Axon.ivy 7.2

Axon.ivy Portal Documentation

Axon.ivy 7.2: Axon.ivy Portal Documentation

Publication date 05.11.2018
Copyright © 2015-2018 AXON Ivy AG

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
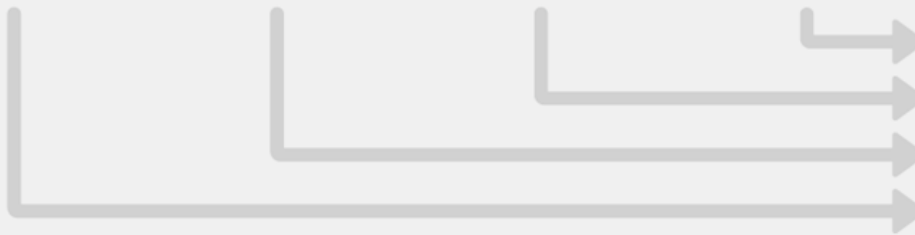
Chapter 2. Installation

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`

Add required libraries

Portal needs Apache POI for exporting to Excel features. Follow these steps to add Apache POI libraries:

1. Copy libraries below and then paste into `webapps/ivy/WEB-INF/lib` folder located on `AxonIvyEngine`, `AxonIvyDesigner.`

   > **Note**
   > You can find libraries inside `AxonIvyDesigner` folder, under `configuration/` `org.eclipse.osgi` (search libraries in this directory), or download from this site.

   - poi-3.15.jar

   - poi-ooxml-3.15.jar

   - poi-ooxml-schemas-3.15.jar

   - xmlbeans-2.6.0.jar

2. Restart AxonIvy Designer/Engine.

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 applications on single engine 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 applications on multi engines 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).

> ⚠️ **Important**
>
> - In single mode, if you deploy a new Portal application with a customer project, you need to deactivate the standard Portal application.
>
> - In multi applications on single engine mode, if you do not need overall dashboard, deactivate the standard Portal application. Otherwise, keep the standard Portal application activated.

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.

Role configuration

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

Table 2.2. Role configuration

Permission settings

Absences

- READ

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

- CREATE/MODIFY

  This function will be disabled if session user does not have `IPermission.USER_CREATE_OWN_ABSENCE` and `IPermission.USER_CREATE_ABSENCE`.

- DELETE

  This function will be disabled if session user does not have `IPermission.USER_DELETE_OWN_ABSENCE` and `IPermission.USER_DELETE_ABSENCE`.

- MANAGE EVERY USER ABSENCES

  User can read, add, delete absences of all users. This function will be disabled if session user does not have all of the following permissions: `IPermission.USER_CREATE_ABSENCE` , `IPermission.USER_DELETE_ABSENCE` , `IPermission.USER_READ_ABSENCES`.

Substitute

- CREATE OWN SUBSTITUTE

  This function will be disabled if session user does not have `IPermission.USER_CREATE_OWN_SUBSTITUTE` and `IPermission.USER_CREATE_SUBSTITUTE`.

- MANAGE EVERY USER SUBSTITUTES

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

Personal task permission

- DELEGATE

  User can delegate his personal or group tasks if he has permission `TaskWriteActivatorOwnTasks` (This permission belongs to Portal permission group). User can delegate all the tasks he see in his task list if he 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` or `IPermission.TASK_RESET`.

  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 TASK NAME

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

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

- CHANGE TASK DESCRIPTION

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

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

- 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 , FAILED .

- CHANGE PRIORITY

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

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

- CHANGE CASE NAME

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

  > ⚠️ **Important**
  >
  > Case state cannot to be: DESTROYED.

- CHANGE CASE DESCRIPTION

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

  > ⚠️ **Important**
  >
  > Case state cannot to be: DESTROYED.

- SEE RELATED TASKS OF CASE

  Session user can see all related tasks of case if he has IPermission.TASK_READ_OWN_CASE_TASKS or IPermission.TASK_READ_ALL .

  > ⚠️ **Important**
  >
  > Case state cannot to be: DESTROYED.

Upload/delete document permission

Upload/delete document function will be enabled if session user has `IPermission.DOCUMENT_WRITE` or `IPermission.DOCUMENT_OF_INVOLVED_CASE_WRITE`.

Express Workflow permission

- CREATE EXPRESS WORKFLOW

Create Express Workflow function will be enabled if session user has `PortalPermission.EXPRESS_CREATE_WORKFLOW` (This permission belongs to Portal permission group, assigned to role Everybody by default).

Statistics permission

- ADD DASHBOARD CHART

Add new charts function will be enabled if session user has `PortalPermission.STATISTIC_ADD_DASHBOARD_CHART` (This permission belongs to Portal permission group, assigned to role Everybody by default).

- ANALYZE TASK

Filter tasks and export data to excel for advanced analysis. This function will be enabled if session user has `PortalPermission.STATISTIC_ANALYZE_TASK` (This permission belongs to Portal permission group, assigned to user admin by default).

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 . 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.

Variable	Default value	Description
PortalSearchDelayInMilliseconds	500	The delay time of search function at top menu.
PortalStartTimeCleanObsoletedDataExpression	0 0 6 * * ?	Cron expression define the time to clean up data of obsoleted users. E.g.: expression for at 6AM every day is 0 0 6 * * ? . Refer to crontrigger . Restart Ivy engine after changing this variable.

Table 2.3. Global variables

Look and feel

Portal doesn't use Modena theme from version 6.3.

All dialogs in Portal and screens of **Self service BPM** and **Axon.ivy Express** have following buttons orders:

- Yes/Ok buttons on the left, No/Cancel buttons on the right

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 make your existing Portal working with current Axon.ivy engine.

How to migrate

> **Important**
> In order to migrate Portal, you need to migrate Axon.ivy, refer Axon.ivy migration notes. Changes in Axon.ivy could lead to problems if the customer project is not migrated properly.

In designer

1. Replace all Portal projects

2. Update PortalTemplate dependency of customer project in pom.xml.

3. If PortalStyle is customized, copy logo, customization.less, font-faces.less, customized stuff from old to new PortalStyle, run maven to compile CSS.

4. Follow migration notes.

5. If customization needs copying code from Portal, merge changes between 2 version of Portal for copied code.

> **Important**
> - Scenario migrating one customer project without customization: Follow the guidelines to step 2.
> - Scenario migrating one customer project with supported customization: Follow the guidelines.
> - Scenario migrating one customer project with (un)supported customization: Follow the guidelines for supported customization. If unsupported customization needs copying code from Portal, merge changes between 2 versions of Portal for copied code. Take care your own unsupported customization.

In engine

1. Convert database schema if needed.

2. Deactivate standard Portal application if it is not needed.

3. Redeploy Portal projects (exclude PortalConnector) and customer project.

4. Follow migration notes to migrate data, if any.

Migrate 7.1.0 to 7.2.0

Portal needs Apache POI for exporting to Excel features. Refer to Add required libraries.

If you overrode task widget's data query described at How to override task widget's data query, follow these steps to migrate

- Add new start method with signature `buildTaskJsonQuery(Boolean)` in your overridden file of BuildTaskJsonQuery.mod (refer to original file BuildTaskJsonQuery.mod).

- If you customized `TaskLazyDataModel`, change `withStartSignature("buildTaskJsonQuery()` to `withStartSignature("buildTaskJsonQuery(Boolean)").withParam("isQueryForHomePage", compactMode)` in your customized `TaskLazyDataModel` class.

There are some changes (DefaultApplicationHomePage, DefaultLoginPage, GlobalSearch) in PortalStart process of Portal Template. If you have customized this process in your project, copy the new PortalStart from Portal Template to your project and re-apply your customization.

> ⚠️ **Important**
>
> In case you already have PortalStart process in your project, delete all elements in that process and copy everything from PortalStart process of Portal Template (to prevent start link id change). `Do not delete PortalStart proccess in your project and copy new again.`
>
> Check map param result of callable process after copy to make sure it's the same as original process.
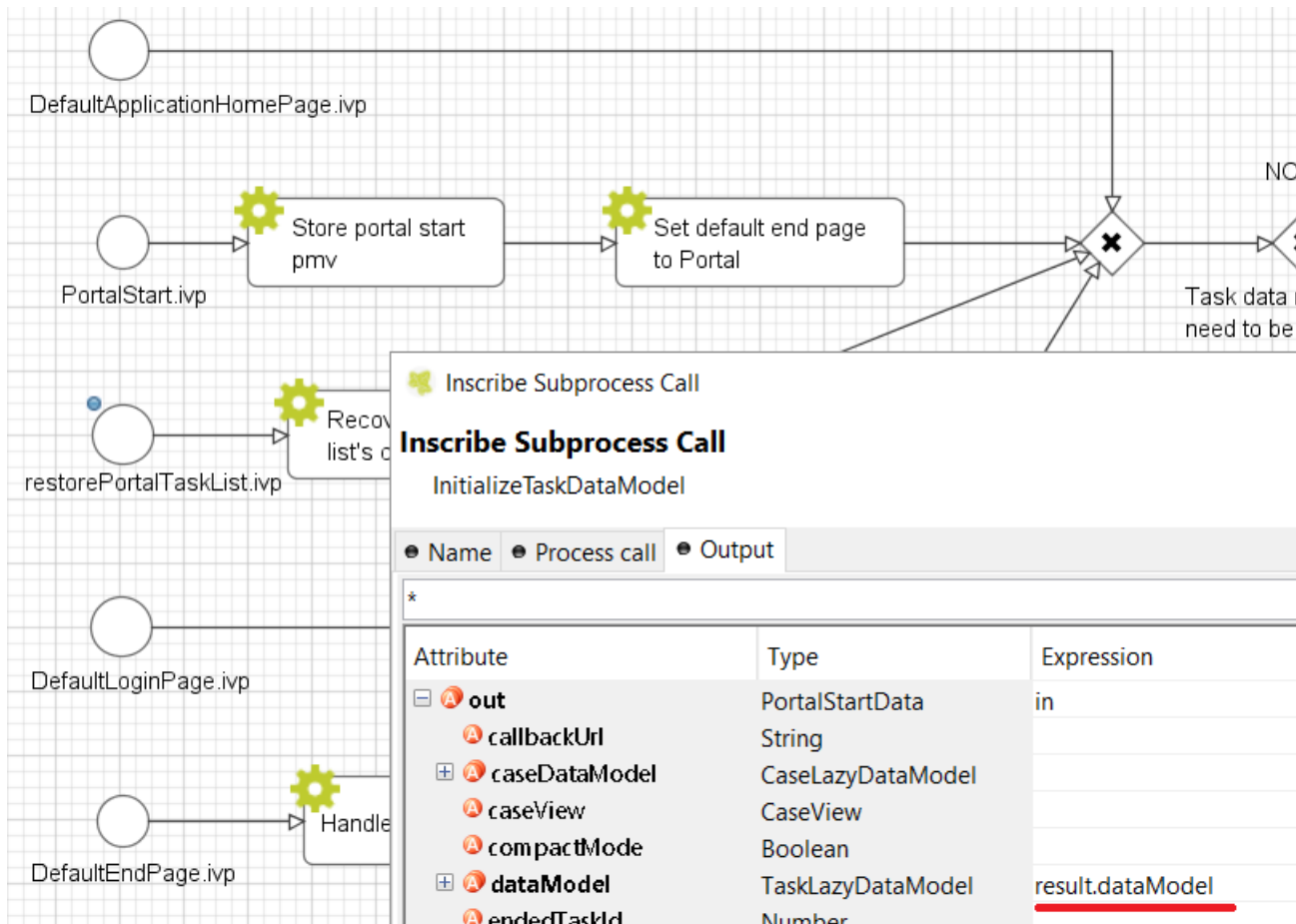
EXPIRY_CHART_LAST_DRILLDOWN_LEVEL global variable is removed. User now can use a context menu to drilldown Task By Expiry chart.

Migrate 7.0.3 to 7.0.5 (or 7.1.0)

There are some changes in PortalStart process of Portal Template. If you have customized this process in your project, copy the new PortalStart from Portal Template to your project and re-apply your customization.

We introduce new method `findStartableLinkByUserFriendlyRequestPath(String requestPath)` in `ProcessStartCollector` class. If your project has customized Default user process, use this method to generate link to your process. If user doesn't have permission to start the process, this method will return empty string.

> ⚠️ **Important**
>
> In case you already have PortalStart process in your project, delete all elements in that process and copy everything from PortalStart process of Portal Template (to prevent start link id change). `Do not delete` PortalStart proccess in your project and copy new again.
>
> Check map param result of callable process after copy to make sure it's the same as original process.
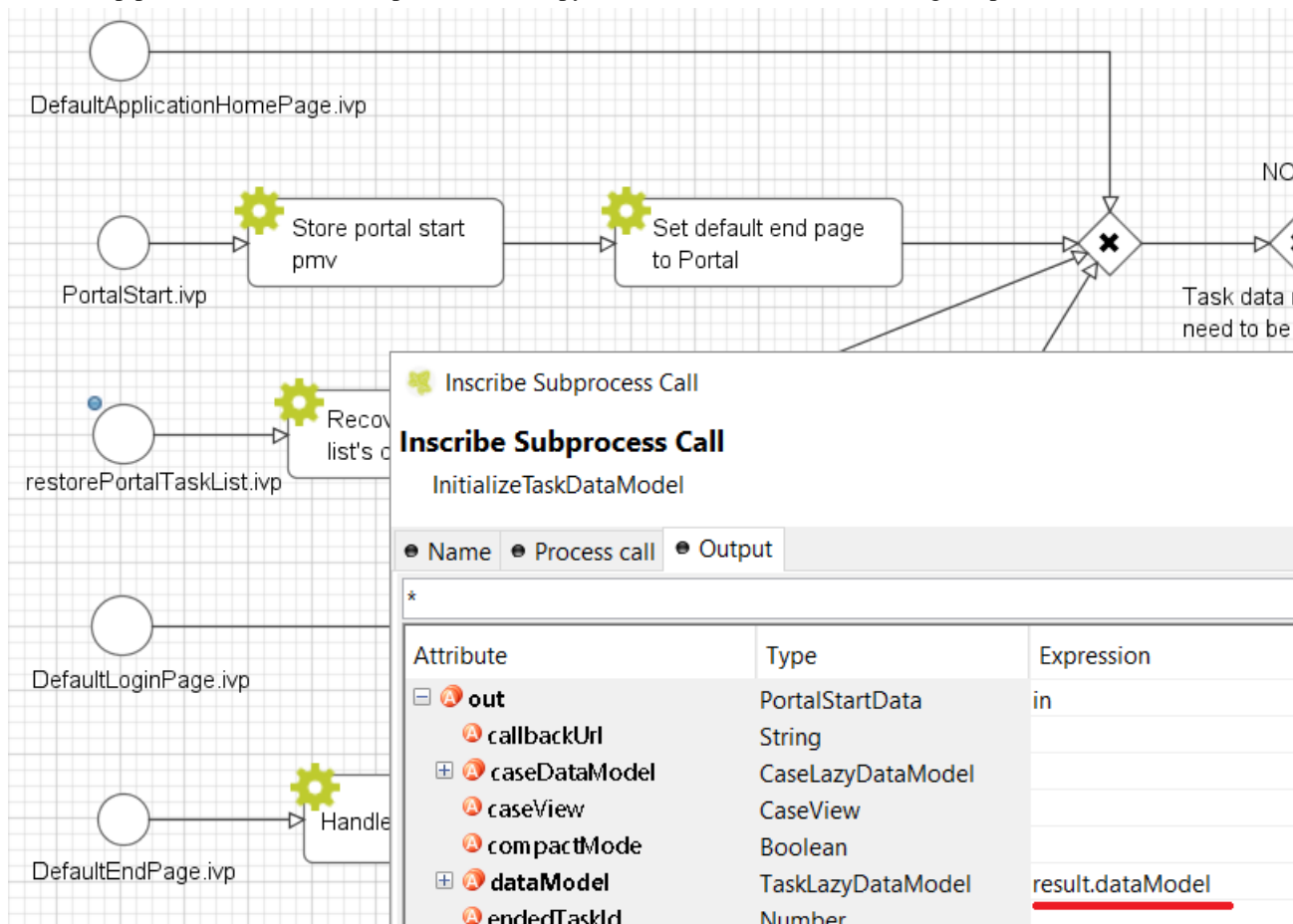
Migrate 7.0.2 to 7.0.3

If you have additional columns in your customized task widget, refer Task widget to adapt your customization in `taskHeader` section.

Migrate 7.0.1 to 7.0.2

In PortalStyle\pom.xml, update project-build-plugin version to `7.1.0` and run maven to compile CSS.

If changing password is customized, change method call `PasswordService.mod#changePassword(String,String)` to `PasswordService.mod#changePasswordService(String, String)` in this customization.

Custom fields in Portal task list can now be sorted properly. The method `extendSort()` of `TaskLazyDataModel` is changed to have a `taskQuery` parameter. If you override this method, change your code to use the new parameter instead of using the `queryCriteria` taskQuery.

Portal does not have separate full task list in the homepage anymore. It's mean that you don't have to customize the task list in `/layouts/DefaultHomePageTemplate.xhtml`. You can remove your task list customization code in `PortalHome.xhtml`.

If you have added new language to Portal by adding cms entry `/AppInfo/SupportedLanguages` in your project. Move this entry to Portal Style.

