Axon.ivy 6.7

Axon.ivy Portal Documentation

Axon.ivy 6.7: Axon.ivy Portal Documentation

Chapter 1. Introduction

Main features

- Central access for end users - Only one portal for all applications

- Repository of reusable components

- Fast integration of process applications to the portal

- Provide customers and other system vendors the flexibility to build their own portals, but reuse portal components of Ivy

Reference architecture

The following image describes the reference architecture of the Portal Kit:

- Organize Ivy Landscape in more Ivy engine based on

  - Critical Process Apps

  - Support Applications

- Ivy Landscape shares a Company wide LDAP

- Ivy Landscape shares a Landscape wide DB for centralize settings

Multi-Server Reference Architecture

IIS for SSO

Axon.ivy Portal

Axon.ivy Engine 1

Application 1

Application 2

Application 3

Application 4

Application ...

Axon.ivy Engine 2

Application 1

Application ...

Axon.ivy Engine 3

Application 1

Application 2

Application 3

Application ...

Axon.ivy Engine ...

Application 1

Application 2

Application ...

Axon.ivy Landscape

LDAP

Landscape Database
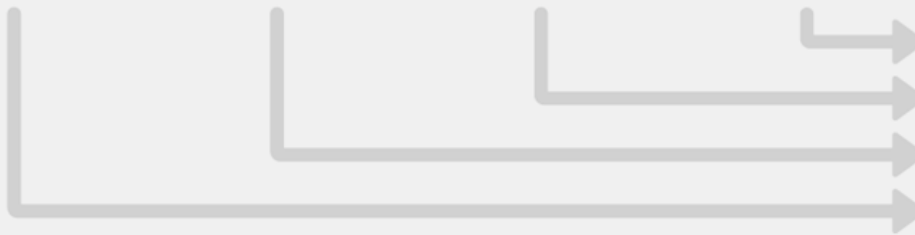
Chapter 2. Installation

Release installation

Installation

The installation section describes all the steps, that are necessary to install and setup the Process Application.

If you install the application for the first time than it's important to start with the Basic installation . It describes all the initial steps, that must be done for the first installation.

Release Installation

If the application is already installed and initial prepared, than refer to the Release Installation Steps, that are provided, here you will only find those steps, that are necessary to install this release.

Basic installation

Project modules

The application consists of 5 process modules. For detailed information of each module, refer to Architecture .

- PortalStyle

- PortalKit

- PortalTemplate

- SelfServiceBPM

- AxonIvyExpress

The project deployment of Ivy project are described in project deployment .

Server configuration

- The minimum required engine version is `6.0.0.49863`

Specify servers, applications used in Portal

General concept

Portal has 3 different configurations:

- **Single mode** : Only one Portal application on one engine. The Portal application must include portalKit, portalTemplate and portalStyle modules. By default, portalConnector is already deployed on the System application of Axon.ivy engine.

- **Multi-application mode** : Multiple Portal applications on one engine. Each Portal application must include portalKit, portalTemplate and portalStyle modules. Only one portalConnector of the System application is needed.

- **Multi-portal mode** : Single or multiple Portal applications on multiple engines. The engines will communicate with each other via the portalConnector of their System application (This mode is not supported yet).

Automatically detect servers, applications

By default, PortalKit will connect to PortalConnector using the address specified by the system property `WebServer.HTTP.Address` . If the property is not set, PortalKit will use the address localhost to connect to PortalConnector. All applications of the server which have the logged-in user are used in Portal.
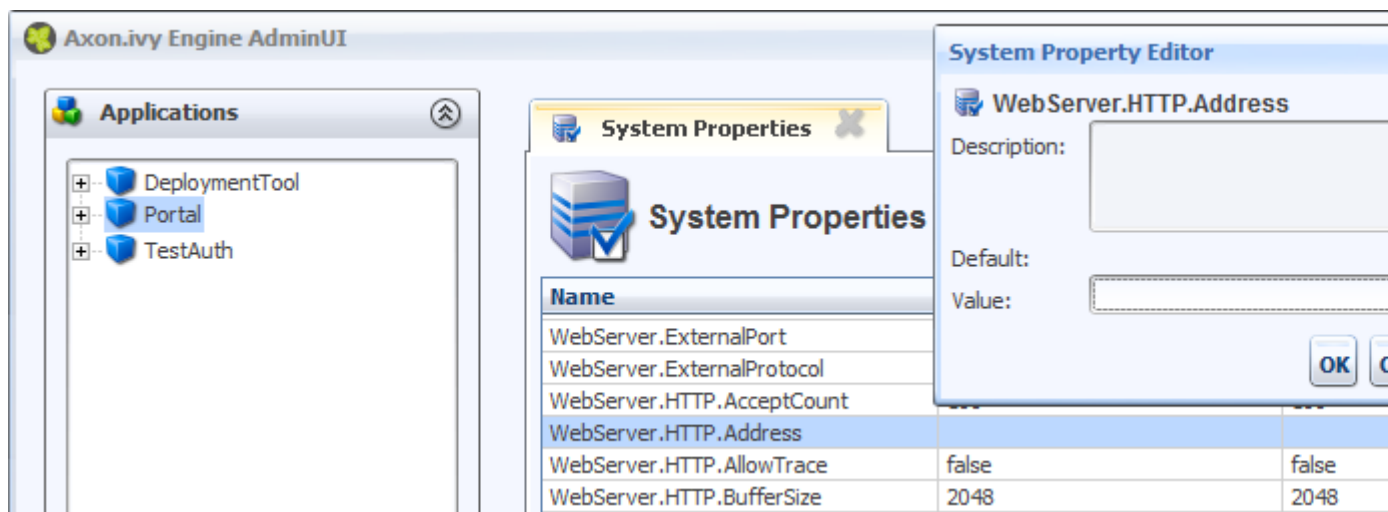
> **Note**
> System property `WebServer.HTTP.Address` specifies the address that will listen on the ivy engine port.

Steps to set the system property `WebServer.HTTP.Address`

1. Open AdminUI and go to System Properties.

2. Set the `WebServer.HTTP.Address` property to the value of the address that will be used to connect to PortalConnector.

3. Restart the ivy engine.

Manually configure servers, applications

Refer Setup multi portals .

Default user credentials

Portal provides 3 default users:

Username	Password	Description
admin	admin	This user has all Portal permissions, can access to Portal Admin Settings.
demo	demo	This user has permission to manage user absences.
guest	guest	Default normal user of portal.

Table 2.1. Default user credentials

> **Tip**
> You can change these accounts info in the AdminUI.

Global settings

Keyword	Value	Description
HOMEPAGE_URL	http://PORTAL_SERVER:PORTAL_PORT /pro/Portal/ portalTemplate /1549F58C18A6C562/ PortalStart.ivp	Portal administrator home page link. Note: Only valid if there are not any registered ivy applications.
HIDE_LOGOUT_BUTTON	TRUE/FALSE	If `true` (none case-sensitive), the logout button on top menu will be hidden, otherwise it will be shown. If this property is not set, the logout menu will be displayed by default.
SHOW_ENVIRONMENT_INFO	TRUE/FALSE	For switch on/off the display of environment information.
HIDE_TIME	TRUE/FALSE	Set to `TRUE` to hide hours and minutes next to date.
SHOW_TASK_BUTTON_LABELS	TRUE/FALSE	Set to `TRUE` to show button labels from task list.

Table 2.2. Global settings

Role configuration

PortalKit roles	Rights
AXONIVY_PORTAL_ADMIN	User belong to this role can handle AdminUI page, configure the internal role properties, users who own this role need some permissions. See Permission settings .

Table 2.3. Role configuration

Permission settings

Absences

- READ

  This function will be disabled if session user does not have `IPermission.USER_READ_OWN_ABSENCES` or `IPermission.USER_READ_ABSENCES` .

- CREATE/MODIFY

  This function will be disabled if session user does not have `IPermission.USER_CREATE_OWN_ABSENCES` or `IPermission.USER_CREATE_OWN_ABSENCES` .

- DELETE

  This function will be disabled if session user does not have `IPermission.USER_DELETE_OWN_ABSENCES` or `IPermission.USER_DELETE_ABSENCE` .

Personal task permission

- DELEGATE

  This function will be enabled if session user has permission `IPermission.TASK_WRITE_ACTIVATOR` .

> ⚠️ **Important**
> Task state cannot be one of the following: `DONE`, `DESTROYED`, `RESUMED`, `FAILED`.

- ADD NOTE

No permission requires.

> ⚠️ **Important**
> Task state cannot be one of the following: `DONE`, `DESTROYED`, `RESUMED`, `FAILED`.

- RESET

This function will be enabled if session user has permission `IPermission.TASK_RESET_OWN_WORKING_TASK`.

> ⚠️ **Important**
> Task state has to be one of following: `RESUMED`, `PARKED`.

- PARK

This function will be enabled if session user has permission `IPermission.TASK_PARK_OWN_WORKING_TASK`.

> ⚠️ **Important**
> Task state has to be `RESUMED`.

- CHANGE DEADLINE

This function will be enabled if session user has `IPermission.TASK_WRITE_EXPIRY_TIMESTAMP`.

> ⚠️ **Important**
> Task state cannot be one of following values: `DONE`, `DESTROYED`, `RESUMED`, `FAILED`.

- CHANGE PRIORITY

This function will be disabled if session user does not have `IPermission.TASK_WRITE_ORIGINAL_PRIORITY`.

> ⚠️ **Important**
> Task state has to be one of following: `DONE`, `DESTROYED`, `RESUMED`, `FAILED`.

Personal case permission

- ADD NOTE

Add note function will be enabled if case state is `RUNNING`.

- DELETE CASE

Delete case function will be enabled if session user has `IPermission.CASE_DESTROY`.

> ⚠️ **Important**
> Case state has to be `RUNNING`.

Administrator permission can see all tasks/cases in the application

Normal users can only see their tasks/cases they can work on.

Administrator can see all tasks/cases in the application.

Permissions needed: `IPermission.TASK_READ_ALL`, `IPermission.CASE_READ_ALL`.

Administrator permission can interact with all workflows in the application

Normal users can updates and deletes workflows which created by him and can interact with workflow's task which assigned to him.

Administrator can creates, updates and deletes all workflows in the application.

Global variables

Variable	Default value	Description
PortalStartTimeSynchUserExpression	0 0 5 * * ?	Cron expression define the time to reload application user datas. E.g.: expression for at 5AM every day is `0 0 5 * * ?`. Refer to crontrigger . Please restart Ivy engine after changing this variable.
PortalCallWebserviceMaxRetry	10	Maximum time that PortalConnector will retry to synchronize data to Portal on the system. If beyond this limit, system will create a Task for `AXONIVY_PORTAL_ADMIN` role to handle this error.
PortalSearchDelayInMilliseconds	500	The delay time of search function at top menu.

Table 2.4. Global variables

Email settings

Task notification mail

Users can receive standard Ivy new task email notification. More information about the email notification can be found here .

Individual Mails

- List of individual mails, that are send in the process.

- Application specific mail is stored in the user profile within the attribute `useCustomMails` .The value of this attribute is `True/False` .

  If `True` then this user will receive notification email send out from the process, otherwise they will not receive email.

Look and feel

Portal doesn't use Modena theme from version 6.3.

Migration notes

This document informs you in detail about incompatibilities that were introduced between Portal versions and tells you what needs to be done to made your existing Portal working with current Axon.ivy engine.

Migrate 6.x to 7.0

If you copy the `PortalStart` process or the `PortalHome` HTMLDialog for customizations, please adapt the changes:

- The whole process is refactored to be clearer. So it is recommended that you copy it again.

- New process is introduced: restorePortalTaskList.ivp

- PortalStart: some new ivy scripts are added to handle the navigation back to the same page before starting a task.

- PortalHome: The `taskView` parameter is added to the start method.

SQL conversion

From Portal `7.0`, we use standard axon.ivy Task Category field to store task category.

To migrate task categories, please deploy MigrateTaskCategorySample.iar to your application and run `Migrate Task Category` process to:

1. Migrate data from column `customVarCharField5` to `category` for all tasks in the application.

2. Delete leftover data in `customVarCharField5` of all tasks in the application.

3. Create CMS entries for task categories in the application.

If you have queries which referring to task category, plese replace `customVarCharField5()` part with `category()` part.

Migrate 6.4 or 6.5 to 6.6

Task header is supported to be customized. The `useOverride` param, which is used to override the task item's body, is changed to `useOverrideBody`

Migrate 6.4 to 6.5

The relative link in default user processes starts with ivy context path instead of "pro"

Migrate 6.x (x < 4) ... 6.4 (Jakobshorn)

Portal appearance

Portal `6.4` are redesigned. Therefore many components look different from the previous version like menu, task list, case list ... . Portal `BasicTemplate` does not use `p:layout` and `p:layoutUnit` anymore. You may need to adapt your pages to this change.

For now the menu customization is not supported.

From `6.4`, Portal applies LESS to support customizing Portal styles. You can customize colors, fonts and Portal's component styles. For more information about customizing Portal's style with LESS, please refer to Portal logos and colors .

Steps to migrate

1. Copy PortalStyle/webContent/resources of Portal `6.4` to PortalStyle/webContent/resources of the current Portal.

2. Modify PortalStyle/webContent/resources/less/theme.less, update value of @body-background-color for the background color and @menu-color for the menu, button color.

3. Put custom styles to PortalStyle/webContent/resources/less/customization.less.

4. Add properties and plugins which are defined in PortalStyle/pom.xml of Portal 6.4 to PortalStyle/pom.xml of the current Portal.

5. Run the maven command `mvn lesscss:compile` in PortalStyle to build CSS file.

6. PortalStyle/webContent/resources/css/theme.css is obsolete, please remove it.

Migrate 5.0 (Rothorn) ... 6.0 (Säntis)

Database conversion

If you are using Portal 5.0 , you have to manual configure all settings (create servers, applications, variables) again since Portal now doesn't use external database. All settings on from Portal 6.0 are stored in Ivy system database. If you are using Portal 6.0 , you don't need to convert database.

Portal appearance

Portal now doesn't use Modena theme, it's a big difference to previous 6.0 . Therefore many things in Portal 5.0 and 6.0 will not look the same in new Portal. Many things have been redesigned like menu, task list, case list ...

Release notes

This part lists all relevant changes since the last official product releases of Axon.ivy.

Changes in 7.0 (Jakobshorn)

- More search criteria for user in Task list are added and allowed to customize.

- Task delegate customization is supported

- The same task list is displayed before and after a task. Set default end page to another project to remove this feature.

- Task category of Portal is now stored in new Task category field of ivy.

  Please refer to Migration notes to learn how to migrate data from `customVarCharField5` to new `category` field.

