hybris software

An SAP Company

hybris software
An SAP Company

hybris Developer Training Part I - Core Platform

Backoffice

Overview

Application orchestrator

Widget development

Backoffice configuration

Actions & Editors

The hybris backoffice

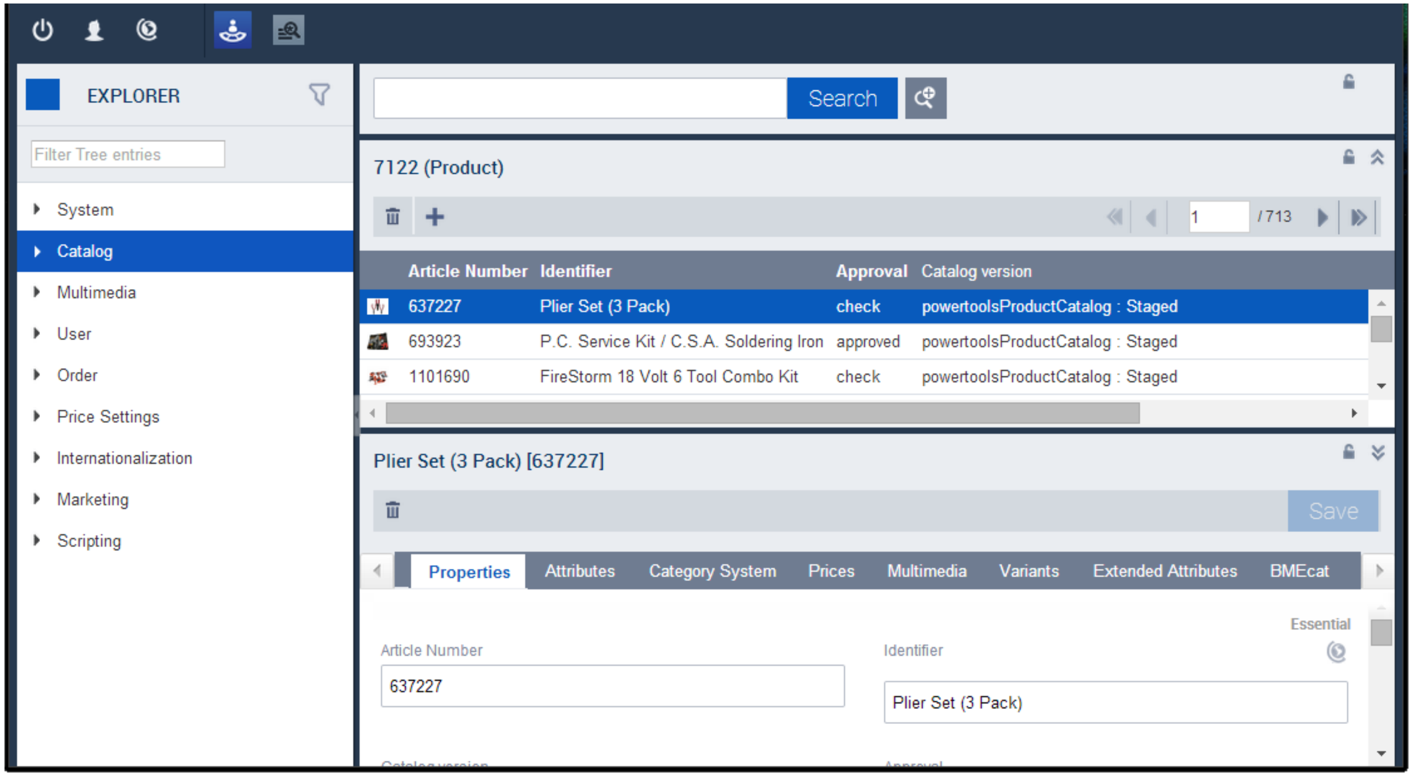
- A **Single** User Interface for all Backend Business Tools
- Built to Manage **any** kind of **data**
- For Business and Administrative Users
- Based on Next Generation Cockpit Framework (Cockpit NG)
- Highly **Extensible** and **Configurable**
- **Areas** (backoffice modules) available as of now:
  - Backoffice admin area
  - Commerce Search
  - OMS Cockpit & Admin