Migrate 7.0.0 to 7.0.1

Ajax error handling: By default, Portal handles all exceptions from ajax requests. Old configuration, customization of ajax error handling should be removed.

Migrate 6.x to 7.0.0

If you copy the `PortalStart` process or the `PortalHome` HTMLDialog for customizations, 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, 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`

- If you customize `TaskLazyDataModel`, remove that customized class and customize as How to override task widget's data query.

Migrate 6.4 to 6.5

- If compilation error "The type org.apache.axis2.databinding.ADBBean cannot be resolved" occurs, refer Project compilation classpath to fix.

- The relative link in default user processes starts with ivy context path instead of "pro". If there are customized default user proceses, append context path at the beginning. E.g. in Portal `6.4`, it is /pro/.../PortalStart.ivp. In Portal `6.5`, change it to /ivy/pro/.../PortalStart.ivp. You may use : `ivy.html.startref(...)` or `RequestUriFactory.createProcessStartUri(...)` to generate links.

Migrate 6.x (x < 4) to 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, refer to PortalStyle customization (logos, colors, date patterns) .

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, remove it.

Migrate 5.0 (Rothorn) to 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 7.2

- Introduced the variables to customize task priority and state colors and header bar colors

- Introduces the new page: Global search result, and supports the customization

- Overrides the DefaultApplicationHomePage.ivp, DefaultLoginPage.ivp, DefaultEndPage.ivp processes, refer to Replacement Project, check migration notes if you have the customized PortalStart.ivp process.

- Portal checks permission when upload/delete document. User needs permission `IPermission.DOCUMENT_WRITE` or `IPermission.DOCUMENT_OF_INVOLVED_CASE_WRITE` to upload/delete document.

- Support to disable upload/delete document when a case is done. This function can be configured by `HIDE_UPLOAD_DOCUMENT_FOR_DONE_CASE` setting.

- Support to configure upload file extension whitelist. Only file extensions appear in this list are allowed to upload to Portal. This function can be configured by `UPLOAD_DOCUMENT_WHITELIST_EXTENSION` setting.

- Support script checking function for upload file. You can enable/disable this function by configuring `ENABLE_SCRIPT_CHECKING_FOR_UPLOADED_DOCUMENT` setting.

Changes in 7.1

- Portal supports client side timeout: informs user when session is about to expire and auto logout when expired.

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

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

- User can add new language. Refer to Language settings for detail.

- Axon ivy express has custom end page. It can be turned off or customized.

- User can create default start process with permission check. If the user doesn't have permission to start the process, it won't appear in favorite processes. Refer to Default user process for detail.

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.

  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.

- Change password is supported to be customized. Refer to Change password process to know how to customize this feature.

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. 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
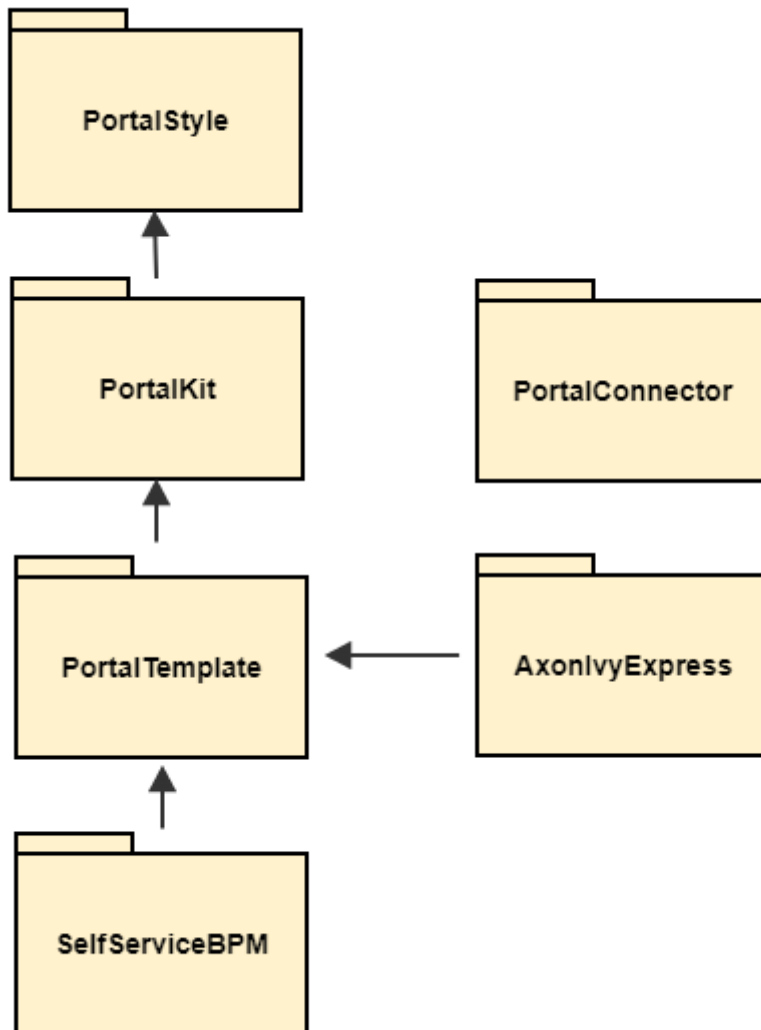
Portal kit

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.
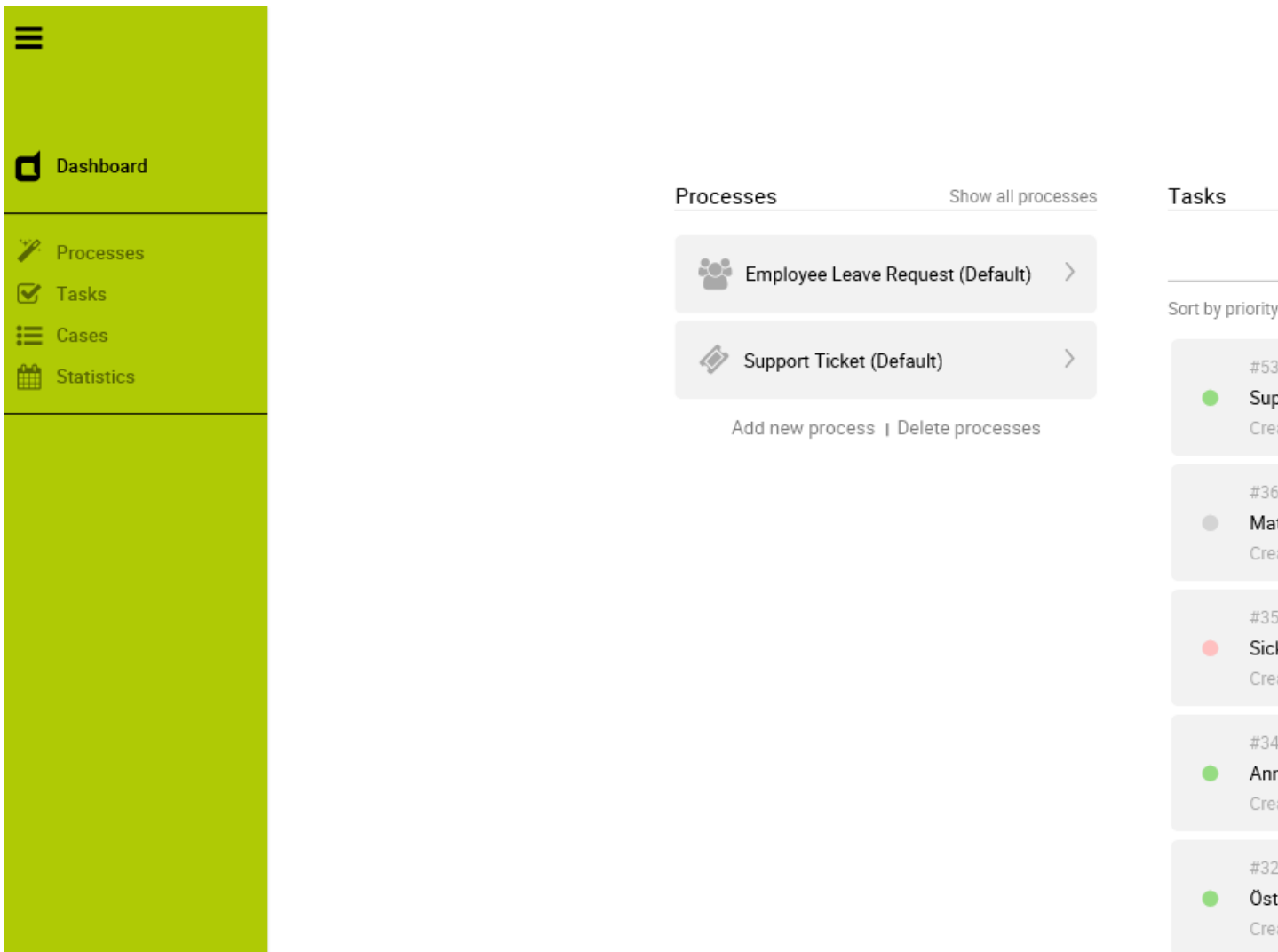
Basic template

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 template TaskTemplate.xhtml

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}" />
   ```

☑ Maternity Leave Request (#4)

Request	Case Information

Name / Description

Leave Request
Leave Request Description

Data

Creator	**Developer user**
Created	**02.07.2018 11:19**
Finished	
Process model	**InternalSupport**
PM version	**1**

Related running tasks an

- ▶ Annual Leave Requ
- ▶ Maternity Leave Re
- ▶ Sick Leave Request

Show details

Show all tasks

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}" />
```

☑ Maternity Leave Request (#4)

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. If your application has multiple tabs, use it and turn off request form by set `showTaskFormTab` to false.

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

    <ui:define name="dynamicTabs">

    <p:tab title="My first tab">

    <p:inputText id="first-name" value="#{data.firstname}"/>

    </p:tab>

    <p:tab title="My second tab">

    <p:inputText id="last-name" value="#{data.lastname}"/>

    </p:tab>

    </ui:define>
    ```

6. Overwrite contents of default tab. Use it when your application need only 1 tab.
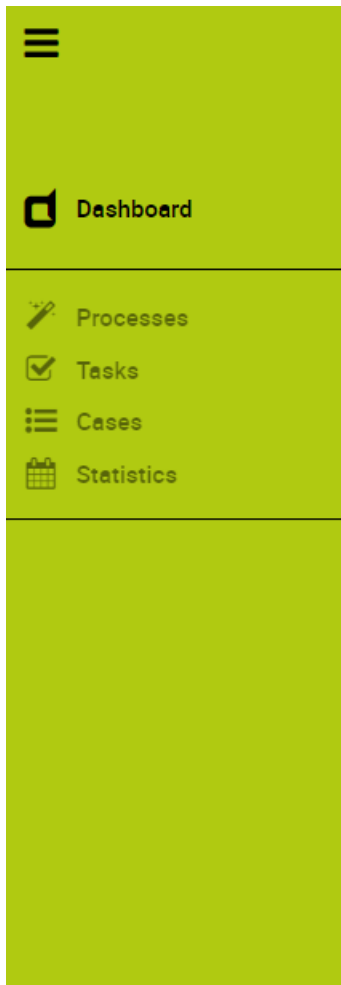
    ```
    <ui:define name="taskForm">

    <h:form>

    <p:outputLabel name="myCustomLabel" />

    ...

    </h:form>

    </ui:define>
    ```

7. Set visible/invisible for default tab case information. Set following variables as `true` if you want to visible and vice versa.

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

8. Inserts left buttons and right buttons which stay at the bottom of the page. It is optional. You can use it to define your action button. Consider using `partialSubmit` to submit your data im tabs.

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

    <p:commandButton         value="Save"         actionListener="#{logic.save}"
    partialSubmit="true"  process="first-name  last-name"  update="first-name  last-
    name" />

    </ui:define>

    <ui:define name="rightButtons">

    <p:commandButton         value="Cancel"         actionListener="#{logic.cancel}"
    immediate="true" />

    </ui:define>
    ```

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.

Tasks >

Tasks (38)

Filter ▾   Save filter 💾

Priority	Name / Description
⚪	**Maternity Leave Request** Maternity Leave Request Description
🔴	**Sick Leave Request** Sick Leave Request Description
🟢	**Annual Leave Request** Annual Leave Request Description
🟢	**Categoried Leave Request** No description

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.

Cases >

Cases (16)

Filter ▼   Save filter 💾

Name / Description

Leave Request
Leave Request Description

Leave Request
Leave Request Description

Data

Creator	**Developer user**
Created	**15.10.2018 15:14**
Finished	
Case Category	**LeaveRequest**

Related running tasks

▶ Annual Leave Re
▶ Maternity Leave
▶ Sick Leave Requ

Show details   Show all tasks

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, Portal handles all exceptions from ajax requests.

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 show details button.

Stacktrace on error messages can be showed/hid depend on ivy system property `Errors.ShowDetailsToEndUser`.

Note

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

Result

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
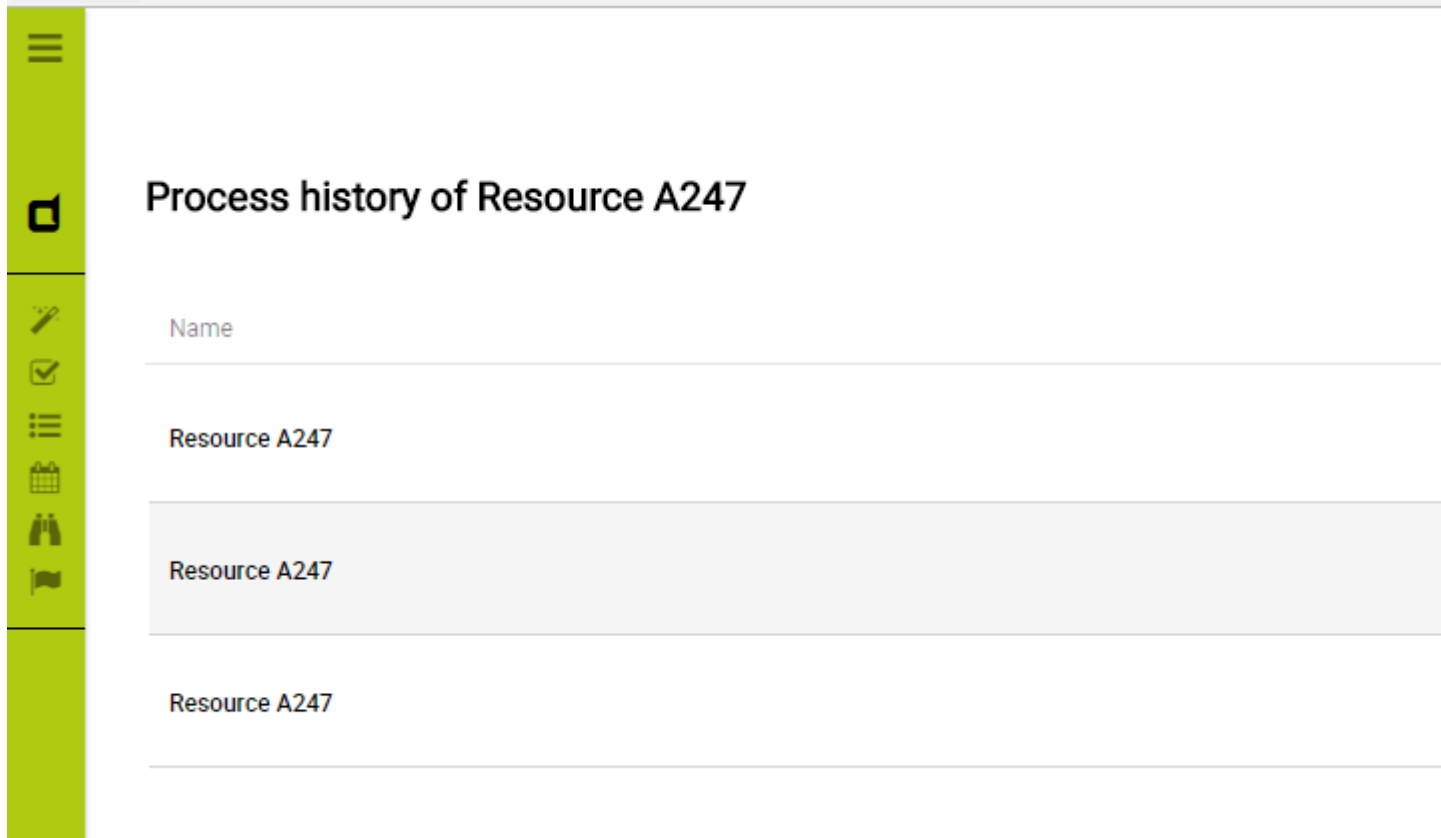
Additional Components

Process history

Introduction

This component is a lazy loading list which displays all business cases of a business entity in your application. You can include this component everywhere:

In a page

In a dialog

41

Process history of Resource A247

Open process history dialog

How to use

First you need to link the cases to the business entity. Call the subprocess `SetBusinessEntityId` in the process which need to be linked and input an identifier unique to your business entity. The subprocess will set the id to the additional property "CASE_BUSINESS_ENTITY_PROPERTY" of the business case.
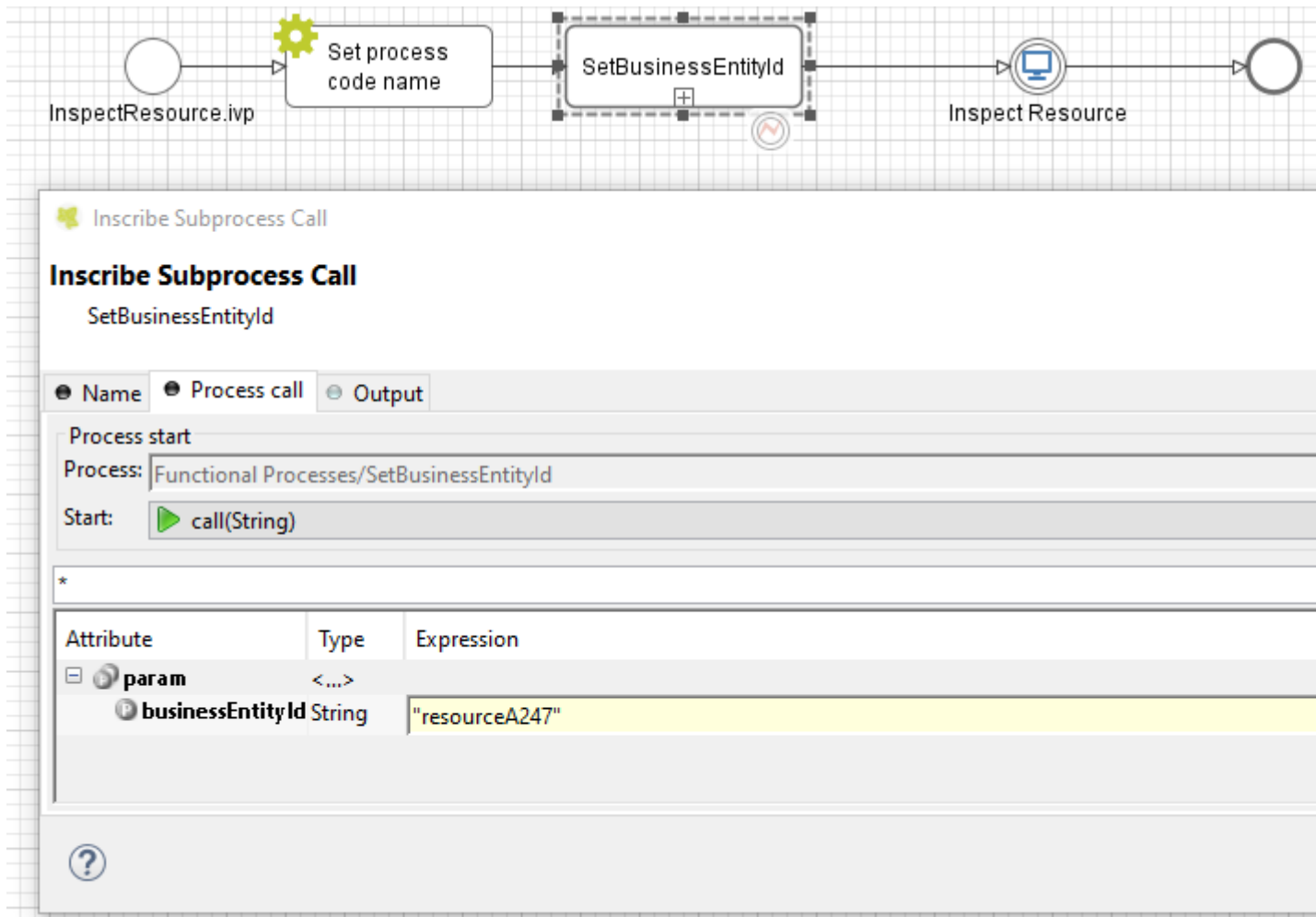
Include the process history component into your page:

```
<ic:ch.ivy.addon.portal.component.ProcessHistory businessEntityId="resourceA247" >
```

The value of the attribute `businessEntityId` must match the id input into the subprocess in the first step.

By default the component will load 20 cases at a time. You can change this by setting the attribute `chunkSize` to the number you want . You should use this attribute alongside with the attribute `scrollHeight` to configure the scroll bar of the list.

> **Note**
>
> If you use this component in a dialog, you must run this script `processHistory.setup();` when the dialog is shown. For example:
>
> ```
> <p:dialog  widgetVar="process-history-dialog"  id="process-history-dialog"
> width="800"  height="500"  header="Process  history  of  Resource  A247"
> onShow="processHistory.setup();">
>
> <ic:ch.ivy.addon.portal.component.ProcessHistory
> businessEntityId="resourceA247" chunkSize="6" scrollHeight="400" />
>
> </p:dialog>
> ```

> **⚠ Important**
>
> If your process has a Trigger component or sends a signal to start another process with the option "Attach to Business Case that triggered this process" selected, the current case of the process will become a technical case and will not be loaded into the process history list. In this case You need to call the `SetBusinessEntityId` subprocess after the first Trigger or signal sending step.

Task Analysis

Introduction

Task Analysis component of Portal provides features for users to analyze not only tasks but also cases. These features are:

1. Set of filters for both tasks and cases which allow user to filter and to find tasks, cases more better. More, user can create and manage their own filter set for future usage.

2. Dynamic result table with lots of information for both task and case.

3. Support export result as Excel files (currently we only support .xlsx extension).

How to use

Task Analysis component is integrated into Statistic widget. You can use this component directly when open Statistic widget. If you want to use this component, you only have to redirect to Task Analysis component with following code:

```
import javax.faces.context.FacesContext;

String    taskAnalysisUrl    =    ivy.html.startref("Start    Processes/TaskAnalysis/
start.ivp");

FacesContext.getCurrentInstance().getExternalContext().redirect(taskAnalysisUrl);
```

Process Chain

Introduction

Process Chain component of Portal provides features for users to know status of all steps in a process: the step's working, these steps are done, these steps is not done. These features are:

1. Support to display all working steps or display only helpful steps as begin, last, current, previous current, next current steps.

2. Support to change the sharp of process chain: circle or line.

3. Support to change direction of process chain: horizontal or vertical.
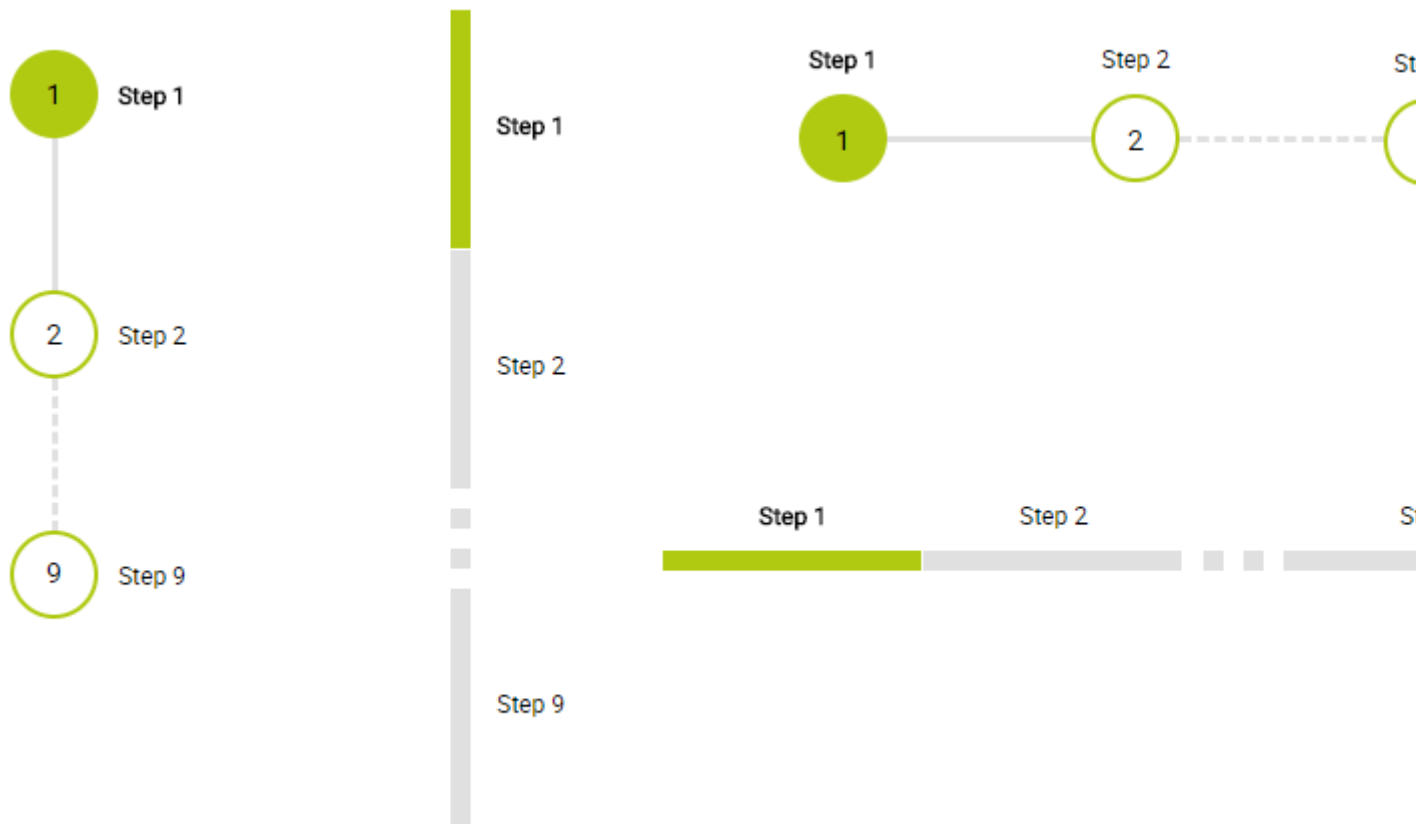
How to use

Process Chain component can be integrated in any widget by including this component into a page. In order to use this component in a page, include this component to this page with following code:

```
<ic:ch.ivy.addon.portalkit.singleapp.process.ProcessChain id="process-chain-circle-
horizontal"

componentId="component-circle-horizontal" shape="CIRCLE" direction="HORIZONTAL"

isShowAllSteps="FALSE"                      actualStepIndex="#{data.actualCurrentIndex}"
steps="#{data.steps}" />
```

1. Must to set value for `actualStepIndex` parameter. This is current step index.

2. Must to set value for `steps` parameter. This is list of working steps.

3. Can change `shape` parameter to `CIRCLE` or `LINE` based on the requirement. Default value of this is `CIRCLE`.

4. Can change `direction` parameter to `HORIZONTAL` or `VERTICAL` based on the requirement. Default value of this is `HORIZONTAL`.

5. Can change `isShowAllSteps` parameter to `TRUE` or `FALSE` based on the requirement. Default value of this is `FALSE`.

Change Last Drilldown Level Of Task By Expiry Chart
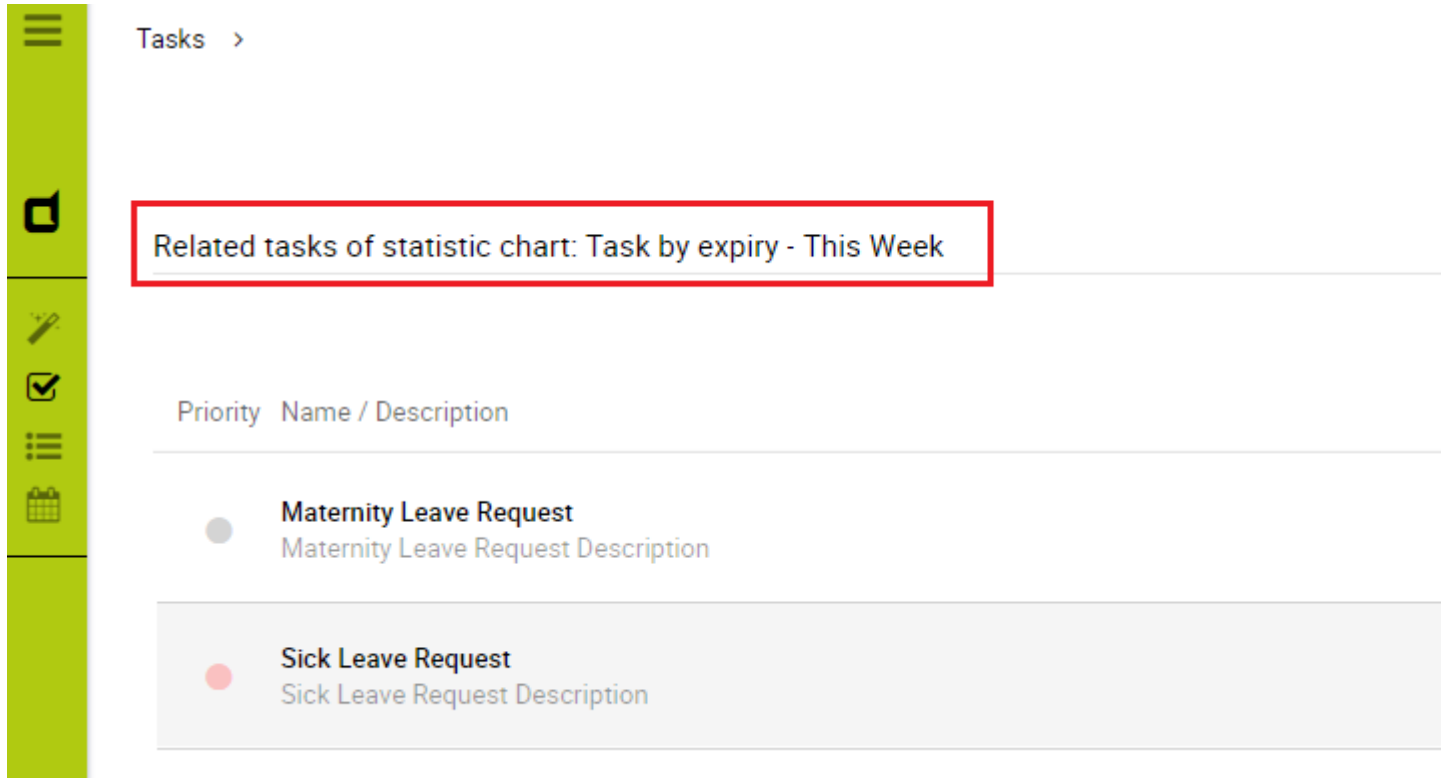
Task by expiry chart

For users who want to change last drilldown level of Task by expiry chart

By default, the last drilldown level of a expiry chart is HOUR. It means that, to open a related tasks list of this chart, users must navigate from YEAR -> MONTH -> WEEK -> DAY -> HOUR, clicking on a column of HOUR chart to open the tasks.

To change this last drilldown level, users can set the value of `EXPIRY_CHART_LAST_DRILLDOWN_LEVEL` in `Global settings`.

For example, to navigate to task list immediately when clicking on a week column, set `EXPIRY_CHART_LAST_DRILLDOWN_LEVEL = WEEK`:

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`.
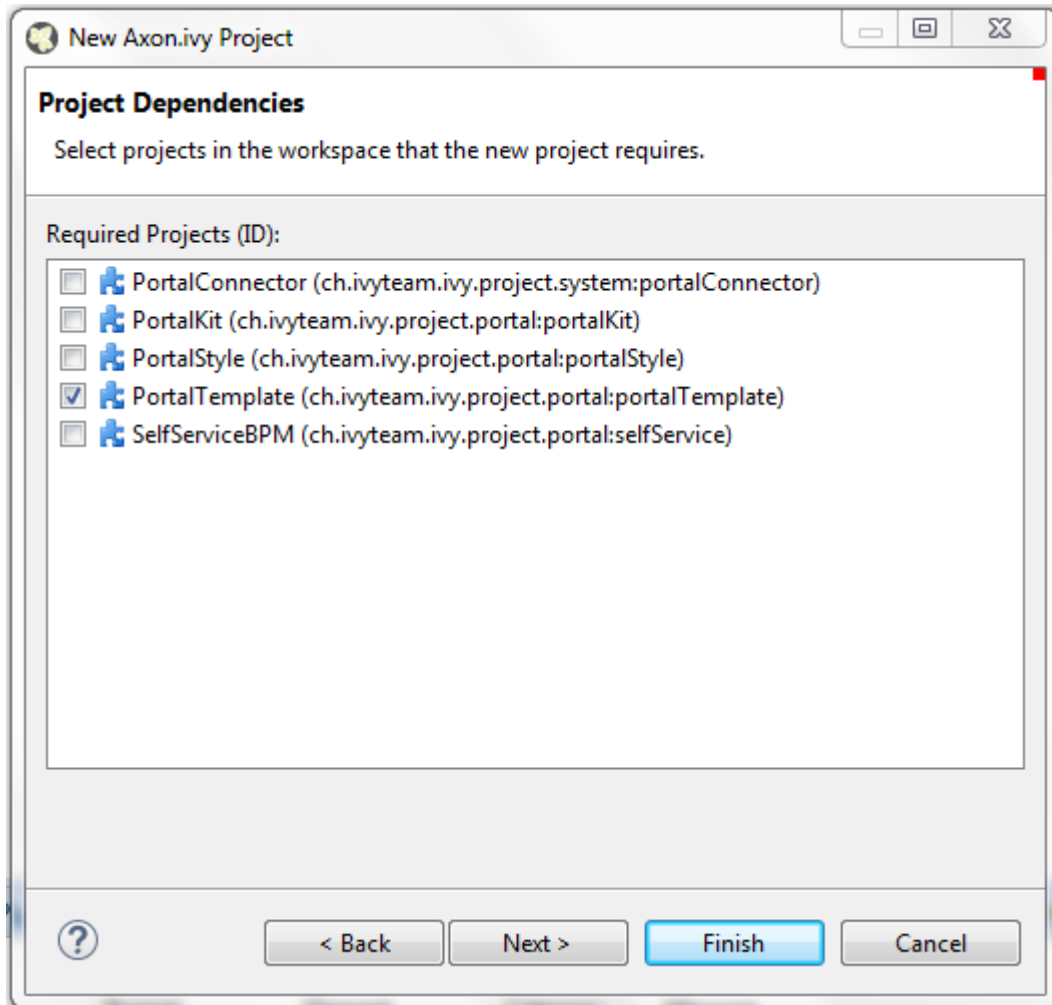
   Project structure
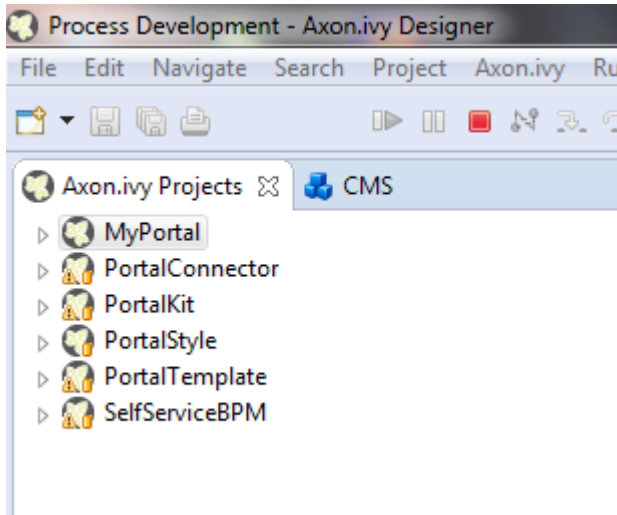
2. Create new home page

    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.