- Hide technical tasks (the HIDE additional property is set), so that they are not displayed in any Portal task lists.

Changes in 6.6 (Jakobshorn)

- Task widget's customization is extended with task header and task data query.

- Hide technical roles (the HIDE property is set), so that they are not displayed anywhere (e.g. delegate, absence mgmt). The default hidden role is AXONIVY_PORTAL_ADMIN

Changes in 6.0 (Säntis)

- Portal has 2 level menu with animation.

- All components such as button, text field ...have been re-styled, not applied Modena's styles.

- Portal supports responsiveness with 3 screen widths: 1920, 1366 and 1024. Please refer to Responsiveness for more details.

- Some customizations are not supported in this release: main menu, case header.

Chapter 3. Architecture

Currently Portal system contains 6 modules
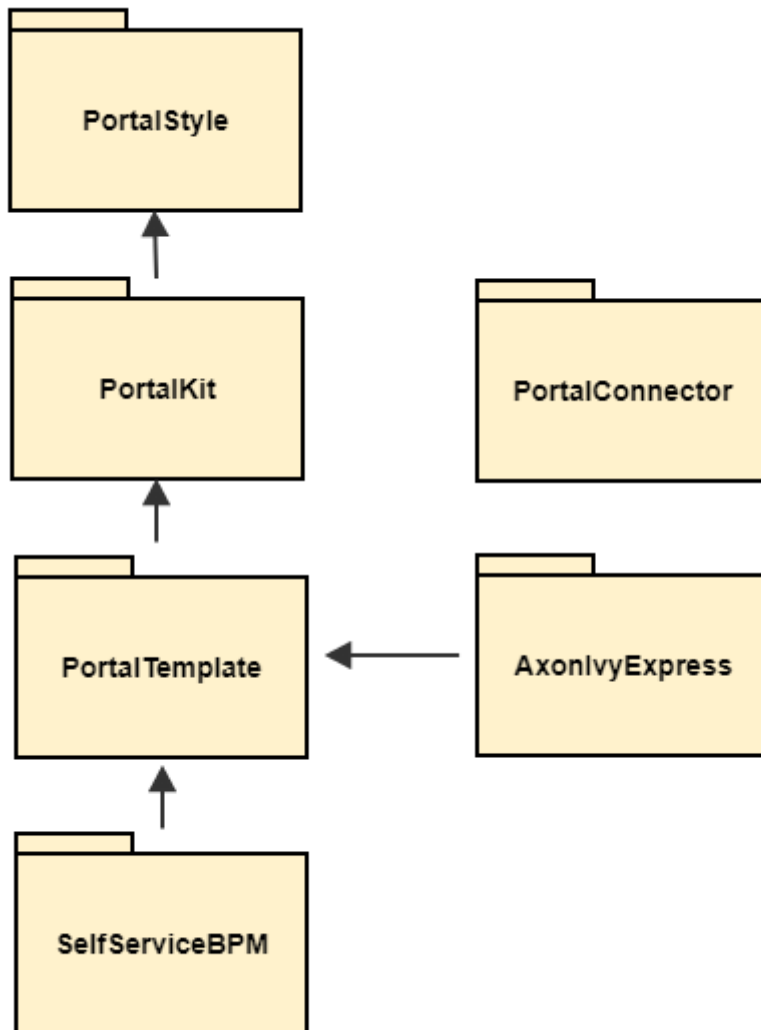
Portal kit

Contains set of UI components. This module contains set of JSF Ivy Component to provide user the usages to work with Ivy Process Data such as: task, case, absence... , styles CSS and JavaScript file for component and Modena theme library. This is the most important module that user needs to use Portal. This module also contains AdminSettings component that is used to configure Portal.

Portal style

Contains definition of styles that can be overridden/customized later. As now Portal supports user to customize various colors of the layout such as: background color, text color, border color, button color, focus/hover color. In the current version you have to change each color one by one. There are no common color definitions.

> **Note**
> This module is prepared for process developers to override and keep customer styles by editing CSS file, CMS's style.

Portal template

Provides default portal's templates and pages. This module contains templates page for Portal's user to use as composition, then they will have supporting features such as : top menu, application menu, user menu. It also contains some start process links to default page such as : Portal home, Portal task list, Portal case list... . Portal's user is advised to depend on this module to use Portal easily.

Self service BPM

This project extends PortalTemplate and is a utility which implements concept of Adaptive Case Management. The idea is that user can start any cases and add simultaneously new roles and tasks while doing his process work.

Axon.ivy Express

The idea is that user can create his own process and can manage it easily, it gives user more flexibility when working with Portal.

This project is an extended project from PortalTemplate. It provides:

- Ability to create his/her own workflow

- Tools to create and modify the web form for his workflow

Portal connector

Communication channel to communicate between portal axon.ivy engine (e.g.: for synchronizing data, read ivy process data like task, case, absence, substitute,...). This module acts like service layer that contains web service's implementation for PortalKit to call and get data, only PortalKit knows about these web services.

Note

By default, this module is integrated to System application of Axon.Ivy engine. To be able to run Portal in designer, you need to import this module.

Chapter 4. Components

Widget concept

Before beginning

This guide assumes that you are already familiar with concepts inherent in JSF programming and in Ivy development.

Introduction

This document provides a high-level explanation of how to develop a Portal widget. The ability to use Portal services and styles can be particularly useful to developers who wish to do one or more of the following:

- Create their own widgets for Portal which have a consistent look and feel with the existing widgets.

- Reuse existing portal services to create their own widgets which can manipulate Portal data, such as: cases, tasks, process starts, users,...

How it is

This section introduces the Html Dialog Component and Portal services, predefined styles used in building a widget, and goes on to describe the process of designing and implementing.

Portal widgets should be implemented using the **Html Dialog Component** technology from Axon.ivy and follow the famous model-view-controller pattern.

Furthermore, to have a clean architecture and avoid a lot of headaches going forward, we suggest that you should separate your widget into layers like below:

- Entities

  Entities are the business objects of the widget. They encapsulate the most general and high-level rules. They are the least likely to change when something external changes. (e.g.: by a change to page navigation, or security).

- Use Cases

  Use case are widget specific business rules. This layer encapsulates and implements all of the use case of the widget. Changes in this layer should not affect the entities. This layer should not be affected by changes to externalities such as the Portal services, the UI, or any of the common frameworks.

- Interfaces Adapters

  Interfaces Adapters are set of adapters that convert data from the format most convenient for the use cases and entities, to the format most convenient for some external agency such as the database or the web. Similarly, data is converted, in this layer, from the form most convenient for entities and use case, into the form most convenient for whatever persistence framework is being used. (The presenters, views, and controllers all belong in here. The models are likely just data structures that are passed from the controllers to the use case, and then back from the use cases to the presenters and views.)

- Frameworks and Drivers

  This layer is generally composed of frameworks and tools such as the database, the web framework, Portal services, etc. This layer is where all the details go. The Web is a detail. The Portal services are detail. We keep these thing on the outside where they can do little harm.

> **Tip**
>
> There's no rule that says you must always have just the four layers above. However, you should always apply that the source code dependencies point from mechanisms to policies:

Frameworks and Drivers > Interfaces Adapters > User Cases > Entities

By doing so, you will create a widget that is intrinsically testable, independent of frameworks, independent of UI, independent of database, and independent of any external agency. When any of the external parts of the system become obsolete, like the database, or the web framework, you can replace those obsolete elements with a minimum of fuss.

Main technology and concept

You should have an understanding of the following technology and concept as you build your widget:

- Managed Beans: in Html Dialog Component it is possible to communicate with normal Java Objects by using Managed Beans.

- User Dialog Concept: an Html Dialog Component follows the model-view-controller pattern of the User Dialog Concept.

  - Model is a data class whose data fields can be bound to widget properties of the view via the special object data.

  - Controller is implemented by a series of UI processes that can be mapped to events on the view such as mouse clicks. Axon.ivy provides the keyword logic to call an event process or a method process in the logic.

  - View of an Html Dialog is defined with the means of an XHTML document.

Services

There are separate services for working with each type of data:

- Application Services: set of services for getting information about applications.

- Absence Services: set of services for manipulating the user's absence.

- Case Services: set of services for working with cases and related data, such as: additional properties, notes,...

- Task Services: set of services for working with tasks.

- Process Start Services: set of services for querying process starts from the Portal system.

- Security Services: set of services for querying users and roles.

- User Setting Services: set of services for manipulating the user settings and related data, such as: email settings, language settings.

- Portal Configuration Services: set of services for controlling the Portal configuration.

Built-in widgets

Portal comes with some useful widgets:

1. Task widget

   Below is the sample how the task widget being use in the default template:

   ```
   <ui:define name="taskWidget">
   ```

   ```
   <ic:ch.ivy.addon.portalkit.component.TaskWidget                id="task-widget"
   tasks="#{logic.getTasksOfSessionUser()}" ... />
   ```

   ```
   </ui:define>
   ```

2. Process widget

   Below is the sample how the process widget being use in the default template:

```
<ui:define name="processWidget">

<ic:ch.ivy.addon.portalkit.component.ProcessWidget          id="process-widget"
compactMode="true" ... .>

</ui:define>
```

3. Statistic widget

Below is the sample how the statistic widget being use in the default template:

```
<ui:define name="statisticWidget">

<ic:ch.ivy.addon.portalkit.component.StatisticWidget       id="statistics-widget"
compactMode="true" ... >

...

</ic:ch.ivy.addon.portalkit.component.StatisticWidget>

</ui:define>
```

Portal setup these widget with the default settings for you, but you can always re-define them in order to match with your needs. Moreover, if you want to turn off a built-in widget, you can simply leave its ui:define container empty like this:

```
<ui:define name="taskWidget">

<!-- leave it empty -->

</ui:define>
```

Predefined styles

There are separate common styles are predefined to ensure every Portal widget has a consistent structure and appearance:

```
<div class="widget">

<div class="widget-header">

<ul class="widget-header-menu">

<li class="widget-header-menu-item">...</li>

<li class="widget-header-menu-item">...</li>

<li class="widget-header-menu-item">...</li>

...

</ul>

...

</div>

<div class="widget-content">

<div class="widget-content-list">

<div class="widget-content-list-item">...</div>

<div class="widget-content-list-item">...</div>

<div class="widget-content-list-item">...</div>
```

```
...

</div>

</div>

<div class="widdget-footer">

...

</div>

</div>
```

Flow

The general flow for developing a widget for portal is as follows:

1. Design your widget, deciding which parts to implement in Ivy component, and which parts to implement as pure JSF.

2. Create an Html Dialog Component.

   The following code fragment defines an example Html Dialog component:

   ```
   <cc:interface componentType="IvyComponent">

   <cc:attribute name="caption" />

   </cc:interface>

   <cc:implementation>

   ...

   </cc:implementation>
   ```

   A component could be inserted with the ic tag.

   ```
   <ic:my.namespace.ComponentName ... />
   ```

   For more information, see the Html Dialog Component section in Axon.ivy Designer - Help: **Designer Guide > User Interface > User Dialogs > Html Dialogs**

3. If you are writing a widget, which manipulates task, case,... consider using Portal built-in services.

4. Optionally, your widgets can have their own configuration. There are separate methods for manipulating widget configuration:

   • You can initiate or update your widget configuration by passing an JSON object to `saveSettings()`.

   • You can load your widget configuration by calling `loadSettings()`.

Integration

The general flow for integrating a widget into Portal homepage is as follows:

1. Create a new home page which uses the `DefaultHomePageTemplate.xhtml` template. By doing this, your new home page will inherit the widget from the previous home page and has a place holder for your own widgets. Your custom home page should look like below:

   ```
   <ui:composition template="/layouts/DefaultHomePageTemplate.xhtml" xmlns="http://
   www.w3.org/1999/xhtml"
   ```

```
xmlns:f="http://xmlns.jcp.org/jsf/core" xmlns:h="http://xmlns.jcp.org/jsf/html"

xmlns:ui="http://xmlns.jcp.org/jsf/facelets"    xmlns:ic="http://ivyteam.ch/jsf/
component">

<ui:define name="customWidget">

...

</ui:define>

</ui:composition>
```

2. Create a new process start for the new home page. Now you will use this process start as the entry point of your portal instead of the default one. To let portal know about your new portal home, you have to go to the portal settings and set the portal home url to the new one.

3. In your new home page, place your widget inside the customWidget section.

```
<ui:define name="customWidget">

<ic:my.namespace.ComponentName ... />

...

</ui:define>
```

For more details, visit Portal home.

Exception handling

Portal separates exception into 2 types: ajax and non-ajax exception.

Portal handle non-ajax exception for you. You do not need to do anything for this type of exception.

Portal also handle ajax exception for you as default, but you can implement your own exception handler by using the Primefaces built-in exception handler: `p:ajaxExceptionHandler`.

Layout templates

Templates for development

Your Portal Project is dependent on PortalTemplate project, in which there are 7 templates that can be used directly.