Key concepts of backoffice

- Web application with ready-to-Use library of UI components called **widgets**
- By combining and connecting widgets, a backoffice **business process** can be designed and put together quickly,
- The mashup can also be designed at runtime using the **Application Orchestrator** interface or **XML configuration**
- Business users will only see set of widgets and tasks that are made available to their **user role.** For e.g. OMS tasks are only visible to fulfillment users.

Backoffice admin area

- Preconfigured data and configuration tools for all platform extensions
- Default Mashup (Explorer Tree, Search, Result List, Editor Area)
- Will replace **hmc** in future



Article Number	Identifier	Approval	Catalog version
637227	Plier Set (3 Pack)	check	powertoolsProductCatalog : Staged
693923	P.C. Service Kit / C.S.A. Soldering Iron	approved	powertoolsProductCatalog : Staged
1101690	FireStorm 18 Volt 6 Tool Combo Kit	check	powertoolsProductCatalog : Staged

Article Number: 637227

Identifier: Plier Set (3 Pack)

Cockpit NG Framework extensions

- **backoffice** extension:
  - backoffice framework and admin area with standard widgets
  - Single web application which is a container for all backoffice modules
  - Contains the Application Orchestrator (**F4**) to design your custom mashup
  - Accessible by default at `http://localhost:9001/backoffice`

- **ybackoffice** extension template
  - Template you can use to generate a custom extension where you can implement your own backoffice module, widgets or UI configuration
  - You can also use the Application Orchestrator to add your custom components to the backoffice
  - Each extension implemented using ybackoffice will contribute to the main backoffice application.