> **Important**
> Portal uses some template files in the folder `"webContent/layouts"` in Portal Template project. Do not create files with the same path and name in your project as they can override the Portal files.

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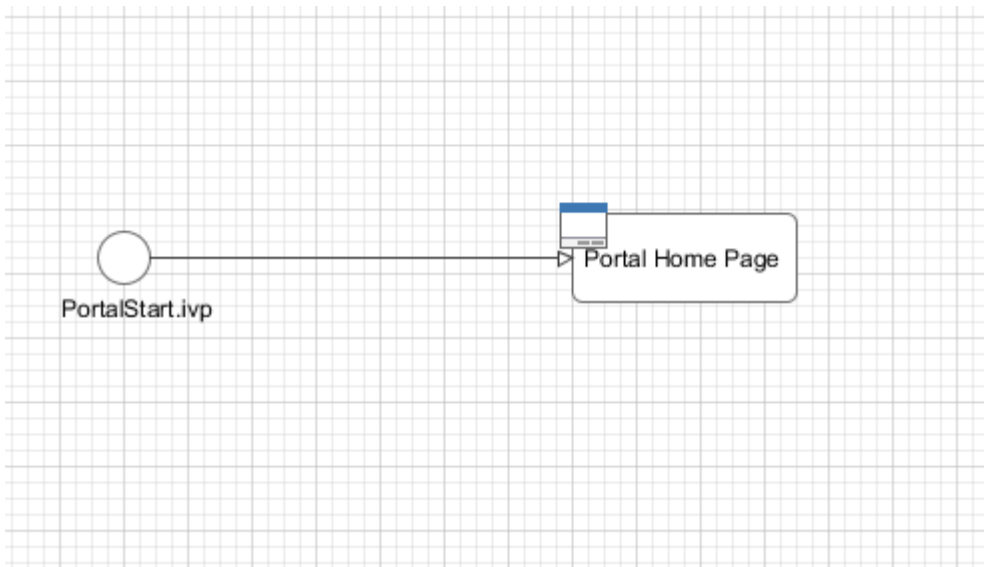
    c. Run your application, start your newly created process and see result.
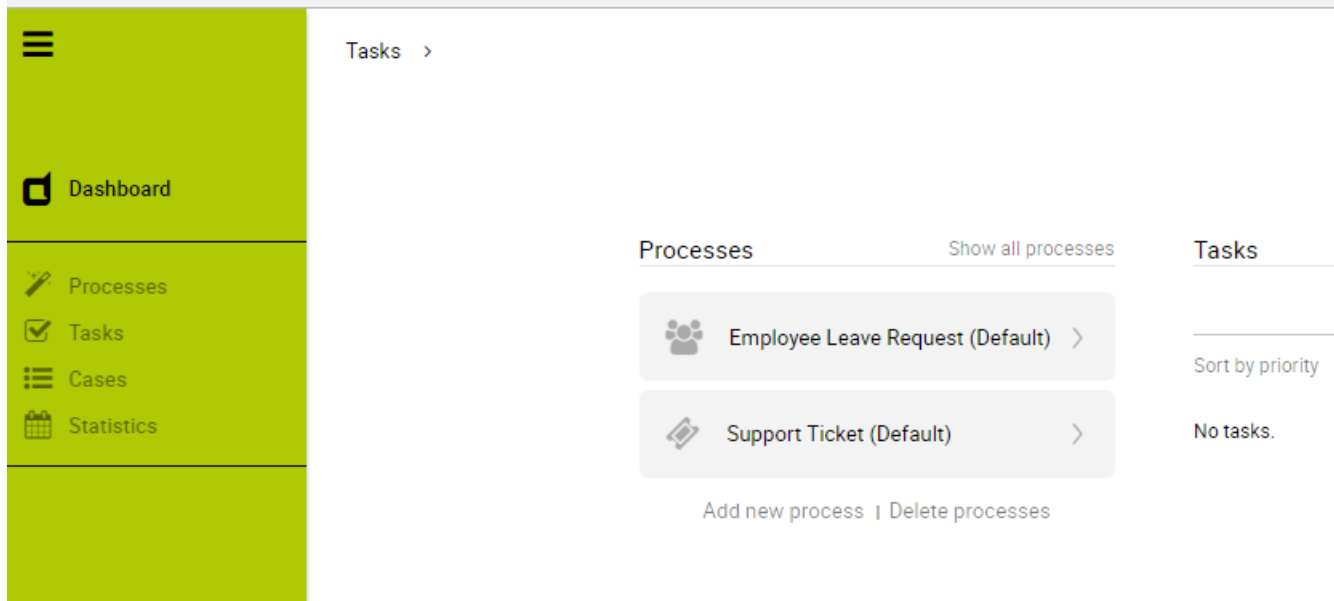
Tip

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.
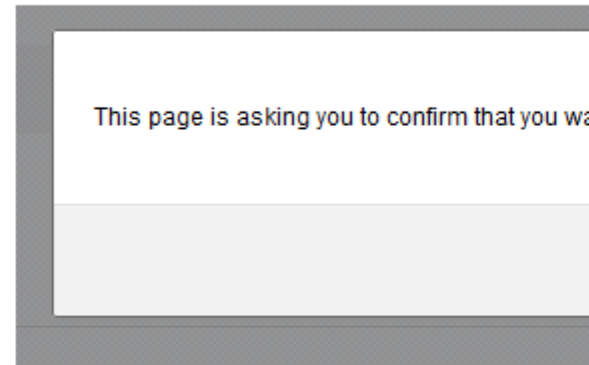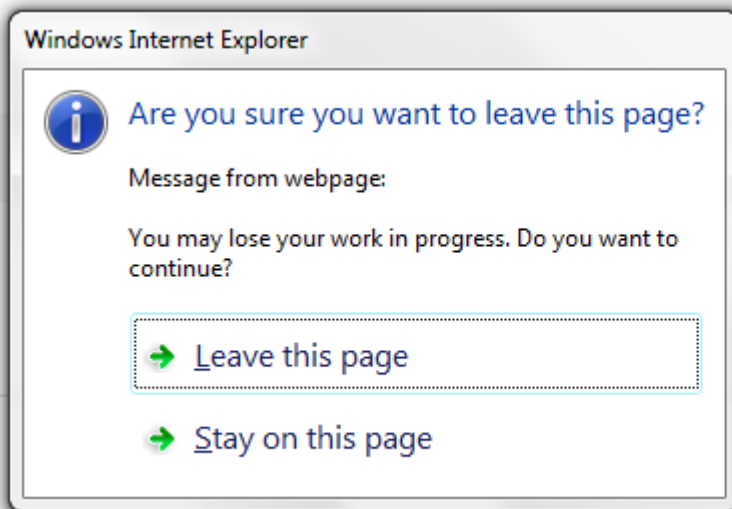
Tip

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;`

PortalStyle customization (logos, colors, date patterns)

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.

- Override the variables: @login-logo-height, @home-logo-height in `customization.less` to scale your logos.

Change Portal styles

Portal applies LESS framework to support you in customizing the styles of Portal. There are 4 files: `theme.less` (shouldn't be modified), `variables.less` (shouldn't be modified), `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.

- `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` , `@process-chain-menu-color` , etc.) 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-opacity-transition-duration` : it is the opacity transition duration of the text on application menu.
>
> - `@process-chain-menu-color` : it is the color of Process Chain component.
>
> - `@action-button-color` : it is the color of action button in Portal e.g: close, add, next ...
>
> - `@action-button-border-radius` : it sets shape of action button, it's rectangle when set @action-button-border-radius:0px;
>
> - `@sidebar-dimension-transition-duration` : it is the expanding/collapsing transition duration of the application menu.

> **Important**
>
> - Do not change `font-family` property values.
>
> - Limitation: the task priority color customization hasn't changed the task priority colors in statistic.

There is additional button type allows developer to configure its color and shape. Use it when you need different button types in your own project. How to: use style class `context-button` for it. For example : `<p:commandButton value="My button" styleClass="context-button"` > There are 2 variables allow developers to change color and shape

- `@context-button-color` : it is the color of context button

- `@context-button-border-radius` : it sets shape of action button, it's rectangle when set @context-button-border-radius:0px;

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.

Change date time pattern

You can change date pattern by modifying CMS in PortalStyle project: `PortalStyle/patterns/datePattern` and `PortalStyle/patterns/dateTimePattern` .

Login

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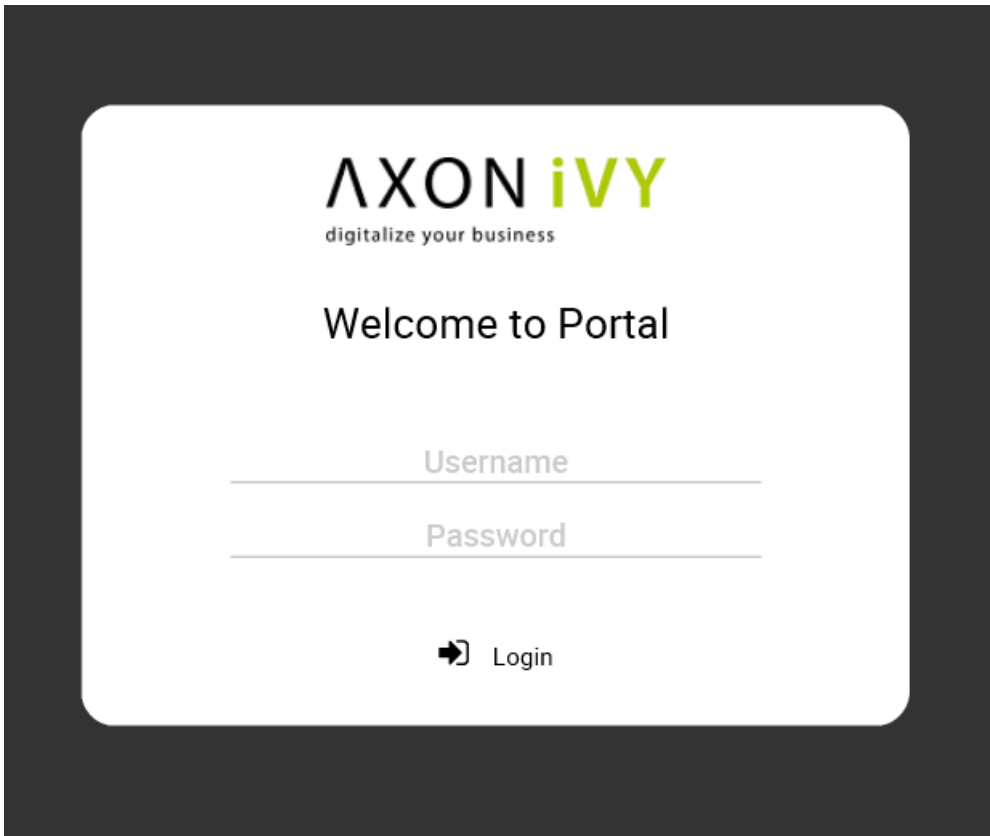
To replace default login page, extends existing templates with `ui:define name="login"` to define your new login component like below

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

<ui:define name="login">

<ic:internaltest.ui.YourOwnLoginComponent />

</ui:define>

</ui:composition>
```

Menu

Introduction

By default Portal main menu has 4 items: Processes, Tasks, Cases and Statistics. You can remove these items or add your own items.

Customization

Create a callable sub process in your project with the `loadSubMenuItems()` signature, make sure this signature is unique in your application and follow the hints.
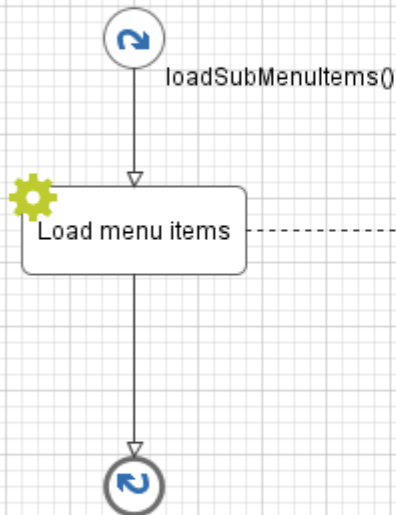
```
TO REMOVE A DEFAULT SUB MENU ITEM:
Remove  one of these lines:
in.subMenuItems.add(new ProcessSubMenuItem());
in.subMenuItems.add(new TaskSubMenuItem());
in.subMenuItems.add(new CaseSubMenuItem());
in.subMenuItems.add(new DashboardSubMenuItem());

TO CREATE A SUB MENU ITEM:

SubMenuItem subMenuItem = new SubMenuItem();
subMenuItem.setMenuKind(MenuKind.CUSTOM);
subMenuItem.setIcon(<SUB_MENU_ICON>);
subMenuItem.setLabel(<SUB_MENU_LABEL>);
subMenuItem.setLink(<SUB_MENU_LINK>);

//add file names of pages where the menu item will be highlighted e.g selfService
selfService.getViews().add(<PAGE_TO_BE_HIGHLIGHT>);

in.subMenuItems.add(subMenuItem);