1. Basic template

2. Two column template

3. Task template

4. Case template

5. Task list template

6. Case list template

7. Default homepage template

These templates have the same header, which is a menu of applications that you configure in Administration page. Since version 6.4, Portal officially supports responsiveness for 3 resolutions: iMac (1920*1050), iPad (1366*1024) iPad Portrait: (1024*1366), every templates has its default responsiveness, you can refer to Responsiveness to override it. Besides, there

are user settings like: Absences, Email, Language Settings and Administration (for admin only). Details about user settings can be found in Settings.
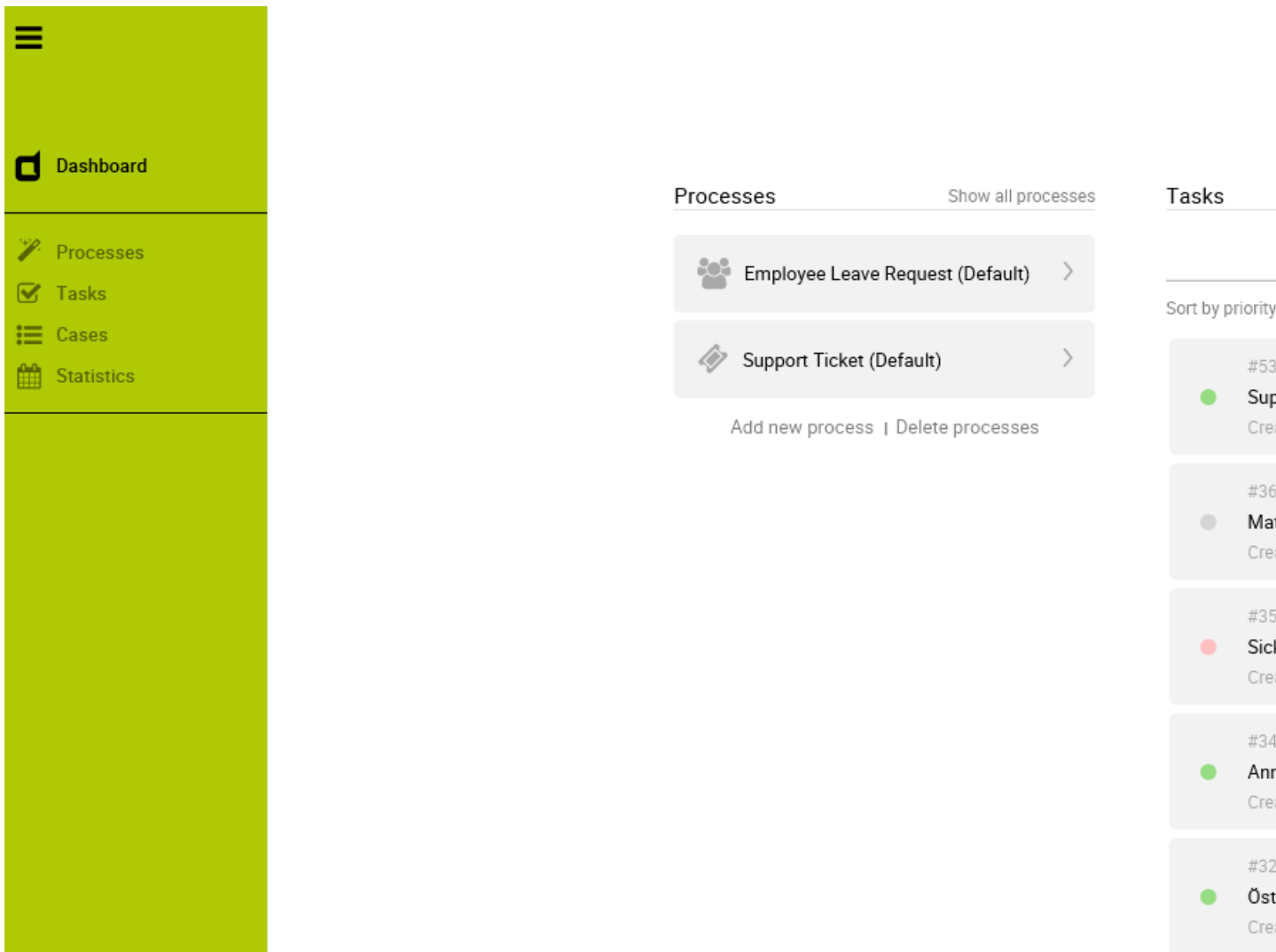
Basic template

Basic template provides basic layout where user can put their custom content. It lacks Portal menu and Case details. We recommend to use task template for your process.

How to use Basic template

1. Create a new HTML User Dialog and then use `ui:composition` to define the template inside and reuse the default responsiveness behavior. To override it, please use `pageContent` instead of `simplePageContent` and Responsiveness.

    ```
    <ui:composition template="/layouts/BasicTemplate.xhtml">

    <ui:define name="pageTitle">Sample Page</ui:define>

    <ui:define name="simplePageContent">

    This is sample content.

    </ui:define>

    </ui:composition>
    ```

2. See the result after using Basic template for example:

This is sample content.

Two column template

Two column template inherits Basic Template. It has 2 columns which user can customize their contents. Normally, the first column is for navigation, the second for displaying corresponding content.

How to use Two column template

1. Create a HTML User Dialog, define template in `ui:composition` and insert content of second column and third column using `ui:define`.

```
<ui:composition template="/layouts/TwoColumnTemplate.xhtml">

<ui:define name="pageTitle">Sample Page</ui:define>

<ui:define name="navigationRegion">

Navigation Region

</ui:define>

<ui:define name="contentRegion">

Content Region

</ui:define>

</ui:composition>
```

2. See the result after using Two column template for example:

Navigation Region   Content Region

Task template

Task template is used for displaying task functionality and related information to support completing the task. There are a lot of regions to be filled with your custom content:

- Request name

- Process chain

- Errors

- Information

- Dynamic tabs

- Request form

- Case information tab

- Buttons at footer

How to use task template

1. Create a new HTML User Dialog and then use `ui:composition` to define template which you use inside.

   ```
   <ui:composition template="/layouts/TaskTemplate.xhtml">
   ```

2. Set `caseId` value so that the `Case information tab` is available to users where they can see info of case, documents, related tasks and history. It is mandatory.

   ```
   <ui:param name="caseId" value="#{ivy.case.id}" />
   ```

3. Set data to `actualStepIndex` and `steps` variables which are used for ProcessChain component in template. It is mandatory.

```
<ui:param name="actualStepIndex" value="#{data.actualStepIndex}" />

<ui:param name="steps" value="#{data.steps}" />
```


4. Inserts contents for `taskName`, `errorsZone`, `infoZone`. It is optional.

```
<ui:define name="taskName">...</ui:define>

<ui:define name="errorsZone">...</ui:define>

<ui:define name="infoZone">...</ui:define>
```

5. Inserts some new tabs, refers some segment of code as below. It is optional.

```
<ui:define name="dynamicTabs">
```

```
<p:tab title="DynamicTab1">Tab1</p:tab>

<p:tab title="DynamicTab2">Tab2</p:tab>

</ui:define>
```

6. Inserts left buttons and right buttons which stay at the bottom of the page. It is optional.

```
<ui:define name="leftButtons">

<p:commandButton value="Left Button" />

</ui:define>


<ui:define name="rightButtons">

<p:commandButton value="Right Button" />

</ui:define>
```

7. Overwrite contents of default tabs. It is optional.
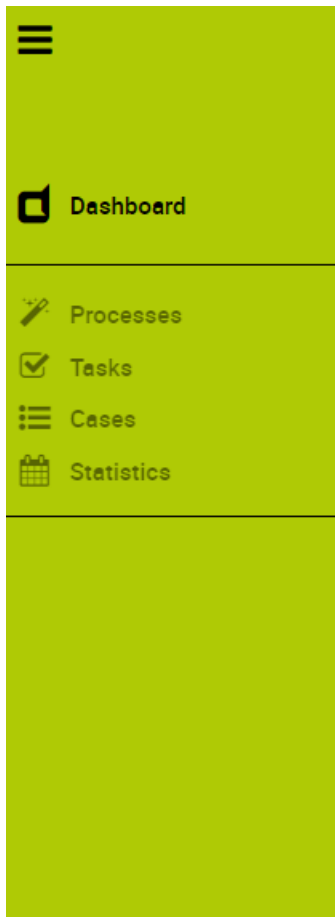
```
<ui:define name="taskForm">

<h:form>

<p:outputLabel name="myCustomLabel" />

...

</h:form>

</ui:define>
```

8. Set visible/invisible for default tabs. Set following variables as `true` if you want to visible and vice versa.

```
<ui:param name="showCaseStatusInfoTab" value="true" />
```

Case template

Case template is similar to Task Template in both UI and usage. The difference is it is used for displaying case details functionality.

How to use case template

Create a new HTML User Dialog and then use `ui:composition` to define template which you use inside.

```
<ui:composition template="/layouts/CaseTemplate.xhtml">
```

Default homepage template

Default homepage template is used to create pages that have the look as default homepage of Portal. Besides, users can customize it by disabling default widgets, add new widgets, change position of widgets. For more details including basic and advanced customization, refer to Portal home

How to use default homepage template

Create a new HTML User Dialog and then use `ui:composition` to define template.

```
<ui:composition template="/layouts/DefaultHomePageTemplate.xhtml">
```

Task list template

Task list template is used to display task list where user can see tasks and their details.

How to use task list template

1. Create a new HTML User Dialog and then use `ui:composition` to define template.

```
<ui:composition template="/layouts/PortalTasksTemplate.xhtml">

</ui:composition>
```

2. Data class of this dialog should have an attribute named `taskView` with type `ch.ivy.addon.portal.generic.view.TaskView`. By changing this attribute, user can modify title of the task list widget, collected tasks (through `dataModel`) and more. The following is a sample to build a taskView.

```
import ch.ivy.addon.portalkit.datamodel.TaskLazyDataModel;

import ch.ivy.addon.portalkit.bo.MainMenuNode;

import ch.ivy.addon.portal.generic.view.TaskView;


TaskLazyDataModel dataModel = new TaskLazyDataModel();

dataModel.setIgnoreInvolvedUser(true);

dataModel.setSortType(ch.ivy.addon.portalkit.enums.SortType.BY_PRIORITY);


MainMenuNode category = new MainMenuNode();

category.value = "My Task List";


out.taskView  =   TaskView.create().dataModel(dataModel).pageTitle("My   Task
List").hideTaskFilter(true).category(category).showHeaderToolbar(false).createNewTaskV
```

Case list template

Case list template is used to display case list where user can see cases and their details.

How to use case list template

1. Create a new HTML User Dialog and then use `ui:composition` to define template.

    ```
    <ui:composition template="/layouts/PortalCasesTemplate.xhtml">

    </ui:composition>
    ```

2. Data class of this dialog should have an attribute named `caseView` with type `ch.ivy.addon.portal.generic.view.CaseView`. By changing this attribute, user can modify title of the case list widget, collected cases (through `dataModel`) and more. The following is an example to build a caseView.

    ```
    import ch.ivy.addon.portalkit.datamodel.CaseLazyDataModel;

    import ch.ivy.addon.portal.generic.view.CaseView;
    ```

```
CaseLazyDataModel dataModel = new CaseLazyDataModel();

out.caseView        =        CaseView.create().dataModel(dataModel).withTitle("My
Cases").buildNewView();
```

Handle required Login in templates

All templates require login to access by default. But templates also provide functionality to access page without login by adding the isNotRequiredLogin parameter.

How to handle required login in template

1. Create a new **HTML User Dialog** and then use ui:param to define the template inside

    ```
    <ui:composition template="/layouts/BasicTemplate.xhtml">

    <ui:param name="isNotRequiredLogin" value="#{data.isNotRequiredLogin}" />

    <ui:define name="pageContent">

    This is sample content.

    </ui:define>

    </ui:composition>
    ```

2. Result after using template for example (All user settings and application menus will not visible).

Responsiveness

Since version 6.4, Portal officially supports responsiveness for 3 screen widths: iMac(width 1920), iPad landscape(width 1366) and iPad portrait(width 1024).

To apply your styles for the above resolutions, you can add your own media query css:

```
/* Small screen */ @media screen and (max-width: 1365px) {/*.....*/}

/* Medium screen */ @media screen and (min-width: 1366px) and (max-width: 1919px)
{/*.....*/},

/* Large screen */ @media screen and (min-width: 1920px) {/*......*/}
```

In Portal's new design, the main container's width should be changed according to menu state (expand/colapse).

To adapt the change, you need to initialize the ResponsiveToolkit Javascript object and introduce 3 objects to handle 3 screen resolutions and each object has to implement the updateMainContainer method.
Portal templates define their own responsiveness, you can redefine the footer section to override:

E.g. Initialize ResponsiveToolkit for TaskList page.

```
<ui:define name="footer">

<script type="text/javascript">

$(function(){

var taskListLargeScreen = new TaskListLargeScreenHandler();
```

```
var taskListMediumScreen = new TaskListMediumScreenHandler();

var taskListSmallScreen = new TaskListSmallScreenHandler();

var responsiveToolkit = ResponsiveToolkit(taskListLargeScreen, taskListMediumScreen,
taskListSmallScreen);

Portal.init(responsiveToolkit);

});

</script>

</ui:define>
```

Enable Portal chat

Chat feature

Information

Chat feature was implemented on Portal.

There are couple of reason, it's intentionally disabled by default:

- Customers have their own Chat, they do not need another.

- For using Chat, system administrator has to configure it on their server (needs additional Libraries).

- With added libraries for using Chat, it slows AxonIvyEngine down.

- It has log-in issue as if AxonIvyEngine is configured with IIS (version `7.5` or older) to manage SSO.

- Not fully support `UNICODE` characters on Ivy `6.3` and upwards.

For users who want to use chat feature

Follow the steps below:

1. First, IvyEngine/Designer needs some configuration to use Chat. Copy two libraries and then paste into `webapps/ivy/WEB-INF/lib` folder located on `AxonIvyEngine, AxonIvyDesigner.`

   > **Note**
   > You can find these libraries inside PortalKit module, located on `PortalKit/lib_chat` or download from this site.

   - portal-chat-endpoints-s125.jar

   - atmosphere-runtime-2.4.2.jar

2. Second, Enables Chat on UI by setting `enablesChat` param

   ```
   <ui:param name="enablesChat" value="true" />
   ```

   Added this line to your pages that are using BasicTemplates of Portal (adds to `DefaultPortalHomeTemplate.xhtml, TaskTempate.xhtml`, etc.)

   E.g.:

   ```
   <ui:composition template="/layouts/BasicTemplate.xhtml">

   <ui:param name="enablesChat" value="true" />

   <...>

   </ui:composition>
   ```

   Or simply change param's value on `BasicTemplate.xhtml`

   <ui:param name="enablesChat" value="#{enablesChat}" /> to <ui:param name="enablesChat" value="true" />

3. Restarts AxonIvy Desinger/Engine. Then enjoy Chat.

Error handling

In this section, we introduce 2 kinds of errors, when and how to handle them in Portal.

- Ajax error : this kind of errors occur during a JSF ajax requests, for example when the user clicks on the show full mode button to tell the task widget switches to full mode, without handling the end user would not get any form of feedback if the action was successfully performed or not.

- Non-ajax error : this kind of errors occur when user access to Portal from a url which could not be handled successfully by server side, or being navigated by a corrupted url. For example, when the user clicks on a link to start a task which does not exist.

Ajax error handling

Introduction

By default, when an exception occurs during a JSF ajax request, Portal would not get any form of feedback if the action was successfully performed or not. It would make sense if exceptions during ajax requests are handled as the way that the end user would see an alert with the exception type and message (exception stack trace might be included for advanced user).

Portal uses the `ajaxExceptionHandler` component provided by PrimeFaces to handle exceptions occur during JSF ajax requests. This document describes when and how to enable this feature.

> **Note**
> This feature is only available if using the portal default template: `BasicTemplate` or its extension.

Configuration

This exception handler must be registered by a factory as follows in `faces-config.xml` in order to get it to run:

Config Ajax Exception Handler of Primefaces

```
<application>

<el-resolver>

org.primefaces.application.exceptionhandler.PrimeExceptionHandlerELResolver

</el-resolver>

</application>

<factory>

<exception-handler-factory>

org.primefaces.application.exceptionhandler.PrimeExceptionHandlerFactory

</exception-handler-factory>

</factory>
```

> **Note**
> The `faces-config.xml` file is located at `../webapps/ivy/WEB-INF` inside your Ivy Designer/Engine. The designer/engine need to be restarted in order to make the changes affect.