hybris software
An SAP Company

Overview

Application orchestrator

Widget Development

Backoffice configuration

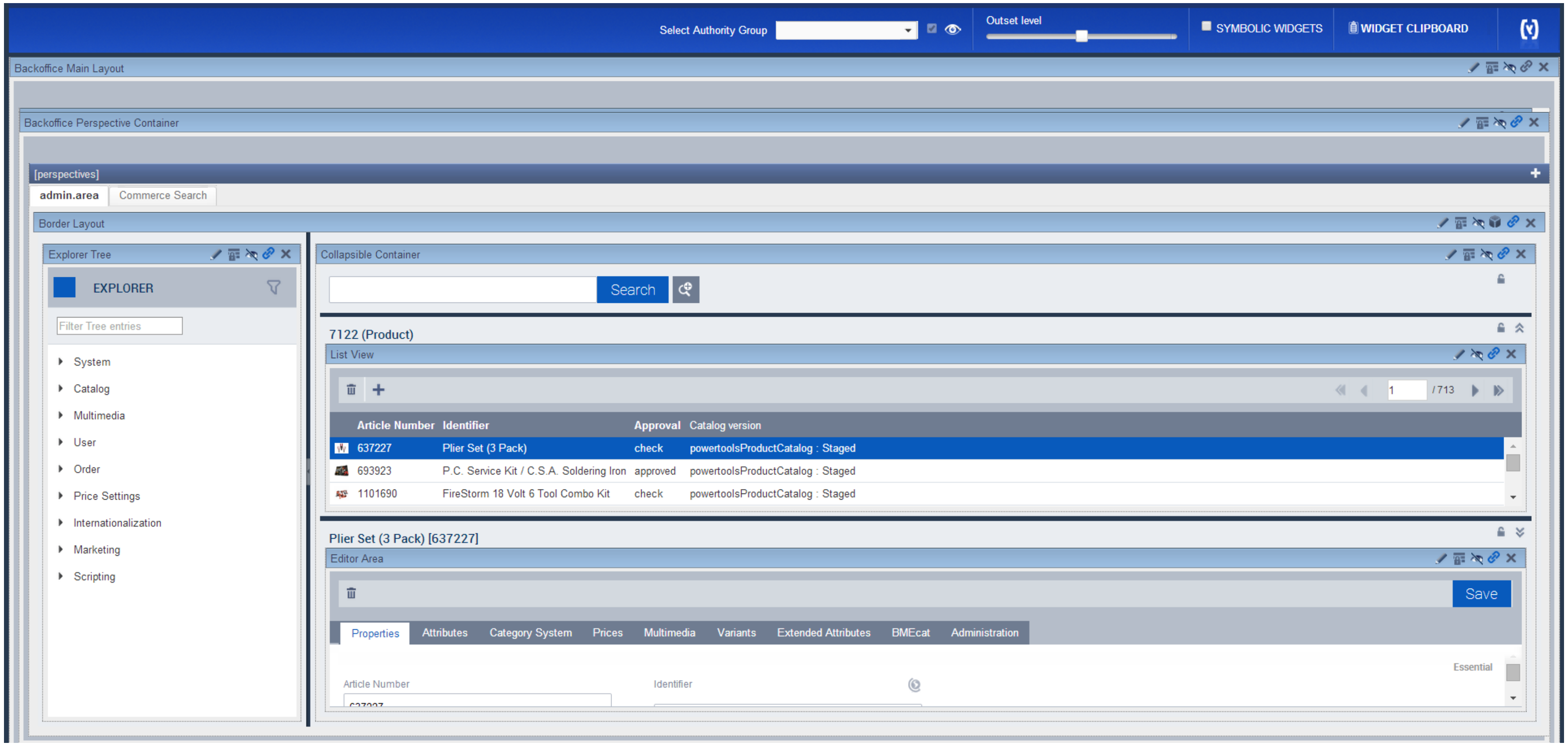
Actions & Editors

Application Orchestrator – key features

- Accessible by pressing **F4** key in the `backoffice` application
  - only for administrators.
- Add, remove and connect widgets, areas and other components
- Add or change settings of a widget instance
- Connect widgets using their sockets
- Manage UI localization by providing values for labels.
- Manage access restrictions (allowed roles)
- Access and reset application configuration (widgets.xml) and UI configuration (cockpit-config.xml)
- Inspect application mash-up and communication flow
- Consists of two views