OUT: subMenuItems: List<SubMenuItem>

HINT: how to build a menu link
Axon.Ivy link
- Absolute path: ivy.html.startref(...)
- Relative path: RequestUriFactory.createProcessStartUri(...)
External link:
- www.yourexternallink.com
- http://www.yourexternallink.com
```

loadSubMenuItems()

Load menu items
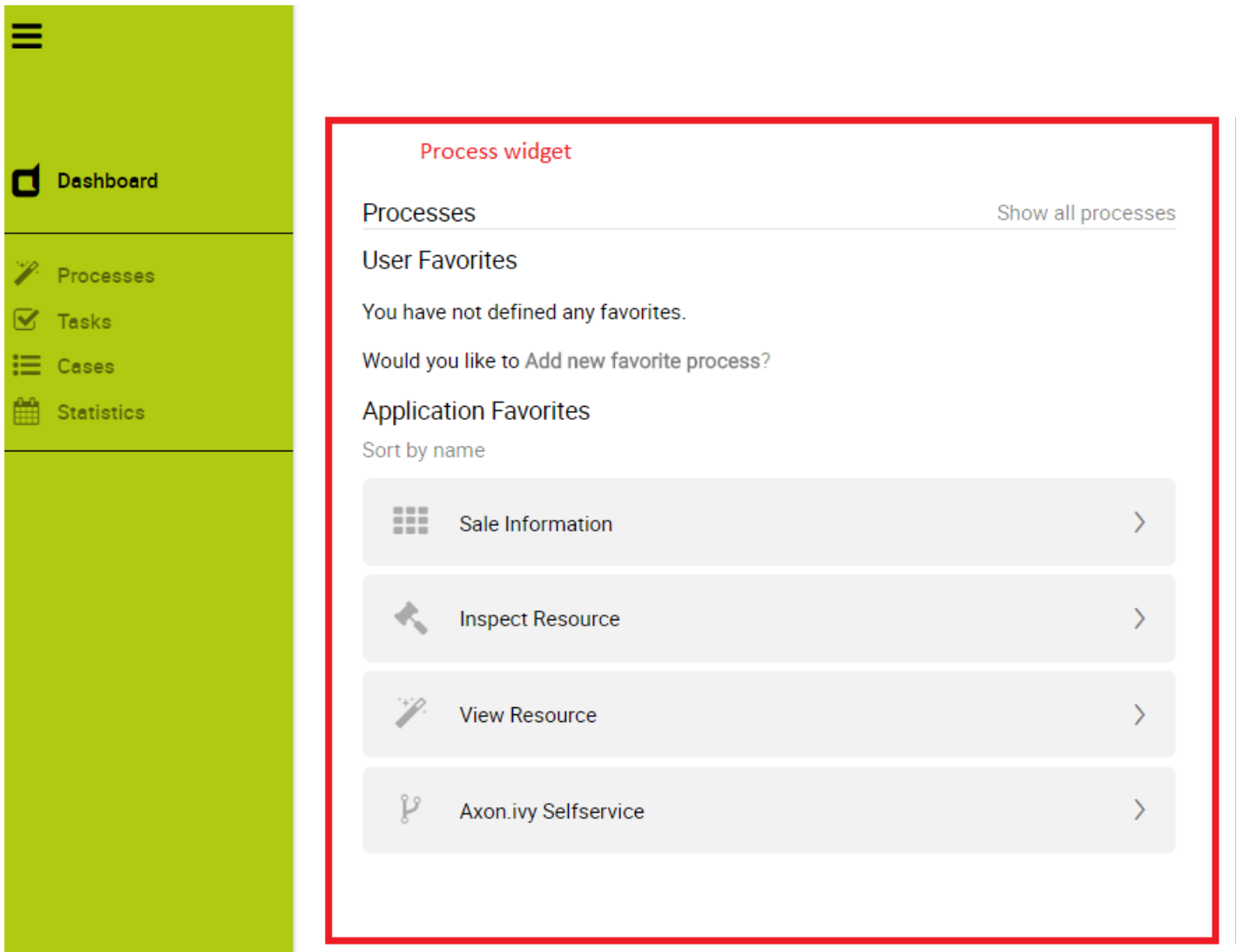
Portal home

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.

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/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

Refer to `PortalExamples` project for examples

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

2. To customize task widget, you must customize Portal Home first. Refer to Customize Portal home to set new home page.

3. Copy the `PortalStart` process from `PortalTemplate` to your project. Point PortalHome element to your custom home page in previous step. This process is new home page and administrator should register this link by Portal's Admin Settings.

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

   - 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`.

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

Task header

Refer to the `taskListHeader` and `taskHeader` sections in `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)

- Default taskList supports user to config display/hide column

AXON iVY
digitalize your business

	Responsible	Ta
	Everybody	79
⌄		
	Everybody	78
⌄		
	Everybody	78
⌄		
	Everybody	79
⌄		
	Everybody	79
⌄		

- In case you has new columns, override method `getDefaultColumns` of the extended class from TaskLazyDataModel to display checkboxes in Config columns panel and display/hide sortFields (see the methods' Javadoc comments)

- To add cms for checkboxes's label, add new entries to folder `/ch.ivy.addon.portalkit.ui.jsf/taskList/ defaultColumns/` in `PortalStyle` or override method `getColumnLabel`(see the methods' Javadoc comments)

- In `taskListHeader` section, use `TaskColumnHeader` component

- In `taskHeader` section, use `TaskCustomField` component for each additional columns. This component will handle display/hide new columns on task list.

  Currently, TaskCustomField only supports text field. If you want to create your own component, remember to add rendered="#{taskView.dataModel.isSelectedColumn('YOUR_CUSTOM_COLUMN')}"

  For example: Show custom field `customer name` which stored in `task.customVarCharField5`

  ```
  <ic:ch.ivy.addon.portalkit.component.task.column.TaskCustomField  id="customer-
  name-component"      componentId="customer-name"      column="customVarCharField5"
  dataModel="#{taskView.dataModel}" labelValue="#{task.customVarCharField5}" />
  ```

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

- 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 `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 `PortalTasks.xhtml` of PortalTemplate.

- In order to introduce new filter, create a new java class extends TaskFilter and override its methods (see javadoc comments)

Tasks >

Tasks (4)

label() value()

Filter ▼   Description: "Request" ▼ ⊗   Expiry (from/to): All ▼ ⊗

☐ Created (from/to)
☑ Description label()
☑ Expiry (from/to)
☐ Priority
☐ Responsible
☐ State
☑ In progress

from

to

Apply

Maternity Leave Request
Maternity Leave Request Description

Sick Leave Request
Sick Leave Request Description
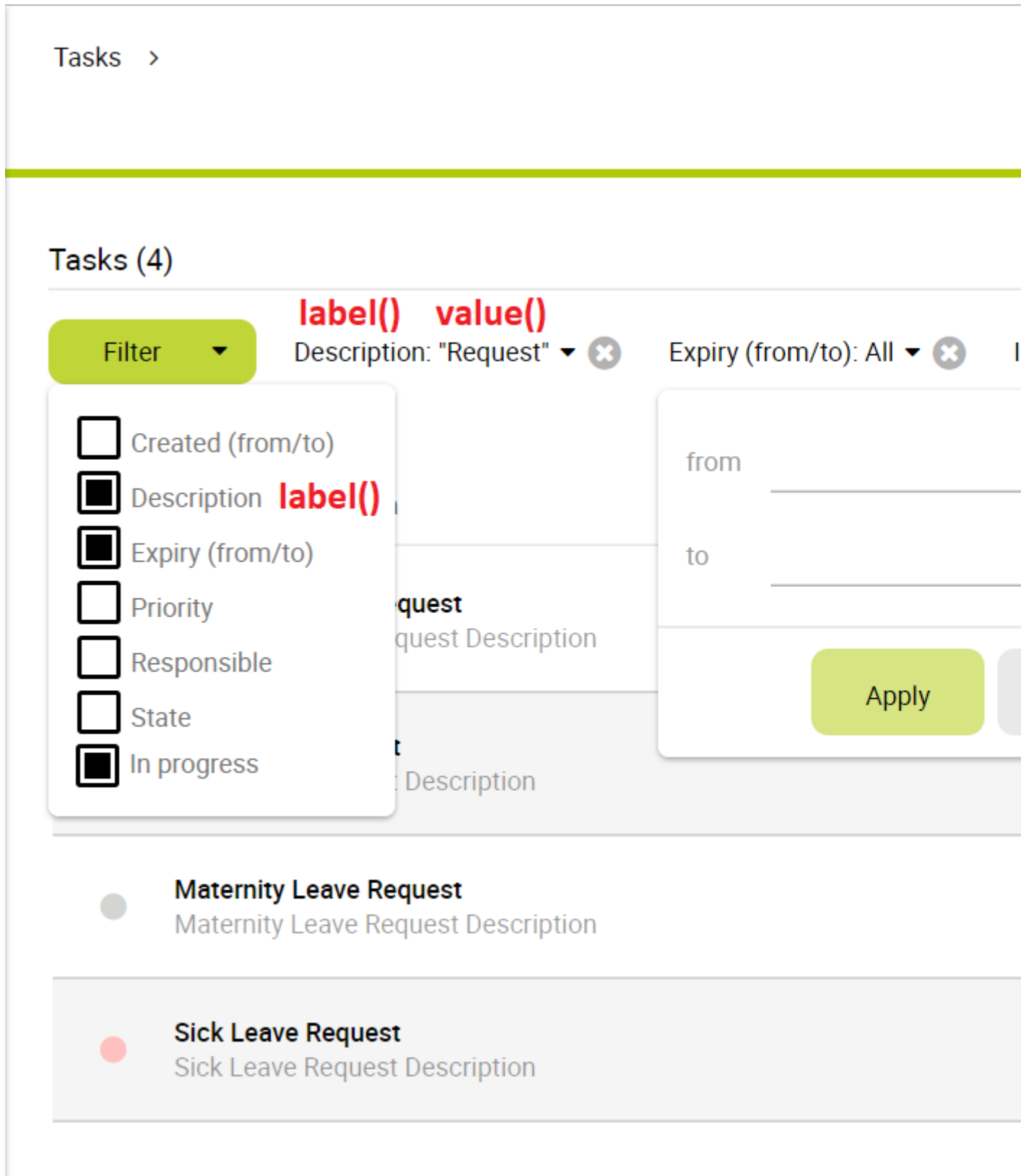
- Introduce a java class extends TaskFilterContainer. This filter container contains your filters, you can reuse default filters, refer to `DefaultTaskFilterContainer.java`

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

- **Advanced usage:** Portal supports storing/restoring filters. Your filter class (extends `TaskFilter`) is stored in business data. Properties stored user input values should be persisted, properties controlled logic should not be persisted to reduce persisted data size in business data. Use annotation `@JsonIgnore` to exclude properties. By default, Portal takes care storing/restoring filters. If you want to customize storing/restoring filter data, do it in your data model class (extends `TaskLazyDataModel` class).

  By default, filters are stored/restored in process model level. You can change this by setting the ui:param `filterGroupId` in `PortalTasks.xhtml` to a new Long value.

> **Tip**
>
> If you have multiple case lists in your project, you may want to set `filterGroupId` to an unique identifier for each of your `PortalTasks.xhtml across your projects`

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.

- If you want to apply a query for only Home page task list, not for Full mode task list, use attribute `isQueryForHomePage` in `BuildTaskJsonQuery` callable process to specify the query for Home page task list, e.g.

```
if (in.isQueryForHomePage) { // in home page
 in.jsonQuery = TaskQuery.create().where().activatorUserId().isNotNull().asJson();
}
```

- 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 wo
out.taskView = TaskView.create().dataModel(dataModel).showHeaderToolbar(false).createNew
```

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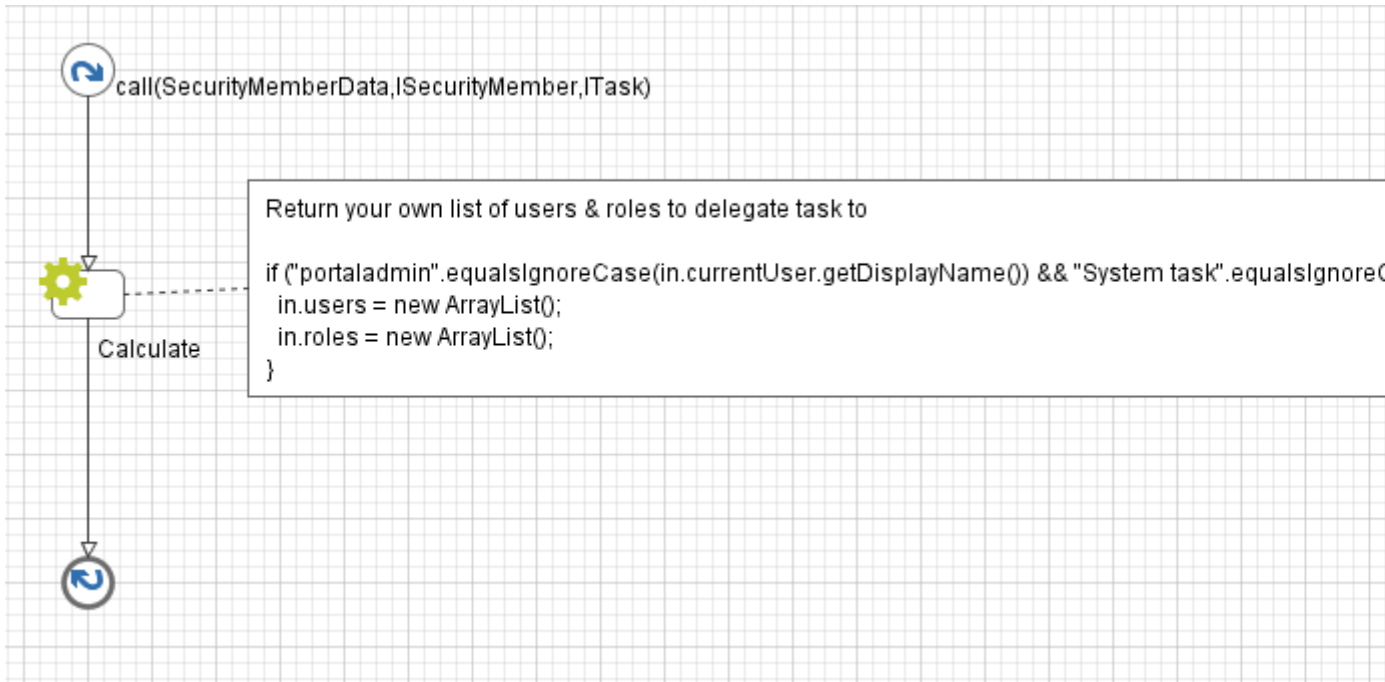
3. The callable subprocess data contains the current user `in.currentUser` and the current task to be delegated `in.task`. The lists `in.users` and `in.roles` contain all possible users and roles that the task can be delegated to. Modify those two to have your own delegate list.

Case widget

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

1. UI : CaseListHeader, CaseHeader, CaseBody and CaseFilter

2. Data query : display the cases as you want by modifying data query

> ⚠️ **Important**
>
> 1. Case header customization does not support responsive design, smaller resolutions (iPad, etc.)
>
> 2. Additional properties cannot be displayed right now as a column
>
> 3. Case header's buttons cannot be modified (they stay where they are)

How to override case widget's UI

Refer to `PortalExamples` project for examples

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

2. To customize case widget, you must customize Portal Home first. Refer to Customize Portal home to set new home page.

3. Copy the `PortalStart` process from `PortalTemplate` to your project. Point PortalHome element to your custom home page in previous step. This process is new home page and administrator should register this link by Portal's Admin Settings.

4.   Override Case widget in: CaseList page, Case Search result.

   • Introduce a new HTMLDialog which uses template `/layouts/PortalCasesTemplate.xhtml` (refer to Responsiveness to override responsiveness). You can take a look at `PortalCases.xhtml` to see how to customize it.

   > **Tip**
   >
   > Highly recommend to copy the `PortalCases` HTMLDialog in PortalTemplate and change the copied one's view.

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

5.   After previous steps, you can override CaseHeader and CaseListHeader and CaseBody and CaseFilter

Case List Header and Case Header

Refer to the `caseListHeader` and `caseHeader` sections in `PortalCases.xhtml` of PortalTemplate. In case your case widget has new columns, you should override CaseLazyDataModel to make the sort function of these columns work:

• Introduce a java class extends CaseLazyDataModel

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

• Default caseList supports user to config display/hide column

- In case you have new columns, override method `getDefaultColumns` of the extended class from CaseLazyDataModel to display checkboxes in Config columns panel and display/hide sortFields (see the methods' Javadoc comments)

- To add cms for checkboxes's label, add new entries to folder `/ch.ivy.addon.portalkit.ui.jsf/caseList/ defaultColumns/` in `PortalStyle` or override method `getColumnLabel`(see the methods' Javadoc comments)

- In `caseListHeader` section, use `CaseColumnHeader` component

- In `caseHeader` section, use `CaseCustomField` component for each additional column. This component will handle display/hide new columns on case list.

Currently, CaseCustomField only supports text field. If you want to create your own component, remember to add `rendered="#{caseView.dataModel.isSelectedColumn('YOUR_CUSTOM_COLUMN')}"`

For example: Show custom field `customer name` which stored in `case.customVarCharField1`

```
<ic:ch.ivy.addon.portalkit.component.cases.column.CaseCustomField    id="case-
customer-name-component"        panelGroupId="customVarCharField1-column-case-
header-panel"        componentId="customVarCharField1-column-case-header-text"
```

```
column="customVarCharField1"                    dataModel="#{caseView.dataModel}"
labelValue="#{case.customVarCharField1}" />
```

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

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

Case body

Refer to the `caseBody` section in `PortalCases.xhtml` of PortalTemplate.

If you need to apply the responsiveness behavior of Portal for case 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: 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.

Case filter

- Refer to the `caseFilter` section in `PortalCases.xhtml` of PortalTemplate.

- In order to introduce new filter, create a new java class extends CaseFilter and override its methods (see javadoc comments)

- Introduce a java class extends CaseFilterContainer. This filter container contains your filters, you can reuse default filters, refer to `DefaultCaseFilterContainer.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 CaseLazyDataModel. Override the `initFilterContainer` method and initialize filter container (see javadoc comments)

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

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

- Portal supports storing/restoring filters. Your filter class (extends `CaseFilter`) is stored in business data. Properties stored user input values should be persisted, properties controlled logic should not be persisted to reduce persisted data size in business data. Use annotation `@JsonIgnore` to exclude properties. By default, Portal takes care storing/restoring filters. If you want to customize storing/restoring filter data, do it in your data model class (extends `CaseLazyDataModel` class).

- By default, filters are stored/restored in process model level. You can change this by setting the ui:param `filterGroupId` in `PortalCases.xhtml` to a new Long value.

> **Tip**
>
> If you have multiple case lists in your project, you may want to set `filterGroupId` to an unique identifier for each of your `PortalCases.xhtml` across your projects

How to override case widget's data query

Override the `BuildCaseJsonQuery` callable process of PortalKit and build your own query to effect the data of case widget.

Apply the following steps in case you would like to provide data for case list after navigating to case list from your page:

- Use the `OpenPortalCases` callable process with the `CaseView` parameter. It is used to define which information are displayed in CaseWidget.