Default exception handler by Portal

The default exception handler of Portal after being configured will transparently handle all exceptions during ajax requests. It is configured to handle the exception type `java.lang.Throwable`, therefore any specific exceptions would match.

When an exception occurs, Portal will show an error notification with the exception type and message to end user. The exception details is available when user click on the show details button.

The default exception handler defined in Portal:

```
<p:ajaxExceptionHandler      type="java.lang.Throwable"      update="exception-dialog"
onexception="portal.notifyError('Action    completed    with    an    error!   Error   message:
#{pfExceptionHandler.message}')" />
```

Customizing exception handler

If more fine grained control is desired for determining the root cause of the caught exception, or whether it should be handled, or determining the error notification, or logging the exception, then the developer can opt to define their own exception handler and override the default exception handler of Portal:

```
<p:ajaxExceptionHandler               type="your.business.domain.MyCustomWidgetException"
update="exception-dialog" onexception="portal.notifyError('MyCustomWidget could not
perform the action completely! Error message: #{pfExceptionHandler.message}')" />
```

> **Note**
>
> Portal provides following Javascript methods for notification:
>
> - `portal.notifyError(String message)`
>
> - `portal.notifyWarning(String message)`
>
> - `portal.notifyInfo(String message)`

Non-ajax error handling

Introduction

By default, when the server has any error such as : `HTTP 404`, `HTTP 500`, or exception while page's loading, AxonIvyEngine will show an default error page. E.g.:

You can find content of this page is the file located on `${AxonIvyEngineFolder}/webapp/ivy/ivy-error-page.xhtml`, but the error page is not user friendly, too much technical information that normal user may not understand. Thus, Axon Ivy Portal provides an alternative solution to make this page nicer.

How to configure

Download the zip file below to configure on your own engine (or designer).

⚠ Important
Read README.txt

PortalErrorPageConfiguration.zip

Result

HTTP 404 Page Not Found

Example testing URL: 404

HTTP 500 Error

Example testing URL: 500

Chapter 5. Customization

Build your own portal

Build your own portal using Portal kit

1. Create new project

   Create a new project that depends on `PortalTemplate`.
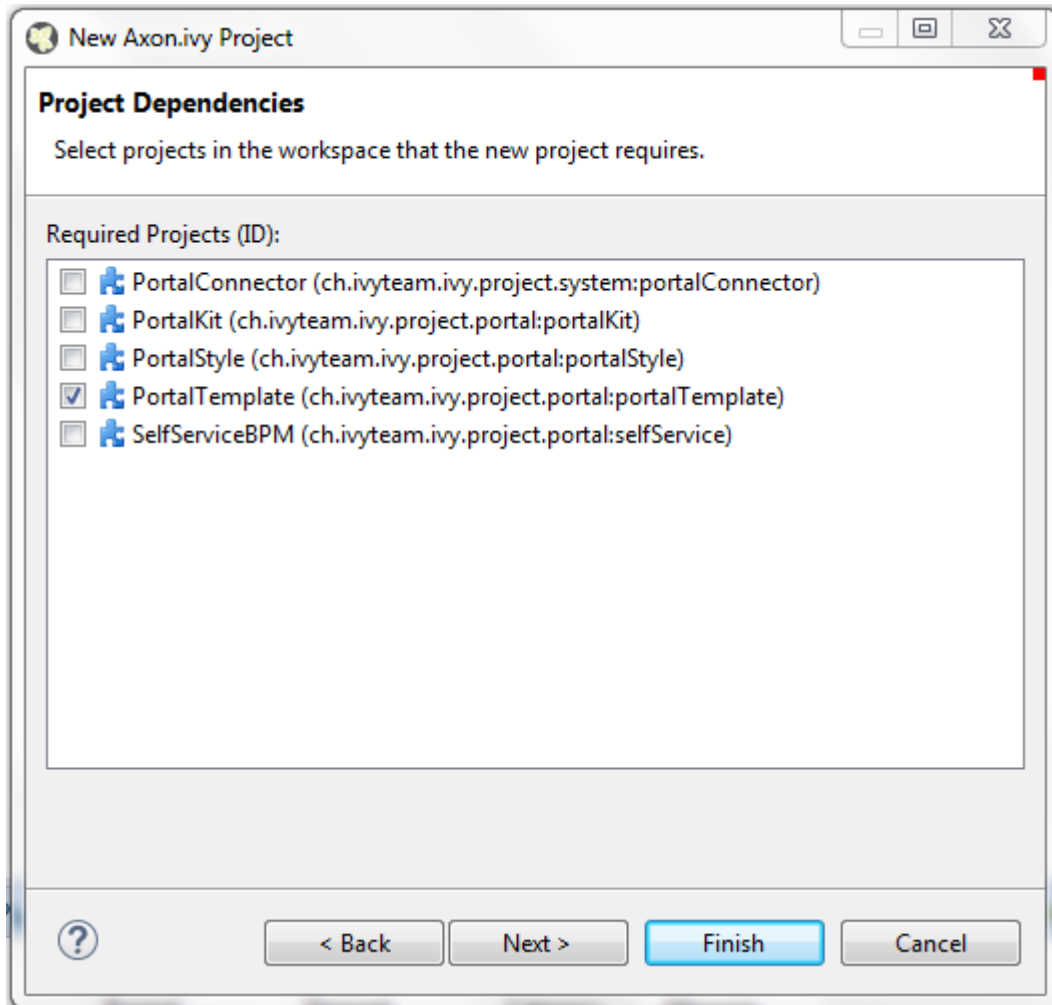
   Project structure
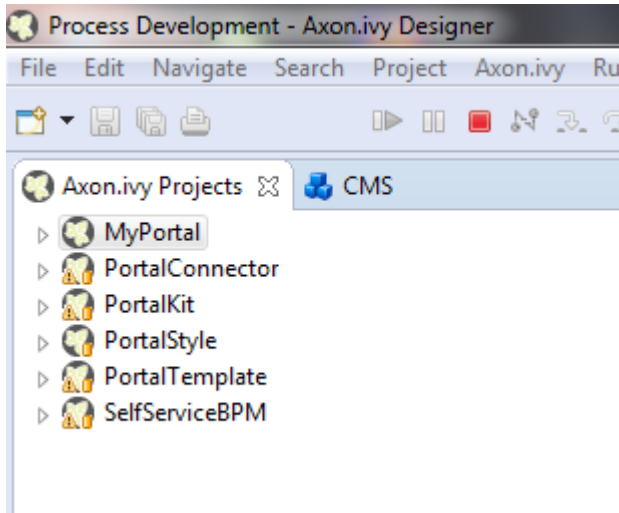
2. Create new home page

    a. Create a new `HTML Dialog` for your home page and then use `ui:composition` to define template which you use inside. If you want to keep the look of default homepage, then choose `DefaultHomePageTemplate`.

    > **Tip**
    > Find more information about templates at Layout templates.

    b. Create a new `Start process` and connect to `User Dialog` for your home page.
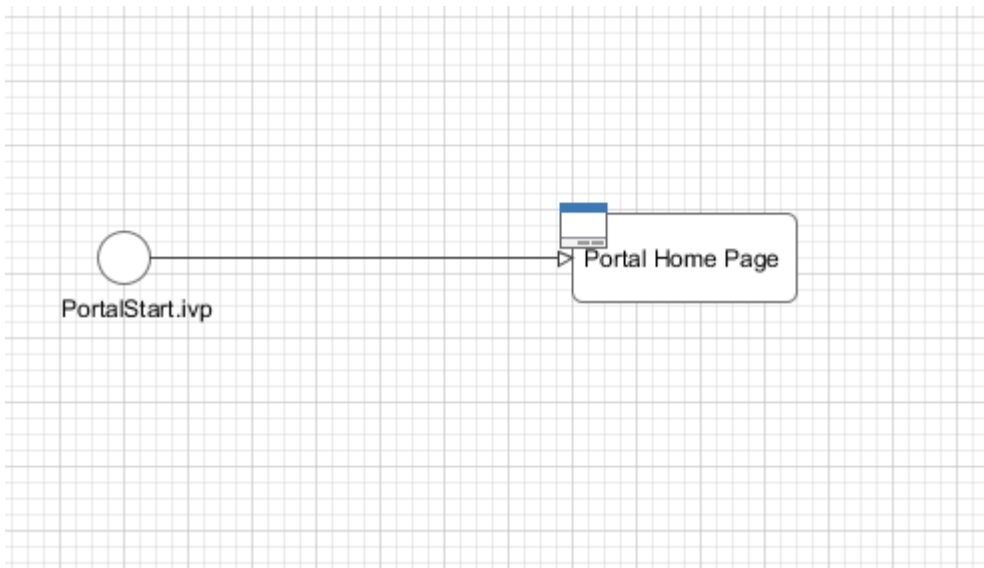
    c. Run your application, start your newly created process and see result.
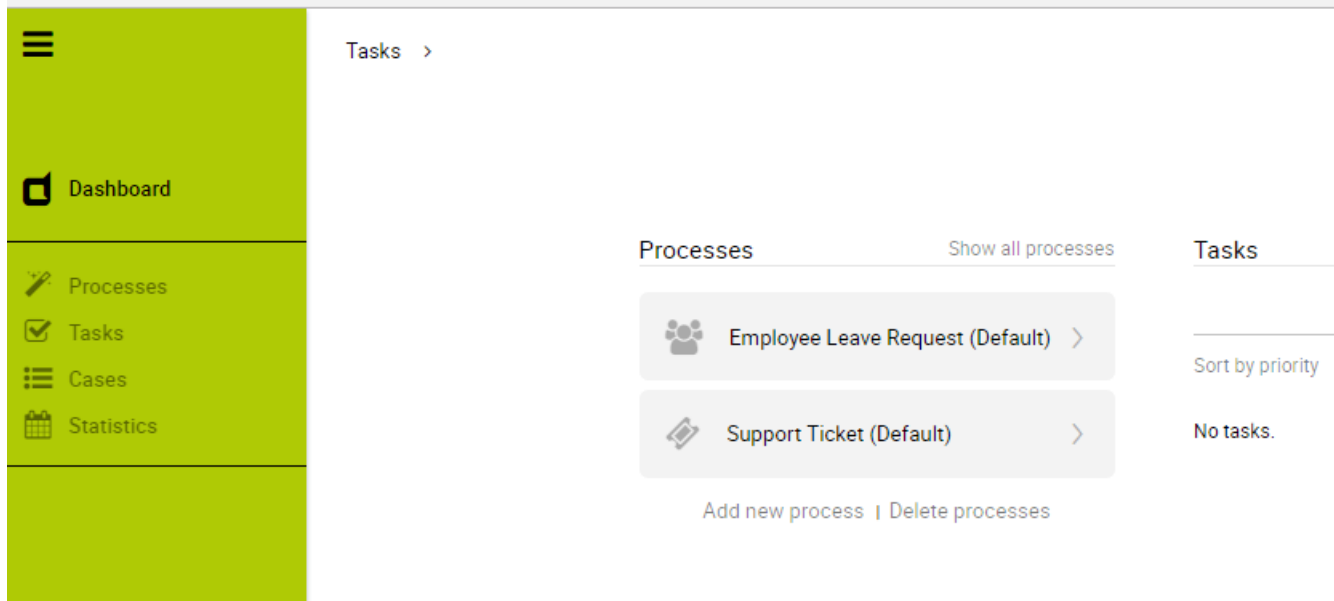
Tip

Your new homepage is the default portal homepage. You can customize it. Reference at Portal home.

3. Set category for tasks

To categorize tasks, set values for `Category` field. Task category can be multi-level if it is separated by slash / signs, as below.

4. How to use feature warning on closing browser/tab

   Sometimes when users are working on a task, if they close tab, browser or refresh page then they may lose their current work. It's a good idea to ask users to verify that they truly want to proceed with the action they just invoke. To use this feature, add `WarnOnClosingBrowserTab` component to the page you want to be affected.

   ```
   <ic:ch.ivy.addon.portalkit.feature.WarnOnClosingBrowserTab   confirmMessage="You
   may lose your work in progress. Do you want to continue?" />
   ```

Depending on browser, content of `confirmMessage` and names of buttons may vary.
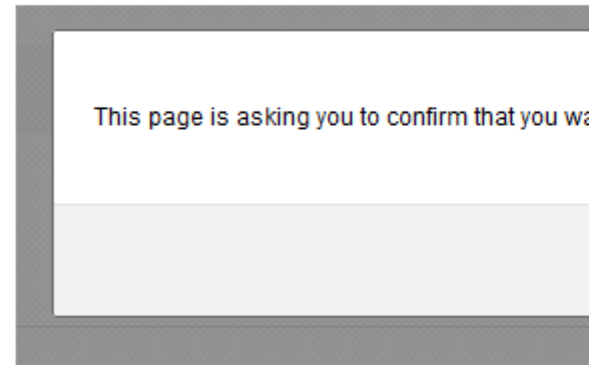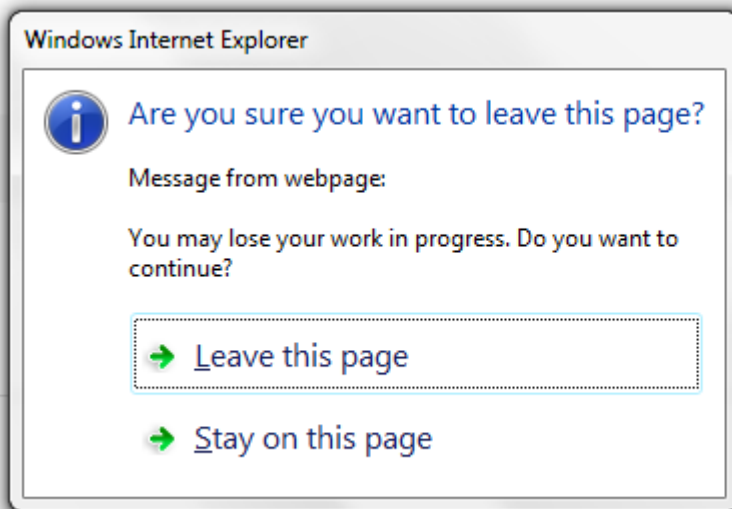
> **Tip**
>
> Normally, when using this component, actions that user invoked outside Portal area like closing tab/browser, refreshing page, clicking on a link on bookmark bar of browser will cause browser to display a confirmation dialog. That might cause the feature work incorrectly.
>
> Actions that lead to navigation that user invoke inside Portal area like clicking on a link/button, submitting a form will not display any confirmation popup.
>
> In some cases, user might use javascript to navigate to another page, for example: set value for `window.location.href` or call `location.reload()`.
>
> If that happens, add this to your javascript function: `showConfirmDialogBeforeUnload = false;`

Portal logos and colors

Change Portal's logos

You can change logo and login logo by modifying default logo in PortalStyle project.