Main view

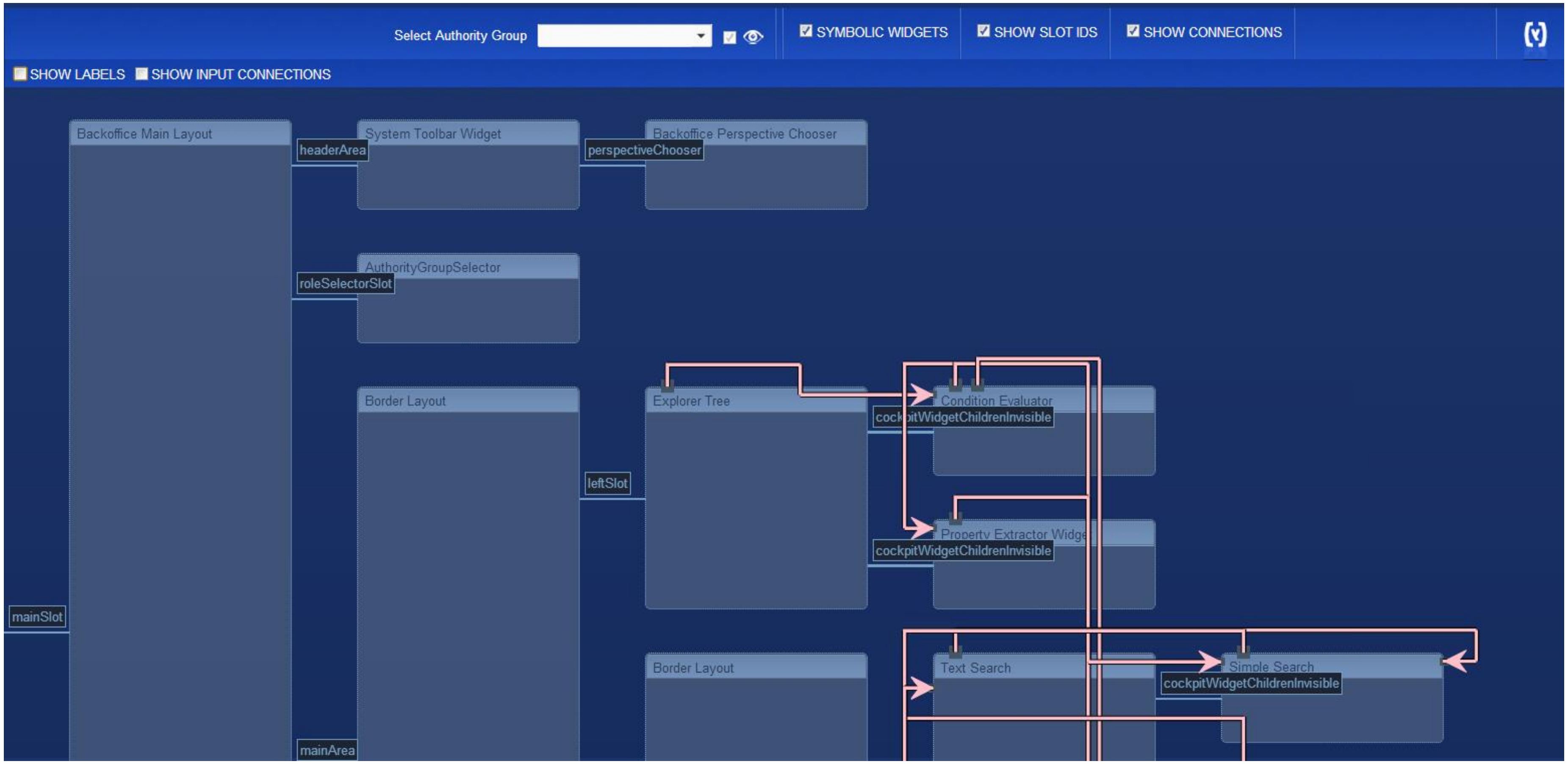
The main widget view is a place where you should add widgets and design a mash-up of your application. Here you can see two types of components: widgets and slots/children.



Article Number	Identifier	Approval	Catalog version
637227	Plier Set (3 Pack)	check	powertoolsProductCatalog : Staged
693923	P.C. Service Kit / C.S.A. Soldering Iron	approved	powertoolsProductCatalog : Staged
1101690	FireStorm 18 Volt 6 Tool Combo Kit	check	powertoolsProductCatalog : Staged

Symbolic widgets view

The symbolic widgets view helps to get an overview of your application's mash-up. It shows widget dependencies and how connections are established. You can also add widgets here.