- Refer to CaseView, CaseSearchCriteria, CaseQueryCriteria to build your CaseView

```
CaseLazyDataModel dataModel = new CaseLazyDataModel();
dataModel.getQueryCriteria().setCaseQuery(YOUR_CASE_QUERY); // Set your CaseQuery
dataModel.getSearchCriteria().setInvolvedUsername(true); // Use this code line if you woul
out.caseView = CaseView.create().dataModel(dataModel).withTitle("My Cases").buildNewView()
```

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 an override which overrides sub process `createDefaultUserProcess()` in Portal Kit. This sub process return a list of user processes. Follow instruction to create default processes.

createDefaultUserProcesses()

create default
processes
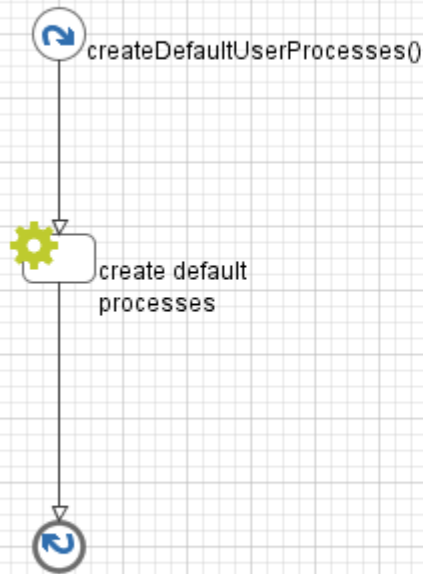
HOW TO CREATE A DEFAULT USER PROCESS:

UserProcess userProcess = new UserProcess();
userProcess.setLink(<PROCESS_LINK>); //Absolute path or relative path starts with:
userProcess.setProcessName(<PROCESS_NAME>);
userProcess.setIcon(<PROCESS_ICON>); //Icons in Font Awesome
userProcess.setIndex(1); // Set the index of the process on the default process list

in.defaultUserProcesses.add(userProcess);

OUT: defaultUserProcesses: List<UserProcess>

HINT: how to build a process url
- Absolute path: ivy.html.startref(...)
- Relative path: RequestUriFactory.createProcessStartUri(...)
- The default processes are sorted by their index attribute. If this attribute is not set, th
- We provide method to get startable link by UserFriendlyRequestPath (If user don't ha
ProcessStartCollector.findStartableLinkByUserFriendlyRequestPath(...)
Example:
ProcessStartCollector collector = new ProcessStartCollector(ivy.request.getApplicatio
String newEmployeeLink = collector.findStartableLinkByUserFriendlyRequestPath("S

Tip

We provide the method to generate link from UserFriendlyRequestPath in `ProcessStartCollector` class: `findStartableLinkByUserFriendlyRequestPath(String requestPath)`. This method will return startable link if user has permission to start the process, otherwise return emtpy string.

Default chart

Introduction

In Portal homepage, by default there is no chart. But you can create your default chart.

Customization

Create an override which overrides sub process `DefaultChart` in Portal Kit. This sub process return a list of default charts. Follow instruction to create charts.

Change password process

Introduction

In Portal, the `Change password process` helps users to change their current password, you can customize this process to add more handling when user change password.
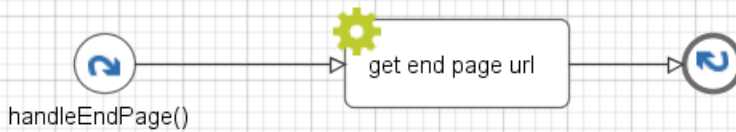
Customization

Create a callable sub process in your project with the `changePassword(String,String)` signature, make sure this signature is unique in your application. It must return an enums `ChangePasswordStatus` and the `message` showing to user (you can override this process in PortalKit):

changePassword(String,String)
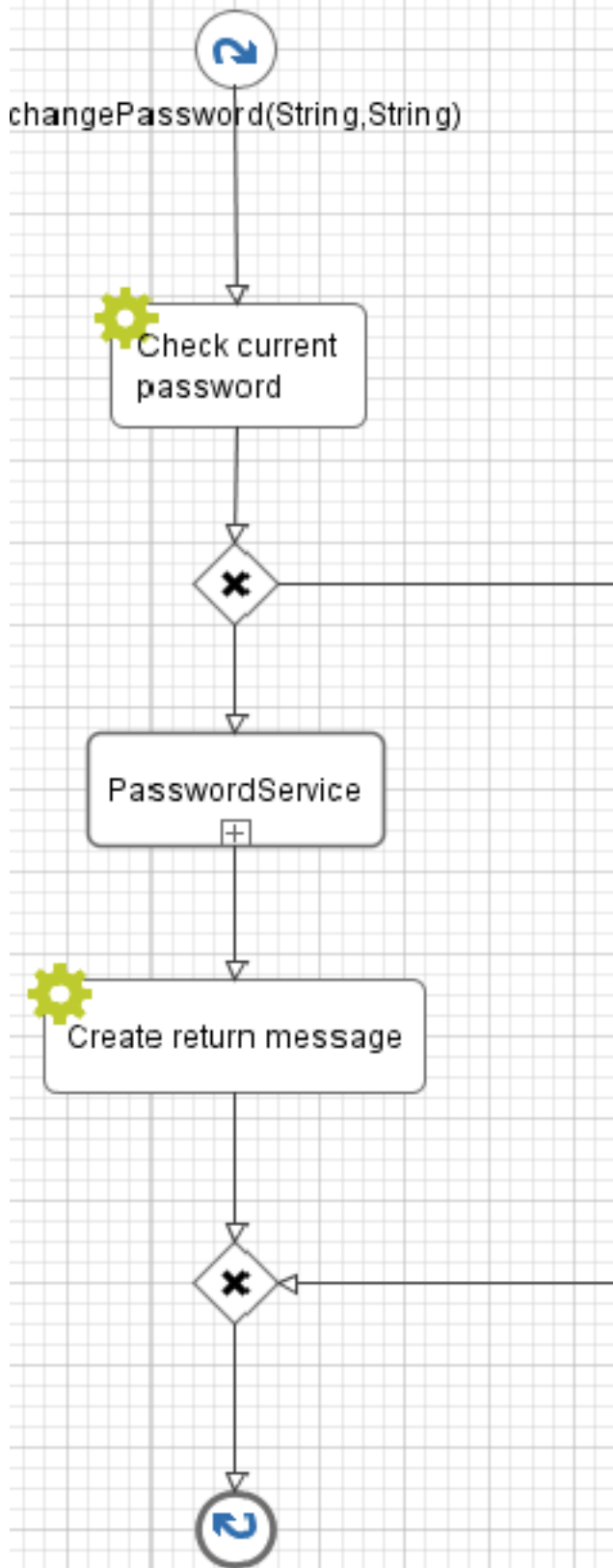
Overwrite this process to add m

INPUT:
- current password
- new password

OUTPUT:
- ChangePasswordStatus (OK,
- message

Check current password

PasswordService

Create return message

Logout process

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 the `Logout` process to customize the logout behavior.

5. Create a callable sub process in your project with the `getLogoutPage()` signature to customize the page which will be redirect to after logout, default is Portal home page. Make sure this signature is unique in your application.

Express end page

Introduction

When the last task of Axon Express finish, express end page will be displayed. You can customize this by provide your own page.

Customization

1. Create a new UI and start link of the new end page.
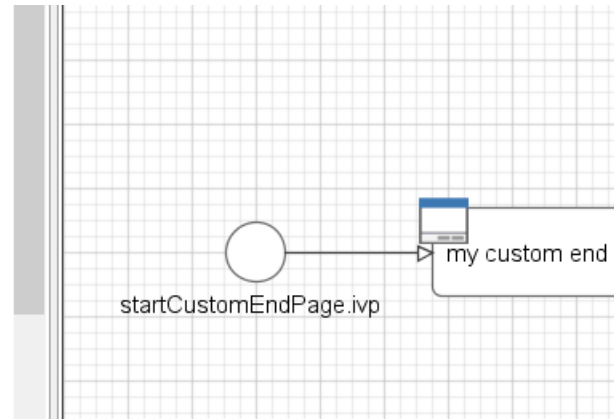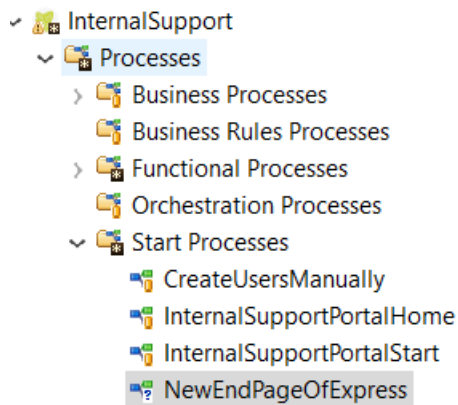
2. Create a callable sub process in your project with the `handleEndPage()` signature, make sure this signature is unique in your application. It must return start link of new end page you define in step 1.

```
HOW TO CREATE CUSTOM END PAGE FOR EXPRESS PROCESS

import ch.ivy.addon.portalkit.service.ProcessStartCollector;

ProcessStartCollector collector = new ProcessStartCollector(ivy.wf.getApplication());
String ourNewEndPageFriendlyRequestPath = "Start Processes/NewEndPageOfExpress/startCustomEndPage.ivp";
in.callbackUrl = collector.findLinkByFriendlyRequestPath(ourNewEndPageFriendlyRequestPath);

OUT: callbackUrl : String
```

Navigate back

- When a task finish, Portal will navigate back to previous page. For example if a task is started from homepage, then go back to homepage. In case task is started from task list, then go back to task list after finish.

- Developer can also apply this feature to their own button e.g Cancel a task.

- Just call `PortalNavigator.navigateToPortalEndPage()` in your button function
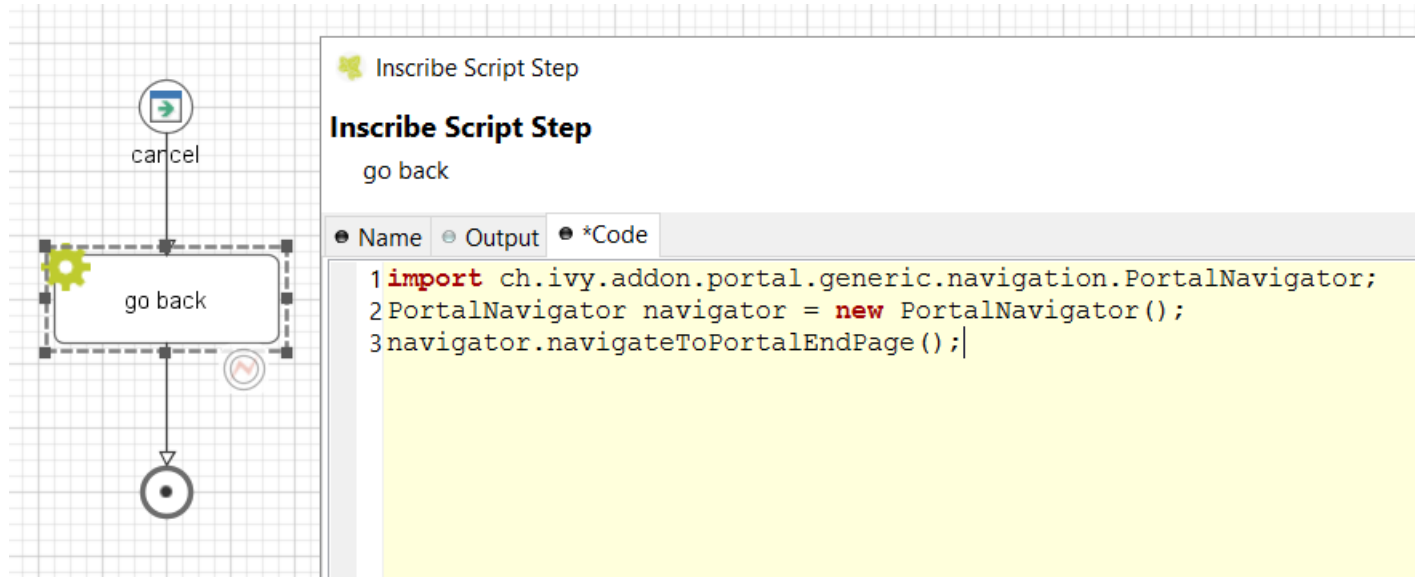
Hide technical stuffs

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)

Hide technical cases

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

Tasks belong to the technical case is considered as technical tasks and should be hide as well.

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

> **Tip**
>
> Use the utility methods of Portal:
>
> - Set property: ch.ivy.addon.portalkit.util.CaseUtils.setHidePropertyToHideInPortal(ICase)
>
> - Remove property: ch.ivy.addon.portalkit.util.CaseUtils.removeHidePropertyToDisplayInPortal(ICase)

Additional case details page

Introduction

The additional case details page shows all custom fields of a case by clicking on "Show details" link in case details.

You can customize this page for each case by providing a relative URL to case.

Customization

1. Create a new additional case details UI and a start process which will display the new UI.
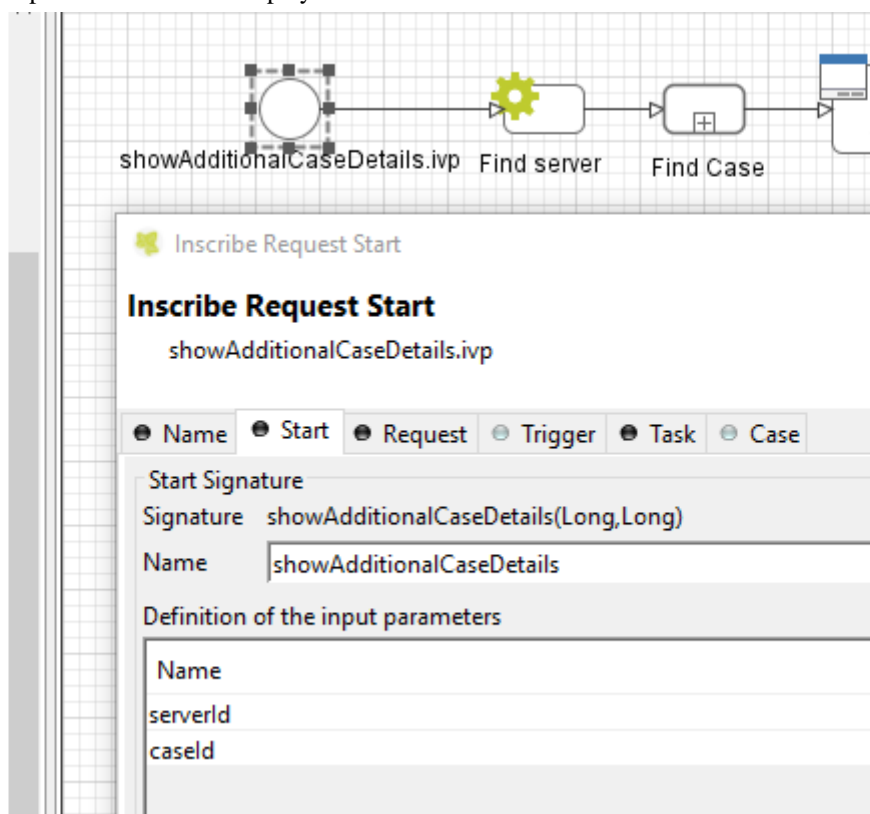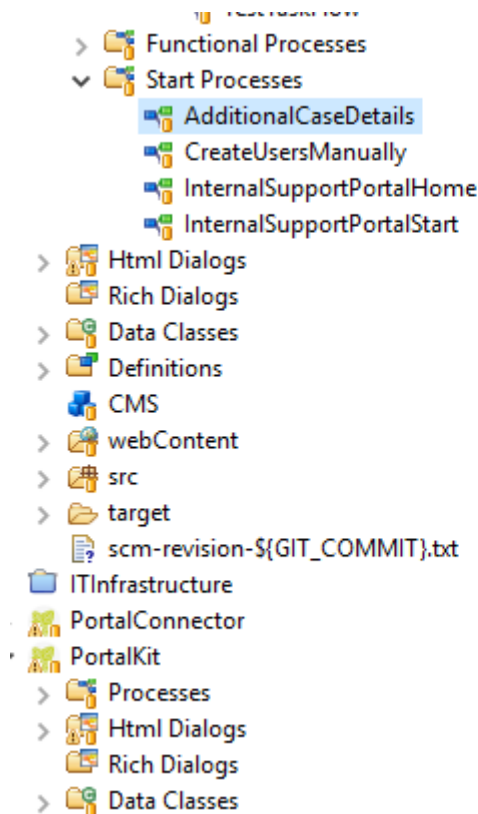
2. Store the URL of start process in "CUSTOMIZATION_ADDITIONAL_CASE_DETAILS_PAGE" additional property of case. You could use `SetAdditonalCaseDetailPage.mod` callable process, and input the friendly URL of process as parameter.

Inscribe Subprocess Call

Inscribe Subprocess Call

SetAdditonalCaseDetailPage

| ⊖ Name | ⊜ Process call | ⊖ Output |

Process start

Process: Functional Processes/SetAdditonalCaseDetailPage

Start: ▶ call(String)

*

Attribute	Type	Expression
⊟ ◉ param	<...>	
⬤ linkToAddtionalCaseDetailPage	String	"Start Processes/AdditionalCaseDetails/showAdditionalCaseDetails.ivp"

Tip

If you want to redirect user to external link, simply store that external link to "CUSTOMIZATION_ADDITIONAL_CASE_DETAILS_PAGE" additional property of case.