- Modify cms entry `PortalStyle/images/logo/CorporateLogo.png` to update homepage logo.

- Modify cms entry `PortalStyle/images/logo/loginLogo.png` to update login logo.

Change Portal styles

Portal applies LESS framework to support you in customizing the styles of Portal. There are 3 files: `theme.less` (shouldn't be customized), `font-faces.less`, `customization.less`, they are placed at `PortalStyle/webContent/resources/less`.

- `font-faces.less`: replace the default font url-s by your font url-s and add/change other font styles to customize the Portal's font styles.

> **Important**
>
> Do not change `font-family` property values.

- `customization.less`: to change the styles of Portal deeply. E.g. Portal's component styles.

> **Note**
>
> Portal provides several variables (`@body-background-color`, `@menu-color`, `@sidebar-dimension-transition-duration`, `@sidebar-opacity-transition-duration`) to change Portal's style. To override the variables, you should put overriding code (e.g. `@body-background-color: red`) at `customization.less` file.
>
> - `@body-background-color`: it is the background color of Portal.
>
> - `@menu-color`: it is the color of the application menu, the color of texts and icons on the menu will be calculated based on the brightness of the menu color.
>
> - `@sidebar-dimension-transition-duration`: it is the expanding/collapsing transition duration of the application menu.
>
> - `@sidebar-opacity-transition-duration`: it is the opacity transition duration of the text on application menu.

After you finish your customization, compile these above less files to build the css file named `theme.min.css` and put it at `PortalStyle/webContent/resources/css`. You are highly recommended to run the `mvn lesscss:compile` maven command in PortalStyle to do it quickly.

Login page
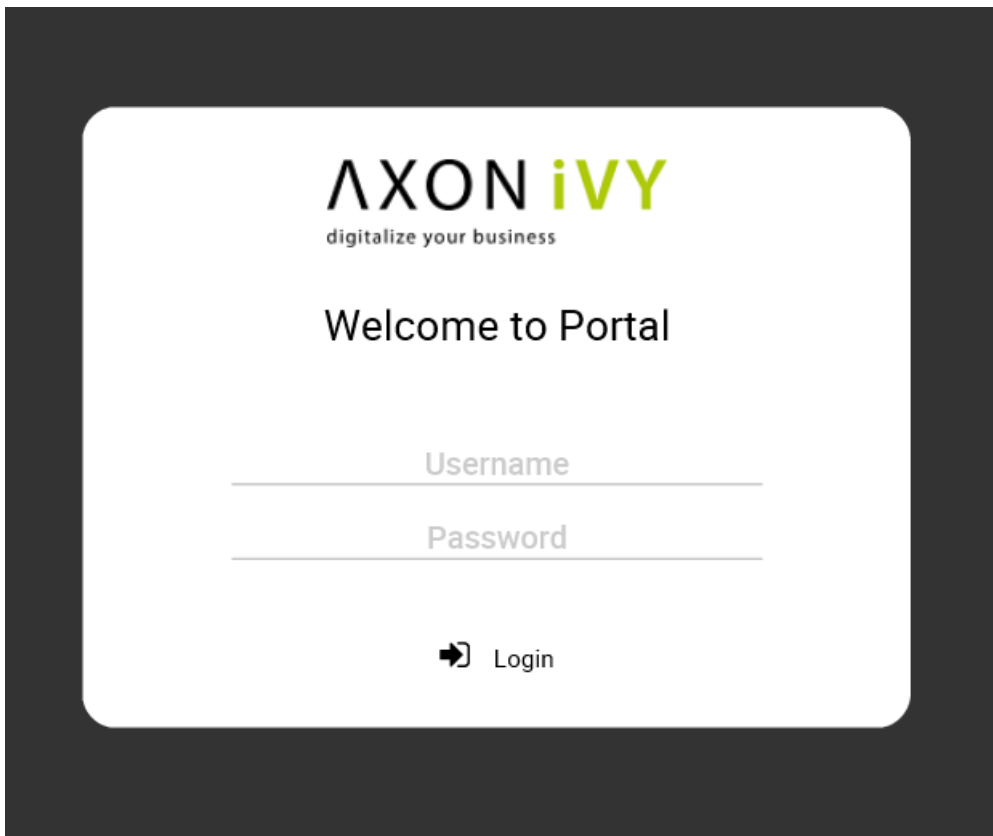
To replace default login page, extends existing templates with `ui:define name="login"` to define your new login component like below

```
<ui:composition template="/layouts/BasicTemplate.xhtml">

<ui:define name="login">

<ic:internaltest.ui.YourOwnLoginComponent />
```

```
</ui:define>
```

```
</ui:composition>
```

Portal home

Before beginning

This guide assumes that you are already familiar with CSS to integrate your own widgets.

Introduction

The default home page of Portal has three built-in widgets in order: `Process widget`, `Task widget`, `Statistic widget`. If it does not fit your needs, you can replace it by your own one. We decided that based on screen size, widget may become hidden, not smaller. So we set **fix length** for each widget. Length of `Process widget` is **270 pixels**, `Task widget` is **520 pixels** and `Task widget` is **270 pixels**.
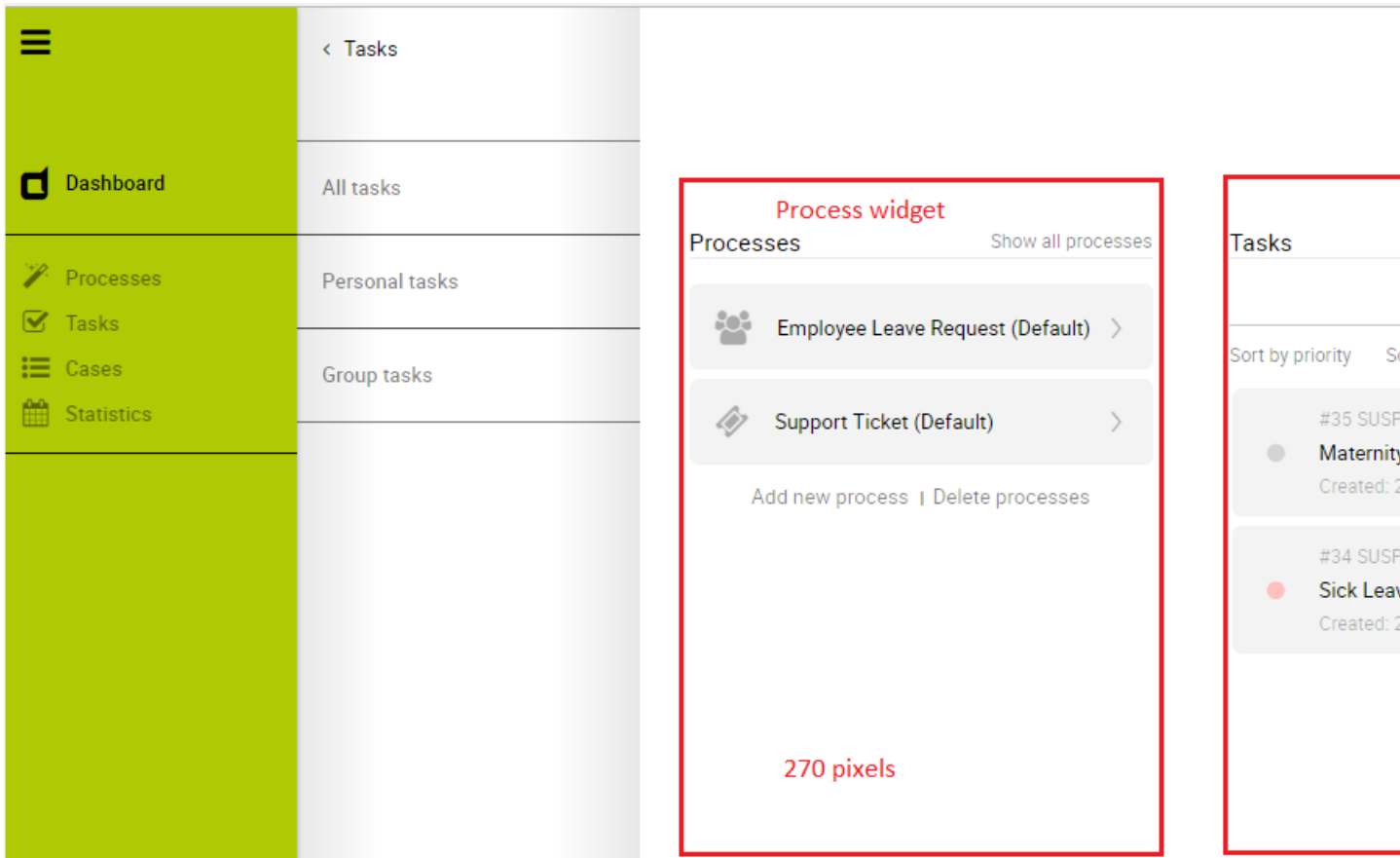
Basic usage

Following these steps to have your own Portal Home:

1. Create a new home page which uses the `DefaultHomePageTemplate.xhtml` template. By doing this, your new home page will inherit the widgets from the default home page and has a place holder for your own widgets.

    Your custom home page should look like below:

    ```
    <ui:composition template="/layouts/DefaultHomePageTemplate.xhtml" ... >
    ```

    ```
    <ui:define name="customWidget">
    ```

```
...

</ui:define>

</ui:composition>
```

2. *In case of single Portal:* Create a new process start for the new home page. Now you will use this process start as the entry point of your portal instead of the default one. **To let portal know about your new portal home, you have to go to the portal settings and set the portal home url to the new one. e.g: HOMEPAGE_URL=http://localhost:8081/ ivy/pro/designer/CustomizePortalHome/157454FCA39C3844/start.ivp**

Admin Settings

Servers	Applications	Settings

Key	Value
HOMEPAGE_URL	<YOUR_CUSTOM_HOMEPAGE>

In case of multi Portal: refer to Setup multi portals to setup.

Note

Currently, responsive custom home page is not supported.

Advanced usage

The `DefaultHomePageTemplate.xhtml` template supports some customizations.

Display your statistics data

The template has a parameter: `statistic` to display the statistic data. Their default values are the tasks which the session user can work on.

```
<ui:composition template="/layouts/DefaultHomePageTemplate.xhtml" ... >

<ui:param name="statistic" value="#{logic.getStatistic()}" /> <!-- A method returns
StatisticData -->

</ui:composition>
```

Display/hide the default widgets

The template has three parameters: `displayProcessWidget`, `displayTaskWidget`, `displayStatisticWidget` to display or hide the default widgets. Their default values are true, you can set them to boolean values (true/false) to display or hide them as you need.

```
<ui:composition template="/layouts/DefaultHomePageTemplate.xhtml" ... >
```

```
<ui:param name="displayTaskWidget" value="false" />
```

```
</ui:composition>
```

> **Tip**
> Task widget now is hidden.

Customize the default widget's sections

The template has the placeholders to redefine the default widgets' sections.

```
<ui:composition template="/layouts/DefaultHomePageTemplate.xhtml" ... >
```

```
<ui:define name="statisticWidget">
```

```
<div    class="js-dashboard-main-content-3rd-col    dashboard-main-content-3rd-col
layout-col">
```

```
<h:panelGroup layout="block" styleClass="js-statistic-widget" id="statistic-widget-
container">
```

```
<!-- KEEP THE STATISTIC WIDGET -->
```

```
<ic:ch.ivy.addon.portalkit.component.StatisticWidget         id="statistics-widget"
compactMode="true" tasks="#{tasks}" ... >
```

```
<!-- ADD THE WEATHER WIDGET BELOW STATISTIC WIDGET -->
```

```
<ic:my.namespace.WeatherWidget />
```

```
</h:panelGroup>
```

```
</div>
```

```
</ui:define>
```

```
</ui:composition>
```

Add your own widgets

The template has a placeholder to add your own widgets. Your own widgets' default positions are next to statistic widget, you can change them by setting the default widgets' positions.

```
<ui:composition template="/layouts/DefaultHomePageTemplate.xhtml" ... >
```

```
<ui:define name="customWidget">
```

```
<ic:my.namespace.ComponentName ... />
```

```
</ui:define>
```

```
</ui:composition>
```

> **Tip**
> This custom widget will show below the 3 default widget

Change the page's title

The default page title is Cockpit. Apply the following code to change it:

```
<ui:composition template="/layouts/DefaultHomePageTemplate.xhtml" ... >
```

```
<ui:define name="pageTitle">YOUR PAGE'S TITLE</ui:define>

</ui:composition>
```

Task widget

TaskWidget is a built-in component of Portal which contains the tasks users can interact with. In order to show needed task's information, Portal supports overriding concept for TaskWidget. Each TaskWidget contains 2 parts:

1. UI: TaskHeader and TaskListHeader and TaskBody and TaskFilter

2. Data query: display the tasks as you want

Important

1. Task header customization does not support responsive design, smaller resolutions (iPad, etc.)

2. Additional properties cannot be displayed right now as a column

3. Task header's buttons cannot be modified (they stay where they are)

How to override task widget's UI

1. Introduce an Axon.ivy project which has `PortalTemplate` as a required library.

2. Copy the `PortalStart` process from `PortalTemplate` to your project. This process is new home page and administrator should register this link by global

3. Refer to Customize Portal home to set new home page.

4. Override Task widget at home page.

   • Introduce a new HTMLDialog which uses template `/layouts/DefaultHomePageTemplate.xhtml` (refer to Responsiveness to override task details's responsiveness). You can take a look at `PortalHome.xhtml` to see how to customize it.

     ## Tip

     Highly recommend to copy the `PortalHome` HTMLDialog in PortalTemplate and change the copied one's view.

   • Replace the old home HTMLDialog of PortalTemplate to the new one at the copied process which you have done at step 2.

5. Override Task widget in other places: TaskList page, Task Search result, Relate tasks of a case, History tasks of a case.

   • It is the same way as home page. You should introduce a new HTMLDialog which uses template `/layouts/PortalTasksTemplate.xhtml` (refer to Responsiveness to override responsiveness). You can take a look at `PortalTasks.xhtml` to see how to customize it.

     ## Tip

     Highly recommend to copy the `PortalTasks` HTMLDialog in PortalTemplate and change the copied one's view.

   • Use Axon.ivy Override to override the `OpenPortalTasks` callable. The original implementation of this callable is calling `PortalTasks`, change it to call the customized Page introduced in the step above. The signature of this callable is `useView(TaskView)` and customized page must receive this `TaskView` instance, put in the dialog's `Data` with the exact name `taskView`.