hybris software
An SAP Company

Overview

Application orchestrator

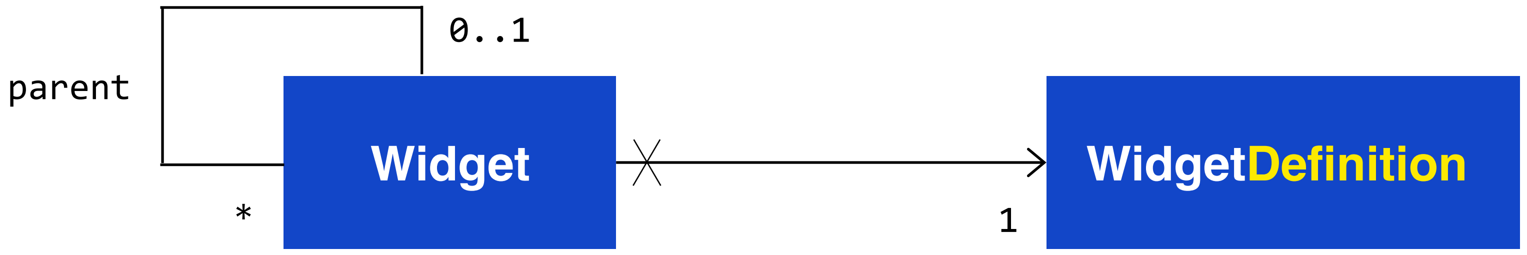
Widget Development

Backoffice configuration

Actions & Editors

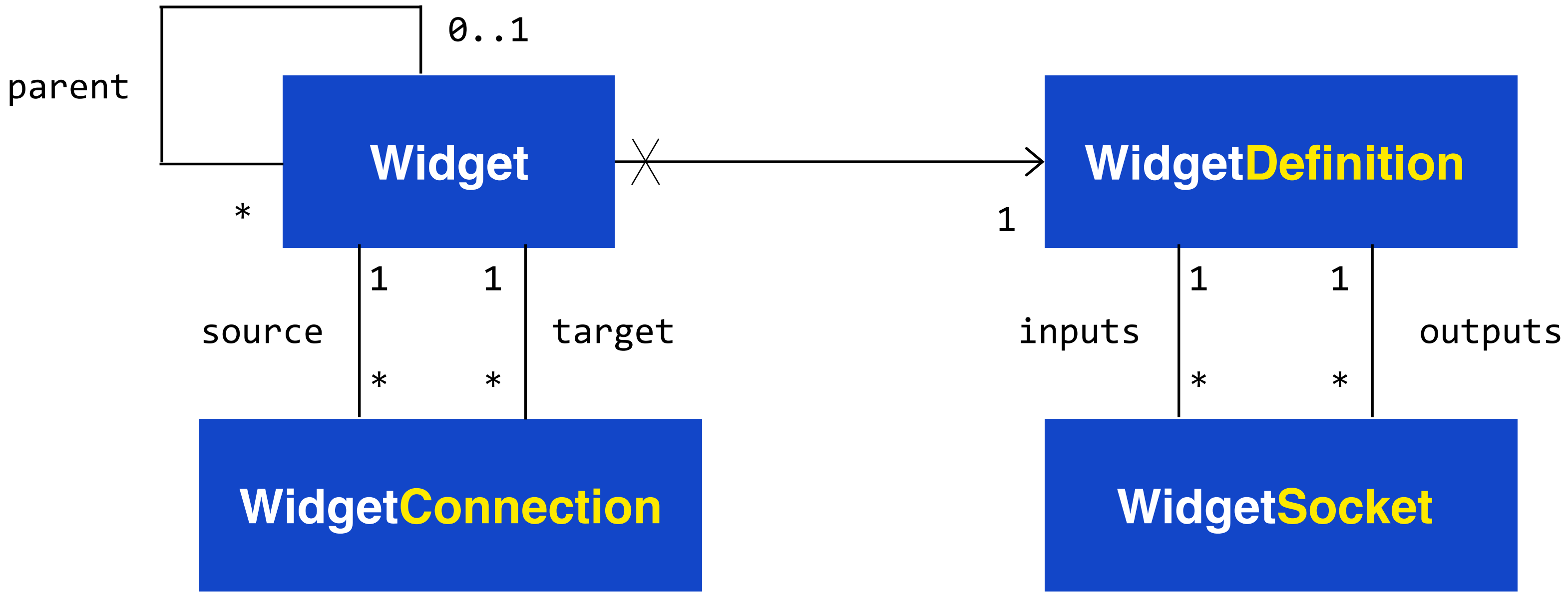
Widgets

- A widget is a stand-alone, **deployable** component with a clearly defined interface and a specific purpose.
- A Widget **definition** describes a widget type, its ID and name, its **view** and **controller** classes. Also any additional settings specific to that widget
- A widget used in an application is an **instance** of a widget definition.
- You can take a widget sub-tree (a widget group) and reuse it in other application, just like a single widget.