Inscribe Script Step

Inscribe Script Step

| ⊖ Name | ⊜ Output | ⊜ Code |

```
import ch.ivy.addon.portalkit.enums.AdditionalProperty;

ivy.case.setAdditionalProperty(AdditionalProperty.CUSTOMIZATION_ADDITIONAL_CASE_DETAILS_
```

Global Search Result

Global Search Result
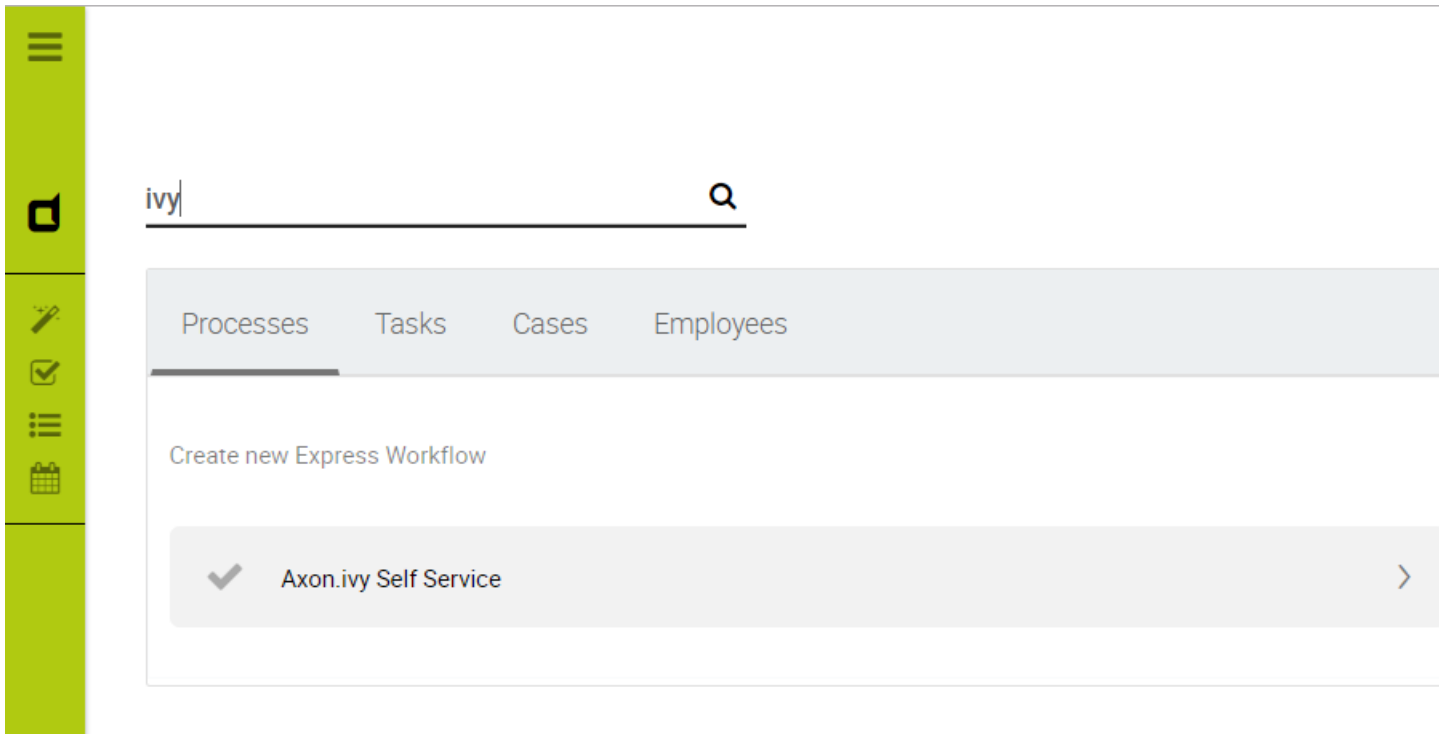
Follow these steps to customize the global search page:

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. Create the customized search data model extends `SearchResultsDataModel`, and override the `search` method to filter your objects.

> 💡 **Tip**
>
> Note: it is recommended that lazy loading or pagination should be applied for custom tabs to have a good performance.

5. Create the HTML dialog using `/layouts/SearchResultsTemplate.xhtml`, recommended to copy the `SearchResults` HTML dialog in PortalTemplate. In dataclass, change the `dataModel` to the above one, and in logic, also cast it.

Inscribe User Dialog Start

Inscribe User Dialog Start

start(SearchResultsDataModel)

| Name | Start | Output | Result |

Call Parameters

*

Attribute	Type	Expression
□ out	CustomizedSearchResultsData	
activeTabIndex	Number	param.activeTabIndex
⊞ dataModel	CustomizedSearchResultsDataModel	param.searchResultsDataModel as ch.ivyteam.ivy.project.portal.exam
⊞ emp	Employee	

6. Define the `custom-search` section to add your customized tabs:

CustomizedSearchResults.xhtml ⊠

```
 1 <html xmlns="http://www.w3.org/1999/xhtml" xmlns:f="http://xmlns.jcp.org/jsf/core"
 2   xmlns:h="http://xmlns.jcp.org/jsf/html" xmlns:ui="http://xmlns.jcp.org/jsf/facel
 3   xmlns:ic="http://ivyteam.ch/jsf/component" xmlns:p="http://primefaces.org/ui"
 4   xmlns:pe="http://primefaces.org/ui/extensions" xmlns:pm="http://primefaces.org/
 5
 6 <h:body>
 7   <ui:composition template="/layouts/SearchResultsTemplate.xhtml">
 8     <ui:define name="custom-search">
 9       <p:tab title="Employees">
10         <p:dataTable id="employee-table" value="#{data.dataModel.employees}" var="
11           <p:column headerText="First Name" sortBy="#{emp.firstName}" filterBy="#{
12             <h:outputText value="#{emp.firstName}" />
13           </p:column>
14           <p:column headerText="Last Name" sortBy="#{emp.lastName}" filterBy="#{em
15             <h:outputText value="#{emp.lastName}" />
16           </p:column>
17           <p:column headerText="Country" sortBy="#{emp.country}" filterBy="#{emp.c
18             <h:outputText value="#{emp.country}" />
19           </p:column>
20           <p:column headerText="Details" style="text-align: center">
21             <p:commandLink id="detail-action" actionListener="#{logic.openDetails
22               title="#{ivy.cms.co('/Labels/Details')}">
23               <i class="fa fa-search fa-lg" />
24             </p:commandLink>
25           </p:column>
26         </p:dataTable>
27       </p:tab>
28     </ui:define>
29   </ui:composition>
30 </h:body>
31 </html>
```

7. Override the `OpenPortalSearch` callable process and change the HTML dialog to your customized one.

Tip

Hint: refer to the example in the PortalExamples project: CustomizedSearchResultsDataModel.java, CustomizedSearchResults HTML dialog, OpenPortalSearchOverride callable process

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

Global settings

Global settings for Portal can be set in Settings tab. All available settings with their default value and description are listed in this place.

Admin settings

Servers	Applications	Settings

Key	Default value	Current value
CLIENT_SIDE_TIMEOUT		
ENABLE_CHAT	false	false
ENABLE_SCRIPT_CHECKING_FOR_UPLOADED_DOCUMENT	false	false
EXPRESS_END_PAGE	true	true
HIDE_CHANGE_PASSWORD_BUTTON	false	false
HIDE_LOGOUT_BUTTON	false	false

You can edit value for a specific setting

You can also get back the default value for each setting by clicking reset button on each row, or reset all values of all settings by clicking `Restore All To Defaults` button

Setup multi portals

Warning

Multi applications on multi engines is not officially supported, only an experimental mode is available.

Important

There is 2 ways to configure portals: `two levels` and `single level`

• Two levels portal
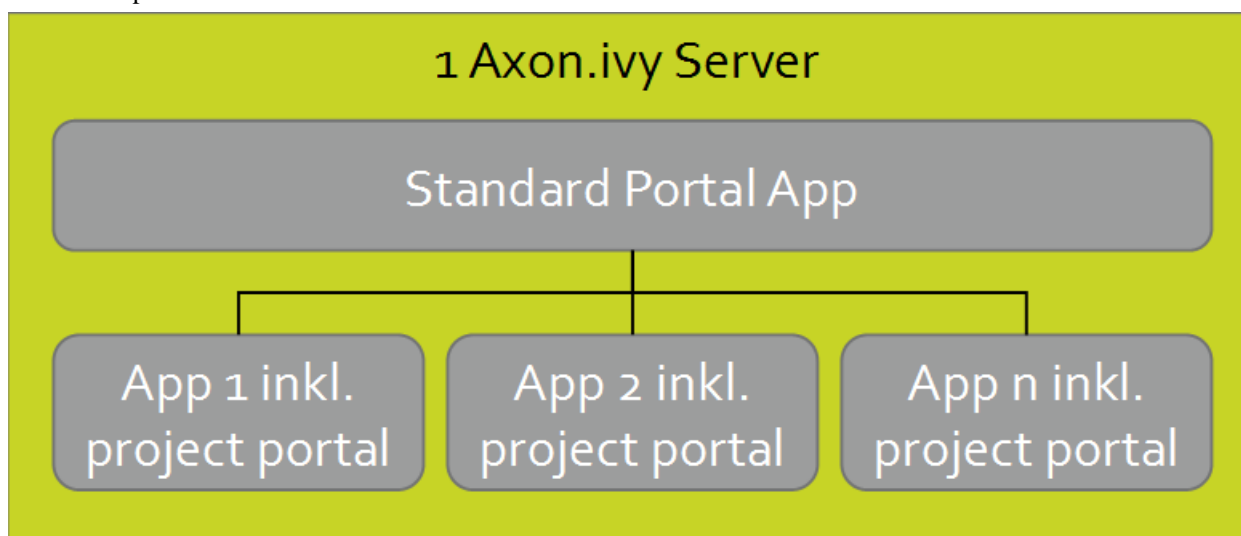
• Used for related applications where we need an overview of all tasks and cases.

• There is default application Portal.

- Create new applications: App1, App2, App3... Deploy portal (kit, template ..) to new applications.

- Configure multi-apps Portal on single server: login by Portal Admin. Configure 1 server then configure applications: App1, App2, App3...

- Note that, `do not` add default Portal application, the default application (Dashboard) is reserved for displaying all tasks/cases... from all configured applications.

- Dashboard menu only visible when login user exists in Portal application.

- Single level portal
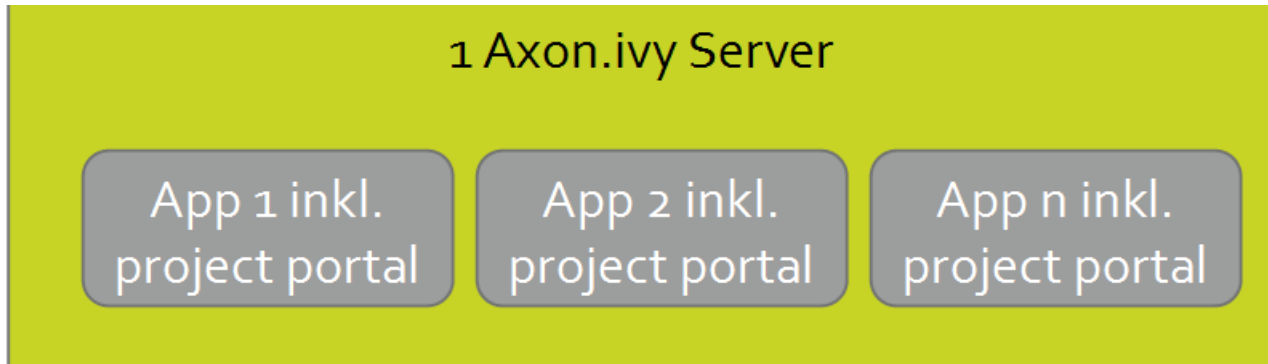
- Used for independent applications.

- Disable the default application Portal.

- Create new applications: App1, App2, App3... Deploy portal(kit, template ..) to new applications.

- Configure multi-apps Portal on single server: login by Portal Admin. Configure 1 server then configure applications: App1, App2, App3...

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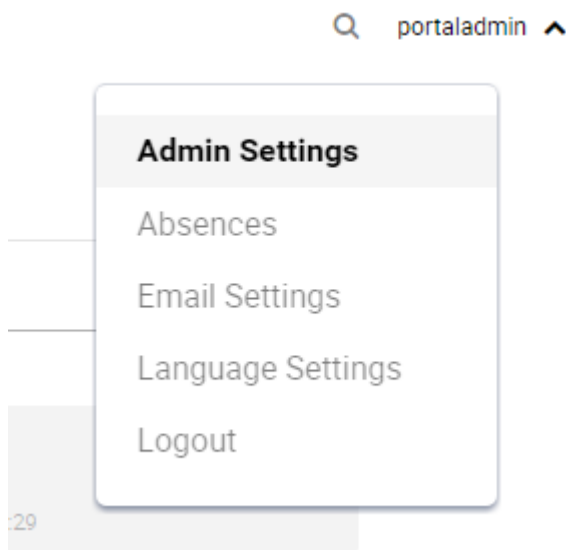
> **⚠ Important**
>
> If your application does not use templates of `PortalTemplate` project, you have to create a page and use `AdminSettings` component inside.

> **💡 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.

Add new server

Server name *	Main server
Server path *	http://localhost:8081/ivy/ws/designer/PortalConnector
	⚠ Portal data will be transferred in plain text. We recommend HTTPS.
External Host	<AUTO_DETECT>
Active	■
Authentication type	HTTP Basic
Username *	AxonIvy
Password *	••••••••

[Test connection]   [Ok]

> **🗒 Note**
>
> - `Server  path` must be the link of the `PortalConnector` application (e.g. `http://localhost:8080/ivy/ws/System/PortalConnector`). Note that the port in the link is the web server port
>
> - 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` (account to login to administration tool) 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.
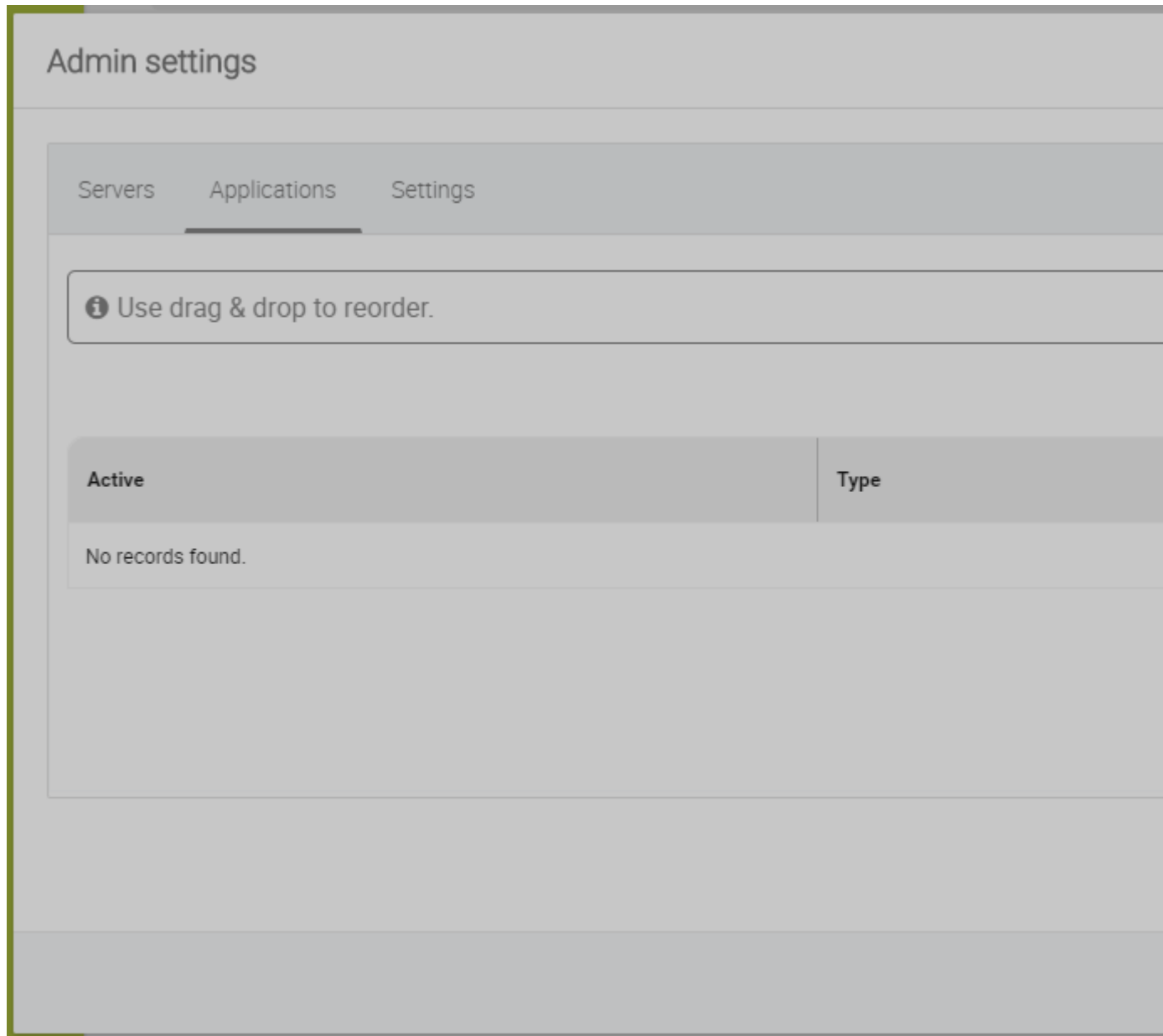
Tip

- `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 selecting the application on the application menu. It could be an absolute link (e.g. http://10.123.1.30:8000/ivy/pro/.../PortalStart.ivp) or relative link (e.g. /ivy/pro/.../PortalStart.ivp). If your application could be accessed from multiple domains, use relative link so that you can access the link from different domains.