6.   After previous steps, you can override TaskHeader and TaskListHeader and TaskBody and TaskFilter

Task header

Refer to the `taskListHeader` and `taskHeader` sections in `PortalHome.xhtml` and `PortalTasks.xhtml` of PortalTemplate. In case your task widget has new columns, you should override TaskLazyDataModel to make the sort function of these columns work:

- Introduce a java class extends TaskLazyDataModel

- Override the `extendSort` method, the `extendSortTasksInNotDisplayedTaskMap` method and extend the sort function for the added columns (see the methods' Javadoc comments)

- Use Axon.ivy Override to override the `InitializeTaskDataModel` callable and initialize data model by your customized one.

- In your customized portal home HTMLDialog, the customized data model should be passed as a parameter to template and components (refer to `PortalHome.xhtml`)

- In your customized portal tasks HTMLDialog, the customized data model should be passed as a parameter to components (refer to `PortalTasks.xhtml`).

Task body

Refer to the `taskBody` section in `PortalHome.xhtml` and `PortalTasks.xhtml` of PortalTemplate.

If you need to apply the responsiveness behavior of Portal for task details, there are 2 components which can be used: `ResponsivenessHandleContainer` and `ResponsivenessHandleButton`:

- ResponsivenessHandleContainer: a container contains `ResponsivenessHandleButton` and your javascript which contains the onstart and oncomplete function of `ResponsivenessHandleButton`.

- ResponsivenessHandleButton: contains the params which handle responsiveness:

  - icon: the icon of button

  - displayedSelectors: css selectors of the components which need to be displayed.

  - hiddenSelectorsInLargeScreen: css selectors of the components which need to be hidden in large screen (width: 1920).

  - hiddenSelectorsInMediumScreen: css selectors of the components which need to be hidden in medium screen (width: 1366).

  - hiddenSelectorsInSmallScreen: css selectors of the components which need to be hidden in small screen (width: 1024).

  - fadeTime: the fade in/out time.

  - onstart: client side callback to execute before responsiveness execution.

  - oncomplete: client side callback to execute after responsiveness execution.

For example:

```
<ui:composition template="/layouts/PortalTasksTemplate.xhtml">

<ui:param name="useOverrideBody" value="true" />

<ui:define name="taskBody">

Customized content

<ic:ch.ivy.addon.portalkit.component.ResponsivenessHandleContainer
styleClass="hidden-lg">
```

```
<ic:ch.ivy.addon.portalkit.component.ResponsivenessHandleButton    icon="fa    fa-
file       js-note-column-responsive-button"       displayedSelectors="['#task-
note']"       hiddenSelectorsInMediumScreen="['.task-details       .replaced']"
hiddenSelectorsInSmallScreen="['.task-details .replaced']" />

<h:outputScript library="js" name="task-detail-default-responsiveness.js" />

</ic:ch.ivy.addon.portalkit.component.ResponsivenessHandleContainer>

</ui:define>

</ui:composition>
```
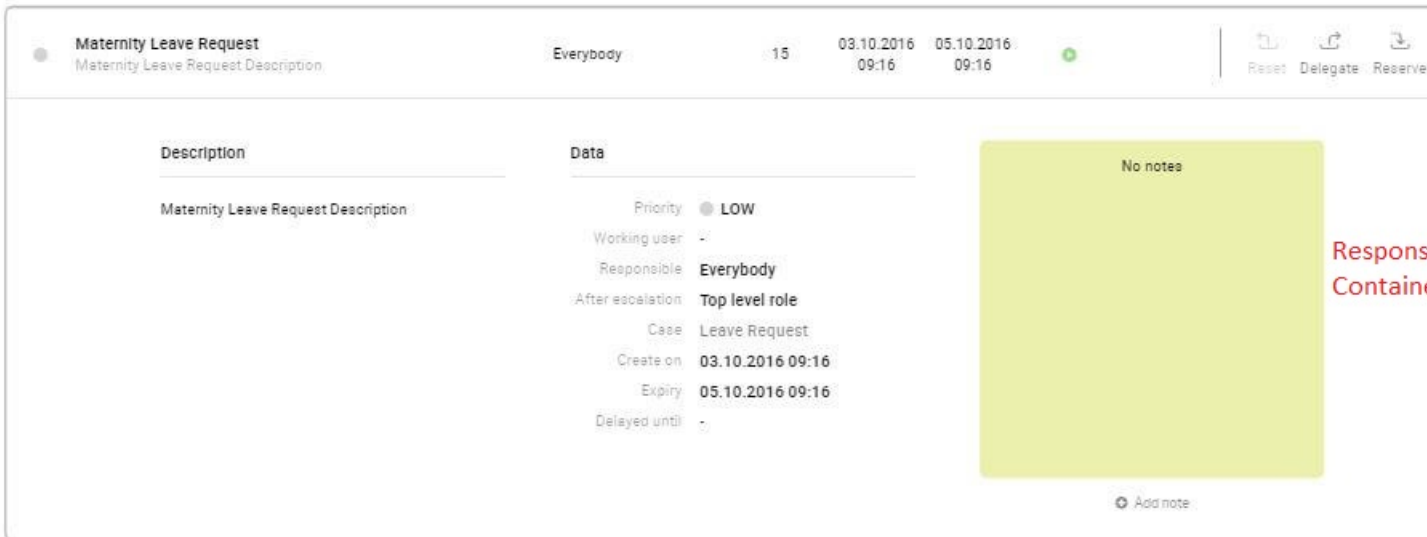

Task filter

- Refer to the `taskFilter` section in `PortalHome.xhtml` and `PortalTasks.xhtml` of PortalTemplate.

- In order to introduce new filter, create a new java class extends TaskFilter and override its methods (see javadoc comments)
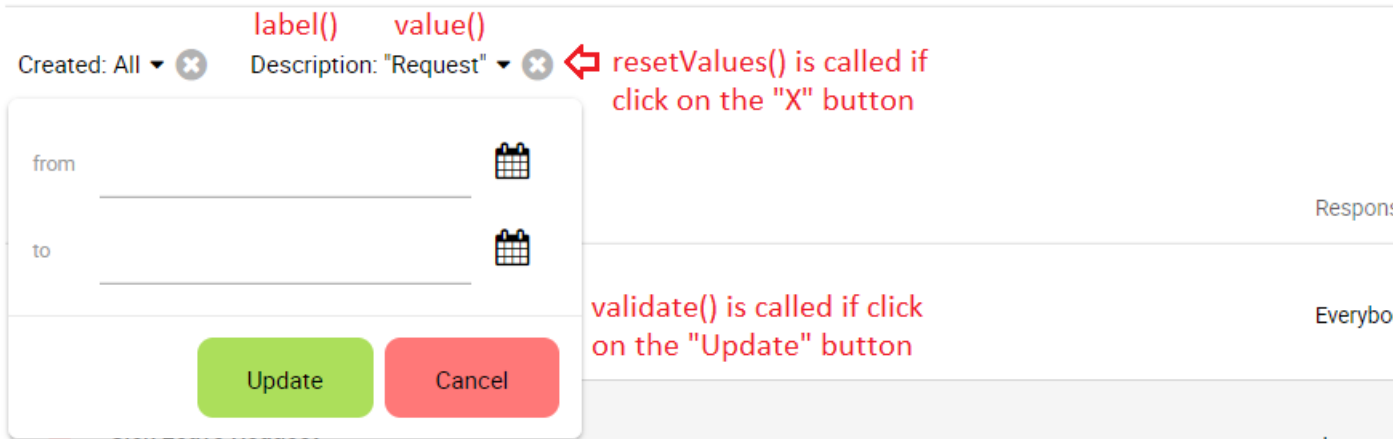
- Introduce a java class extends TaskFilterContainer. This filter container contains your filters, you can reuse default filters, refer to `DefaultTaskFilterContainer.java`

> **Tip**
>
> StateFilter is added as default to container. If you don't need it, use this code in constructor:
> `filters.remove(stateFilter);`

- Introduce a java class extends TaskLazyDataModel. Override the `initFilterContainer` method and initialize filter container (see javadoc comments)

- Use Axon.ivy Override to override the `InitializeTaskDataModel` callable and initialize data model by your customized one.

- In your customized portal home HTMLDialog, the customized data model and filter container should be passed as parameters to template and components (refer to `PortalHome.xhtml`)

- In your customized portal tasks HTMLDialog, the customized data model and filter container should be passed as parameters to components (refer to `PortalTasks.xhtml`).

How to override task widget's data query

Override the `BuildTaskJsonQuery` callable process of PortalKit and build your own query to effect the data of task widget, task categories and statistic widget.

Apply the following steps in case you would like to provide data for task list after navigating to task list from your page, e.g. clicking on a bar chart then opening the tasks of that bar:

- Use the `OpenPortalTasks` callable process with the `TaskView` parameter. It is used to define which information are displayed in TaskWidget.

- Refer to TaskView, TaskSearchCriteria, TaskQueryCriteria to build your TaskView

```
TaskLazyDataModel dataModel = new TaskLazyDataModel();
dataModel.getQueryCriteria().setTaskQuery(YOUR_TASK_QUERY); // Set your TaskQuery
dataModel.getSearchCriteria().setInvolvedUsername(true); // Use this code line if you woul
out.taskView = TaskView.create().dataModel(dataModel).showHeaderToolbar(false).createNewTa
```

Custom task delegate

Portal allows to customize the list of users and roles that a task can be delegated to. This can be done following these steps:

1. Introduce a Axon.ivy project which has `PortalTemplate` as a required library and its own `PortalStart` process. Refer to step 1, 2, 3, 4 in override task widget's UI guide.

2. In your project, override the callable subprocess `CalculateTaskDelegate`
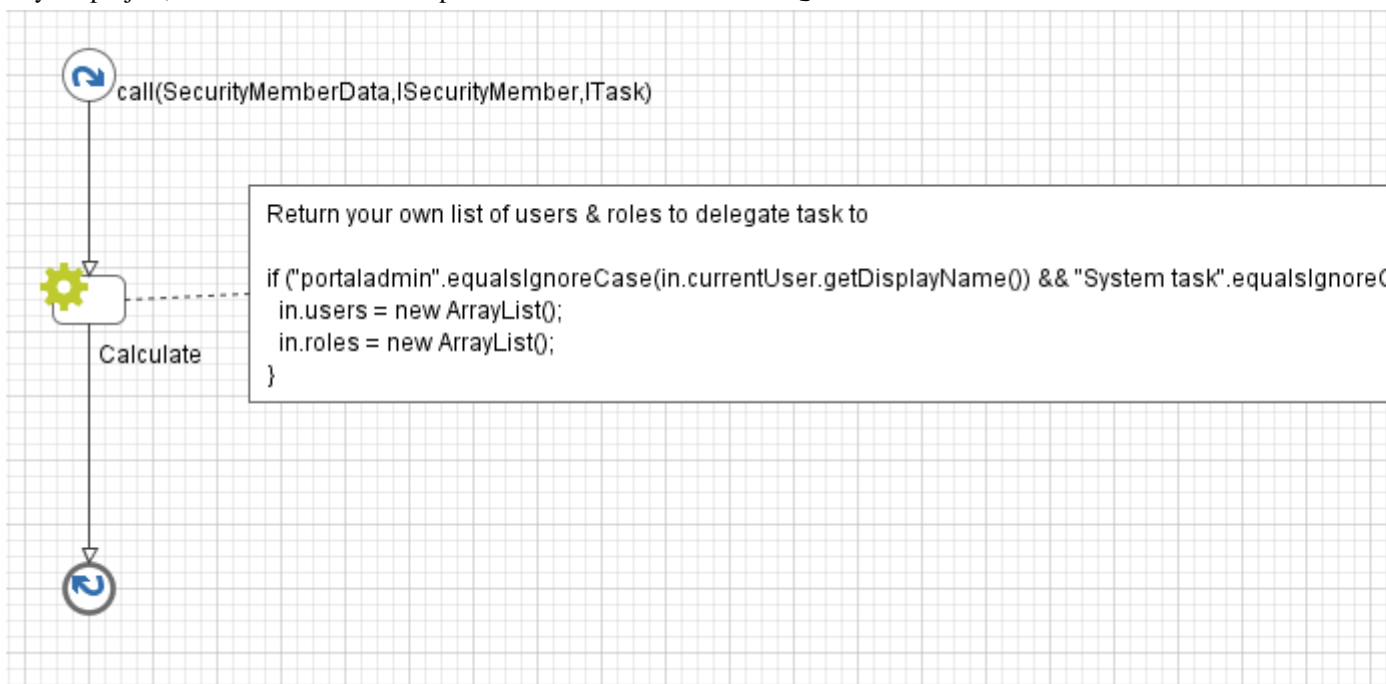
3. The callable subprocess data contains the current user `in.currentUser` and the current task to be delegated `in.task`. The lists `in.users` and `in.roles` contain all possible users and roles that the task can be delegated to. Modify those two to have your own delegate list.

Case widget

CaseWidget is a built-in component of Portal which contains cases users can interact with. In order to show needed case's information, Portal supports overriding concept for CaseItem (inside CaseWidget). Each CaseItem contains two part: `CaseHeader` and `CaseBody`.

⚠️ Important

`CaseHeader` area is not supported to override yet.

How to override case item

1. Introduce an Axon.ivy project which has `PortalTemplate` as a required library.

2. Copy the `PortalStart` process from `PortalTemplate` to your project.

3. Introduce a new HtmlDialog Page which uses the template `/layouts/PortalCasesTemplate.xhtml` (refer to Responsiveness to override responsiveness). The `PortalCasesTemplate` is designed such that it will insert the content between its predefined place holder. Put your customized content in there. There is a placeholder named: `caseBody`. The effect only takes place when the `ui:param` named `useOverrideBody` is set to `true`. Clients can set this parameter to a value expression for flexibility. For example:

```
<ui:composition template="/layouts/PortalCasesTemplate.xhtml">

<ui:param                    name="useOverrideBody"                    value="#{not
case.getCustomVarCharField1().isEmpty()}" />
```

```
<ui:define name="caseBody">

This is the customized content of the case: #{case.getId()}

<ui:fragment    rendered="#{case.getCustomVarCharField1().equals('I    am    a
secret')}">

<h:outputText value="I'm a secret case so I'm displayed differently" />

</ui:fragment>

<ui:fragment  rendered="#{case.getCustomVarCharField1().equals('I  am  even  more
secret')}">

<!-- Put your secret content here -->

</ui:fragment>

</ui:define>

</ui:composition>
```

4. If you need to apply the responsiveness behavior of Portal, there are 2 components which can be used: `ResponsivenessHandleContainer` and `ResponsivenessHandleButton`:

- ResponsivenessHandleContainer: a container contains `ResponsivenessHandleButton` and your javascript which contains the onstart and oncomplete function of `ResponsivenessHandleButton`.