Widget communication

- To have a meaningful `backoffice` application, widgets have to communicate with each other.
- A widget can have inputs and outputs, called sockets, which have an ID and a type definition. This helps with compatibility checking when connecting two widgets.
- Every widget **connection** references both source and target widgets, and the IDs of the input and output sockets.



What constitutes a Widget?

- **Widget definition (definition.xml)**
  - Component interface i.e. inputs and outputs
  - Other meta info e.g. controller class, view file, author, name
- **Widget view (.zul)**
  - Standard ZK XML syntax with added hybris **tags**
  - Change and refresh browser. No compile required!
- **Controller**
  - **Java** class containing business logic
  - Standard syntax allowing you to react to inputs and component events

Example widget definition: Chat widget

myextension/backoffice/resources/widgets/mychat/definition.xml

```xml
<widget-definition id="org.cuppytrailbackoffice.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>

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

    <controller
        class="com.hybris.cuppytrailbackoffice.widgets.MyChatController"/>

</widget-definition>
```

→ **`myextension`** is a custom extension generated using **`ybackoffice`**

Example widget view: Chat widget

view.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>
```

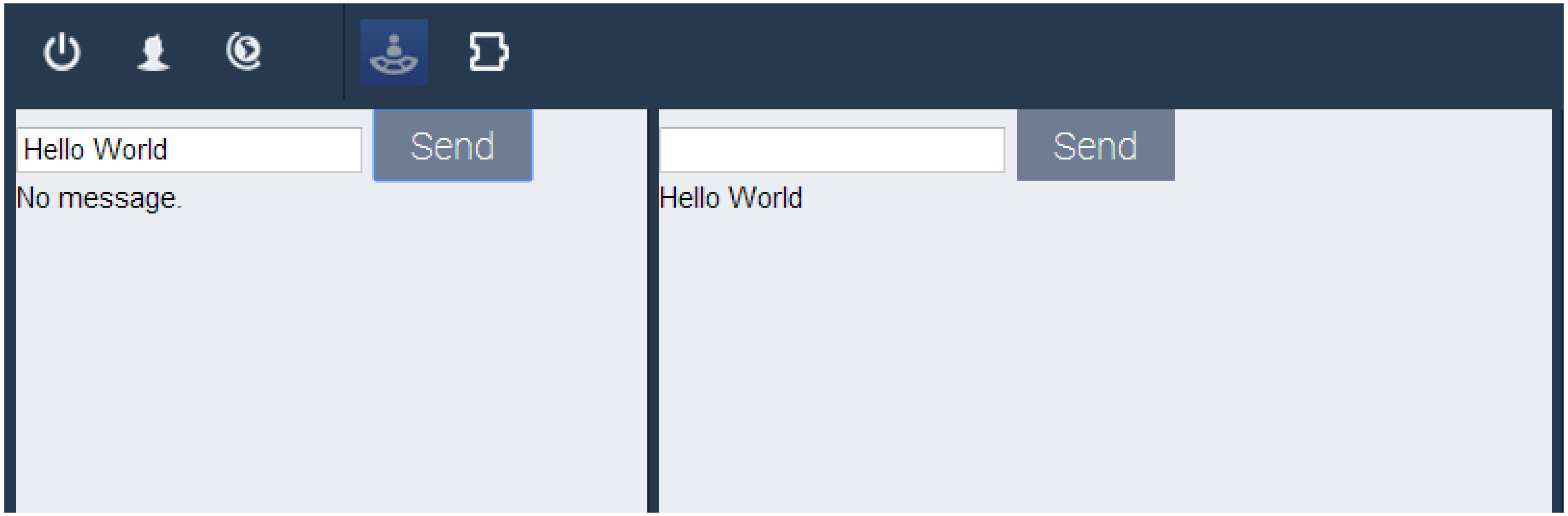
Example widget controller: Chat widget