- `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.
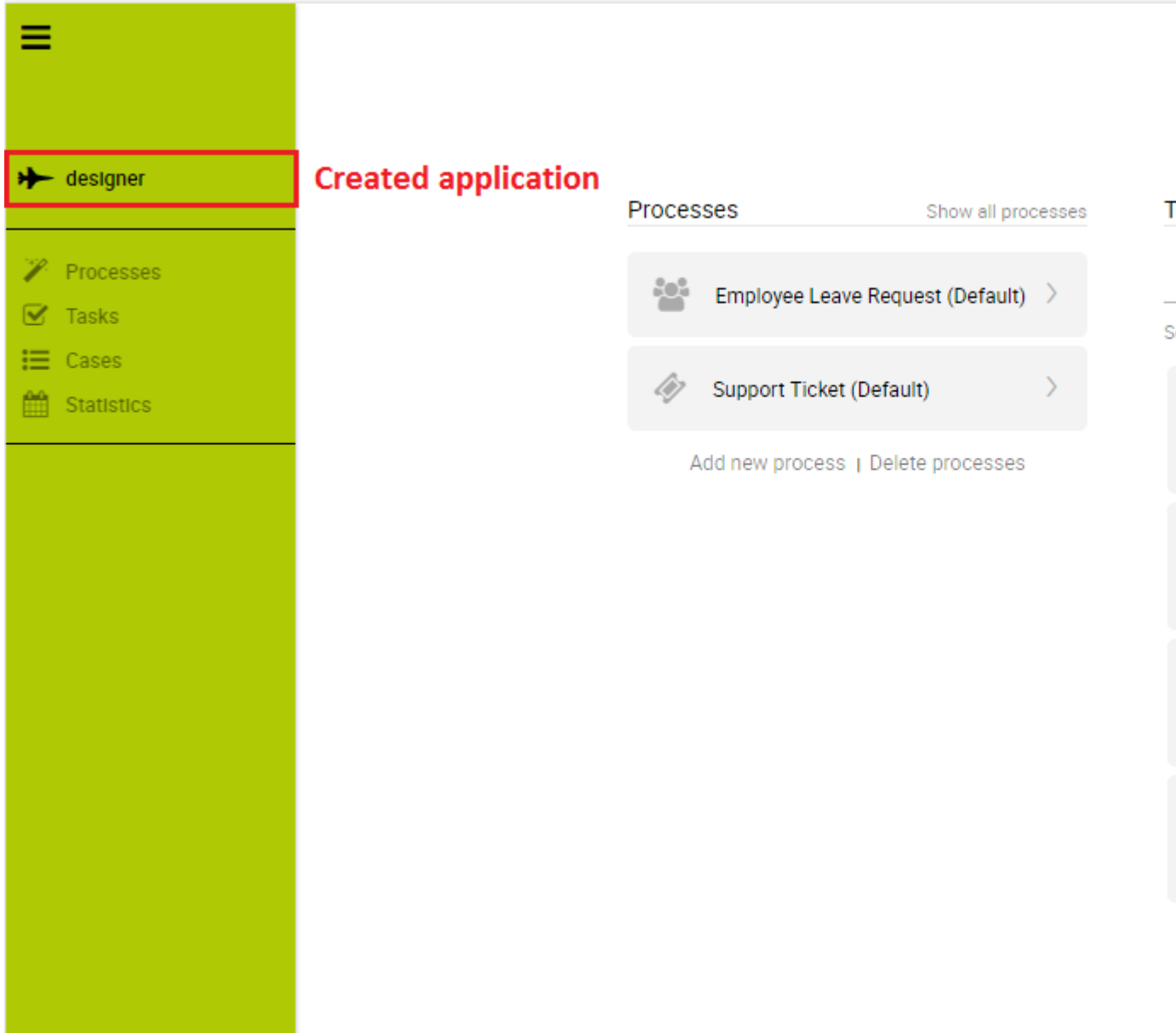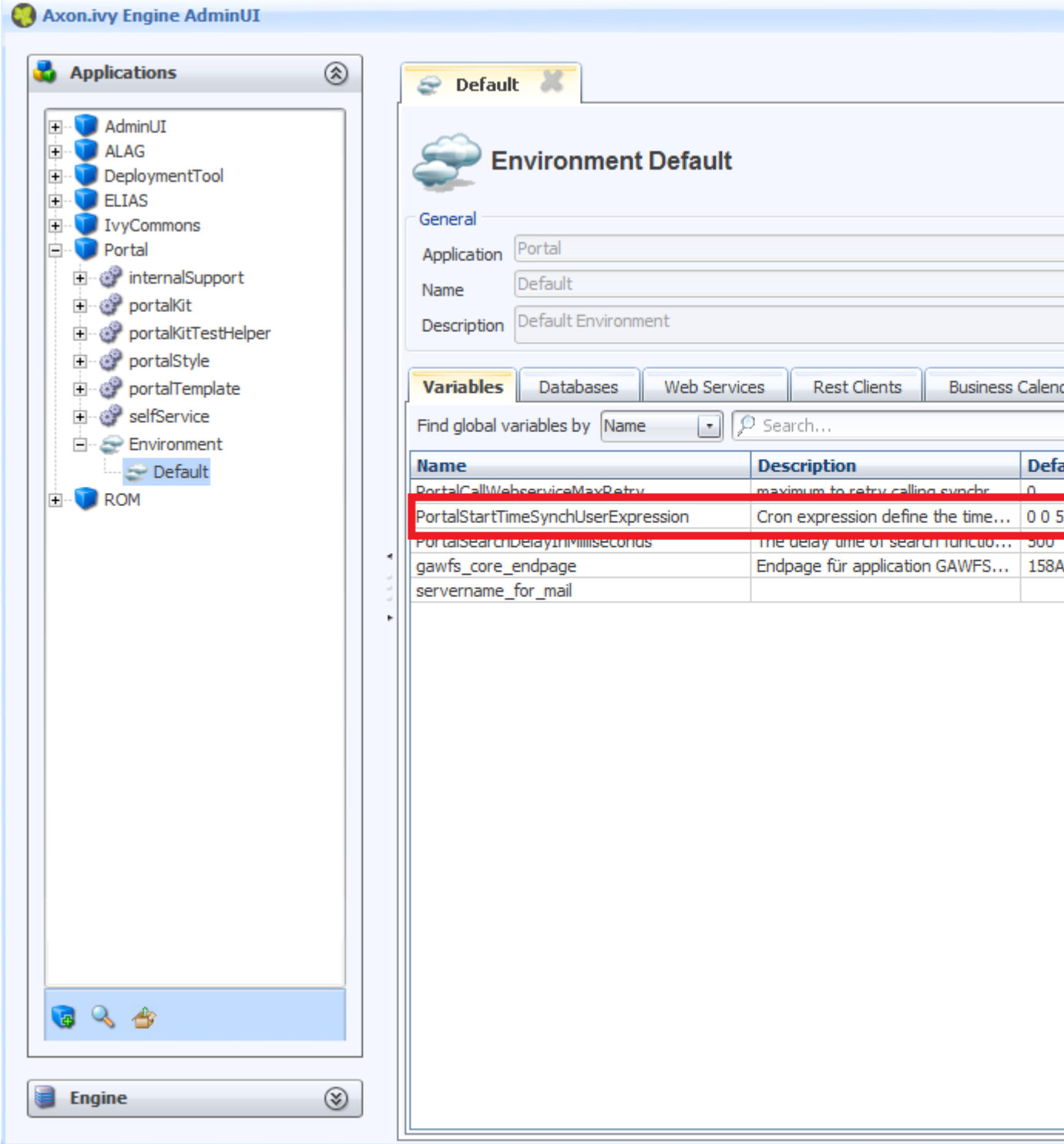
Tip

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

Processes

User Favorites

You have not defined any favorites.

Would you like to Add new favorite process?

Application Favorites

Sort by name

Inspect Resource

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

Pr

Absences

☐ Display absences in the past

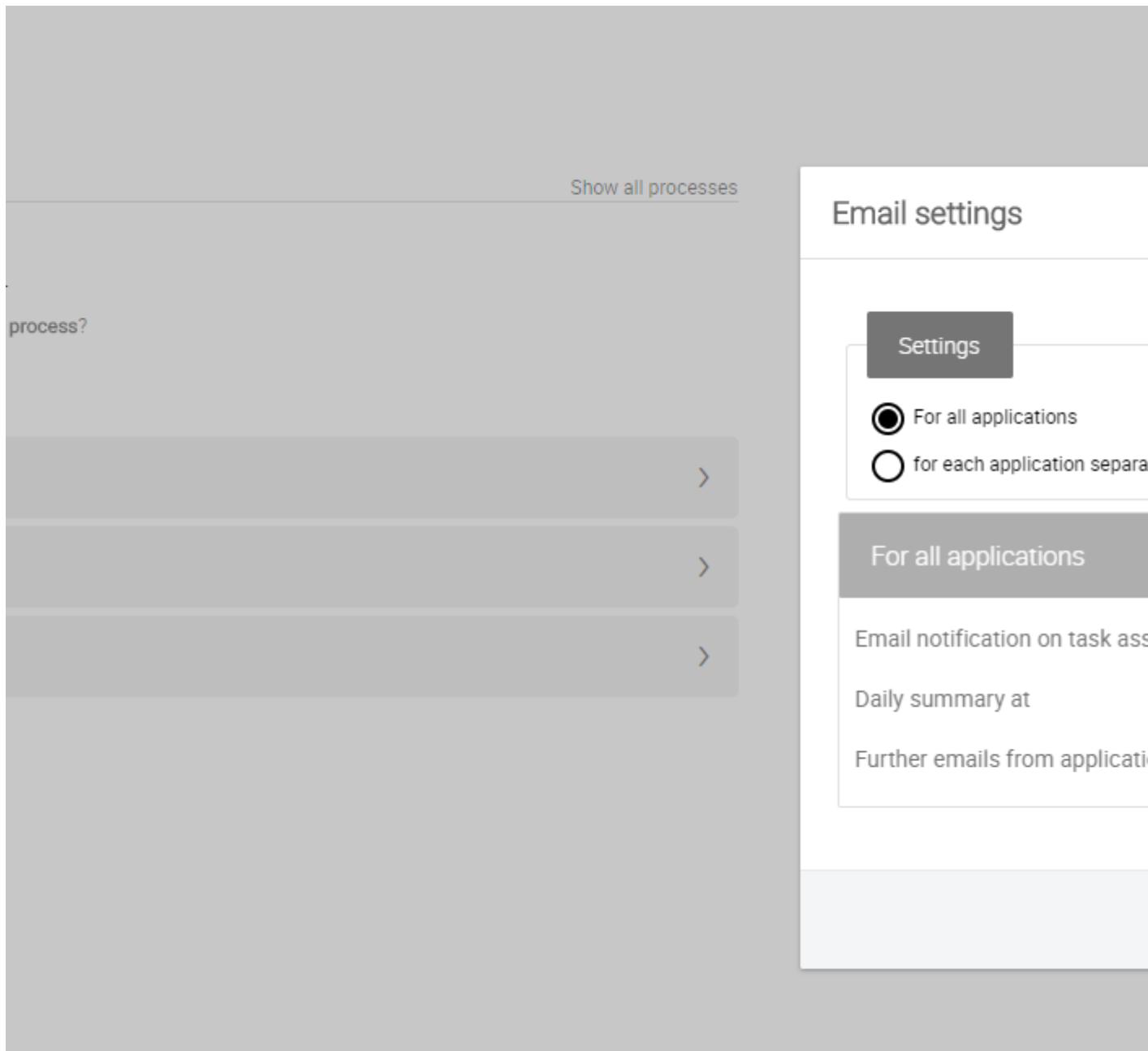| Name ⇕	From ⇕	Until ⇕
No records found.		

- User cannot set deputies for hidden roles.

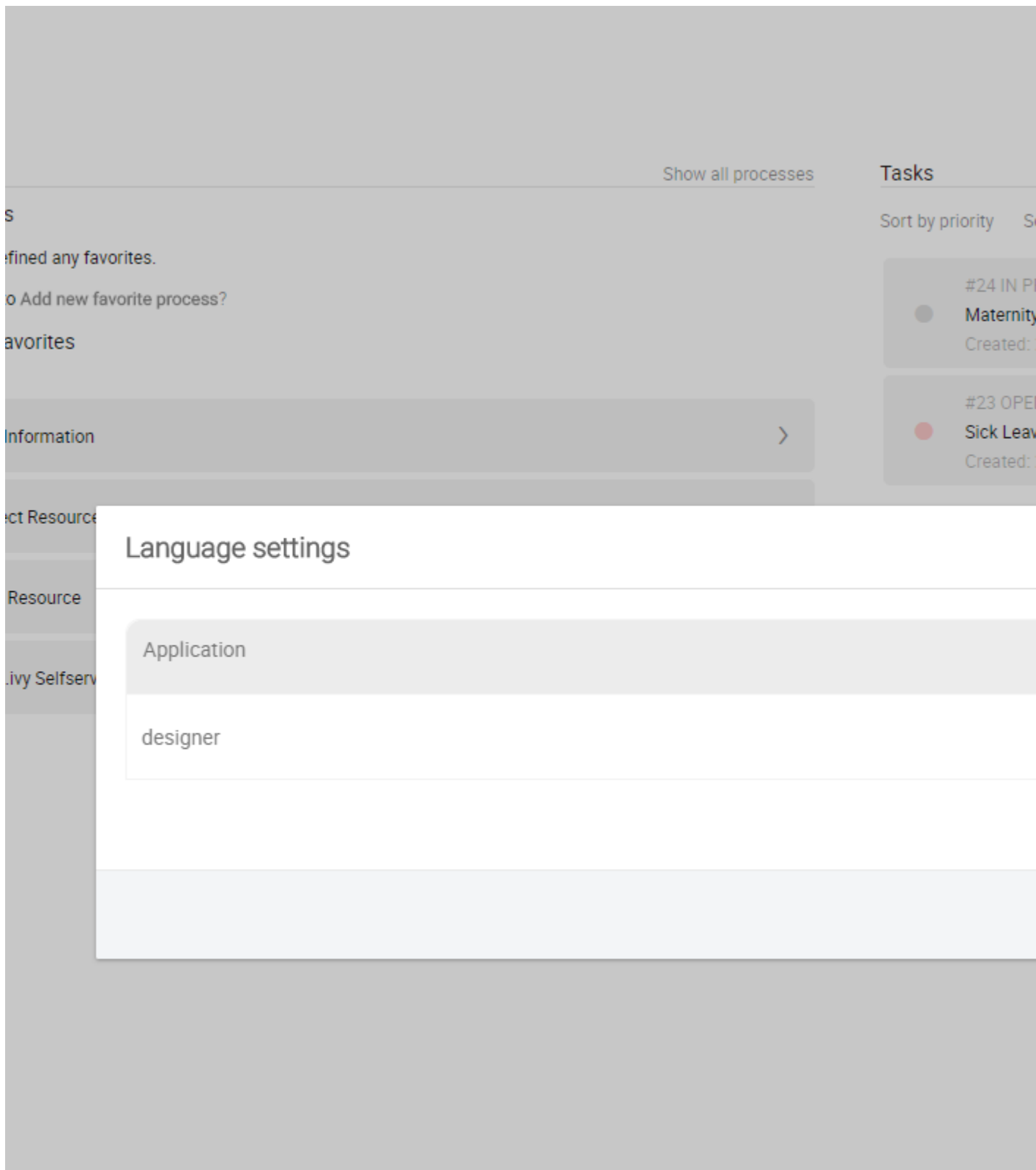- All the items will be saved after click Save button.

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.

- There are 2 types of emails notification: Ivy email notification sent when task is assigned (More information about the email notification can be found here) and other emails sent from processes of customer project.

- 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).

Show all processes

Tasks

Sort by priority

s...fined any favorites.

...o Add new favorite process?

...avorites

#24 IN P...
Maternity
Created:

Information   >

#23 OPE...
Sick Leav...
Created:

...ect Resource

Language settings

Resource

Application

.ivy Selfserv...

designer

- For multiple languages, the CMS key `/AppInfo/SupportedLanguages` must exist in your application. From Portal 7.1, this CMS entry is moved to Portal Style. It 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

- To add new language to Portal, what you have to do is

  - Add new language locale to cms entry of Portal Style `/AppInfo/SupportedLanguages`

  - Export all CMS entries of Portal Style to excel file

  - Add translation of new language for all CMS entries

  - Import file excel back, then redeploy Portal Style

  - This is sample how to add new Spanish to portal

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.