- ResponsivenessHandleButton: contains the params which handle responsiveness:

  - icon: the icon of button

  - displayedSelectors: css selectors of the components which need to be displayed.

  - hiddenSelectorsInLargeScreen: css selectors of the components which need to be hidden in large screen (width: 1920px).

  - hiddenSelectorsInMediumScreen: css selectors of the components which need to be displayed in medium screen (width: 1366px).

  - hiddenSelectorsInSmallScreen: css selectors of the components which need to be displayed in small screen (width: 1024px).

  - fadeTime: the fade in/out time. (metric: millisecond)

  - onstart: client side callback to execute before responsiveness execution.

  - oncomplete: client side callback to execute after responsiveness execution.

5. Use Axon.ivy Override to override the `OpenPortalCases` callable . The original implementation is calling `PortalCases`, change it to call the Page introduced in the step above. The signature of this callable is `useView(CaseView)` and your customized Page must receive this `CaseView` instance, put in the dialog's `Data` with the exact name `caseView`.

6. Introduce a Business Process which starts the page `PortalHome`. To make the overriding take effect, the client must use this process to start `Portal` (instead of the original one on `PortalTemplate`). Remember to set the home page variable to the new home page url (see Portal home for more details).

7. In case you want to use the `CaseTempate` template, you may want to define the `caseStatusTab` with your customized content. For example:

```
<ui:composition template="/layouts/CaseTemplate.xhtml">

<ui:define name="caseStatusTab">

This is the customized content of the case: #{caseId}

</ui:define>

</ui:composition>
```

Default user process

Introduction

In Portal homepage, the `Process  widget` displays default processes, you can customize them so that project important starts are displayed.

Customization

Create a callable sub process in your project with the `createDefaultUserProcesses()` signature. It must return a list of user processes: `defaultUserProcesses(List<UserProcess>)` (you can override this process in PortalKit):
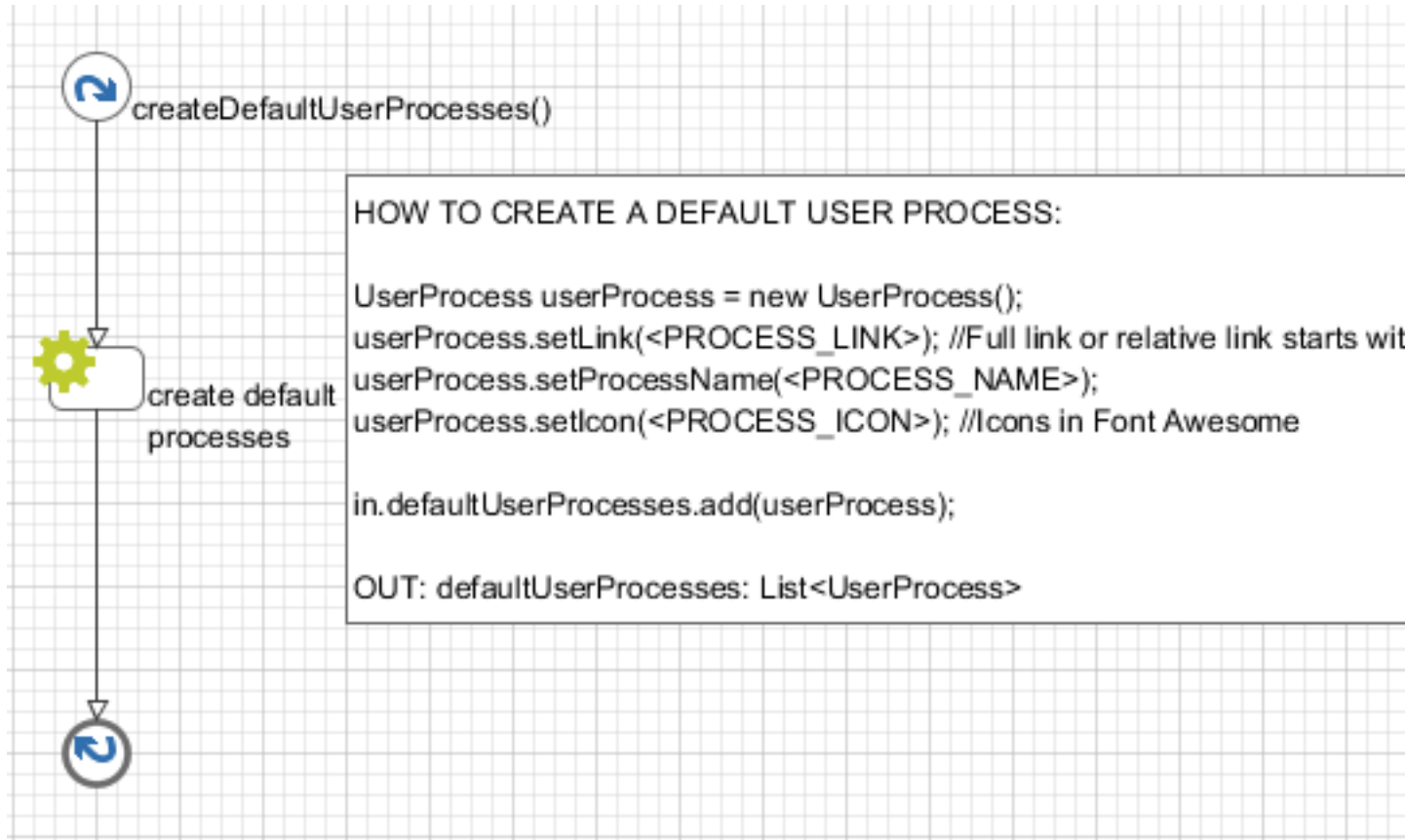
HOW TO CREATE A DEFAULT USER PROCESS:

UserProcess userProcess = new UserProcess();
userProcess.setLink(<PROCESS_LINK>); //Full link or relative link starts wit
userProcess.setProcessName(<PROCESS_NAME>);
userProcess.setIcon(<PROCESS_ICON>); //Icons in Font Awesome

in.defaultUserProcesses.add(userProcess);

OUT: defaultUserProcesses: List<UserProcess>

Hide technical stuffs

Hide technical roles

A technical role is the role which is not displayed anywhere (e.g. delegate, absence management). AXONIVY_PORTAL_ADMIN is a technical role by default.

To mark a role as a technical role, set the **HIDE** property with any value to the role.

Tip

Use the utility method of Portal to set property:

ch.ivy.addon.portalkit.util.RoleUtils.setProperty([YOUR_ROLE],   ch.ivy.addon.portalkit.util.HIDE, [ANY_VALUE])

Hide technical tasks

A technical task is the task which is not displayed in any task lists of Portal.

To mark a task as a technical task, set the **HIDE** additional property with any value to the task.

Tip

Use the utility methods of Portal:

- Set property: ch.ivy.addon.portalkit.util.TaskUtils.setHidePropertyToHideInPortal(ITask)

- Remove property: ch.ivy.addon.portalkit.util.TaskUtils.removeHidePropertyToDisplayInPortal(ITask)

Chapter 6. Settings

User Settings provide options for configuring Portal applications:

- Admin settings

- Absence and substitute settings

- Email settings

- Language settings
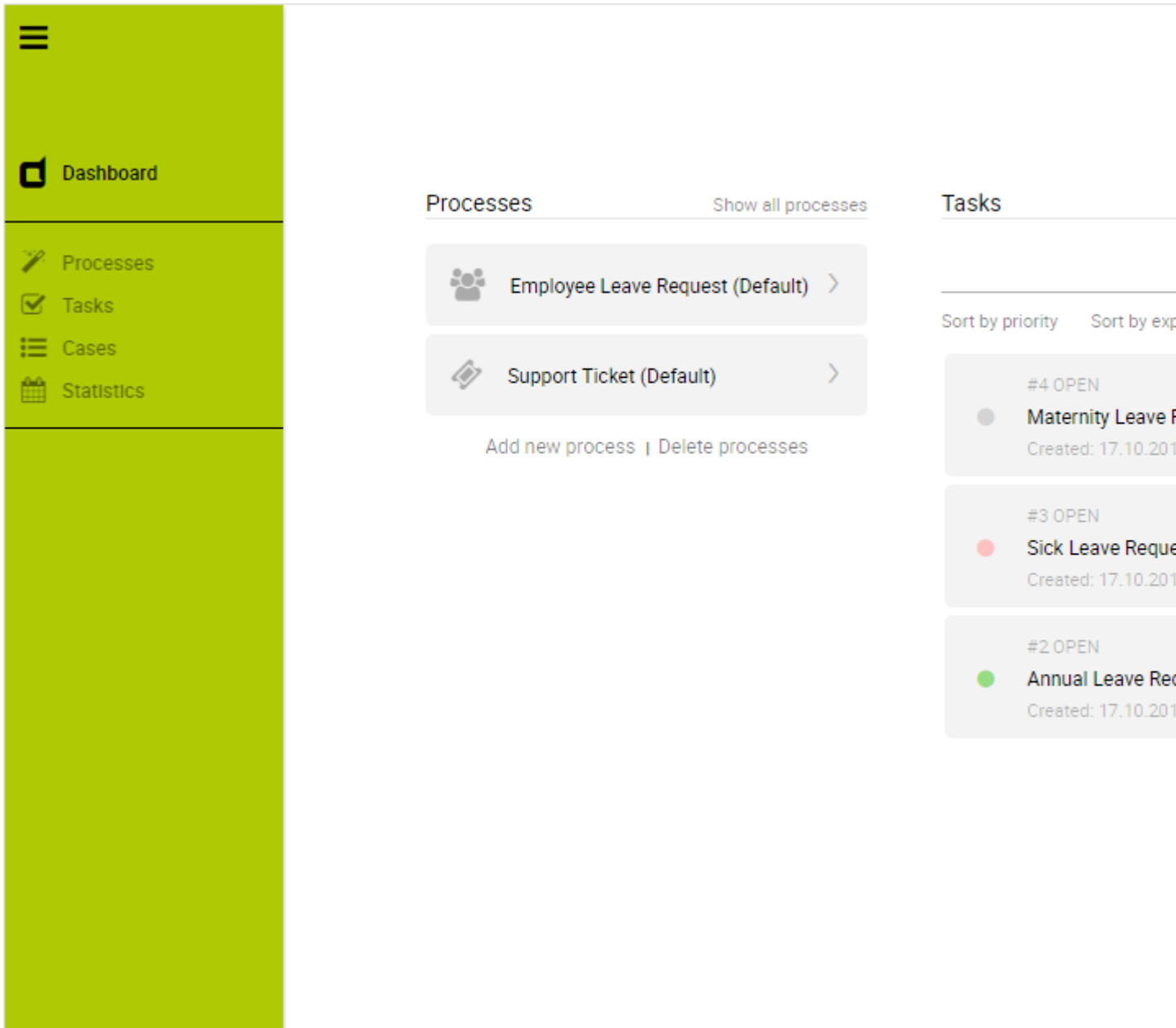
Admin settings

User needs to have role `AXONIVY_PORTAL_ADMIN` to see this menu item, it is used to configure Portal configuration, see different Portal configurations in General concept

Setup multi portals

⊗ Warning

Multi-portal is not officially supported, only an experimental mode is available.

1. Add a new server

   a. Open `Admin Settings` by selecting the item in `UserMenu` on the topbar, if your page using layout of `PortalTemplate`.
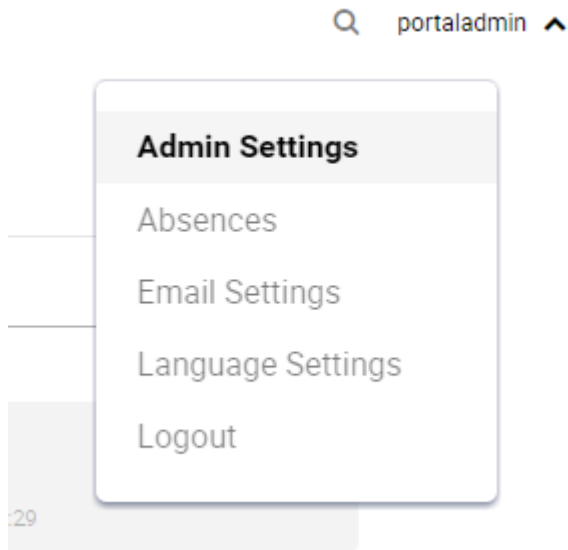
   ⚠ Important

   If your application does not use templates of `PortalTemplate` project, you have to create a page and use `AdminSettings` component inside.

   💡 Tip

   To be able to open Admin Settings dialog, user needs to have role `AXONIVY_PORTAL_ADMIN`.

   b. Click on `New` button to add new server.
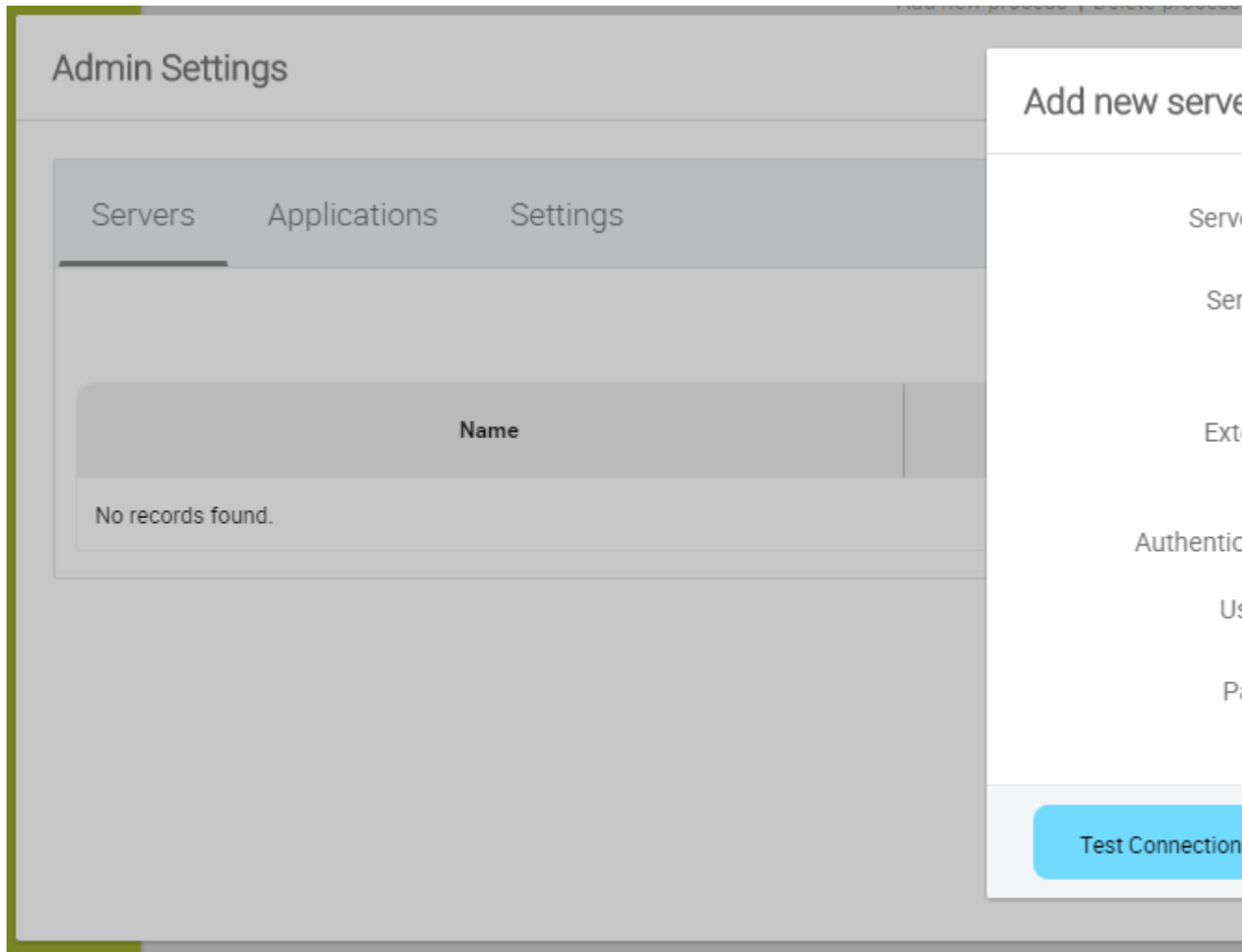
Note

- `Server path` must be the link of the `PortalConnector` application (e.g. `http://path-to-ivy:8080/ivy/ws/System/PortalConnector`)

- Web services authentication of the built-in PortalConnector of the Axon.ivy engine are configured as `HTTP Basic` . Therefore, when adding a new server using the built-in PortalConnector, use authentication type `HTTP Basic` .

