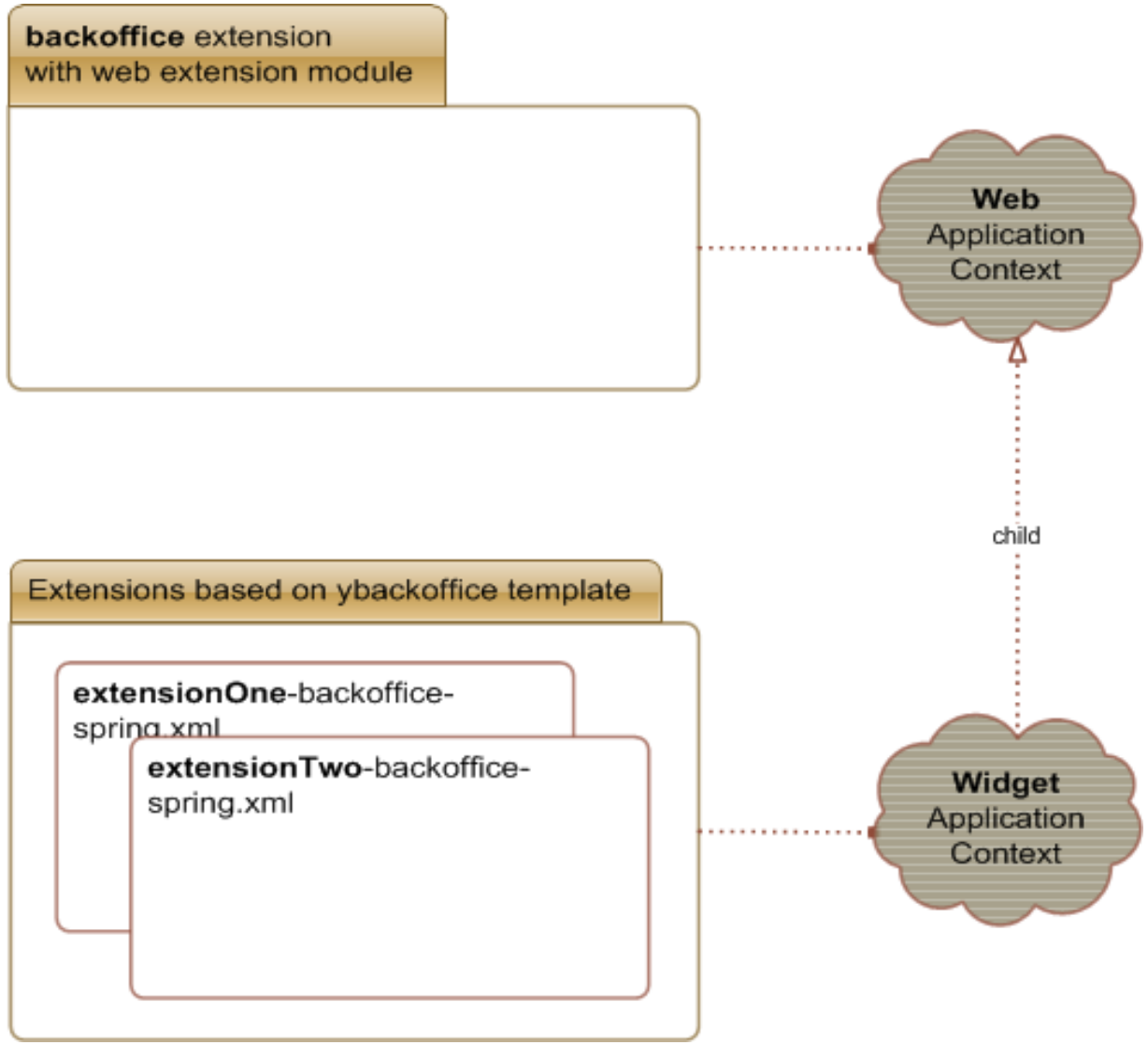
- Modifying data in Model

```
model.setValue("product.code", "prod1234");
```



Widget Application Context

Widget Application Context



Wiring Beans

- **`@WireVariable`** is used to access a bean inside a widget controller
  - Only used in controllers that extend `DefaultWidgetController`, `SelectorWidgetController`, or `BindWidgetController`, though!

```
@WireVariable
protected BackofficeBookstoreService backofficeBookstoreService;
```

- For all other wirings use the service locator - **`BackofficeSpringUtil`**

```
BackofficeBookstoreService backofficeBookstoreService =
        BackofficeSpringUtil.getBean(
         "backofficeBookstoreService", BackofficeBookstoreService.class);
```

Exercise 5

Exercise 5 – Create a New Widget

Create a widget for previewing a selected book!

- Complete the view’s definition
- Complete the controller’s implementation
- Add the new widget to the cockpit

Thank you.