```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);
    }
}
```

Chat widget in backoffice



hybris software
An SAP Company

Overview

Application orchestrator

Widget Development

Backoffice configuration

Actions & Editors

UI Configuration

UI configuration is responsible for the user interface configuration. Here you define the UI of each widget instance. There are three different versions of this configuration, consulted in the following order:

- cockpit-config.xml: Merged configuration of the whole `backoffice` application. You can modify it at runtime from the Application Orchestrator `-->` Show cockpit-config.xml. This file is stored as a hybris Media item.
- mybackoffice/backoffice/resources/mybackoffice-backoffice-config.xml: Configuration of components defined in the mybackoffice extension. It can also be loaded at runtime by doing a reset in the Application Orchestrator
- mybackoffice/backoffice/resources/widgets/mywidget/cockpit-config.xml (optional): Configuration of a specific widget. Think of it as the widget's default configuration.

UI Configuration: Example

Base Configuration for a type

```
<context type="Match" component="base">
    <y:base xmlns:y="http://www.hybris.com/cockpit/config/hybris">
        <y:labels>
            <y:label>homeTeam?.name + ' - ' + guestTeam?.name</y:label>
        </y:labels>
    </y:base>
</context>
```

List view configuration (Type specific)

```
<context type="Stadium" component="listview">
    <list:list-view xmlns:list="http://www.hybris.com/cockpitng/component/listView">
    <list:column spring-bean="previewListCellRenderer" width="26px"/>
        <list:column qualifier="code"/>
        <list:column qualifier="StadiumType"/>
        <list:column qualifier="capacity"/>
        <list:column qualifier="matches"/>
        <list:column qualifier="modifiedtime"/>
    </list:list-view>
</context>
```

Backoffice – Application configuration

Application configuration holds information about all widgets added to your application, keeps information things like viewing restrictions and socket connections. There are two different versions of this configuration, consulted in the following order:

- ${HYBRIS_DATA_DIR}/backoffice/config/backoffice-widgets.xml: Merged configuration of the whole backoffice application. Not to be modified directly, it updates automatically when you add/remove widgets and create widget connections using the Application Orchestrator.

- mybackoffice/backoffice/resources/mybackoffice-backoffice-widgets.xml: Default configuration of components defined in the mybackoffice extension.

Performing a reset of widgets.xml in the Application Orchestrator will result in a loss of the dynamic configuration and the default configuration from all backoffice module extensions will be loaded.

hybris software
An SAP Company

Overview

Application orchestrator

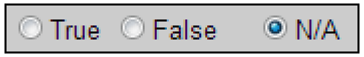
Widget Development

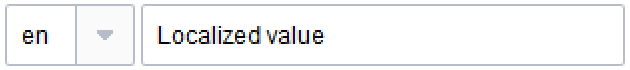
Backoffice configuration

Actions & Editors

Actions & Editors

- **Actions**
  - Actions are used to trigger an action that might perform a CRUD operation on objects or trigger other widgets.
  - Some default actions available in backoffice are:
    - Create/Delete action: Create or Delete an object of a Type
    - Principal permission action: Manage permissions for a *Principal*

- **Editors**
  - Editors are used to properly handle different types of data for editing. For example, special editors might be required for editing dates and ranges.
  - Some available editors:
    - Default Boolean editor: 

      True False [✓ N/A]
    - Localized simple editor: 

      en | Localized value

Creating custom actions/editors

1. **Create Action/Editor definition (definition.xml)**
   - For Actions it contains
     - The implementing class **`<actionClassName>`**
     - Data type of input and output
     - Icons to be displayed in the view for the action
   - For Editors it contains
     - The data type of the property being edited
     - The implementing class **`<editorClassName>`**
2. **Create implementing class**
   - For Actions, it should extend **`CockpitAction`**
   - For Editors, the class should extend **`CockpitEditorRenderer`**

Your custom action/editor can be used directly in a widget’s view (zul)