- The username, password are used by PortalKit to invoke the web services of PortalConnector.

  These are accounts of the application that deploys PortalConnector.

  The built-in PortalConnector are deployed to application System therefore use the `username/ password of system administrators` when connecting to built-in PortalConnector.

  To create system administrators, refer to the document **EngineGuide > chapter Configuration > Engine Configuration > System Administrators** . E.g. in the latest EngineGuide, refer to this guideline .

Tip

- In multi-Portal mode, Portal uses the External Host value as host value for task or process starts URLs. If this value is empty, Portal will use the host value as configured in the server path field. For other mode, Portal will use the host value of the web address of the engine for the URLs.

- External Host value is returned from configured server's system property `WebServer.ExternalHostName` (you might want to change it via Engine's `AdminUI`).

  You can sync this value by clicking `Test Connection` button, if `WebServer.ExternalHostName` is set the value would be updated, otherwise displays `AUTO_DETECT`.

  Configured server information is needed for calling service to get this value.

2. Add a new application

Choose `Application` tab on `Admin Settings` dialog and click on `New` button to add new application. Here you can choose application type either as Ivy application or Third Party application.

💡 **Tip**

- `Application name` is the name of the application when you create it. `Display name` is the name of the application to be shown on Portal UI.

- If `Active` is not checked, the application will be disabled on the application menu. To prevent its tasks, cases and processes to be shown, you must leave the `Visible` box unchecked.

- `Portal link` specifies the link you will be redirect to when select the application on the application menu.

- `Absence planning` if checked will enable absence feature for the application.

- `Mail notification` If checked will enable email setting feature for the application.

- For multiple languages of application display name, you need to create the "AppInfo/SupportedLanguages" CMS which defines how many languages your application supports. See the below "Language settings" for more details.

3. Synchronize data

After adding new settings (server, application), click on `Synchronize Settings` to synch new settings and information about users to registered servers. Close dialog (logging out may be required) and check if newly created application is available on left menu.
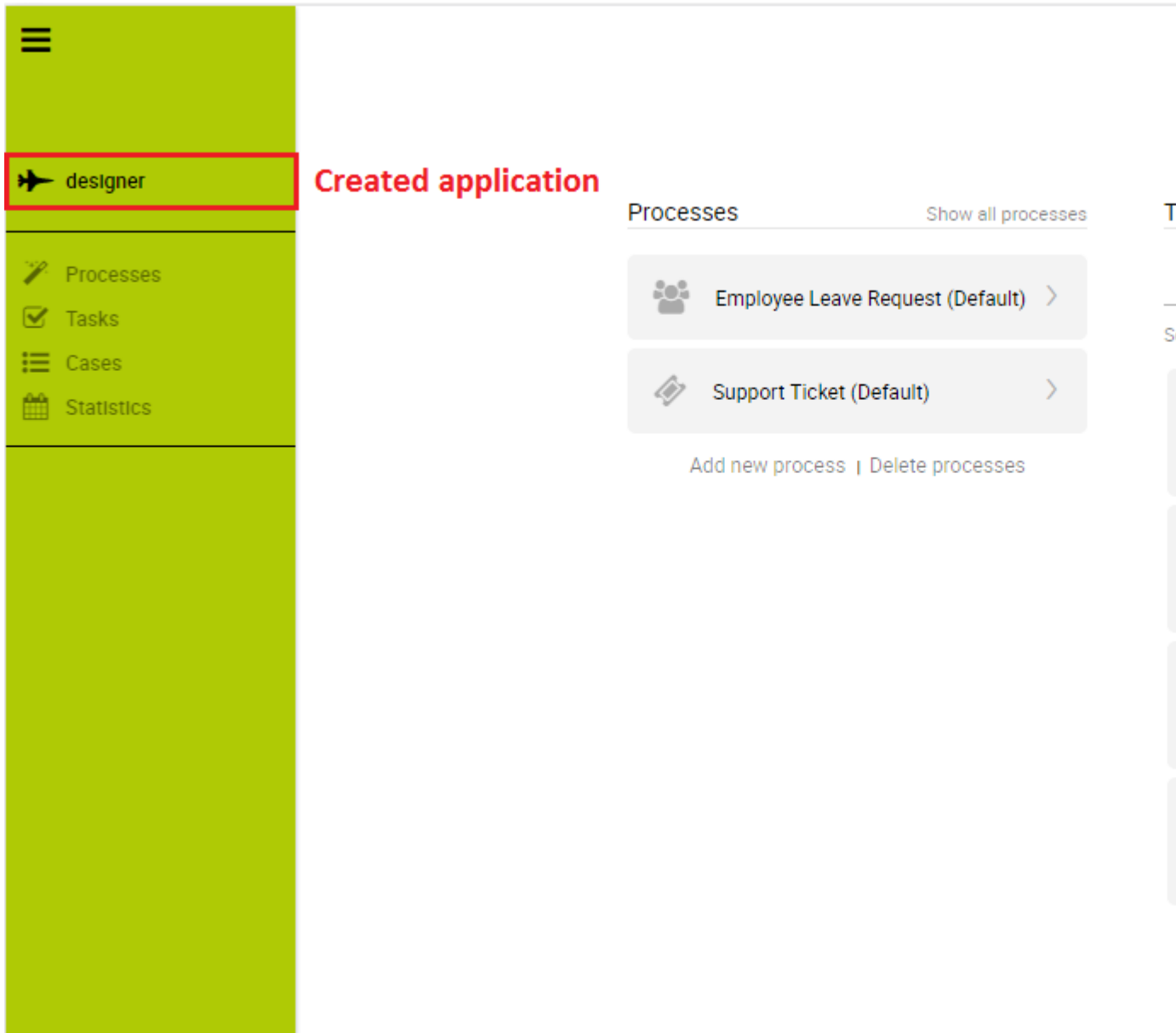
Tip

In single mode, Portal detects the applications that have the login user in the current engine. Based on that, Portal will collect tasks, cases of those applications. In multi-portal mode, Portal will only collect on registered applications.

By default, Portal will automatically synchronize user settings to registered servers at 5AM everyday. This synchronization process is needed to run at only one application for each server. If there are multiple applications registered for one server, the first application in alphabetical order will run the synchronization process. You can change the time when the synchronization start by going to AdminUI and change the value of the global variable `PortalStartTimeSynchUserExpression` of the application that will run the synchronization process.

Absence and substitute settings

- Choose **Absences** from **User Settings** options.

- To create new Absence, click **New Absence** to open the dialog as below:

≡

Dashboard

Processes
Tasks
Cases
Statistics

Pro

Absences

☐ Display absences in the past

Name ⇕	From ⇕	Until ⇕
No records found.		

- `till/from date` must not be empty.

- `till date` must be greater than or equal `from`.

- `till date` must not be in the past (greater than or equal today).

- User can edit or delete Absences by click edit/delete on the Absences list.

- Deputies area will display all the information of deputy on each application.

☰

Dashboard

Processes

Tasks

Cases

Statistics

Pro

Absences

☐ Display absences in the past

Name ⇅	From ⇅	Until ⇅
No records found.		

- User cannot set deputies for hidden roles.

- All the items will be saved after click Save button.

Email settings

- To configure mails of Portal applications, select **Email Settings** in **User Settings** .

- You can configure one email setting for all applications or each application separately.

-
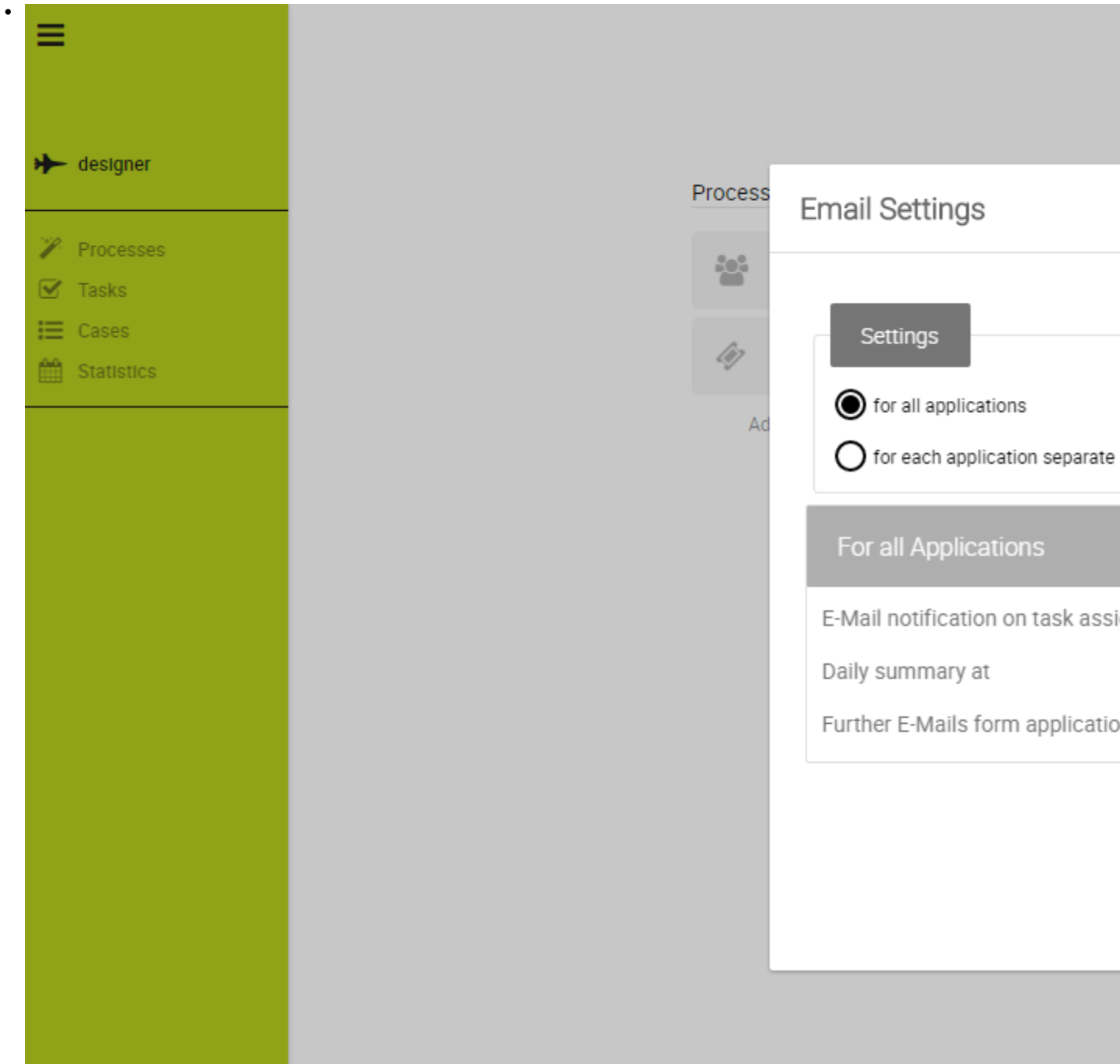
- All the items will be saved when click save button.

Language settings

- To configure languages of Portal applications, select **Language Settings** in **User Settings** .

- UI reads current languages settings for all applications.

- To change language for application, select one in the languages dropdown list of application. When the change is saved, the language will be set for application (Click on the application in header menu to reload application and see the change of language).

- For multiple languages, the CMS key **/AppInfo/SupportedLanguages** must exist in your application. This entry contains list of all languages supported by your application, separated by comma.

- Must not contain spaces

- Same as display name of Locale

- Separated by comma

- Process Model version, which has this CMS, must active

Chapter 7. Troubleshooting

Here you will find solutions to some of the most common problems related to Axon.ivy Portal.

If you can't find your solution here there are some other sources which could help:

- Axon.ivy Q&A

  The Axon.ivy Q&A contains a considerable amount of questions and answers related to Axon.ivy Designer and Engine.

- Stack Overflow

  Problems related to common technologies like Java, JSF, Primefaces are answered on the web, e.g. on Stack Overflow.

- Support

  You can get support via support@soreco.ch (support may be subject to charging, depending on your licence agreement).

IE Security Problem

If you start Portal in Internet Explorer installed on Windows Server then Portal application may not work correctly. The reason could be **Internet Explorer Enhanced Security Configuration** is enabled by default which means ActiveX Controls and scripting are disabled, so Internet sites may not display in Internet Explorer as you expect.

To fix this, you may turn off **Internet Explorer Enhanced Security Configuration** if you are running in Windows Server. Another way is adding that site to the Trusted sites zone in Internet Explorer.

Portal install with IIS

It could be a problem when install portal with IIS with proxy, depends on your environment. Consider to configure if your IIS is called via proxy. Add `-Dhttp.proxyHost` to `VM argument` could help.